

# Data-connector: An agent-based framework for autonomous ML-based smart management in cloud-edge continuum

Peini Liu<sup>\*†</sup>, Joan Oliveras Torra<sup>\*†</sup>, Marc Palacín<sup>\*</sup>, Jordi Guitart<sup>\*†</sup>, Josep Ll. Berral<sup>\*†</sup>, Ramon Nou<sup>\*</sup>

<sup>\*</sup>Barcelona Supercomputing Center, Barcelona, Spain

<sup>†</sup>Universitat Politècnica de Catalunya, Barcelona, Spain

E-mail: {peini.liu, joan.oliveras, marc.palacin, ramon.nou}@bsc.es, {jordi.guitart, josep.ll.berral}@upc.edu

**Abstract**—Machine Learning (ML) is becoming pervasive and integrated into different kinds of intelligent applications, and the collaborative Cloud-Edge continuum has been introduced as an emerging trend to support their adoption into use cases. However, managing these ML applications in the Cloud-Edge continuum is challenging due to the ML application’s dynamic resource usage with different user loads and Cloud and Edge’s dynamic resource availability. We envision machine learning methods that can be used for smart management in this dynamic environment, but how to deploy and utilize them for the adaptation scenario in Cloud-Edge continuum is unknown. This paper proposes an agent-based framework to enable autonomous smart management mechanisms that can be broadly enabled in diverse adaptation scenarios. The agent acts as a data-connector<sup>1</sup>, connecting different sources of data, utilizing ML models for decision-making and triggering adaptations in Cloud-edge platforms. The case study shows the feasibility of our proposed data-connector for smart migration of an ML workload in the Cloud-edge continuum. The result shows that the smart migration-enabled Cloud-edge scenario has 11.9% ML application prediction time better than the Cloud scenario without migration. Moreover, with minimal customization, the data connector agent can be adapted for more use cases.

**Index Terms**—Agent, Kubernetes, Machine Learning.

## I. INTRODUCTION

Modern computing infrastructure is evolving at a fast pace to a computing continuum. Containerization, as a fundamental virtualization technology, allows efficient utilization and easy maintenance of the infrastructure within the cloud-edge continuum.

The deployment of containerized applications in the continuum is managed by container orchestrators, which have the capability to launch and manage containers and their entire lifecycle, and leverage resource availability and user specifications to decide the placement of containers. Typical resource orchestrators are available nowadays such as Docker Swarm [1], Mesos [2], and Kubernetes [3]. Among these, Kubernetes has been extensively adopted in commercial production systems, such as Google Kubernetes Engine [4], and provides a wide and active toolkit ecosystem. In addition, ML applications can be served by different runtimes such as TensorFlow Serving [5], TorchServe [6]. More complex

applications may require specialized application managers to manage a group of containers, such as ML workflows which may need a workflow manager such as Argo [7] or serverless ML inference manager Kserve [8] etc. However, the containerized application is not running in a known context with static requirements, and consequently bringing autonomous management to these platforms to serve these applications to meet dynamic changes is essential nowadays.

To this end, autonomic computing [9]–[12] offers inspiring approaches to adapt the application and the infrastructure at runtime. For example, by changing application packing or configurations at the application layer or by rescheduling and auto-scaling the application at the infrastructure layer. However, decision-making crossing various factors and objectives remains challenging. Therefore, by incorporating machine learning (ML) into the management of containerized applications, systems can become more resilient, efficient, and adaptive to changing conditions, ultimately providing a more robust and scalable solution within the computing continuum.

In this paper, we present an agent-based approach leveraging autonomic computing to achieve autonomous smart management for containerized applications. The goal is to leverage our smart algorithm and integrate the agent-based management framework with container orchestration technologies to improve the performance of the application. Our main contributions are:

- We extend our Scanflow agent framework to fit more general autonomous management scenarios and integrate ML into a decision-maker. So that, the current agent can connect with various data sources, generate knowledge and decisions using different ML models and trigger platforms to do adjustments.
- We establish a use case on a real platform (based on Scanflow-Kubernetes [13] [14]), and evaluate our smart agent for migrating containerized AI applications in a Cloud-edge continuum.

