Efficient Microservice Deployment in Kubernetes Multi-Clusters through Reinforcement Learning

José Santos*, Mattia Zaccarini†, Filippo Poltronieri†, Mauro Tortonesi†,
Cesare Stefanelli†, Nicola Di Cicco‡, Filip de Turck*
* Ghent University - imec, IDLab, Department of Information Technology, Gent, Belgium
Email: {josepedro.pereiradossantos, filip.deturck}@UGent.be
† Distributed Systems Research Group, University of Ferrara, Ferrara, Italy
Email: {filippo.poltronieri, mauro.tortonesi, mattia.zaccarini}@unife.it
‡ Department of Electronics, Information, and Bioengineering (DEIB), Politecnico di Milano, Italy
Email: nicola.dicicco@polimi.it

***Abstract*—Microservices have revolutionized application deployment in popular cloud platforms, offering flexible scheduling of loosely-coupled containers and improving operational efficiency. However, this transition made applications more complex, consisting of tens to hundreds of microservices. Efficient orchestration remains an enormous challenge, especially with emerging paradigms such as Fog Computing and novel use cases as autonomous vehicles. Also, multi-cluster scenarios are still not vastly explored today since most literature focuses mainly on a single-cluster setup. The scheduling problem becomes significantly more challenging since the orchestrator needs to find optimal locations for each microservice while deciding whether instances are deployed altogether or placed into different clusters. This paper studies the multi-cluster orchestration challenge by proposing a Reinforcement Learning (RL)-based approach for efficient microservice deployment in Kubernetes (K8s), a widely adopted container orchestration platform. The study demonstrates the effectiveness of RL agents in achieving near-optimal allocation schemes, emphasizing latency reduction and deployment cost minimization. Additionally, the work highlights the versatility of the DeepSets neural network in optimizing microservice placement across diverse multi-cluster setups without retraining. Results show that DeepSets algorithms optimize the placement of microservices in a multi-cluster setup 32 times higher than its trained scenario.**

***Index Terms*—Kubernetes, Orchestration, Microservices, Reinforcement Learning, Resource allocation**

I. Introduction

In the last few years, containers have revolutionized application deployment and life-cycle management [1]. Applications evolved from a single monolith to a complex composition of loosely-coupled microservices, resulting in remarkable improvements in deployment flexibility and operational efficiency [2]. However, managing these modern, microservice-based applications requires extremely sophisticated orchestration solutions. The emergence of novel paradigms such as Fog Computing [3] and Edge Computing [4] and new use cases (e.g., autonomous vehicles [5], virtual reality services [6]) demanding computing resources closer to devices and end-users adds further complexity and puts even more pressure on popular cloud infrastructures (e.g., Amazon ECS, Kubernetes (K8s), and Red Hat OpenShift). The lack of efficient multi-cluster management features has hindered the deployment of these applications due to their stringent requirements (e.g., low latency, high bandwidth) [7].

Current literature (e.g., [8], [9]) focuses mainly on a single-cluster scenario where applications are deployed based on the available computing resources. Existing works studying multi-cluster orchestration are still scarce, however, the scheduling problem becomes significantly more challenging in these scenarios. For instance, the orchestrator has to determine where to deploy each microservice and decide whether to place all its instances in a single cluster or distribute them across multiple ones. An efficient strategy is crucial to choose when to distribute microservice instances across different clusters, aiming to enhance resource utilization and decrease the application's latency. Only a few works (e.g., [10]–[13]) address multi-cluster orchestration by proposing either theoretical formulations or heuristic-based methods that are typically evaluated via simulations or small testbeds, making their applicability in popular platforms difficult.

This paper strives to tackle the orchestration challenge in a multi-cluster infrastructure by proposing an Reinforcement Learning (RL)-based Global Topology Manager (GTM) for efficient application deployment in K8s, a widely adopted container orchestration platform [14]. An RL environment has been developed to provide a scalable and cost-effective solution to train RL agents for the multi-cluster orchestration problem. Numerous works [15], [16] reported that online training in RL is significantly expensive for complex tasks in the network management domain. It allows training an RL agent with a valuable dataset collected over a specific time period (e.g., several days) or by creating a realistic simulation-based environment. In addition, this work leverages the capabilities of the open-source project Kubernetes Armada (Karmada) [17], which acts as a control-plane solution for managing multi-cluster applications across hybrid cloud settings. This study aims to make Karmada's behavior more adaptive and intelligent than its current one by developing new components and novel orchestration policies to accomplish more efficient multi-cluster scheduling. The main contributions of the paper are the following:

- ***gym-multi-k8s* framework**: Implementation of an offline RL-based framework for proper scheduling of microservice-based applications in multi-cluster scenarios. The proposed framework[1] has been open-sourced, allowing researchers to evaluate their scheduling ideas. Sec. IV presents the RL-based GTM, including observation state, action space, and the reward functions. The approach addresses multi-cluster orchestration focused on two opposing strategies: reducing deployment costs and minimizing latency.

- **Evaluation with microservice-based applications**: The evaluation considers a real-world application named *Cloud2Edge (C2E)*. Experiments in multiple multi-cluster K8s setups show that the RL-based GTM can find near-optimal allocation schemes for the selected strategy while achieving a high percentage of accepted requests.
- **RL generalization**: The paper also evaluates the generalization potential of the DeepSets neural network architecture by applying it to different problem sizes without retraining. Results show that the RL-based GTM can optimize microservice placement in a multi-cluster scenario 32 times higher than its trained setup (Sec. VI).

The remainder of the paper is organized as follows: the state-of-the-art on multi-cluster orchestration is discussed in the next section. Sec. III highlights the importance of efficient multi-cluster orchestration, describing the proposed approach focused on its integration with the Karmada open-source project. Sec. IV details the RL-based GTM orchestration solution, including its observation and action spaces. Sec. V describes the evaluation setup, followed by the results in Sec. VI. Sec. VII concludes this paper.

II. Related Work on Multi-Cluster Orchestration

Cloud orchestration has been an active research topic in recent years. Several studies have proposed scheduling policies to optimize container allocation in popular cloud platforms. This section reviews the most relevant works on application scheduling, mainly focusing on orchestration methods for multi-cluster infrastructures. The awareness of the scheduler plays a crucial role in these scenarios since it will allow more refined scheduling decisions in order to improve the performance and responsiveness of the system.

