2023 IEEE International Conference on Robotics and Automation (ICRA 2023)
May 29 - June 2, 2023. London, UK

# KubeROS: A Unified Platform for Automated and Scalable Deployment of ROS2-based Multi-Robot Applications

Yongzhou Zhang[1,2], Christian Wurll[1], Björn Hein[1,2]

***Abstract*— As advanced algorithms enable robots to handle more challenging tasks and operate more autonomously, the on-board computer cannot meet the increased demands regarding computing power and memory storage in an efficient way. Leveraging the massive computing power of the cloud and low-latency connectivity to the edge can compensate for this lack of computing resources. However, this introduces a new challenge related to the deployment of complex robotic software across multiple devices, especially in a large-scale system. This paper presents KubeROS, a unified and fully managed platform for automated deployment of robotic applications developed on top of Robot Operating System 2 (ROS2), in a hybrid computing infrastructure with robots, edge and cloud. KubeROS uses Kubernetes from Cloud Native Computing as its underlying software orchestration framework. It aims to help researchers and developers with no prior cloud computing knowledge deploy their ROS2-based robotic applications at any scale. KubeROS eliminates the need for system configuration and network setup. We demonstrate the applicability of KubeROS by deploying a fleet of simulated mobile manipulators in a classical pick-and-place application. The experiments demonstrate the effects of different deployment strategies for vision-based motion planning under different fleet sizes and workloads. In addition, KubeROS improves task performance by using high-performance computing at the edge and in the cloud, and achieves high resource efficiency when using the shared deployment strategy.**

## I. Introduction

Recently, the demand for intelligent autonomous robots has increased significantly to address challenges in various fields: Manufacturing, logistics, support in medical environments, agricultural applications, or even as household helpers. Many scenarios require a fleet of robots to handle the workload. In order for the robot to operate autonomously and skillfully, various state-of-the-art techniques are used for object recognition [1], localization [2], and grasping [3], which require different types of resources, such as CPU/GPU for computation or memory (RAM) or storage for data handling. As the system gets larger and the software gets more complex, two problems need to be addressed. *Problem A* - a robot's on-board computing resources are often inadequate due to battery capacity and lack of space. Furthermore, once the robot is built, it does not make sense from a technical or economic perspective to upgrade the hardware to meet a later increased need for computing power. *Problem B* - concerns the lifecycle management of the robot software. This includes efficient development and automatic deployment to frequent upgrades with minimal system downtime.

[1]Karlsruhe University of Applied Sciences, 76133 Karlsruhe, Germany. yongzhou.zhang@h-ka.de
[2]Karlsruhe Institute of Technology, 76131 Karlsruhe, Germany

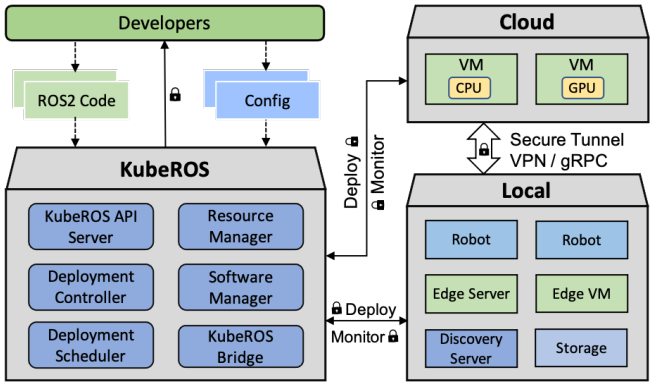



Fig. 1. **High-level overview of KubeROS platform:** KubeROS with its core components abstracts and manages all computing resources (robots/devices, edge and cloud). It provides an easy-to-use interface for developers to automatically deploy and monitor their ROS2-based robotic applications, from a single robot to a large fleet of robots. The complexity of the underlying infrastructure is invisible to the user as it is hidden by the KubeROS platform.

With regard to Problem A, cloud robotics [4], [5], [6] has emerged as a collaborative technology to leverage the enormous computing power and large storage space of cloud computing for robotics. However, this introduces new challenges caused by insufficient network bandwidth, high latency, and unreliable wireless communications. To complement this, the concept of fog robotics [7], [8], [9] has been proposed to balance the computing power and network resources between cloud and edge.

To address Problem B, the robot development process was significantly accelerated by the introduction of the Robot Operating System (ROS) along with various reusable software packages and tools [10]. The successor system, ROS2, addresses reliability, real-time performance, and safety challenges. It has been redesigned from the ground up to be industrial-grade and deployable in production [11]. However, how the software can be deployed on a large-scale system is not the main goal of ROS2. Advantageously, it is software deployment that is the main goal of Kubernetes (K8s). Kubernetes comes from cloud native computing and is widely and successfully used to automate the deployment, orchestration, scaling, and management of containerized applications in the cloud [12], [13]. It is designed for large-scale software services, enables the deployment of thousands of containers per week, and can scale services on demand.