## II. BACKGROUND AND RELATED WORK

### A. Container and Container Orchestration

Containerization is a lightweight virtualization technology that builds upon resource isolation and limitation features

<sup>1</sup><https://github.com/bsc-scanflow/data-connector>

of the Linux kernel, such as `namespaces` and `cgroups`, respectively. Currently, containers are widely used to pack applications because of the benefits of portability, isolation, and high availability. Numerous platforms support container orchestration, with Kubernetes [3] being dominant in the current computing Continuum. Kubernetes supports the orchestration of containers and provides resource management for its minimal unit Pod. From the users’ perspective, they can submit simple specifications of applications (e.g., Deployments, Jobs) to Kubernetes, the application can be encapsulated into containers and wrapped as Pods to be deployed in the nodes. From the providers’ perspective, all the nodes and resources are controlled by Kubernetes. Upon receiving a deployment request, Kubernetes resource manager checks the Pod resource requirements, and selects the node (using a scheduling policy to filter and rank nodes) to run each Pod, so that the Kubelet can launch the Pod on the selected node.

### B. Application in Kubernetes

Kubernetes was initially designed to orchestrate only simple micro-service applications. Other application managers can be used on top of Kubernetes to help manage more complex containerized applications. For example, the Volcano MPI [15] controller can manage HPC applications. Similarly, Argo workflows [7] can manage ML pipelines, Flink [16] can handle streaming applications, Kserve [8] can manage serverless ML inference, and Seldon [17] can serve ML models. Our previous work, the Scanflow-Kubernetes deployer<sup>2</sup>, uses different backends to support deploying different applications, including ML workflows, ML inference services, and Scanflow(MPI) for HPC applications.

### C. Autonomous Smart Management for Containerized Applications

Once the applications are containerized, they could run on the Continuum by directly using their application controllers and container orchestration platforms. To enable the orchestration and management, Kubernetes natively supports various scheduling algorithms and HorizontalPodAutoscaler (HPA) for workload auto-scaling. Also, the community has developed Kubernetes event-driven auto-scaling Keda [18], and other schedulers such as Volcano [15] that support rescheduling, preemption scheduling, etc. Application controllers can also handle some management parts; for example, Seldon [17] can reallocate traffic, and Torchserve [6] can adjust the number of workers internally. However, these management strategies are mostly reactive, and the actions are driven by service level agreements (SLAs) or events with constraints.

Beyond basic allocation mechanisms, well-known combinatorial optimization heuristics, such as integer linear programming, bin packing, etc., can be used for smart decision-making. However, recent works are advancing ML-based smart management for resource provisioning and orchestration to solve the multi-objective NP-hard problem. This

includes using Deep Reinforcement Learning (DRL) for resource allocation [19]–[21] and workload scheduling [22], as well as using ML-based methods for workload characterization predictions [23], [24] to enable proactive auto-scaling.

Our previous work Scanflow-Kubernetes agent<sup>2</sup> [13], [14], [25] provided reactive agents which enabled autonomous management of ML workflows. In this work, we evolve these agents into more general data-connector proactive agents. These agents can connect with various data sources and platforms to do management and integrate machine learning strategies for decision-making.

## III. AGENTS FOR AUTONOMOUS SMART MANAGEMENT

In this section, we introduce the architecture of the Data-connector agent and its features. This agent is a Scanflow agent [14] variant, with enhancements to abstract to different data and platforms, and integrate ML strategies. Specifically, we define agent intelligence with model inference, triggers, and actuators for smart workload management in a dynamic environment.

### A. Agent Architecture

We use the concept of agent, which does not implement a global model or plan but only some simple behaviours. These behaviours allow the agent to observe the environmental changes proactively. An agent includes a sensor that detects changes in internal and external states, an ML model that responds to relevant observations, and an actuator that activates specific processes within the environment or other agents.