Heuristics and Theoretical Formulations are vastly explored in the literature [10]–[13]. For example, in [12], the authors propose three task scheduling algorithms for heterogeneous cloud environments. These algorithms find the most suitable location for each task while optimizing makespan, resource utilization, and throughput. Also, in [13], S. Qin et al. define a reliability-aware multi-objective algorithm demonstrating the effectiveness of the approach in comparison with other analogous algorithms in solving multi-objective workflow scheduling problems in multi-cloud systems. The main drawback of these methods is that they are designed and developed for a specific platform, reducing its potential applicability in practice.

Scheduling Optimizations in K8s is an active topic lately [17]–[20]. Most efforts aim to improve resource efficiency [18], [20] or reduce the application response time by focusing on the network latency between geo-distributed clusters [19], showing the benefits of network-aware placement. Karmada [17] scheduling focuses on deployment preferences specified by cloud administrators. Microservice replicas can all be deployed into a single cluster or distributed across different clusters. If the spreading policy is selected, a simplified cluster resource modeling is applied to decide how to spread replicas across the clusters. The proposed RL GTM aims to find the optimal decision based on the current status of the infrastructure, without cloud administrators having to decide beforehand how they prefer to deploy these replicas across their infrastructure.

RL algorithms have been proposed in recent years as an alternative to current heuristics [21]–[23]. These techniques aim to teach an agent how to deploy microservices in a multi-cluster setting by giving it the current status of the infrastructure after each applied action. RL approaches are typically robust to dynamic demands since the algorithm adjusts the model parameters if any notable event occurs (i.e., online learning). Nonetheless, the main drawback of RL techniques is the high execution time to converge to a stable model and thus trigger inefficient scheduling actions during the learning period. The proposed GTM enables a more scalable and cost-effective solution for RL training via its *gym-multi-k8s* framework.

Table I compares all works referred to in this section. These methods have been classified based on their main characteristics. Nonetheless, the quantitative assessment is challenging since these techniques are designed for a particular system or virtualization technology. To the best of our knowledge, no standard testing framework for multi-cluster scheduling exists. In our previous work, numerous studies have proposed scheduling optimizations for a single cluster addressing microservice-based applications (e.g., [24]). This paper builds on those efforts to propose an RL-based approach focused on multi-cluster scenarios. The work differs from the current literature by addressing multiple factors (e.g., resource efficiency, network latency) for application scheduling and its potential integration with the Karmada project. In addition, this paper addresses the need for RL generalization by evaluating the DeepSets neural network architecture. The aim is to teach an agent in a small-scale scenario and directly apply the learned policy to large-scale setups.

III. Toward efficient multi-cluster orchestration

A. System Overview

This paper envisions a multi-cluster scenario as an aggregation of multiple heterogeneous K8s clusters managed singularly by a control-plane entity (Fig.1). It will enable a dynamic methodology for developing, deploying, and managing all the clusters distributed between all the computing

[1] https://anonymous.4open.science/r/gym-multi-k8s-8643

TABLE I: Comparison of existing works related to multi-cluster application scheduling.

Authors	Year	Virtualization	Dimension	Main Focus	Generalization	Evaluation Method
Bhamare, D. el al. [10]	2017	VMs	N	R & NL	✕	S
Guerrero, C. el al. [11]	2018	VMs & C	MO	R & NL	✕	S
Panda, S. K. et al. [12]	2019	VMs	MO	R & M	✕	S
Qin, S. et al. [13]	2023	VMs	MO	RL	✕	S
Lee, S et al. [18]	2020	C	R	R	✓	K8s
Rossi, F, et al. [19]	2020	C	N	NL	✓	K8s
Tamiru, M. A. et al. [20]	2021	C	R	R	✓	K8s
Karmada [17]	2020	C	L	R	✓	K8s
Zhang, Y. et al. [21]	2020	N/A	R	R & NL	✕	S
Shi, T. et al. [22]	2021	VMs	L	R & NL	✕	S
Suzuki, A. et al. [23]	2023	N/A	N	R & NL	✕	S
Our RL-based GTM	2023	C	N + R	R & NL	✓	RL

Virtualization: VMs = Virtual Machines, C = Containers, N/A = no clear distinction.
Dimension: N = Network-aware, MO = Multi-objective, L = Location-aware, R = Resource-aware.
Main Focus: R = Resources, RL = Reliability, NL = Network Latency, M = Makespan.
Generalization: ✓= addressed, ✕= not considered.
Evaluation Method: K8s = Kubernetes, S = Simulation, RL = RL environment.

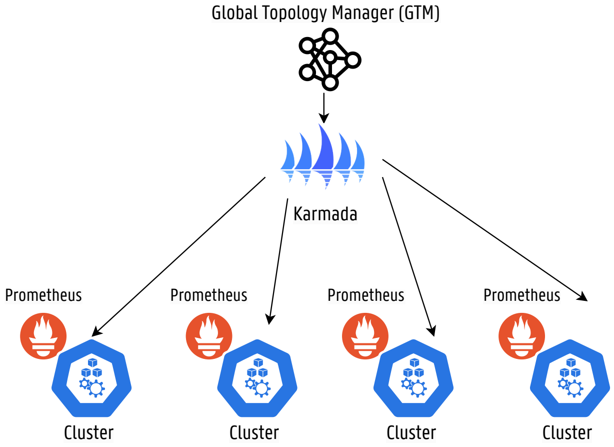


Fig. 1: Envisioned Global Topology Manager (GTM) for Multi-Cluster K8s Orchestration.

layers in the cloud continuum (i.e, edge, fog, and cloud). The proposed architecture employs K8s at every layer of the cloud continuum due to its various heterogeneous distributions such as MicroK8s, Kubedge, and K3s [25]. With a large variety of managed K8s setups, it is logical to consider this scenario as a federation of multi-cluster environments. Thus, our approach adopts Karmada [17] as a Federation Layer, a Cloud Native Computing Foundation (CNCF) project developed in continuation of Kubernetes Federation (KubeFed) consisting of a control-plane management system capable of deploying cloud-native applications across multiple K8s clusters. Its main objective is to provide autonomous management for multi-cluster applications in multi-cloud and hybrid cloud scenarios, with key features such as centralized management, high availability, failure recovery, and traffic scheduling [17]. The main reason to choose Karmada as our federation layer is that it seems a more mature solution than others available as open-source, such as Open Cluster Management (OCM). For instance, Karmada can already exploit the K8s Native API in the resource templates, making it easier to integrate with the plethora of existing K8s tools and extending it with plugins.

