Life Cycle Automation of IoT Services over Multiple Edge Clouds

Mafeni Vitumbiko
School of Electronic Engineering
Soongsil University
Seoul, Korea
vitumafeni@dcn.ss.ac.kr

Younghan Kim
School of Electronic Engineering
Soongsil University
Seoul, Korea
younghak@ssu.ac.kr

***Abstract*—In the rapidly evolving landscape of edge computing for IoT service provisioning, the challenges of managing and deploying applications across a diverse array of edge devices in multi-edge setups are evident. Edge devices, often situated on users' premises, exhibit heterogeneity in hardware and software requirements. Furthermore, manual cluster setup and software packaging for each edge site become impractical when trying to cater to the specific needs of each edge device. Developers are often required to set up an edge cluster, continuously create software packages tailored to individual edge devices and end-user requirements, all while managing multiple application versions and redeploying them across a fleet of edge devices. To address these challenges, this paper proposes an innovative solution that simplifies IoT microservice deployment across multiple edge sites, accommodating the intricacies of device diversity and different end user requirements.**

***Index Terms*—IoT, Multi-edge, Kubernetes, CI/CD**

I. Introduction

The Internet of Things (IoT) is revolutionizing the conventional methods of conducting business in sectors like precision agriculture, automotive manufacturing, health monitoring, and more. IoT systems are typically extensively dispersed across multiple geographic areas, often incorporating cloud and edge computing methods that operate in proximity to where intelligence and processing capabilities are deployed [1]. Edge computing brings the services and cloud utilities closer to the devices and end users enabling fast processing, quick application response time and low latency to delay-sensitive applications [2]. Companies initially begin with small-node clusters and eventually scale up to deploy hundreds, or even thousands, of clusters to support their applications [3]. A multi-cluster setup involves several clusters deployed across one or many data centers separating development and production instances. Nonetheless, overseeing multiple Kubernetes clusters presents distinct challenges. Each cluster has its own control plane, necessitating individual configuration for networking, security, and policies. Additionally, achieving automated deployments and efficient resource management across these clusters is exceedingly difficult without a centralized solution. This would involve manual deployment and the synchronization of configurations for each cluster, which is not a scalable approach [4]. One way to effectively manage multi clusters is through Continuous Integration and Continuous Deployment (CI/CD) platforms for Kubernetes [5]. CI/CD is used to support the collaborative software development process. CI/CD automates a wide range of activities in the development workflow such as testing, linting, updating dependencies, creating and deploying releases, and so on [6]. The IoT industry faces the challenge of rapid innovation, necessitating faster software updates to edge devices. However, as the industry grows, complexities arise in delivering large-scale updates, like customizing service software as a service for diverse edge devices and different user requirements [7]. This article discusses the concept of CI/CD for IoT edge, which entails not only automating the setup of IoT clusters and the building and testing of software modules for edge devices but also automating the release process. The article presents an architectural model for a highly distributed IoT system spanning both cloud and edge components, along with a CI/CD workflow for customized services on edge nodes. This approach enables software vendors to deliver their software services on demand through cloud providers, while the installation and configuration of software at the edge are managed through a defined set of pipelines. We select an open source project Nephio to bootstrap IoT edge clusters backed up by Kubernetes Controllers and Custom Resource Definitions (CRDs) for provisioning IoT ”leaf” nodes. Furthermore, we leverage Kpt and Config sync to accomplish a CI/CD approach to enable rapid and automated software deployment. Our main contributions are as follows;

- Bootstrapping IoT Infrastructure: We utilize open-source projects to bootstrap Kind clusters for IoT deployment, ensuring efficient and reliable operations.
- CI/CD Automation for IoT Edge: Our solution automates the deployment of IoT services across multi-edge IoT environments.
- Heterogeneous Edge Management: We integrate modules or services for seamless management of diverse edge IoT requirements.

The rest of the article is structured as follows. Section II presents related work. Section III presents the proposed system description. Section IV presents preliminary experiment and evaluation of the proposed work. Finally, section V presents our future planned directs and conclusions.