By leveraging the advantages of both Kubernetes and ROS2, we developed KubeROS to address both challenges. KubeROS is built on top of Kubernetes and serves as a

platform to make the deployment of ROS2-based multi-robot applications simple, reliable, and scalable. The high-level overview is shown in Fig. 1. KubeROS abstracts all computing devices, including the robot's on-board computers, edge servers/virtual machines (VMs), and cloud VMs, as a unified computing infrastructure. The system administrator uses extensible libraries for system setup, network configuration, and VM provisioning.

Researchers and developers can easily use this platform to deploy their ROS2-based applications with a configuration file that specifies the deployment preferences, resource requirements, and allowed network conditions. Depending on the deployment configuration and the system status, KubeROS orchestrates the software across the entire cluster to achieve improved performance and efficient resource utilization. For example, a motion planning module that prefers more CPU resources for better planning results is deployed to the edge or cloud if the network latency satisfies the requirements. To our knowledge, KubeROS is the first unified platform that fuses the technologies from cloud native computing for deploying the ROS2-based robotic applications and is ideal for the large-scale system.

KubeROS aims to be a Platform as a Service (PaaS) for robotics. It has been developed to meet the following design principles:

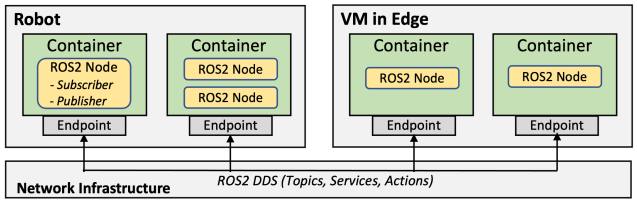
- *Unified Resource Abstraction*: All robot hardware and computing devices (onboard, edge, and cloud) are managed as resources.
- *Automated and scalable deployment*: Depending on the configuration and system state, the containerized software can be automatically deployed to the appropriate device. KubeROS should be able to allocate more compute resources to handle the peak workload.
- *Easy of use*: After setting up the system infrastructure, users only need to write a simple deployment description. No modification of the existing code is required.
- *Security*: Robots and edge computers should be on an isolated local network. Communication between the local network and the cloud should be secured by a VPN tunnel or using secure gRPC as a bridge.
- *high availability*: The core components of KubeROS should be deployed redundantly on multiple dedicated edge servers to avoid single-point of failure.
- *Multi-Fleet Support*: KubeROS should support flexible deployment of the software across multiple robot fleets.
- *high extensibility*: Kubernetes community tools should integrate easily with KubeROS, e.g. Prometheus [14] for system monitoring.

## II. Related Work

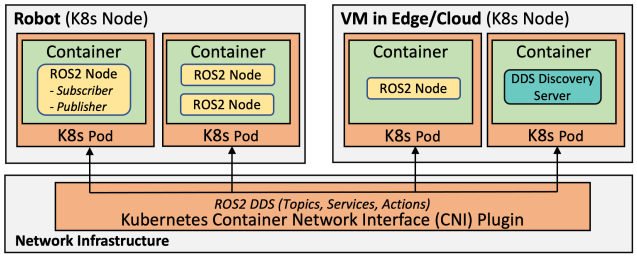
Recent research on cloud robotics has focused on leveraging the massive computing capacity, large-scale data storage, and shared knowledge base to help robots perform various tasks [5], [6]. Lee *et al.* proposed moving Deep Learning-based object recognition processing to the cloud to achieve real-time performance for unmanned aerial vehicles (UAVs) [15]. Beksi *et al.* presented a cloud-based object recognition engine for training large data sets and classifying 3D point clouds for robotics [16]. Kehoe *et al.* presented a cloud-based method for object detection and pose estimation [17]. To facilitate robotic grasping, researchers have proposed to use cloud computing from various perspectives, such as for grasp analyzing with parallelized Monte Carlo Sampling [18], tolerance bounds estimating [19], and prediction of grasping uncertainties [20]. In terms of motion planning, researchers explored the potential of using cloud computing to improve planning performance, such as roadmap precomputation [21], multi-query motion planning [22], [23], and muti-robot path planning in a crowded environment [24].

In addition to the benefits of leveraging massive computing power, researchers are exploring the model of software-as-a-service (SaaS) solutions for robotics, such as Robot Control as a Service (RCaaS) [25], Berkeley Robotics and Automation as a Service (BRaaS) [26], Dex-Net as a Service (DNaaS) [27], and Robot Inference and Learning as a Service (RILaaS) [28]. These SaaS solutions allow robots to access advanced algorithms and massive computing power without complicated installation and configuration. In comparison, KubeROS is designed as a Platform as a Service (PaaS) to deploy ROS2 software that can serve as a service to robots.