Data-connector agents are the fundamental components to implement autonomous ML-based smart management. Each agent is an independent computational unit that is able to run actions according to state changes. Therefore, an agent can be defined as a set of state-to-action mappings (i.e.,  $Agent = States(s) \rightarrow Actions(a)$ ), that is, state changes could result in the execution of actions (recommended by ML-based strategies). However, an agent usually cannot directly perceive the states but compute them from observations  $o_t$  using a function  $F$ . Also, the agent performs actions through rules with the computed states  $s_t$  ( $a_t = R(s_t)$ ). Figure 1 shows the agent-environment interaction: At time  $t$ , the agent computes the states  $s_t$  from the observations  $o_t$  using the function  $F$ . Then, it chooses actions  $a_t$  according to the rules  $R$  to achieve the agent’s goal.

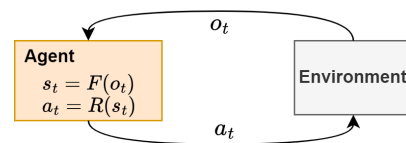


Fig. 1. Agent-Environment interaction.

Scanflow agent implements its autonomy by defining strategies that include events, constraints, and actions, expressed as

<sup>2</sup><https://github.com/bsc-scanflow/scanflow>

3-tuples  $Strategy = (Events, Constraints, Actions)$ , specifically, the autonomic management strategy is described as: when *Event* occurs, if *Constraint* is satisfied, then *Action* will be executed. A Data-connector agent primarily uses ML-based strategies, so that the tuples become  $Strategy = (Data, Models, Constraints, Actions)$ , specifically, the autonomic ML-based smart management strategy is: load the *Data*, inference them with the *Model* for predictions or recommendations, evaluate rules *Constraints* when generating decision-making, and then execute *Actions* on specific platforms. The workflow of the Data-connector agent to use the strategies is shown in Figure 2.

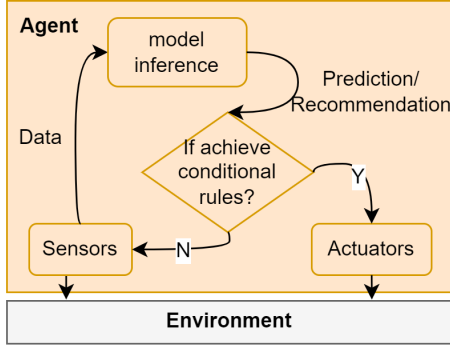


Fig. 2. Data-connector agent abstract workflow.

### B. Agent Model Inference

To enable smart management, ML models are used to generate knowledge for the current system. Depending on different use cases and different autonomous management requirements, the developer should gather the history data from their platform and train different models to analyse target scenarios. After that, this model can be used in a pipeline or served as a service for predictions. For example, Figure 3

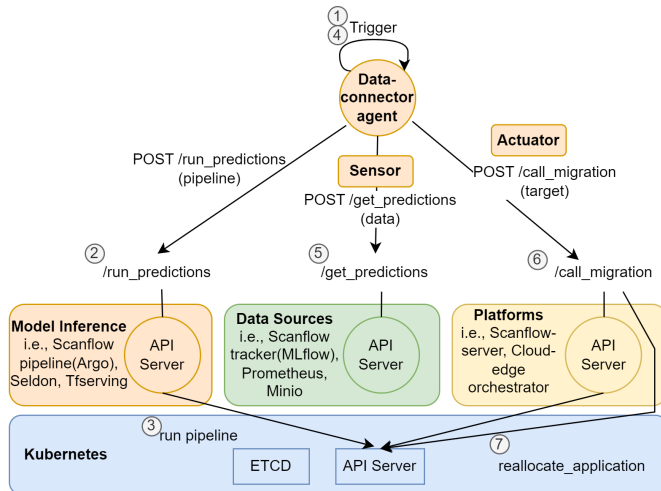


Fig. 3. Data-connector agent implementation workflow.