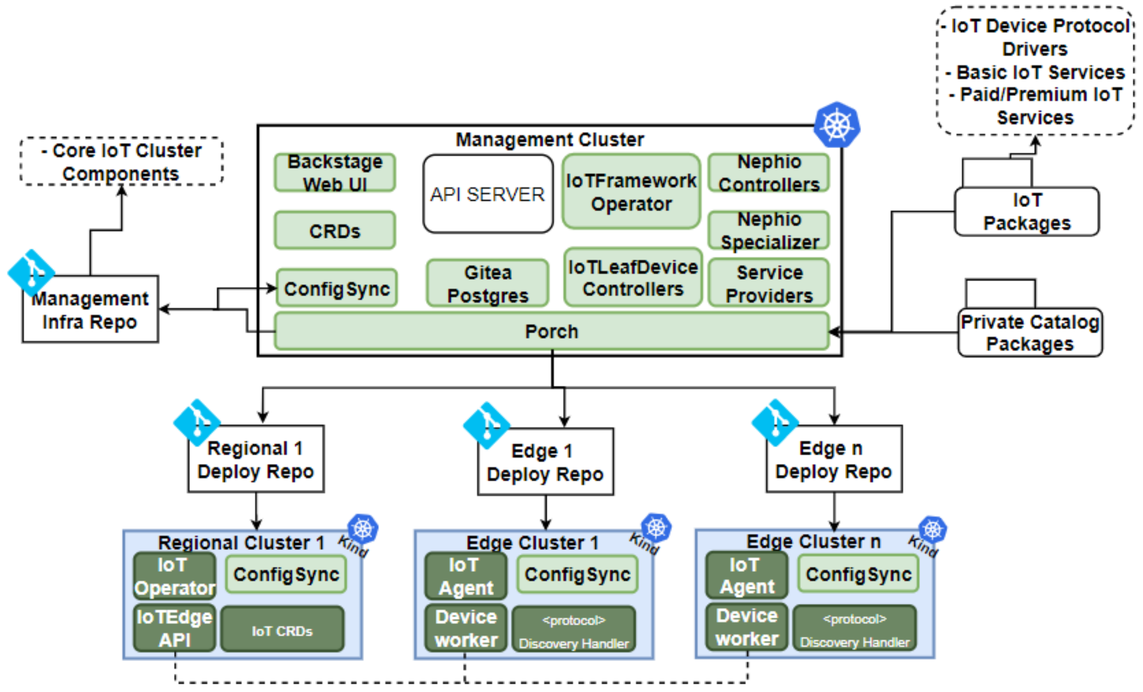

Fig. 1. Proposed Architecture

II. Related Work

This section presents various existing studies in relation to our study with distinct components and focus.

The paper [7] discusses implementing DevOps in IoT edge computing, emphasizing precision agriculture as a case study. It showcases how DevOps enables continuous delivery of tailored software updates to edge devices, potentially leading to new business models at the IoT edge. The study outlines an architectural model for a distributed IoT system and a continuous delivery process for edge nodes, demonstrating its practical application.

The literature in [8] addresses the lack of proper tool support for orchestrating and deploying software systems across IoT, edge, and cloud environments. It introduces a framework called GeneSIS, which enables the continuous orchestration and deployment of IoT systems. GeneSIS is designed to handle the heterogeneity present at each level (IoT, edge, and cloud) and facilitates the control of software systems across these infrastructures.

Authors in [9] examines multi-domain Kubernetes setups, which use multiple clusters for better workload management. It introduces an automatic method, the Multi-Cluster Orchestrator, to generate network security policies in each cluster. This reduces configuration errors, promotes consistent policies, and enables seamless communication between clusters in complex deployments.

Another study in [10], examines the use of GitOps, a DevOps practice, in Edge Computing for IoT. It investigates the feasibility of applying GitOps to IoT Edge Computing using CNCF tools and identifies drawbacks, including limitations in infrastructure provisioning and support for edge devices.

III. System Description

In this section, we introduce the main conceptual parts of our proposal and select an open source project, Nephio [12], as the framework software. We select an open source project Nephio to achieve our goals in this manuscript. Nephio is a Kubernetes-based system that automates network functions and infrastructure, enabling high-level intent expression and intelligent automation for cloud and edge setup and network configuration delivery [11]. We chose the approach used in the Nephio project and apply it for IoT management.