B. Karmada Integration

Despite these aspects and considerations, there is still plenty of room for improvement in the standard behavior of Karmada, especially regarding application scheduling. Karmada supports two modes for deploying replicas in a K8s cluster: *duplicated* and *divided*. The first mode implies deploying the number of requested instances in all clusters, and the second strategy splits the number of requested replicas across all the clusters. Depending on the strategy favored by the cloud administrator, extra options (e.g., *ClusterAffinities*, *LabelSelectors*) can be inserted to the *PropagationPolicy* object to fine-tune the behavior of the Karmada scheduler. Karmada decides to divide replicas mainly by the resource availability of each cluster, but typically does not consider fragmented resources leading to suboptimal scheduling. Also, the duplication policy typically leads to resource wastage since the demand is lower than the number of reserved resources. The proposed GTM aims to automate microservice deployment in multi-cluster scenarios by finding an optimal balance between deploying the number of requested replicas in a single cluster or distributing them across several ones. Two opposing orchestration policies have been developed focused on resource efficiency and network latency to find near-optimal multi-cluster placement for different application scenarios.

The GTM communicates directly with Karmada, influencing the operations performed by the Karmada Controller Manager. Consequentially, the Karmada Controller manager selects the correct controller that manages the corresponding resources of the underlying clusters through their API servers. For example, the Policy Controller monitors the deployed *PropagationPolicy* objects by creating *ResourceBinding* objects for each resource object of the group that matches the *ResourceSelector* field. The necessary deployment information for the GTM is given by extended *PropagationPolicy* objects, describing the requirements of the deployed services to permit the comprehensive evaluation of their performance requirements and adapt the resource allocation dynamically. In addition, Prometheus is applied as a monitoring agent since it provides a higher level of visibility into workloads, APIs, and distributed applications running in the cluster. The GTM takes advantage of monitoring information from Prometheus regarding cluster resource availability at every moment to make efficient scheduling decisions.

IV. Reinforcement Learning (RL)-based multi-cluster orchestration

A. Problem Overview - Efficient multi-cluster orchestration

In the last few years, RL has become an active research topic in networking [26], often applied to solve decision-making problems in which an agent learns to choose an action based on the current state of the network. The agent typically learns an optimal policy by receiving a reward for each applied action. This reward corresponds to the new observation state after applying the selected action. On the one hand, in microservice scheduling, the reward is positive if the action increases the cluster's or the application's performance (e.g., high resource usage, low response time). On the other hand, the agent receives a penalty (i.e., typically a negative reward) if the cluster's or application's performance degrades. Thus, the agent learns through repeated interactions with the environment and determines the inherent synergies between states, actions, and subsequent rewards. Based on our expertise, RL is well-suited for scheduling problems as the efficient multi-cluster orchestration addressed in this paper since it can learn a winning strategy based on a given goal, and applied for long-term decision-making in repeated scheduling problems. RL agents can adjust their action selection and achieve long-term objectives in complex situations by receiving adequate feedback (i.e., rewards). The following subsections describe the RL approach for solving the scheduling of microservices in a multi-cluster K8s environment.

B. Reinforcement Learning (RL) Environment

An OpenAI Gym-based framework [27] named *gym-multi-k8s* has been developed to train the RL-based GTM in a scalable and cost-efficient manner. The framework enables RL agents to learn how efficiently deploy microservices in multi-cluster scenarios. The environment consists of a discrete-event RL scenario to reenact the behavior of multiple deployment requests for a given microservice deployed via Karmada on several K8s clusters. The deployment requirements (e.g., CPU and Memory requests) and the number of available resources in each cluster are updated during training based on the scheduling actions of the agent. Sec. V shows the deployment requirements used for the RL environment based on a realistic microservice-based application to create near-real experiments. In addition, the proposed approach adopts the DeepSets methodology presented in [28], [29]. Deep RL methods based on Multi-Layer Perceptrons (MLPs) operate in fixed-length vector spaces, which cannot support variable input and/or output dimensionalities. In other words, for the microservice scheduling problem, if an MLP-based RL agent learns on a multi-cluster setup with four clusters, it cannot be directly applied to another multi-cluster scenario that manages eight clusters. Instead, DeepSets assume that inputs and outputs can be arbitrarily-sized sets, meaning that the learned policy by the RL agent is not bound to a fixed number of clusters. Because of this, a DeepSets-based RL agent can generalize its learned policy to different multi-cluster scenarios without retraining. The aim is that the proposed GTM, by applying DeepSets, generalizes well to problem sizes larger than training, which would be beneficial to cloud providers scaling their infrastructure by adding further computing power.

TABLE II: The structure of the Observation Space.

Set	**Metric**	**Description**
App	R	The number of requested replicas.
	ω_{cpu}	The CPU request of each replica.
	ω_{mem}	The memory request of the replica.
	Δ	The latency threshold of the request.
	T	The expected execution time of the request.
Cluster	Π_{cpu}	The cluster's cpu capacity.
	Π_{mem}	The cluster's memory capacity.
	Θ_{cpu}	The CPU allocated in the cluster.
	Θ_{mem}	The memory allocated in the cluster.
	δ_c	The latency of cluster c to cluster c_j.

TABLE III: The hardware configuration of each cluster based on Amazon EC2 On-Demand Pricing [30].