(steps 1-3) shows that the data-connector agent can trigger an inference pipeline to predict the QoS of an application. This proposed combination of model inference and the agent leverages a distributed decision-making, optimizing performance and reducing the need of extensive telemetry data transfers. This approach is particularly advantageous for application and nodes where statistical predictions can be performed locally, mitigating the necessity for constant data communication with centralized servers.

### C. Agent Sensors

To actively monitor current *States*, data-connector agents are required to trigger tasks to detect useful observations. Data-connector agent has different types of built-in triggers as **Sensors**, namely interval triggers, date triggers, and cron triggers (see Table I). In addition, basic triggers can be combined using 'and' or 'or' logic to produce more complex hybrid triggers. These triggers can be scheduled at a specific time or time intervals to execute tasks so that agents could get required observations to evaluate the changes of *States*. Note that each Scanflow agent contains an asynchronous I/O scheduler with multiple queued tasks. Tasks are run by the scheduler in a thread-pool.

TABLE I  
TYPES OF AGENT TRIGGERS.

Types		Definition
Scheduled Triggers	Interval	Trigger at the specified frequency.
	Date	Trigger once on the given date and time.
	Cron	Trigger when current time matches all specified time constraints (similarly to UNIX cron).

Data-connector has a flexible way to connect with different sources of data, such as real-time data from Prometheus, artifacts from Minio, or different types of data from MLflow. Figure 3 (steps 4-5) shows a sensor to get\_predictions and aggregate prediction results. Different use case will need to prepare the relevant data and define their customized sensors.

### D. Agent Actuators

After a change in *States*, agents need to perform *Actions* (i.e.,  $a_t = R(s_t)$ ) through **Actuators** to adapt changes to the environment. Different use cases will need to define their operation primitives for customized actuators and make sure the operations can be implemented or realized by the platform they used in their testbed (i.e., Kubernetes, multi-clustering orchestrator, etc.). For instance, Figure 3 (steps 6-7) shows that an actuator's call to migrate applications, the corresponding Cloud-edge orchestrator/Kubernetes should accomplish this operation through their internal implementations.

### E. Agent Communication

The Data-connector agent's communication methods are aligned with the Scanflow agent. Below, we introduce two different ways for the Data-connector agent to communicate with the environment: through RESTful APIs and shared artifacts [14], [25].

- Interaction through RESTful APIs: In this approach, the sensors and actuators of an agent are exposed as interfaces. The agent is registered into a service discovery system, allowing it to call the well-defined interfaces from the environment data sources to get data through REST. Additionally, the remote call leads to changes in the agent’s belief/state and ultimately calls platform APIs to drive an action.
- Interaction through shared artifacts: The data-connector has a database for shared artifacts within a knowledge base, which receives queries from the model inference pipeline and the sensors, and delivers the results from its database. These data include the metadata and logs from the prediction service/pipelines, as well as the metrics, scores, parameters, and different versions of the ML model. For instance, the model inference process will download the model from the database for inference, and the results will be aggregated as knowledge and saved back into a database. This approach is mainly used for model inference within a Data-connector.

#### IV. USE CASE: SMART QoS-AWARE MIGRATION IN SCANFLOW-KUBERNETES

The previously proposed Data-connector agent is a framework that can be used for different use cases and satisfy different autonomous management requirements. In this section, we build a use case on top of Scanflow-Kubernetes platform and enable a data-connector agent to perform smart QoS-aware migration for an image classification ML workload.

##### A. Experimental Settings

**Platform Settings:** Our experiments are executed on a three-node Kubernetes cluster, where the control-plane consists of 12 cores, 48Gi; one worker nodes of 8 cores, 16Gi, and one edge node of 4 cores, 8Gi. Each node is a Openstack-based VM with Rocky Linux 9.3.