A. System Architecture

In our architecture, Fig. 1, we have three cluster types: the management cluster, the regional cluster, and the edge cluster. The management cluster is the central hub where Ansible scripts, using kpt packages, install all necessary components. All the components for bootstrapping are stored in Management Infra Repository. The management cluster serves as the central management point for both regional and edge clusters. Here, we introduce custom resources and controllers to manage and enable interaction with IoT devices on the edge. IoT devices are controllable from the management cluster, providing information about their connection to specific edge clusters and their operational status. On the other hand, the regional cluster comprises the control plane, IoT custom controllers, and custom resources, facilitating local management of IoT leaf and edge nodes. It also incorporates ConfigSync to

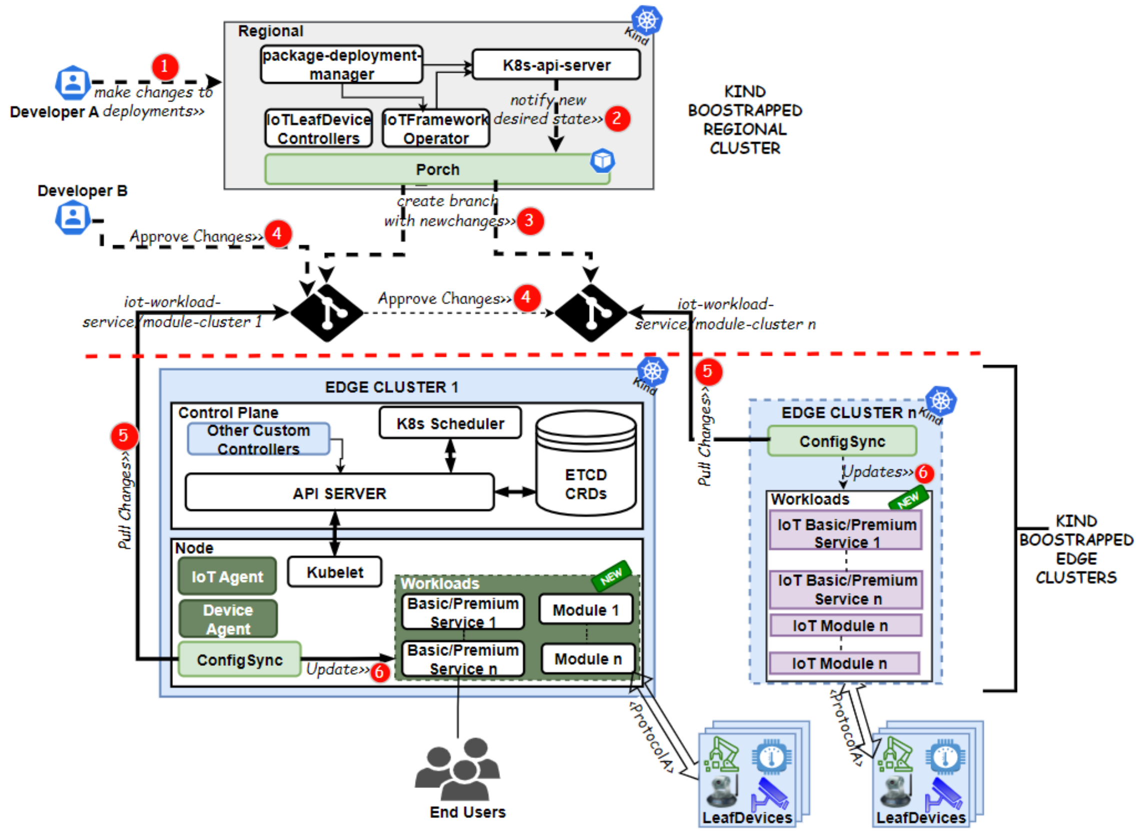

Fig. 2. CI/CD process step by step

ensure synchronization of IoT applications and Git repository updates. Similarly, the edge clusters utilize ConfigSync for the same synchronization purposes.

B. Automated CI/CD Pipeline

After bootstrapping the clusters and installing the IoT framework during setup, IoT services can be deployed depending on the site requirements. All services are bundled up in Kpt packages format stored in the github repositories for each site. Once we create//modify a resource, the deployment controller will fetch from here. In our pipeline, we assume the involvement of two developers interacting with two edge clusters where our IoT services are deployed. Different users may require various types of services, either free or paid, and IoT device modules are deployed based on the available leaf devices. Consequently, developer A is responsible for updating services on edge devices by modifying and customizing the configuration files. When service modifications are made to meet specific requirements, Porch is notified via the Kubernetes API server. This triggers Porch to fetch packages from the IoT repository, create a corresponding Git branch in the iot-workload-service-modules branch. To deploy these changes or updates, developer B must first approve them. Upon approval, ConfigSync, which monitors the repository, retrieves the latest changes and replaces the service on the edge device. This process is further illustrated in Fig. 2