Cluster Type	Amazon Cost ($/h)	Cost (τ_c)	CPU	RAM
Cloud	t4g.2xlarge (0.2688)	16.0	8.0	32.0
Fog Tier 2	t4g.xlarge (0.1344)	8.0	4.0	16.0
Fog Tier 1	t4g.large (0.0672)	4.0	2.0	8.0
Edge Tier 2	t4g.medium (0.0336)	2.0	2.0	4.0
Edge Tier 1	t4g.small (0.0168)	1.0	2.0	2.0

C. Observation Space

Table II shows the observation space considered for the multi-cluster orchestration problem, describing the environment at a given step. It includes two sets of metrics: *App* and *Cluster*. The first set *App* corresponds to the deployment requirements of the microservice-based application, such as the requested number of replicas (R), and its CPU and memory requests (ω_{cpu} and ω_{mem}). Each request also has a latency threshold, which the cluster hosting the request should respect. The second set *Cluster* corresponds to the current status of the infrastructure in terms of resource capacity (Π_{cpu} and Π_{mem}), the current amount of allocated resources (Θ_{cpu} and Θ_{mem}), among others. Also, the cluster latency consists of several latency metrics depending on the number of available clusters in the multi-cluster setup, translating into C latency metrics for each cluster. Latency values are represented as values in $[1.0, 500.0]$ milliseconds. Table III shows the resource capacities for each cluster based on different cluster types and their corresponding deployment cost. Resource capacities are then represented as values in $[2.0, 32.0]$, and allocated resources are initiated as values in $[0.0, 0.2]$ since each cluster has a reserved amount of resources for background services (e.g., monitoring). This information helps the agent to select adequate actions at a given moment from the action space described next.

TABLE IV: The structure of the Action Space.

Action Name	Description
Deploy-all-c	Deploy all replicas in cluster *c*.
Spread	Divide and spread replicas across different clusters.
Reject	The agent rejects the request. Nothing is deployed.

D. Action Space

Table IV shows the action space designed for *gym-multi-k8s* as a discrete set of possible actions, where a single action is chosen at each timestep. Given a deployment request, the GTM can decide to allocate the total number of requested replicas to a single cluster, divide the number of instances across all available clusters, or reject the request. Nevertheless, the size of the action space depends on the total number of clusters in the multi-cluster scenario. Let's assume the multi-cluster setup consists of C clusters, the action space length is then $C+2$. Rejection is allowed since computational resources might be scarce at a certain moment, and no cluster can satisfy the request. The agent should not be penalized in these cases. Regarding penalties (i.e., negative reward), a simple approach commonly followed in the literature [31] is to penalize the agent if it selects an invalid action since these are typically known beforehand based on the allocated computing resources. In contrast, action masking [32] can teach the agent that depending on the current state s specific actions are invalid. This approach has recently shown significantly higher performance and sample efficiency than penalties. Action masks for each cluster $c \varepsilon C$ at state s can be defined as follows:

$$mask(s)[c] = \begin{cases} true, & \text{If cluster } c \text{ has enough resources.} \\ false, & \text{Otherwise.} \end{cases} \tag{1}$$

Whereas for spread and reject actions, the action mask is always $true$, avoiding the lock in case all actions were marked invalid. It is noteworthy that the current Karmada does not make this decision between *deploy-all* and *spread* placement. The cloud administrator decides by indicating the preferred strategy in the *PropagationPolicy* object. The GTM aims to find the optimal balance between both policies by following a First Fit Decreasing (FFD) approach for the *spread* action (Alg. 1). This balance depends on the selected reward function, in which two opposing strategies were designed for the GTM as described next.

Algorithm 1 First Fit Decreasing (FFD) for spread placement

```
Input: $R$, the number of requested replicas.
       $C$, the number of clusters.
       $\omega_{cpu,mem}$, the replica's requested cpu/memory.
       $\Omega_{cpu,mem}$, the cluster's amount of free cpu/memory.
Output: $\alpha$, the distribution of replicas across all clusters
  if $R = 1$ then
      $penalty \leftarrow$ true                          ▷ Penalize the agent
      return $\alpha = 0$
  end if
  $min \leftarrow 1$, $max \leftarrow R$, $\Delta \leftarrow R$   ▷ Get min and max replicas
  for each $c \in C$ do                                  ▷ Calculate min factor
      $f \leftarrow \min(\Omega_{\text{cpu}}[c]/\omega_{\text{cpu}}[c], \Omega_{\text{mem}}[c]/\omega_{\text{mem}}[c])$
      $\Delta \leftarrow \min(f, \Delta)$
  end for
  if $\Delta \geq R$ then
      $\Delta \leftarrow R - 1$                          ▷ To really distribute replicas
  end if
  $S \leftarrow$ sorted($\Omega_{\text{cpu}}$)           ▷ Sort by decreasing order of CPU
  for each $c \in S$ do                                  ▷ $DistLoop$: distribute replicas
      if $R = 0$ then
          break
      else if $R > 0$ & $\Delta < R$ & $(\omega_{\text{cpu}} \times \Delta < \Omega_{\text{cpu}}[c])$ &
      $(\omega_{\text{mem}} \times \Delta < \Omega_{\text{mem}}[c])$ then
          $\alpha[c] = \alpha[c] + \Delta$
          $R = R - \Delta$
      else if $R > 0$ & $(\omega_{\text{cpu}} < \Omega_{\text{cpu}}[c])$ & $(\omega_{\text{mem}} <$
      $\Omega_{\text{mem}}[c])$ then
          $\alpha[c] = \alpha[c] + min$
          $R = R - min$
      end if
  end for
  if $R = 0$ then
      return $\alpha$
  else if $R \neq 0$ then
      repeat
          $DistLoop$                                     ▷ Repeat the $DistLoop$
      until $R = 0$
  end if
```

E. Reward

The purpose of a reward function is to guide the agent towards maximization of accumulated rewards by choosing suitable actions depending on the current observation state. Two reward functions have been designed based on different objectives: cost-aware (2), and latency-aware (3). The cost-aware function leads the agent to deploy requests on clusters focused on minimizing the allocation cost (i.e., τ_c). The cloud type is significantly more expensive than fog and edge types, so the agent will prefer to deploy requests to the edge or fog since it receives a higher reward. The request's deployment cost is given by c while the maximum allocation cost is given by max. The agent is penalized if the request is rejected, and computing resources are available even when the agent supports action masking. The latency-aware function aims to satisfy the latency threshold (i.e., th) of each request. If the agent chooses a cluster that meets the latency requirements, it receives a positive reward (i.e., $+1$). In contrast, the agent is penalized if the threshold is not respected. The request's latency (i.e., l) corresponds to the average latency of the cluster by considering all latency metrics.