Regarding the general framework for leveraging cloud computing, the RoboEarth project focuses on building a knowledge repository to store the object models descriptions, and manipulation instructions in the cloud [29]. As its cloud engine, Rapyuta allows robots to move the computation to the cloud through a full duplex web-socket communication and provides access to the RoboEarth knowledge repository [30], [31]. Toffetti *et al.* proposed an Enterprise Cloud Robotics Platform (ECRP) solution for developing robotic applications based on ROS1 and discussed the design requirements and open challenges [32]. To gain access to the cloud, the frameworks FogROS [33] and FogROS2 [34] provide an easy way to establish a secure connection, automatically provision a cloud computer, and deploy the ROS nodes to the cloud by using an extended ROS launch script. The experiments with different tasks show that the additional latency can be eliminated by the use of high-performance cloud computing resources. Both frameworks allow researchers to deploy their software component in the cloud and require basic knowledge for configuration, such as selecting the instance type and region and setting up the authorization token. In comparison, KubeROS focuses on automatically deploying the entire application, which contains multiple software modules, especially also on a scalable fleet of robots. Developers do not have to explicitly select the resource type. KubeROS selects the location and instance type based on the deployment configuration, resource availability and current network conditions. No prior knowledge of cloud computing is required to use KubeROS. The infrastructure administrator performs the setup and configuration of KubeROS and establishes the connection to the cloud. In addition, KubeROS supports deployment with edge-only resources when the connection to the cloud is not viable due to high security requirements.




(a) Running containerized ROS2 application across multiple machines




(b) Running containerized ROS2 application on a Kubernetes cluster

Fig. 2. **Deployment of ROS2 on multiple machines:** (a) shows the classic setup for deploying containerized ROS2 nodes to robots and edge servers using the Docker host network model. (b) shows the deployment of containerized ROS2 nodes in a Kubernetes cluster. A pod with DDS discovery server is created to provide the DDS discovery service.

## III. Background

KubeROS is designed to deploy ROS2-based robotic applications and uses Kubernetes as the underlying framework for software orchestration.

### A. ROS2 and DDS

In ROS2, the *node* is the smallest functional unit. A software module contains several nodes, that communicate with each other via predefined communication patterns: *topics*, *services*, and *actions*. As communication middleware, ROS2 uses Data Distributed Service (DDS), a specification defined by the Object Management Group (OMG), for which there are different implementations from different vendors [35]. DDS uses UDP/IP for data transmission by default and introduces Quality of Service (QoS) to expose these settings to optimize message transmission performance.

The software modules deployed on the robots are packaged into an isolated container to avoid dependency conflicts. To enable communication between different containers, certain ports must be exposed, and an suitable network driver has to be selected. Fig. 2 (a) shows an simplified example of deploying multiple ROS2 nodes on a robot and a VM using the docker host network model. In this example, the container uses host networking directly. IP multicasting must be enabled for the DDS to discover the nodes.

### B. Kubernetes

In Kubernetes, the *node* refers to a physical or virtual machine managed by the control plane and contains necessary components, including *kubelet*, *kube-proxy*, and container runtime for running the workload [36]. The *pod* is the smallest deployable unit that includes one or more containers. The pod appears within the cluster as a logical host with its own unique cluster-wide IP address for communicating with other pods on any other node without NAT (Network Address Translation). Kubernetes specifies the Container Network Interface (CNI) to dynamically configure network resources and supports various plugins from different vendors. In KubeROS, we use the open source solution Calico [37] to create an encapsulated overlay network.

As IP multi-casting is not supported by most Kubernetes CNI plugins and is by default disabled in most WiFi environments due to its excessive amounts of traffic. The latter can often be the causes for a negative impact on network performance. KubeROS uses the discovery server from FastDDS [38] for participants, instead of the standard DDS simple discovery protocol, see Fig. 2 (b). KubeROS creates redundant discovery servers for each robot fleet and exposes these using the Kubernetes *service*. By deploying the ROS2 modules, KubeROS extends the command arguments in the pod description by setting the environment variable `ROS_DISCOVERY_SERVER` to point to discovery servers.

## IV. Architecture and Paradigm

KubeROS consists of the following core components: KubeROS API server, resource manager, software manager, deployment controller and scheduler, and the KubeROS-Bridge. Fig. 3 shows its overall architecture.

### A. Infrastructure Abstraction and Communication

In most robotics applications, especially in the industrial sector, security is paramount and robots may not have direct open access to the cloud, so the core components of KubeROS are designed to run in the main cluster at the edge. The control plane is deployed on one or more dedicated physical servers to improve the platform availability. Three types of computing resources are abstracted: the robot's onboard computers, the edge computers, and the computing resources in the cloud. All local computing units are added as nodes to the main KubeROS cluster. To gain a secure access to the cloud, KubeROS provides two mechanisms:

*Virtual Private Network (VPN):* For robotic deployment in manufacturing, logistics, and in facilities such as hospitals, the connection to the public cloud are usually performed by the IT administrator for security reasons. In this case, additional VMs can be added to the cluster as normal work nodes. Communication between the cloud and the local network is established through a secure VPN tunnel.

*KubeROS-Bridge over gRPC:* For the use case where the robots retrieve the resource from an another cluster by using the same ROS2-based software modules without adaptation, KubeROS introduces a message bridge (called KubeROS-Bridge) [39] between clusters. Before deploying the ROS2 modules in the third-party cluster, KubeROS instantiates a gRPC server in the main cluster to subscribe to the ROS2 messages and encode them into protocol buffers (Protobuf). Meanwhile, a gRPC client is created to republish the received messages in ROS2. With this approach, the ROS2 modules deployed in third-party clusters serve as a cloud software service to robots.