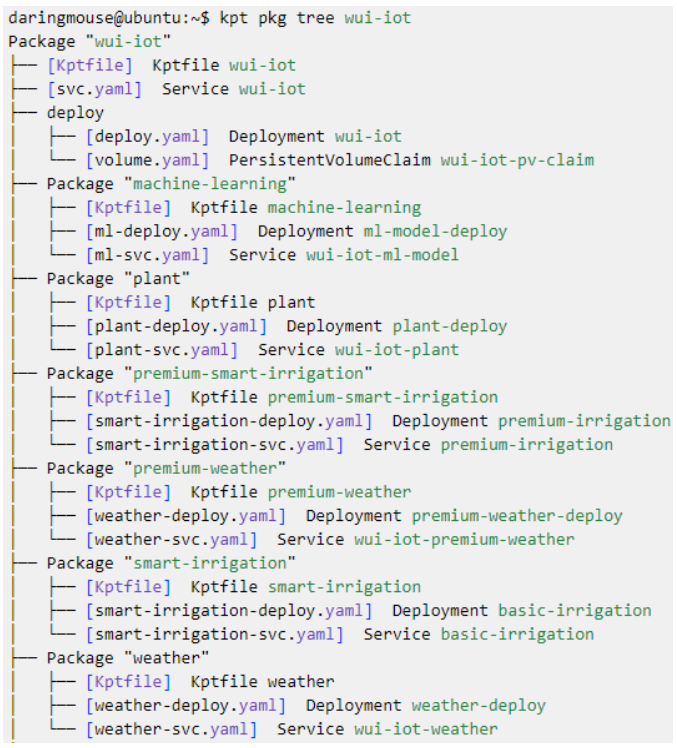
```
daringmouse@ubuntu:~$ kpt pkg tree wui-iot
Package "wui-iot"
├── [Kptfile]  Kptfile wui-iot
├── [svc.yaml]  Service wui-iot
├── deploy
│   ├── [deploy.yaml]  Deployment wui-iot
│   └── [volume.yaml]  PersistentVolumeClaim wui-iot-pv-claim
├── Package "machine-learning"
│   ├── [Kptfile]  Kptfile machine-learning
│   ├── [ml-deploy.yaml]  Deployment ml-model-deploy
│   └── [ml-svc.yaml]  Service wui-iot-ml-model
├── Package "plant"
│   ├── [Kptfile]  Kptfile plant
│   ├── [plant-deploy.yaml]  Deployment plant-deploy
│   └── [plant-svc.yaml]  Service wui-iot-plant
├── Package "premium-smart-irrigation"
│   ├── [Kptfile]  Kptfile premium-smart-irrigation
│   ├── [smart-irrigation-deploy.yaml]  Deployment premium-irrigation
│   └── [smart-irrigation-svc.yaml]  Service premium-irrigation
├── Package "premium-weather"
│   ├── [Kptfile]  Kptfile premium-weather
│   ├── [weather-deploy.yaml]  Deployment premium-weather-deploy
│   └── [weather-svc.yaml]  Service wui-iot-premium-weather
├── Package "smart-irrigation"
│   ├── [Kptfile]  Kptfile smart-irrigation
│   ├── [smart-irrigation-deploy.yaml]  Deployment basic-irrigation
│   └── [smart-irrigation-svc.yaml]  Service basic-irrigation
├── Package "weather"
│   ├── [Kptfile]  Kptfile weather
│   ├── [weather-deploy.yaml]  Deployment weather-deploy
│   └── [weather-svc.yaml]  Service wui-iot-weather
```

Fig. 3. Kpt IoT packages for different Agriculture IoT service requirements

Edge computing at the edge exhibits heterogeneous characteristics, primarily stemming from the presence of diverse IoT edge devices that demand various IoT modules for interfacing with these devices. Moreover, end-users may seek access to different categories of IoT services, including both premium and free offerings. Premium services often encompass advanced features, assisted by machine learning, in contrast to free services. To address these requirements, we consolidate all IoT services into a single Kpt package (see Fig. 3) and make necessary adjustments. The PackageVariantControllerSet is responsible for determining the deployment location for each service, which is orchestrated through Porch and subsequently pulled to the edge using ConfigSync.

IV. Preliminary Experiment and Evaluation

This section will focus on deploying microservices and verifying the feasibility and performance of our proposal on the multi edge setup. Our primary focus is on comparing the time required to update and deploy IoT applications at the edge once deployments and configurations are modified in the repositories and the central management cluster. To ensure the validity of our findings, we perform these tests using identical applications and services. Additionally, we scrutinize and compare the complexity associated with configuring and managing the edge clusters in the context of IoT deployment.