$$r = \begin{cases} max - c & \text{if req. is accepted} \vee (\text{req. is rejected} \wedge \\ & \text{no resources.)} \\ -1 & \text{if req. is rejected} \wedge \text{avail. resources.} \end{cases} \tag{2}$$

$$r = \begin{cases} 1.0 & \text{if } l \leq th \vee (\text{req. is rejected} \wedge \text{no resources}). \\ -1 & \text{if } l > th \vee (\text{req. is rejected} \wedge \text{avail. resources}). \end{cases} \tag{3}$$

F. Agents - Implementation Details

Multiple agents have been evaluated in the *gym-multi-k8s* environment. Most of these algorithms have been implemented based on the stable baselines 3 [33] library, a set of reliable implementations of RL algorithms written in Python. The evaluation consists mainly of four agents that support discrete action spaces: Advantage Actor Critic (A2C) [34], maskable Proximal Policy Optimization (PPO) [35], DeepSets PPO [29], and DeepSets Deep Q-Network (DQN). A2C is a synchronous, deterministic algorithm that combines policy and value-based algorithms. Policy-based agents learn a policy mapping input states to output actions (i.e., actors), and value-based algorithms select actions based on the predicted value of the input state (i.e., critic). The evaluated version of A2C does not support action masking. PPO is a policy gradient method for RL vastly used today for different scenarios (e.g., robot control and video games), and maskable PPO adds support for action masking. DQN combines the classical Q-Learning RL algorithm with deep neural networks. The DeepSets version of both PPO and DQN support the DeepSets neural network to evaluate its generalization potential.

V. Evaluation Setup

This section presents an overview of the C2E application used to validate the offline *gym-multi-k8s* RL environment. Sec. V-A describes C2E and shows its deployment requirements, and Sec. V-B details the dynamics of the *gym-multi-k8s* environment.

A. Cloud2Edge (C2E) application

The C2E package provides a scalable, cloud-based Internet of Things (IoT) platform, connecting sensor style devices and processing their respective data with a Digital Twin (DT) platform. As shown in Fig. 2, its architecture includes two main applications: Eclipse Hono and Eclipse Ditto. The first relates to an open-source framework that enables the connection of several IoT devices through remote service interfaces and the communication between them thanks to various protocol implementations (e.g., HTTP REST, MQTT). Thus, lower-end devices are connected to the back-end in the Cloud to publish or report data like telemetry. At the same time, Eclipse Hono facilitates their usage to update the DT provided by Eclipse Ditto and other functional operations, such as sending commands or communicating events. It is important to note that Hono has become the subject of different studies in the last few years (e.g., [36], [37]) due to its capability to support functionalities like device authentication and machine-to-machine management. On the other end, Ditto presents numerous services to realize DT of IoT devices. Therefore, it permits the definition of IoT solutions without the need for managing a custom back end, allowing the users to focus only on business requirements and the implementation of their applications. Through the years, Ditto has been applied in several works in the literature, especially in combination with other system and tools to realize more sophisticated IoT environments [38], [39]. We argue that C2E is a convenient application to test the proposed multi-cluster orchestration approach since it consists of several microservices, aiming to ease the proper life-cycle management of IoT applications. Table V shows the deployment requirements for the several microservices of the C2E application applied in the *gym-multi-k8s* environment.

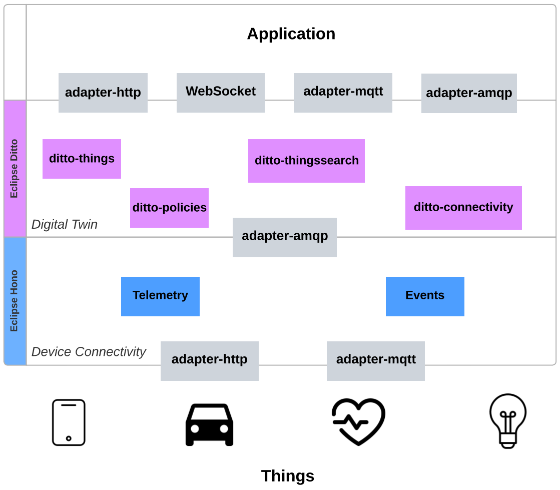


Fig. 2: The Eclipse Cloud2Edge (C2E) architecture: processing sensor data with a digital twin cloud platform.

TABLE V: Deployment properties of the C2E application.

Application	Deployment	CPU	MEM	Latency Th.
Cloud2Edge (C2E)	adapter-amqp	0.20	0.30	200 ms
	adapter-http	0.20	0.30	200 ms
	adapter-mqtt	0.20	0.30	200 ms
	artemis	0.20	0.60	200 ms
	dispatch-router	0.20	0.64	200 ms
	ditto-connectivity	0.20	0.75	100 ms
	ditto-gateway	0.20	0.50	100 ms
	ditto-nginx	0.05	0.016	100 ms
	ditto-policies	0.20	0.50	100 ms
	ditto-swaggerui	0.05	0.016	400 ms
	ditto-things	0.20	0.5	200 ms
	ditto-thingssearch	0.20	0.50	200 ms
	ditto-mongodb	0.015	0.25	200 ms
	service-auth	0.20	0.20	300 ms
	service-comm-router	0.015	0.25	300 ms
	service-dev-registry	0.20	0.40	200 ms

TABLE VI: The execution time per episode during training.

Algorithm	Execution Time (in s)
A2C	0.121 ± 0.011
Maskable PPO	0.148 ± 0.227
Deepsets PPO	0.317 ± 0.053
Deepsets DQN	0.215 ± 0.042

B. gym-multi-k8s RL environment dynamics

The *gym-multi-k8s* framework has been implemented in Python to ease the interaction with both the OpenAI Gym and the stable baselines 3 libraries. In the evaluation, an episode consists of 100 steps where the agent attempts to maximize the reward based on the current deployment request. If the agent deploys the request in one of the clusters, its average latency increases by increasing its corresponding latency metrics. In contrast, if a microservice is terminated based on a mean service duration (as default one time unit), the average latency decreases by decreasing the corresponding metrics. Also, the action selected by the agent directly impacts the latency calculation since a spread policy means all used clusters for deployment will increase its latency. During training for all algorithms, the multi-cluster setup consists of four clusters. The agents have been executed on a 14-core Intel i7-12700H CPU @ 4.7 GHz processor with 16 GB of memory. The performance of the agents has been evaluated based on the following metrics:

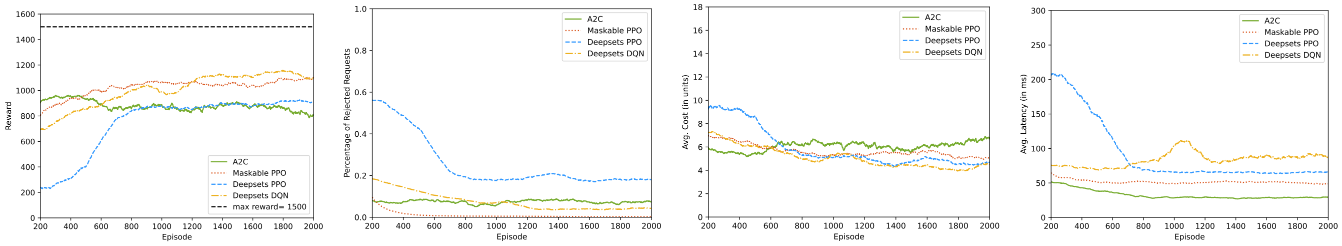
- **Accumulated reward** during each episode.
- **Percentage of rejected requests** represented as $[0, 1]$. 1 equals to 100% rejection rate.
- **Average deployment cost** of deploying all requests in the multi-cluster setup.
- **Average latency** expected by each accepted request.

VI. Results

Time Complexity has been accessed based on the training execution time for the multiple RL agents (Table VI). The results highlight that training RL agents in near-real environments is significantly faster, and that RL environments can speed up the applicability of RL methods in operational environments. A2C and Maskable PPO are considerable faster than Deepsets PPO and Deepsets DQN. **Training** results for 2000 episodes are shown in Fig. 3 and Fig. 4 for both reward functions. The number of available clusters during training corresponds to four for all algorithms, and a smoothing window of 200 episodes is applied to reduce spikes in the graphs. All algorithms seem to converge around 1000 episodes despite the fact that DeepSets PPO for the latency reward function had a period with significantly lower rewards. After this period, DeepSets PPO seems to achieve even higher rewards. Regarding the number of rejected requests, all algorithms seem to converge to a state where the percentage is below 20%, being Maskable PPO capable of reaching 0%. In terms of deployment costs, all algorithms reach average deployment costs between 6 and 12, and slightly higher values for the latency reward function, as expected. In addition, during training, the number of available clusters is set to four, which means that it might be difficult to accept a high percentage of requests while keeping the overall deployment cost low. Lastly, concerning latency, all algorithms are able to reduce it significantly during training, obtaining significantly lower values for the latency reward function (on average below 50 ms).

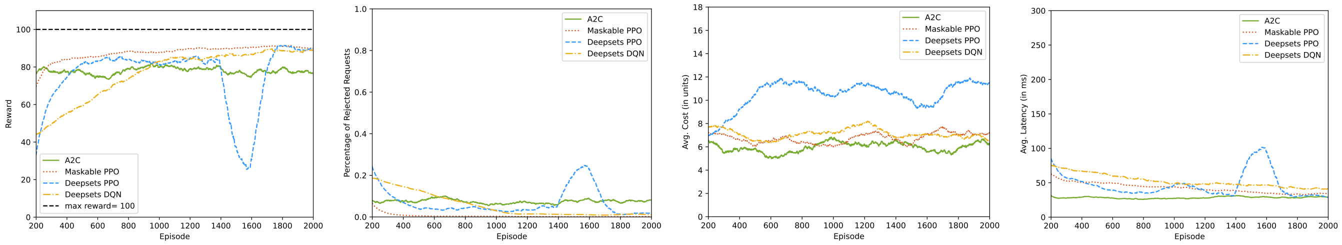
Testing has been executed for all algorithms during 100 episodes with the saved configuration after 2000 training episodes. Table VII shows the results obtained during the testing phase concerning accumulated rewards, the percentage of rejected requests, the average deployment costs, and the average latency for the different algorithms. Both DeepSets algorithms achieve higher performance than A2C and maskable PPO for both reward functions though the slightly worse training. For the cost-aware strategy, accumulated rewards of 1100 were obtained on average, corresponding to a low percentage of rejected requests of 3.9%, and an overall deployment cost of 4.3 units. For the latency-aware function, DeepSets PPO on average achieves 0% rejection rate while attaining a deployment cost of 11.7 units and an average latency of 28.57 ms. DeepSets DQN on average obtains a lower deployment cost of 5.76 units, corresponding to a rejection rate of 0.3% and an average latency of 42.16 ms. **Generalization** has been assessed for both DeepSets algorithms by varying the cluster size $[4, ..., 128]$. Results demonstrate the enormous potential of the DeepSets neural network. Both algorithms can find near-optimal allocation schemes for both strategies even when trained in a small-scale setup. The latency goal is considerably more complex than the cost objective while the number of clusters increases. Latency increases throughout the experiment, but both agents achieve adequate latency values for both strategies, lower than 100 ms for latency-aware, and lower than 300 ms for cost-aware. DQN achieves slightly lower latency than PPO at a cost of a rejection rate of almost 4%. In conclusion, both algorithms can optimize the placement of microservices in a multi-cluster setup 32 times higher than its trained setup.

In summary, this paper investigates efficient multi-cluster orchestration strategies focused on the well-known K8s platform and in recent trends as RL. Two opposing objectives demonstrate that RL algorithms can find appropriate actions that maximize the accumulated reward. An offline RL environment validated the RL approach since most algorithms achieved significantly high performance. The Karmada multi-cluster orchestration solution would benefit from our GTM since it finds a near-optimal balance between deploying all replicas into a single cluster or distributing them into multiple ones. This trade-off has been found for different objectives, as shown in this paper. In the testing phase, all algorithms achieved high rewards for both strategies. Last but not the least, the DeepSets neural network has shown its enormous potential. These RL algorithms can be applied directly into different multi-cluster environments with varying cluster sizes, significantly reducing the training time. Without DeepSets, RL algorithms need retraining for that particular cluster size, which is considerably more costly.



(a) Accumulated Reward. (b) Number of Rejected Requests. (c) Deployment Cost. (d) Avg. Latency.

Fig. 3: The training results for the multiple agents evaluated for the cost reward function.