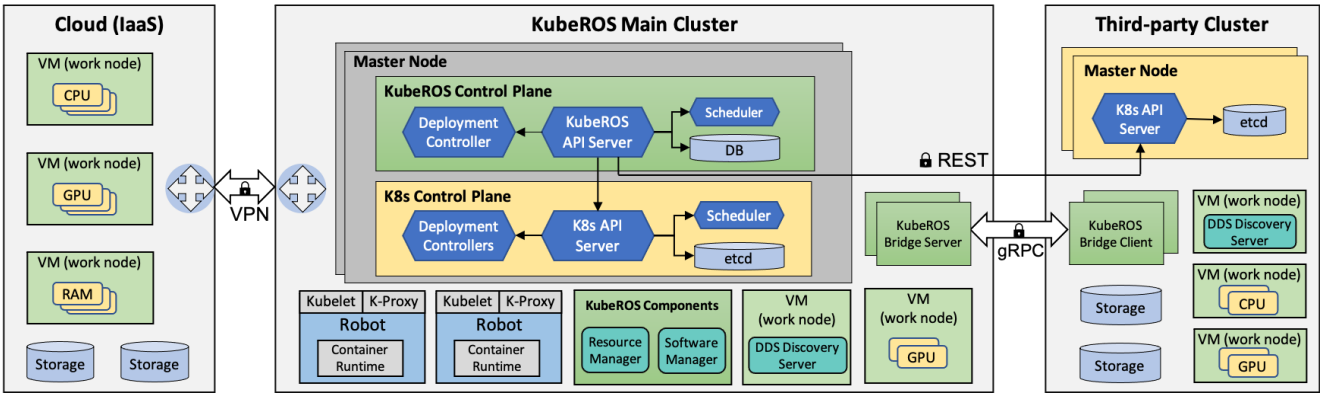



Fig. 3. **Architecture of the KubeROS platform:** The core components of KubeROS are installed at the edge. The robots and edge computers form the main cluster. The VMs in the cloud are added to the main cluster through a secure VPN tunnel. KubeROS obtains computing resources from third-party clusters through the KubeROS-Bridge over gRPC. The KubeROS API Server provides REST services to receive and process the deployment requests.

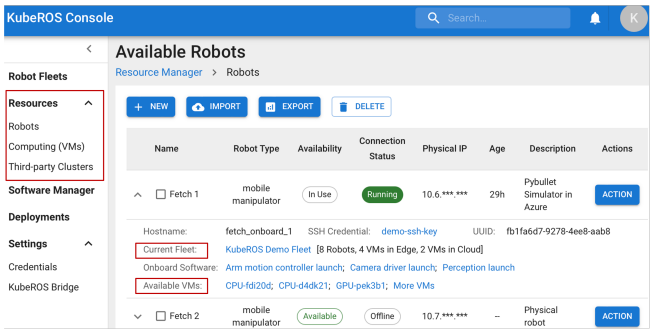



Fig. 4. **Snapshot of resource manager:** The front-end provides a graphic interface for registering robots, managing computing resources and monitoring the system state.

## B. Resource Manager

Compared to pure software deployments in Kubernetes, the hardware resources in robotic applications are not agnostic. Deployment depends on the system state and task requirements, such as network latency for real-time control loops. For this reason, KubeROS includes a built-in hardware manager to store the hardware specifics and to provide essential system state for software orchestration, see the graphic interface in Fig. 4. To register new robots or hardware resources, the system administrator can either use the command *kubeadm join* manually or upload the ssh public keys to KubeROS for an automated configuration. To connect to a third-party cluster, KubeROS provides an interface to upload the cluster certificate and service token. As the hardware and network setup is usually complex, the resource manager is designed to be maintained by the system administrator. For developers and researchers, the resource manager provides an easy-to-use interface for exploring available computing resources, creating robot fleets and monitoring the software modules that are deployed.

## C. ROS2 Integration and Deployment YAML

ROS2 has a startup system that allows multiple nodes to be started simultaneously as a software module. A robotics application is made up of many such modules. KubeROS introduces a deployment configuration in the YAML manifest to describe the software module to be launched and to specify deployment requirements and preferences. An example is shown in Listing 1. The KubeROS API server receives and processes the deployment requests via the REST API. Once the deployment request is received, the KubeROS API server parses the YAML file to the corresponding Kubernetes deployment description by adding device metadata, such as device ID in `nodeSelector`, and then calls the Kubernetes API server to deploy the containerized modules to the appropriate devices.

```
metadata:
  name: kuberos-demo-picking
  targetFleet: test-fleet
  robotSelector:
    robot: [fetch-1, fetch-2]
rosModules:
- name: fetch-motion-planner
  image: container-registry/planner:tag
  launch: [ros2 launch package launch_file]
  preference: cloud
  requirements:
    max_latency: 200 ms
- name: fetch-controller-foxy
  image: container-registry/controller:tag
  launch: [ros2 run package script]
  preference: onboard
  requirements:
    onboard: true
```

Listing 1. **KubeROS deployment configuration snippet:** This example shows the basic deployment description of two ROS2 modules in the target fleet, which is preconfigured via the resource manager interface. The motion planning module is preferably deployed in the cloud. The motion control module must be installed on the robot's on-board computer.

## D. Software Manager

As the application is broken down into a number of small modules and deployed in different locations, software complexity increases dramatically. To improve the software manageability and reusability, KubeROS introduces a software manager in combination with the resource manager.