The Scanflow-Kubernetes platform is built based on Kubernetes v1.28.2 (Network: Calico v3.26.4, Runtime: containerd v1.6.25, DNS: CoreDNS v1.10.1, Storage: etcd 3.5.9). Scanflow corresponding toolkits are Argo workflow v3.5.2, Mlflow 2.9.2, Minio v5.0.11, and PostgreSQL 1.17.

**Application Settings:** We use a target ML-serving application for image classification. The application is running a Resnet\_18 model, wrapped within TorchServe and deployed by Kubernetes Deployment controller. We are using a client sending pictures with a concurrency of two to the TorchServe service and receiving classification results.

##### B. Smart Migration Scenario Settings

Figure 4 shows the use case of smart migration using a data-connector agent. The data-connector agent proactively calls model inference to predict the QoS of a target application in the cloud and the edge, and also periodically watches the application QoS prediction results with a performance-cost comparison policy to decide if triggering the application migration.

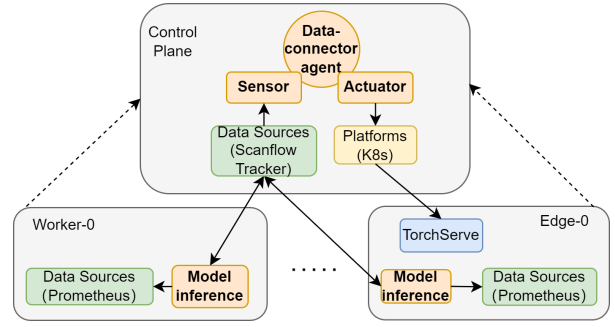


Fig. 4. Smart migration of TorchServe using data-connector agent.

The autonomous strategies of the agent for application migration are described in detail in Table II. The proposed data-connector requires the Data Engineer team to enable the model inference and provide custom functions for sensors and actuators.

TABLE II  
AGENT AUTONOMOUS MANAGEMENT STRATEGY

Agent	Analyzing and Planning Strategies
Data-connector agent	Analyzing Strategy: $QoS\_predictions$ <b>WHEN</b> $intervalTrigger(5min, QoS\_prediction(data))$ <b>IF</b> $successful\_call$ <b>THEN</b> $runWorkflow(prediction-pipeline, data)$ Planning Strategy: $migrate\_app$ <b>WHEN</b> $intervalTrigger(5min, watch\_app\_qos)$ <b>IF</b> $target\_node\_app\_qos > current\_node\_app\_qos$ <b>THEN</b> $Call(Platform-API : migrate\_app)$

##### C. Data-connector Agent Implementation

To implement the agent, custom functions of these main components should be developed, i.e., model inference, sensor, and actuator.

1) *Pre-steps:* Pre-steps mainly contain data collection and model training. The model used for QoS prediction should be pre-trained, how to train the model is out-of-scope of the data-connector agent. In this paper, we run TorchServe application in both cloud and edge and each of the node has dynamic resource usage data and the application prediction latency data to train a Random Forest(RF) model. The trained model is saved and can be used by the data-connector agent for QoS predictions.

2) *Model inference:* Model inference is to use the pre-trained model for predictions. In our implementation, we define a model inference that Loads data uses Prometheus API to get the local node resource usage data in the last time window, preprocesses the unknown values etc., and predicts the application QoS in the edge and cloud using the pre-trained RF model. Note that the model inference can be distributed into the edge and cloud if the data is not aggregated as shown in Figure 4.

3) *Sensor:* Sensor defines which data in the shared artifacts the agent should watch, and the policy to decide if triggering the actuator. Listing 1 shows the agent is watching the QoS

predictions and having a policy of performance cost trade-off (i.e., performance/cost). If QoS constraints are satisfied, chooses the node with the best placement recommendation. Listing 2 shows the frequency of this watch, and how the sensor can mount a timer, in this case 1 minute.

```

1 #example 1: watch app QoS predictions
2 @sensor(nodes=["predictor"])
3 async def watch_qos(runs: List[mflow.entities.Run], args,
4     kwargs):
5     qos = 0
6     input_data = get_qos()
7     if input_data:
8         qos, node_index = choose_better_nodes(input_data)
9         if qos_constraints(qos):
10            await call_migrate_app(max_qos_index, "icresnet",
11                "torch-deployment")
12        else:
13            logging.info("all_machine_can_not_achive_qos_sla,
14                no_actions")
15    else:
16        logging.info("no_data_in_last_check")
17    return max_qos