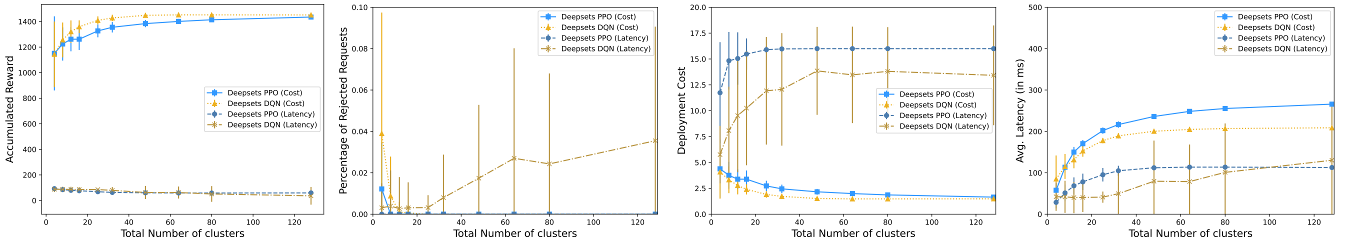


(a) Accumulated Reward. (b) Number of Rejected Requests. (c) Deployment Cost. (d) Avg. Latency.

Fig. 4: The training results for the several evaluated agents for the latency reward function.

TABLE VII: Results obtained during the testing phase.

Algorithm	Reward Function	Acc. Reward	Number of Rejected Requests	Avg. Deployment Cost	Avg. latency
A2C	Cost	893.25 ± 300.23	7.30% ± 8.8%	6.30 ± 3.55 (units)	191.61 ± 53.71 (ms)
Maskable PPO	Cost	941.58 ± 183.50	9.08% ± 7.73%	5.46 ± 2.15 (units)	191.42 ± 45.85 (ms)
Deepsets PPO	Cost	1150.89 ± 289.90	1.22% ± 2.24%	4.38 ± 2.81 (units)	57.84 ± 17.34 (ms)
Deepsets DQN	Cost	1143.66 ± 256.59	3.90% ± 5.84%	4.08 ± 2.58 (units)	85.14 ± 56.70 (ms)
A2C	Latency	6.04 ± 58.23	7.90% ± 13.29%	6.51 ± 5.74 (units)	183.64 ± 63.46 (ms)
Maskable PPO	Latency	33.22 ± 36.85	6.61% ± 7.89%	6.96 ± 3.55 (units)	147.79 ± 44.26 (ms)
Deepsets PPO	Latency	92.64 ± 7.11	0.0% ± 0.0%	11.73 ± 4.88 (units)	28.57 ± 14.55 (ms)
Deepsets DQN	Latency	87.98 ± 21.61	0.31% ± 1.18%	5.76 ± 2.77 (units)	42.16 ± 34.09 (ms)



(a) Accumulated Reward. (b) Number of Rejected Requests. (c) Deployment Cost. (d) Avg. Latency.

Fig. 5: The results for the trained Deepsets agents for both reward functions while varying the number of available clusters.

VII. Conclusions

This paper studies the efficient scheduling of microservices in a multi-cluster scenario. An RL-based approach inspired on the OpenAI Gym library has been proposed to handle efficient multi-cluster orchestration in the well-known K8s platform. The evaluation considers two opposing strategies that show the feasability of RL for the multi-cluster ocrhestration problem addressed in the paper. Results also show that generalization is attainable by incorporating the DeepSets neural network in typical RL algorithms, achieving higher performance for scenarios 32 times higher than the trained one. Multi-objective formulations and multi-agent RL scenarios will be studied as future work to find optimal combinations of opposing scehduling strategies. Our work contributes to the field by providing a framework released in open-source, allowing researchers to evaluate their scheduling ideas and potentially guide the development of more efficient scheduling algorithms.

Acknowledgment

This work has been partially supported by the Spoke 1 “FutureHPC & BigData” of the Italian Research Center on High-Performance Computing, Big Data and Quantum Computing (ICSC) funded by MUR Missione 4 - Next Generation EU (NGEU). José Santos is funded by the Research Foundation Flanders (FWO), grant number 1299323N.

References

[1] J. Thönes, "Microservices," *IEEE software*, vol. 32, no. 1, pp. 116–116, 2015.

[2] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, "Microservices: yesterday, today, and tomorrow," *Present and ulterior software engineering*, pp. 195–216, 2017.

[3] J. Santos, T. Wauters, B. Volckaert, and F. De Turck, "Fog computing: Enabling the management and orchestration of smart city applications in 5g networks," *Entropy*, vol. 20, no. 1, p. 4, 2017.

[4] B. Varghese, N. Wang, S. Barbhuiya, P. Kilpatrick, and D. S. Nikolopoulos, "Challenges and opportunities in edge computing," in *2016 IEEE international conference on smart cloud (SmartCloud)*. IEEE, 2016, pp. 20–26.

[5] S. Liu, L. Liu, J. Tang, B. Yu, Y. Wang, and W. Shi, "Edge computing for autonomous driving: Opportunities and challenges," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1697–1716, 2019.

[6] Y. Siriwardhana, P. Porambage, M. Liyanage, and M. Ylianttila, "A survey on mobile augmented reality with 5g mobile edge computing: Architectures, applications, and technical aspects," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 2, pp. 1160–1192, 2021.

[7] J. Santos, T. Wauters, B. Volckaert, and F. De Turck, "Towards low-latency service delivery in a continuum of virtual resources: State-of-the-art and research directions," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 4, pp. 2557–2589, 2021.

[8] Z. Zhong and R. Buyya, "A cost-efficient container orchestration strategy in kubernetes-based cloud computing infrastructures with heterogeneous resources," *ACM Transactions on Internet Technology (TOIT)*, vol. 20, no. 2, pp. 1–24, 2020.

[9] Y. Han, S. Shen, X. Wang, S. Wang, and V. C. Leung, "Tailored learning-based scheduling for kubernetes-oriented edge-cloud system," in *IEEE INFOCOM 2021-IEEE conference on computer communications*. IEEE, 2021, pp. 1–10.

[10] D. Bhamare, M. Samaka, A. Erbad, R. Jain, L. Gupta, and H. A. Chan, "Optimal virtual network function placement in multi-cloud service function chaining architecture," *Computer Communications*, vol. 102, pp. 1–16, 2017.