A. Experiment use case scenario

We assume that multi-edge sites can be used in smart farms. Edge sites are considered as different farm sites belonging to different farmers. These sites can exhibit varying characteristics, such as geographical position, network bandwidth, and the crops grown. Consequently, each edge site would require services directly tailored to its specific needs. For instance, farmers might require premium services coupled with machine learning capabilities. Thus, we have created two types of services: basic and premium services, with the premium services enhanced by a machine learning model for predicting future conditions in the field.

B. Experiment setup

In our initial implementation, we have established a multi-cluster architecture to efficiently manage IoT devices and services. All the components are running on Kind clusters on a Virtual Machine (VM) running Ubuntu 20.04 LTS OS.

The Management Cluster, which acts as the central command center for all other clusters and deployments. We assume that regional clusters are spread across different geographic regions, serve as the workhorses for application workloads. These Regional Clusters are augmented with custom IoT components to streamline application deployment and management for leaf devices on the edge clusters.

Strategically positioned at the edge sites, edge clusters are dedicated to local processing and serving IoT devices, including device mappers. To enable precise management of leaf node devices, we have introduced Custom Resource Definitions (CRDs) on the clusters to enable accessing the status of the IoT devices (sensors and actuators). An example of the custom resource introduced is shown in Listing 1.

```yaml
apiVersion: iot-nephio.iistrc.dcn.com/v1alpha1
kind: IoTDevice
metadata:
  name: iot-device-example
  namespace: default
spec:
  configuration:
    edgeSite: edge01
    deviceMapper: collection-mapper-demo:v1
    targetDevices:
      - model: DHT11
      - model: RelayModule
    telemetry:
      dataFrequency: 60s
      dataFormat: JSON
      dataStorage:
        type: Cloud
        endpoint: http://192.168.1.102/api/store
status:
  iot-devices:
    - model: DHT11
      properties:
        - name: temperature
          accessMode: ReadOnly
          type: float
          units: "°C"
        - name: humidity
          accessMode: ReadOnly
          type: float
          units: "%"
      reported:
        - name: temperature
          value: 25
          timestamp: "1699431851"
        - name: humidity
          value: 65
          timestamp: "1699431851"
    - model: RelayModule
      properties:
        - name: RM-pump-01
          accessMode: ReadOnly
          type: bool
      reported:
        - name: RM-pump-01
          value: Off
          timestamp: "1699431851"
```

Listing 1: An example of a Custom Resource on Management Cluster for connected IoT Devices on Edge site.

C. Results and Evaluation

This section presents preliminary results of our proposed framework for deploying an agricultural prototype using the pipeline approach described. We compared our work with an existing study cited in [1] and utilized their code for some of the modules available at https://github.com/rlopezv/aaas/aaa-azure. We measured the average time it takes to deploy services to the edge sites from the moment a Kpt package is approved in the management cluster. Subsequently, we compared the time required for our framework to deploy the same services or modules to the framework proposed in [1]. The results are depicted in Fig. 4. We calculate the Average Pipeline Execution Time (APET) of the frameworks by using the formula in 1

$$APET(t)_{FW} = \frac{\sum_{i=1}^{n} \mathrm{PET}_i}{n} \tag{1}$$

Where

$$\begin{aligned} APET(t)_{\mathrm{FW}} &: \text{Average Pipeline Execution in time } t \text{ for a Framework.} \\ \sum_{i=1}^{n} &: \text{Summation over } n \text{ stages in the pipeline.} \\ \mathrm{PET}_i &: \text{The execution time of each stage in the pipeline.} \\ n &: \text{The total number of pipeline stages.} \end{aligned} \tag{2}$$

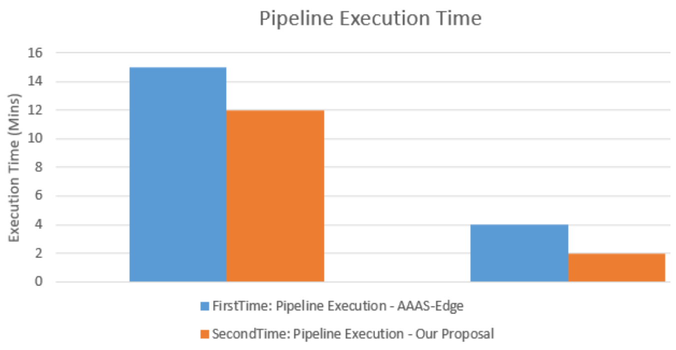

Fig. 4. Average Pipeline Execution Time

The first bars displays the average execution time required to complete the deployment of packages or modules for the first time. The second part presents the execution time for applying and replacing packages when deployments and configurations are modified. The first time takes much time due to docker pulling images from remote registry. The results indicate that our proposed framework outperforms the compared solution when deploying similar modules to the edge, with a significantly lower average execution time.