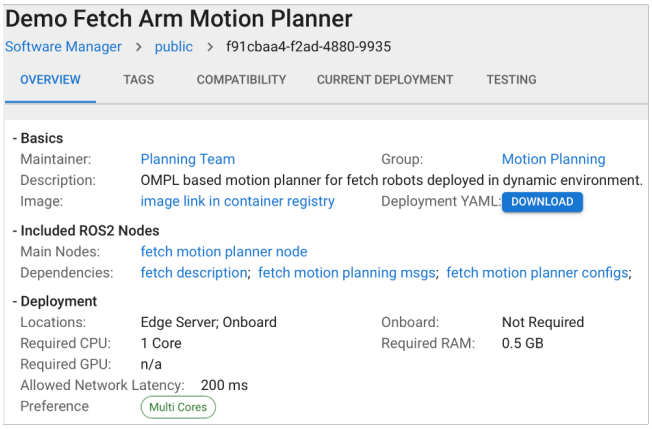



Fig. 5. **Snapshot of software manager**: The front-end interface allows users to maintain the software version, dependencies, compatibility, and deployment requirements.

The graphic user interface is shown in the Fig. 5. It stores the essential deployment configuration, including the container image address, dependencies, hardware requirements, and preferences. This allows users to better manage the software lifecycle and to track the deployment status, thereby improving the overall software stability in large-scale deployments.

### *E. Deployment and Orchestration*

Following the orchestration concept in Kubernetes, KubeROS uses its own scheduler to select the appropriate nodes to run the software modules depending on the specifics of the robotic tasks. For each deployment, the scheduler retrieves the system state from the resource manager and matches the requirements and the preferences for node assignment. The controller runs periodically to check the status of the ROS2 module, and in the event of a failure or unstable network, it will redeploy the module to the next appropriate location. Generally, for a module running on an on-board computer or on a VM in the main cluster, KubeROS sets the `nodeSelector` to the robot device ID, which is unique across the cluster. For a module running in a third-party cluster as a cloud service, KubeROS creates a bridge server and client on both sides and then deploys the software module by calling the third-party cluster's API server with the stored credential and service token.

### *F. KubeROS Workflow*

KubeROS workflow is summarized in Fig. 6. The infrastructure engineer is responsible for: (1) creating the Kubernetes cluster using *Kubeadm*, installing KubeROS components, and configuring the edge VMs; (2) setting up the provision automation module to get access to the cloud; (3) registering the robots as nodes into the KubeROS main cluster. The developers use the platform without prior knowledge of cloud computing: (4) select a robot from the pool to develop and test the software; (5) create the robot fleet for production and write the deployment description file. (6) KubeROS deploys and orchestrates all software modules across the robot fleet.

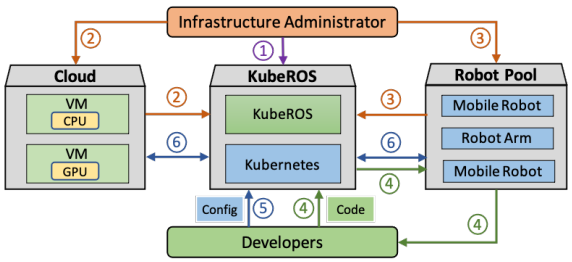



Fig. 6. **KubeROS Workflow:** The infrastructure administrator is responsible for (1) installing KubeROS and setup edge VMs, (2) provisioning VMs from the cloud provider, and (3) adding robots. The developers use KubeROS to (4) select the robots to develop software and (5) configure the deployment. (6) KubeROS deploys the software to the entire fleet.

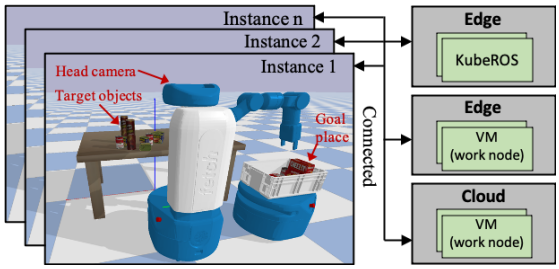



Fig. 7. **Example Application**: A fleet of simulated fetch mobile manipulators in a pick-and-place application. The simulator runs in an isolated container in a VM in Azure cloud. We use VMs in the Azure cloud es edge resources and VMs from AWS as cloud resources. The fetch robot is controlled by a set of ROS2 software modules that deployed by KubeROS.

## V. Use-Case and Evaluation

We choose a classical pick-and-place application to demonstrate the applicability of KubeROS and to investigate different deployment strategies, see Fig. 7. In each task period, the mobile manipulator [40] detects a target object, grabs it, and then drops it on another mobile platform. To study the scalability of the KubeROS platform, we conduct experiments using VMs from Azure and AWS to simulate the system at different scales. A simulation environment was created in Pybullet [41] to simulate the behavior of the real robot. Only two interfaces are provided via gRPC for ROS2: the sensor interface, which publishes the RGB-D image of the head camera, and the control interface, which publishes the joint state and receives the control commands.