```

Listing 1. Custom sensor to get app QoS predictions and enable recommendation policy

```

1 trigger = client.ScanflowAgentSensor_IntervalTrigger(
2     minutes=1)
3 sensor = client.ScanflowAgentSensor(
4     name='watch_qos',
5     isCustom=True,
6     func_name='watch_qos',
7     trigger=trigger
8 )

```

Listing 2. Set trigger to watch\_qos sensor

4) *Actuator and Platform*: Actuator is used to connect different platforms to execute migrations. In our use case, we use native Kubernetes to deploy our application, thus the migrate operations should be done by using Kubernetes API. Listing 3 shows the migration operations execution, where the agent can generate a patch of the nodeSelector for a target pod to update the application deployment, so that the application can be finally migrated from one node to another.

```

1 async def call_migrate_app(max_qos_index, namespace,
2     deployment_name):
3     nodeName_list=['cloudskin-k8s-control-plane-0.novalocal',
4         'cloudskin-k8s-worker-0.novalocal',
5         'cloudskin-k8s-edge-worker-0.novalocal']
6     # Prepare the patch
7     patch_body = {
8         "spec": {
9             "template": {
10                "spec": {
11                    "nodeSelector": {"kubernetes.io/hostname":
12                        nodeName_list[int(max_qos_index)]}
13                }
14            }
15        }
16    }
17    logging.info(f"agent_is_patch_deployment_to_node_{
18        patch_body}")
19    #connect k8s
20    config.load_incluster_config()
21    api_instance = client.AppsV1Api()
22    try:
23        api_instance.patch_namespaced_deployment(
24            name=deployment_name,
25            namespace=namespace,
26            body=patch_body
27        )
28    logging.info("update_deployment_with_patch_succeeded")

```

```

26 return True
27 except client.api_client.rest.ApiException as e:
28     logging.error(f"update_deployment_with_patch_failed:_{
29         e}")
30 return False

```

Listing 3. Custom actuator to connect k8s platform

#### D. Data-connector Agent Deployment

1) *Model inference deployment*: Previously we mentioned the model inference which can be deployed using the Scanflow pipeline, similar to an Argo workflow. If a use case requires customized model inference as well as the deployment, other online serving solutions like Seldon, Torch, or TensorFlow Serving can be used.

2) *Agent service deployment*: The central agent has an HTTP server, so it can be deployed as a service in Kubernetes environment. Additionally, the shared artifacts knowledge database should be accessible from both the edge and the cloud.

#### E. Experimental Results

In this section, we evaluate the effectiveness of our data-connector agent in a scenario of application smart migration. In particular, a TorchServe-based image classification application migration according to the application QoS smart prediction and performance-cost trade-off policy in a dynamic cloud-edge continuum.

Figure 5 shows the application image classification prediction time latency trends of two runs in Cloud and Cloud-edge scenarios. The red one is a baseline which shows all the requests processed in the Cloud. The blue (the Cloud) and green (the Edge) ones show all the requests processed in the Cloud-edge continuum, where application remains in the Cloud at night due to a high cost of energy in the Edge, and in the daytime, application can be migrated between Cloud and Edge based on the QoS prediction.

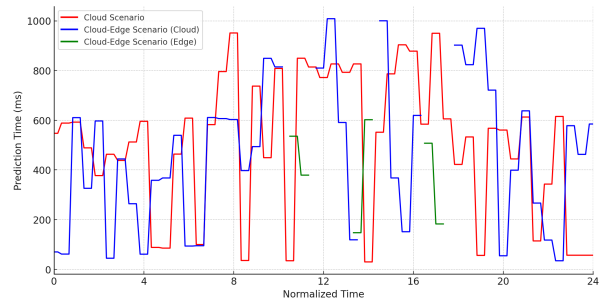


Fig. 5. Application prediction time latency trends in Cloud and Cloud-edge scenarios.