[11] C. Guerrero, I. Lera, and C. Juiz, "Resource optimization of container orchestration: a case study in multi-cloud microservices-based applications," *The Journal of Supercomputing*, vol. 74, no. 7, pp. 2956–2983, 2018.

[12] S. K. Panda, I. Gupta, and P. K. Jana, "Task scheduling algorithms for multi-cloud systems: allocation-aware approach," *Information Systems Frontiers*, vol. 21, pp. 241–259, 2019.

[13] S. Qin, D. Pi, Z. Shao, Y. Xu, and Y. Chen, "Reliability-aware multi-objective memetic algorithm for workflow scheduling problem in multi-cloud system," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 4, pp. 1343–1361, 2023.

[14] B. Burns, J. Beda, and K. Hightower, *Kubernetes: up and running: dive into the future of infrastructure*. O'Reilly Media, 2019.

[15] J. Santos, T. Wauters, B. Volckaert, and F. De Turck, "gym-hpa: Efficient auto-scaling via reinforcement learning for complex microservice-based applications in kubernetes," in *NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2023, pp. 1–9.

[16] R. Galliera, A. Morelli, R. Fronteddu, and N. Suri, "Marlin: Soft actor-critic based reinforcement learning for congestion control in real networks," in *NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2023, pp. 1–10.

[17] "Karmada documentation." accessed on 26 June 2023. [Online]. Available: https://karmada.io/.

[18] S. Lee, S. Son, J. Han, and J. Kim, "Refining micro services placement over multiple kubernetes-orchestrated clusters employing resource monitoring," in *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2020, pp. 1328–1332.

[19] F. Rossi, V. Cardellini, F. L. Presti, and M. Nardelli, "Geo-distributed efficient deployment of containers with kubernetes," *Computer Communications*, vol. 159, pp. 161–174, 2020.

[20] M. A. Tamiru, G. Pierre, J. Tordsson, and E. Elmroth, "mck8s: An orchestration platform for geo-distributed multi-cluster environments," in *2021 International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 2021, pp. 1–10.

[21] Y. Zhang, B. Di, Z. Zheng, J. Lin, and L. Song, "Distributed multi-cloud multi-access edge computing by multi-agent reinforcement learning," *IEEE Transactions on Wireless Communications*, vol. 20, no. 4, pp. 2565–2578, 2020.

[22] T. Shi, H. Ma, G. Chen, and S. Hartmann, "Location-aware and budget-constrained service brokering in multi-cloud via deep reinforcement learning," in *Service-Oriented Computing*, H. Hacid, O. Kao, M. Mecella, N. Moha, and H.-y. Paik, Eds. Cham: Springer International Publishing, 2021, pp. 756–764.

[23] A. Suzuki, M. Kobayashi, and E. Oki, "Multi-agent deep reinforcement learning for cooperative computing offloading and route optimization in multi cloud-edge networks," *IEEE Transactions on Network and Service Management*, 2023.

[24] J. Santos, C. Wang, T. Wauters, and F. De Turck, "Diktyo: Network-aware scheduling in container-based clouds," *IEEE Transactions on Network and Service Management*, 2023.

[25] M. Fogli, T. Kudla, B. Musters, G. Pingen, C. Van den Broek, H. Bastiaansen, N. Suri, and S. Webb, "Performance evaluation of kubernetes distributions (k8s, k3s, kubeedge) in an adaptive and federated cloud infrastructure for disadvantaged tactical networks," in *2021 International Conference on Military Communication and Information Systems (ICMCIS)*. IEEE, 2021, pp. 1–7.

[26] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y.-C. Liang, and D. I. Kim, "Applications of deep reinforcement learning in communications and networking: A survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3133–3174, 2019.

[27] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[28] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola, "Deep sets," *Advances in neural information processing systems*, vol. 30, 2017.

[29] N. D. Cicco, G. F. Pittalà, G. Davoli, D. Borsatti, W. Cerroni, C. Raffaelli, and M. Tornatore, "Drl-forch: A scalable deep reinforcement learning-based fog computing orchestrator," in *2023 IEEE 9th International Conference on Network Softwarization (NetSoft)*, 2023, pp. 125–133.

[30] Amazon AWS, "Amazon ec2 on-demand pricing." accessed on 28 September 2023. [Online]. Available: https://aws.amazon.com/ec2/pricing/on-demand/.

[31] H. Sami, A. Mourad, H. Otrok, and J. Bentahar, "Demand-driven deep reinforcement learning for scalable fog and service placement," *IEEE Transactions on Services Computing*, vol. 15, no. 5, pp. 2671–2684, 2021.

[32] S. Huang and S. Ontañón, "A closer look at invalid action masking in policy gradient algorithms," *arXiv preprint arXiv:2006.14171*, 2020.

[33] A. Raffin, A. Hill, M. Ernestus, A. Gleave, A. Kanervisto, and N. Dormann, "Stable baselines3," 2019.

[34] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*. PMLR, 2016, pp. 1928–1937.

[35] C.-Y. Tang, C.-H. Liu, W.-K. Chen, and S. D. You, "Implementing action mask in proximal policy optimization (ppo) algorithm," *ICT Express*, vol. 6, no. 3, pp. 200–203, 2020.

[36] M. Aly, F. Khomh, and S. Yacout, "Kubernetes or openshift? which technology best suits eclipse hono iot deployments," in *2018 IEEE 11th Conference on Service-Oriented Computing and Applications (SOCA)*, 2018, pp. 113–120.

[37] J. Lee, S. Kang, and I.-G. Chun, "miotwins: Design and evaluation of miot framework for private edge networks," in *2021 International Conference on Information and Communication Technology Convergence (ICTC)*, 2021, pp. 1882–1884.

[38] Y. Lim, Y. K. Lee, J. Yoo, and D. Yoon, "An open source-based digital twin broker interface for interaction between real and virtual assets," in *2022 13th International Conference on Information and Communication Technology Convergence (ICTC)*, 2022, pp. 1657–1659.

[39] J. Kristan, P. Azzoni, L. Römer, S. E. Jeroschewski, and E. Londero, "Evolving the ecosystem: Eclipse arrowhead integrates eclipse iot," in *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*, 2022, pp. 1–6.