Many tasks in this application can benefit from the use of edge and cloud resources. In this paper, we use a vision-based motion planning task to evaluate the system performance in terms of latency and resource utilization. For this purpose, we deploy four software modules in ROS2 (distribution: foxy) via KubeROS, see Fig. 8. First, a perception module computes the sparse point cloud based on the RGB-D image and publishes them as collision objects in the planning scene. Then, a motion planning module based on MoveIt! [42] finds a collision-free path using the parallel planning tools of OMPL [43]. Third, a control module computes the trajectory using the time-optimal trajectory generation approach [44] and sends it to the simulated robot. Finally, a task coordinator module simulates the dynamic workload and sends the planning request.

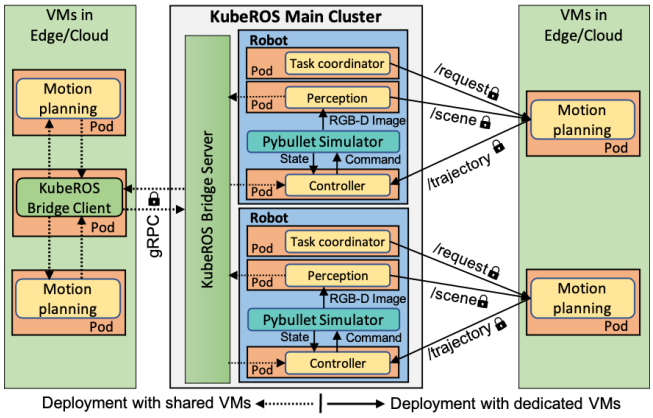
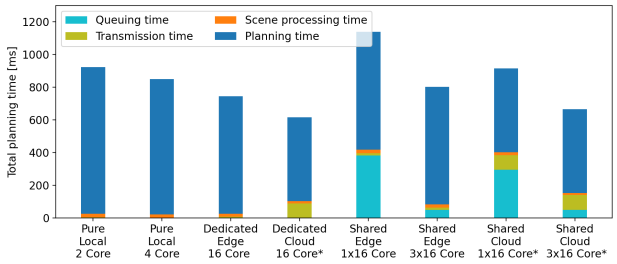
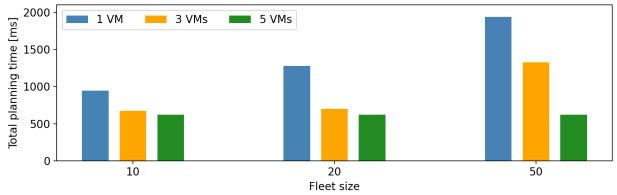



Fig. 8. **Deployment strategies:** Four software modules are deployed by KubeROS: task coordinator, perception, controller and motion planning. All modules are deployed on the robot with the exception of motion planning. Depending on the deployment strategy, the motion planning module is deployed on the robot, on dedicated or shared VMs in the edge/cloud.




(a) Comparison of different deployment strategies




(b) Comparison of different fleet sizes

Fig. 9. **Motion planning performance:** A motion planning instance receives the planning request, adds the collision objects to the planning scene, and finds a feasible collision-free path. The queuing time is the waiting time of the planning request in receive buffer before being processed when using the shared instances. (a) compares the impact of the different deployment strategies. (b) shows the influence of the scalable deployment in the cloud under different fleet sizes.

### A. Deployment Strategies and Performance

The performance of the robot fleet depends on the fleet size and the workload. To compare the effects of different deployment strategies, we conducted an experiment with 10 robots. Each robot randomly requests motion planning every 30 seconds. The software modules except motion planning are installed on the robots. We compare the performance of the PRM* planner [45] with different deployment strategies:

- *Pure Local*: Motion planning is deployed on the robot. Azure general purpose `D2ds_v4` and `D4ds_v4` (west europe) is used as simulated onboard computer.
- *Dedicated VMs*: Each robot owns a dedicated VM for motion planning. The general purpose Azure `D16ds_v4` is used as edge VM, and computing optimized AWS `c6i.4xlarge` (us-east-1) as cloud VM, which has higher performance for planning execution.
- *Shared VMs*: The fleet shares a certain number of VMs.

We ran 100 task periods in each deployment and measured the total motion planning time as the difference between sending the request and receiving the response. Fig. 9 (a) shows the average total planning time. Using high performance VMs from edge and cloud reduces the planing time[1]. The additional network latency can be compensated. Deploying with dedicated VMs in the cloud achieves the best performance. Deployment with shared VMs caused high queuing time, which could be reduced by allocating more motion planning instances.

### B. Scalability and Resource Utilization

We deployed the motion planning module in the cloud for the robots in different fleet sizes. Each robot randomly sends a planning request to the cloud every 30 seconds. KubeROS-Bridge is used as a load balancer to distribute the requests across multiple instances. As shown in Fig. 9 (b), as the fleet size increases, more instances are needed to handle the concurrent requests and reduce the queuing time. The overall performance of motion planning is improved by using a certain number of work instances, but it is strongly depending on the fleet size and workload. Therefore, the number of allocated instances for motion planning must be dynamically adjusted to achieve an optimized resource utilization.