Figure 6 shows the distribution of application image classification prediction time in Cloud and Cloud-edge scenarios, and the average prediction time latency of Cloud-edge scenario is 11.9% better than Cloud scenario.

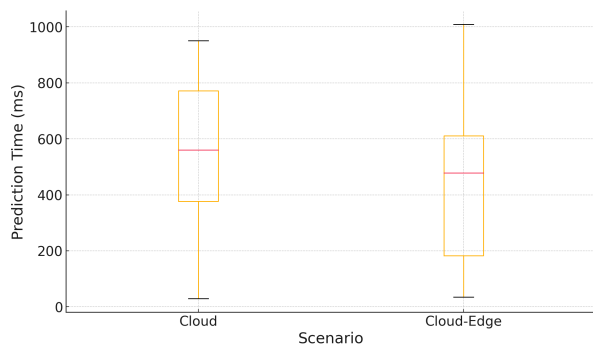


Fig. 6. Distribution of application prediction time latency in Cloud and Cloud-edge scenarios.

## V. CONCLUSION

This paper presented an agent-based framework for autonomous smart management based on ML. We first extended the Scanflow agent to make the new data-connector agent connecting with different data, producing different knowledge, and triggering adaptations across different platforms. We established a data-connector agent on the Scanflow-Kubernetes platform for application migration as a showcase of the proposed agent, and experimental results show the effectiveness of the agent for autonomous management in a cloud-edge continuum. The next step will be to leverage this prototype for large-scale validation and to integrate more intelligent mechanisms (e.g., Long short-term memory model).

## ACKNOWLEDGMENT

This work is financed by the EU-HORIZON programme under grant agreements EU-HORIZON GA.101092646, EU-HORIZON MSCA GA.101086248, by Generalitat de Catalunya (AGAUR) GA.2021-SGR-00478, and the Spanish Ministry of Science (MICINN), the Research State Agency (AEI) and European Regional Development Funds (ERDF/FEDER) PID2021-126248OB-I00, MCIN/AEI/10.13039/501100011033/FEDER, UE.

## REFERENCES

- [1] Docker, "Deploy to Swarm," 2022. [Online]. Available: <https://docs.docker.com/get-started/swarm-deploy/>
- [2] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for Fine-Grained resource sharing in the data center," in *8th USENIX Symposium on Networked Systems Design and Implementation (NSDI'11)*, 2011.
- [3] Kubernetes, "Production-Grade Container Orchestration," 2022. [Online]. Available: <https://kubernetes.io/>
- [4] Google, "Google Kubernetes Engine (GKE)," 2022. [Online]. Available: <https://cloud.google.com/kubernetes-engine>
- [5] Tensorflow serving, "A flexible, high-performance serving system for machine learning models," 2024. [Online]. Available: <https://github.com/tensorflow/serving>
- [6] Torch Serve, "Serve, optimize and scale pytorch models in production," 2024. [Online]. Available: <https://github.com/pytorch/serve>
- [7] Argo, "Argo workflows - the workflow engine for kubernetes," 2021. [Online]. Available: <https://argoproj.github.io/argo-workflows/>
- [8] "Serverless ML Inference Platform on Kubernetes," 2024. [Online]. Available: <https://kserve.github.io/website/>