TABLE I
DEPLOYMENT WITH VARYING CONDITIONS IN MINUTES

	First	Second	Third	Fourth
Edge01	20	15	14	5
Edge02	18	11	7	3
Edge03	23	18	14	8

As shown in table I, we evaluate the time it takes to deploy packages to multiple sites all at once, including the time it takes for the services to start running and become available for usage. We modify IP tables to emulate poor connections in the edge clusters while deploying the applications. The first attempt will take longer to apply the configurations compared to subsequent attempts. Nevertheless, the platform demonstrates its ability to work effectively even in poor network conditions, deploying services to multiple edges simultaneously.

V. Conclusion and Future Directions

In this study, we have introduced our architectural framework designed to automate and manage IoT service delivery and IoT device management following the CI/CD approach.

Our future plans include extending this architecture to encompass a broader range of IoT deployment and management scenarios at every stage of the multi-edge environment. Additionally, we intend to conduct more experiments to benchmark our framework for performance and scalability. We are also exploring the integration of additional features and algorithms to ensure accurate software deployment. This may involve the implementation of machine learning techniques to detect performance issues in new software releases.

Acknowledgment

This work was partly supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grants funded by the Korea government (MSIT) (No. 2022-0-01015, Development of Candidate Element Technology for Intelligent 6G Mobile Core Network and No. 2020-0-00946, Development of Fast and Automatic Service recovery and Transition software in Hybrid Cloud Environment)

References

[1] R. López-Viana, J. Díaz, V. H. Díaz and J. -F. Martínez, "Continuous Delivery of Customized SaaS Edge Applications in Highly Distributed IoT Systems," in IEEE Internet of Things Journal, vol. 7, no. 10, pp. 10189-10199, Oct. 2020, doi: 10.1109/JIOT.2020.3009633.
[2] Khan, W., Ahmed, E., Hakak, S., Yaqoob, I. & Ahmed, A. Edge computing: A survey. *Future Generation Computer Systems*. **97** pp. 219-235 (2019)
[3] B. Reselman, "3 questions to answer when considering a multi-cluster Kubernetes architecture," RedHat, (accessed October 29, 2023).
[4] J. Dwyer, "Simplifying Multi-Cluster Kubernetes Management: A Guide to Key Solutions and Strategies," RedHat, (accessed October 29, 2023).
[5] Tran, T. A Systematic Literature Review on Secure IoT Data Sharing. (2022)
[6] Rostami Mazrae, P., Mens, T., Golzadeh, M. & Decan, A. On the usage, co-usage and migration of CI/CD tools: A qualitative analysis. *Empirical Software Engineering*. **28**, 52 (2023)
[7] Olsson, H. Challenges and strategies for undertaking continuous experimentation to embedded systems: Industry and research perspectives. *Agile Processes In Software Engineering And Extreme Programming*. **277** (2018)
[8] Ferry, N., Nguyen, P., Song, H., Novac, P., Lavirotte, S., Tigli, J. & Solberg, A. GeneSIS: Continuous Orchestration and Deployment of Smart IoT Systems. *2019 IEEE 43rd Annual Computer Software And Applications Conference (COMPSAC)*. **1** pp. 870-875 (2019)
[9] Bringhenti, D., Sisto, R. & Valenza, F. Security automation for multi-cluster orchestration in Kubernetes. *2023 IEEE 9th International Conference On Network Softwarization (NetSoft)*. pp. 480-485 (2023)
[10] López-Viana, R., Díaz, J. & Pérez, J. Continuous Deployment in IoT Edge Computing : A GitOps implementation. *2022 17th Iberian Conference On Information Systems And Technologies (CISTI)*. pp. 1-6 (2022)
[11] Marinova, S. & Leon-Garcia, A. Open-Source Network Slice Orchestration for B5G/6G: Assurance Use Case. *2023 International Balkan Conference On Communications And Networking (BalkanCom)*. pp. 1-6 (2023)
[12] "Kpt Official Docs," Kpt, https://kpt.dev/ (accessed October 30, 2023).