## VI. CONCLUSION AND OUTLOOK

We introduced KubeROS, a unified and easy-to-use platform built on top of Kubernetes to facilitate automated and scalable deployment of computationally intensive ROS2-based applications to a robot or a fleet of robots by leveraging cloud and edge computing. KubeROS abstracts the robots, edge and cloud into a hybrid computing infrastructure. With the built-in resource and software manager, KubeROS enables developers to deploy their software on multiple devices with minimal system setup effort. KubeROS offers different deployment strategies to meet the application's needs. In the example application with a fleet of simulated mobile manipulators, we demonstrate the ease of use of KubeROS and the performance improvement by using an appropriate deployment strategy. As the system size increases, KubeROS uses shared VMs to improve resource utilization.

KubeROS is an ongoing project that spans two areas: Robotics and Cloud Computing. After implementing the prototype, we are currently working on the stable version. We plan to release this project as open source under Apache License, version 2.0. Meanwhile, we will make the sample application available in the cloud and present further evaluation on real robots in various applications.

## ACKNOWLEDGMENT

This research is being conducted as part of the KI5GRob project funded by the German Federal Ministry of Education and Research (BMBF) under project number 13FH579KX9.

[1]Using a multi-core motion planner can provide significant improvement [46]. We use the parallel planners provided by OMPL to show generality.

## REFERENCES


[1] L. Liu, W. Ouyang, X. Wang, P. Fieguth, J. Chen, X. Liu, and M. Pietikäinen, "Deep learning for generic object detection: A survey," *International journal of computer vision*, vol. 128, no. 2, pp. 261–318, 2020.

[2] S. G. Tzafestas, "Mobile robot control and navigation: A global overview," *Journal of Intelligent & Robotic Systems*, vol. 91, no. 1, pp. 35–58, 2018.

[3] J. Bohg, A. Morales, T. Asfour, and D. Kragic, "Data-driven grasp synthesis—a survey," *IEEE Transactions on robotics*, vol. 30, no. 2, pp. 289–309, 2013.

[4] G. Hu, W. P. Tay, and Y. Wen, "Cloud robotics: architecture, challenges and applications," *IEEE network*, vol. 26, no. 3, pp. 21–28, 2012.

[5] B. Kehoe, S. Patil, P. Abbeel, and K. Goldberg, "A survey of research on cloud robotics and automation," *IEEE Transactions on automation science and engineering*, vol. 12, no. 2, pp. 398–409, 2015.

[6] O. Saha and P. Dasgupta, "A comprehensive survey of recent trends in cloud robotics architectures and applications," *Robotics*, vol. 7, no. 3, p. 47, 2018.

[7] V. C. Pujol and S. Dustdar, "Fog robotics—understanding the research challenges," *IEEE Internet Computing*, vol. 25, no. 5, pp. 10–17, 2021.

[8] A. K. Tanwani, N. Mor, J. Kubiatowicz, J. E. Gonzalez, and K. Goldberg, "A fog robotics approach to deep robot learning: Application to object recognition and grasp planning in surface decluttering," in *2019 international conference on robotics and automation (ICRA)*. IEEE, 2019, pp. 4559–4566.

[9] N. Tian, A. K. Tanwani, J. Chen, M. Ma, R. Zhang, B. Huang, K. Goldberg, and S. Sojoudi, "A fog robotic system for dynamic visual servoing," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 1982–1988.

[10] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng, *et al.*, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[11] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, 2022.

[12] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, omega, and kubernetes," *Communications of the ACM*, vol. 59, no. 5, pp. 50–57, 2016.

[13] The Kubernetes Authors, "Kubernetes v1.22 documentation," v1-22.docs.kubernetes.io, Accessed: 2022-09-06.

[14] J. Turnbull, *Monitoring with Prometheus*. Turnbull Press, 2018.

[15] J. Lee, J. Wang, D. Crandall, S. Šabanović, and G. Fox, "Real-time, cloud-based object detection for unmanned aerial vehicles," in *2017 First IEEE International Conference on Robotic Computing (IRC)*. IEEE, 2017, pp. 36–43.

[16] W. J. Beksi, J. Spruth, and N. Papanikolopoulos, "Core: A cloud-based object recognition engine for robotics," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015, pp. 4512–4517.

[17] B. Kehoe, A. Matsukawa, S. Candido, J. Kuffner, and K. Goldberg, "Cloud-based robot grasping with the google object recognition engine," in *2013 IEEE International Conference on Robotics and Automation*. IEEE, 2013, pp. 4263–4270.

[18] B. Kehoe, D. Warrier, S. Patil, and K. Goldberg, "Cloud-based grasp analysis and planning for toleranced parts using parallelized monte carlo sampling," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 2, pp. 455–470, 2014.

[19] B. Kehoe, D. Berenson, and K. Goldberg, "Estimating part tolerance bounds based on adaptive cloud-based grasp planning with slip," in *2012 IEEE International Conference on Automation Science and Engineering (CASE)*. IEEE, 2012, pp. 1106–1113.

[20] B. Kehoe, D. Berenson, K. Goldberg, "Toward cloud-based grasping with uncertainty in shape: Estimating lower bounds on achieving force closure with zero-slip push grasps," in *2012 IEEE International Conference on Robotics and Automation*. IEEE, 2012, pp. 576–583.