- [9] P. Liu, X. Mao, S. Zhang, and F. Hou, "Towards reference architecture for a multi-layer controlled self-adaptive microservice system," in *Proceedings of the 30th International Conference on Software Engineering and Knowledge Engineering (SEKE)*, 2018, pp. 236–241.
- [10] T. De Wolf and T. Holvoet, "Towards autonomic computing: agent-based modelling, dynamical systems analysis, and decentralised control," in *Proceedings of the IEEE International Conference on Industrial Informatics (INDIN)*, 2003, pp. 470–479.
- [11] G. Tesauro, D. Chess, W. Walsh, R. Das, A. Segal, I. Whalley, J. Kephart, and S. White, "A multi-agent systems approach to autonomic computing," in *Proc. of the 3rd Intl. Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2004, pp. 464–471.
- [12] F. M. Brazier, J. O. Kephart, H. V. D. Parunak, and M. N. Huhns, "Agents and Service-Oriented Computing for Autonomic Computing: A Research Agenda," *IEEE Internet Computing*, vol. 13, no. 3, pp. 82–87, 2009.
- [13] P. Liu, G. Bravo-Rocca, J. Guitart, A. Dholakia, D. Ellison, and M. Hodak, "Scanflow: An end-to-end agent-based autonomic ml workflow manager for clusters," in *22nd International Middleware Conference: Demos and Posters*, ser. Middleware'21. ACM, 2021, pp. 1–2.
- [14] —, "Scanflow-k8s: Agent-based framework for autonomic management and supervision of ML workflows in kubernetes clusters," in *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, 2022, pp. 376–385.
- [15] Volcano, "Volcano: Cloud native batch scheduling system for compute-intensive workloads," 2024. [Online]. Available: <https://volcano.sh/en/>
- [16] Flink, "An stream processing framework with powerful stream- and batch-processing capabilities," 2024. [Online]. Available: <https://github.com/apache/flink>
- [17] Seldon, "Machine learning deployment for enterprise," 2024. [Online]. Available: <https://www.seldon.io/>
- [18] Keda, "Keda - kubernetes event-driven autoscaling," 2021. [Online]. Available: <https://keda.sh/>
- [19] T. Danino, Y. Ben-Shimol, and S. Greenberg, "Container allocation in cloud environment using multi-agent deep reinforcement learning," *Electronics*, vol. 12, no. 12, 2023. [Online]. Available: <https://www.mdpi.com/2079-9292/12/12/2614>
- [20] N. Liu, Z. Li, J. Xu, Z. Xu, S. Lin, Q. Qiu, J. Tang, and Y. Wang, "A hierarchical framework of cloud resource allocation and power management using deep reinforcement learning," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. Los Alamitos, CA, USA: IEEE Computer Society, jun 2017, pp. 372–382. [Online]. Available: <https://doi.ieeecomputersociety.org/10.1109/ICDCS.2017.123>
- [21] Y. Ju, Y. Chen, Z. Cao, L. Liu, Q. Pei, M. Xiao, K. Ota, M. Dong, and V. C. M. Leung, "Joint secure offloading and resource allocation for vehicular edge computing network: A multi-agent deep reinforcement learning approach," *IEEE Transactions on Intelligent Transportation Systems*, vol. 24, no. 5, pp. 5555–5569, 2023.
- [22] Z. Jian, X. Xie, Y. Fang, Y. Jiang, Y. Lu, A. Dash, T. Li, and G. Wang, "Drs: A deep reinforcement learning enhanced kubernetes scheduler for microservice-based system," *Software: Practice and Experience*, 2023.
- [23] J. L. Berral, D. Buchaca, C. Herron, C. Wang, and A. Youssef, "Thetascan: Leveraging behavior-driven forecasting for vertical auto-scaling in container cloud," in *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*, 2021, pp. 404–409.
- [24] J. L. Berral, C. Wang, and A. Youssef, "Ai4dl: Mining behaviors of deep learning workloads for resource management," in *Proceedings of the 12th USENIX Conference on Hot Topics in Cloud Computing*, ser. HotCloud'20. USA: USENIX Association, 2020.
- [25] G. Bravo-Rocca, P. Liu, J. Guitart, A. Dholakia, D. Ellison, J. Falkanger, and M. Hodak, "Scanflow: A multi-graph framework for machine learning workflow management, supervision, and debugging," 2021. [Online]. Available: <https://arxiv.org/abs/2111.03003>