[21] K. Bekris, R. Shome, A. Krontiris, and A. Dobson, "Cloud automation: Precomputing roadmaps for flexible manipulation," *IEEE Robotics & Automation Magazine*, vol. 22, no. 2, pp. 41–50, 2015.

[22] J. Ichnowski, J. Prins, and R. Alterovitz, "The economic case for cloud-based computation for robot motion planning," in *Robotics Research*. Springer, 2020, pp. 59–65.

[23] J. Ichnowski, J. Prins, R. Alterovitz, "Cloud-based motion plan computation for power-constrained robots," in *Algorithmic Foundations of Robotics XII*. Springer, 2020, pp. 96–111.

[24] N. Zagradjanin, D. Pamucar, and K. Jovanovic, "Cloud-based multi-robot path planning in complex and crowded environment with multi-criteria decision making using full consistency method," *Symmetry*, vol. 11, no. 10, p. 1241, 2019.

[25] A. Vick, V. Vonásek, R. Pěnička, and J. Krüger, "Robot control as a service—towards cloud-based motion planning and control for industrial robots," in *2015 10th International Workshop on Robot Motion and Control (RoMoCo)*. IEEE, 2015, pp. 33–39.

[26] N. Tian, M. Matl, J. Mahler, Y. X. Zhou, S. Staszak, C. Correa, S. Zheng, Q. Li, R. Zhang, and K. Goldberg, "A cloud robot system using the dexterity network and berkeley robotics and automation as a service (brass)," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 1615–1622.

[27] P. Li, B. DeRose, J. Mahler, J. A. Ojea, A. K. Tanwani, and K. Goldberg, "Dex-net as a service (dnaas): A cloud-based robust robot grasp planning system," in *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*, 2018, pp. 1420–1427.

[28] A. K. Tanwani, R. Anand, J. E. Gonzalez, and K. Goldberg, "Rilaas: Robot inference and learning as a service," *IEEE Robotics and Automation Letters*, vol. 5, no. 3, pp. 4423–4430, 2020.

[29] M. Waibel, M. Beetz, J. Civera, R. d'Andrea, J. Elfring, D. Galvez-Lopez, K. Häussermann, R. Janssen, J. Montiel, A. Perzylo, *et al.*, "Roboearth," *IEEE Robotics & Automation Magazine*, vol. 18, no. 2, pp. 69–82, 2011.

[30] D. Hunziker, M. Gajamohan, M. Waibel, and R. D'Andrea, "Rapyuta: The roboearth cloud engine," in *2013 IEEE international conference on robotics and automation*. IEEE, 2013, pp. 438–444.

[31] G. Mohanarajah, D. Hunziker, R. D'Andrea, and M. Waibel, "Rapyuta: A cloud robotics platform," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 2, pp. 481–493, 2014.

[32] G. Toffetti and T. M. Bohnert, "Cloud robotics with ros," in *Robot operating system (ROS)*. Springer, 2020, pp. 119–146.

[33] K. E. Chen, Y. Liang, N. Jha, J. Ichnowski, M. Danielczuk, J. Gonzalez, J. Kubiatowicz, and K. Goldberg, "Fogros: An adaptive framework for automating fog robotics deployment," in *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*. IEEE, 2021, pp. 2035–2042.

[34] J. Ichnowski, K. Chen, K. Dharmarajan, S. Adebola, M. Danielczuk, V. Mayoral-Vilches, H. Zhan, D. Xu, R. Ghassemi, J. Kubiatowicz, *et al.*, "Fogros 2: An adaptive and extensible platform for cloud and fog robotics using ros 2," *arXiv preprint arXiv:2205.09778*, 2022.

[35] Object Management Group, "Data distribution service (dds), version 1.4," https://www.omg.org/spec/DDS/1.4, Accessed: 2022-09-06.

[36] M. Luksa, *Kubernetes in action*. Simon and Schuster, 2017.

[37] Tigera, Inc., "Calico," https://www.tigera.io/project-calico/, Accessed: 2022-09-06.

[38] eProsima, "Fast dds documentation, release 2.1.1," https://fast-dds.docs.eprosima.com/_/downloads/en/v2.1.1/pdf, Accessed: 2022-09-06.

[39] grpc.io, "gRPC," https://grpc.io, Accessed: 2022-09-06.

[40] M. Wise, M. Ferguson, D. King, E. Diehr, and D. Dymesich, "Fetch and freight: Standard platforms for service robot applications," in *Workshop on autonomous mobile service robots*, 2016.

[41] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," http://pybullet.org, 2016–2022.

[42] D. Coleman, I. A. Sucan, S. Chitta, and N. Correll, "Reducing the barrier to entry of complex robotic software: a moveit! case study," *Journal of Software Engineering for Robotics*, 2014.

[43] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, December 2012, https://ompl.kavrakilab.org.

[44] T. Kunz and M. Stilman, "Time-optimal trajectory generation for path following with bounded acceleration and velocity," *Robotics: Science and Systems VIII*, pp. 1–8, 2012.

[45] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.

[46] J. Ichnowski and R. Alterovitz, "Motion planning templates: A motion planning framework for robots with low-power cpus," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 612–618.