Demo: Managing Far-Edge Devices using Kubernetes

Carlos Resende*
João Oliveira*
carlos.resende@fraunhofer.pt
joao.oliveira@fraunhofer.pt
Fraunhofer Portugal AICOS
Porto, Portugal

Filipe Sousa
Waldir Junior
filipe.sousa@fraunhofer.pt
waldir.junior@fraunhofer.pt
Fraunhofer Portugal AICOS
Porto, Portugal

Luís Almeida
lda@fe.up.pt
CISTER / FEUP - University of Porto
Porto, Portugal

Abstract

This demo presents FITA, a platform that automates the management of resource-constrained Far-Edge devices by integrating them into Kubernetes. The demo setup is composed of an Edge gateway that runs FITA and the consumer application, as well as five commercial Far-Edge devices that host the services managed by Kubernetes. We demonstrate FITA capability to deploy and migrate services based on service requirements.

CCS Concepts

• **Computer systems organization** → *Cloud computing*; **Embedded software**; • **Software and its engineering** → **System administration**.

Keywords

Internet of Things Orchestration, Far-Edge IoT Device Management, Constrained Embedded Systems

1 Introduction

The concept of the Internet of Things (IoT) as a fully interconnected digital environment is materializing with a steady 13% to 20% annual increase in connected devices [1]. This growth enhances the role of Edge Computing in IoT ecosystems, enabling local (pre-)processing of data to enhance network efficiency, safeguard privacy, and improve response times [2].

Edge Computing can leverage TinyML and advancements in microcontroller architectureto expand computation offloading beyond local data centers and network devices, enabling AI tasks to run directly on the resource-constrained devices that interact with their environment, hereupon referred to as Far-Edge devices. Far-Edge devices are typically equipped with pre-installed software, requiring on-site maintenance and updates [3]. Integrating them with current practices for Internet applications, such as orchestration, continuous integration/continuous deployment (CI/CD), and Service-Oriented Architectureswould maximize their potential by enabling unified management across IoT applications, from the Cloud to the Far-Edge, facilitating seamless service deployment, updates, and connectivity.

Efforts to extend orchestration tools beyond the Cloud have mainly focused on the Edge, often neglecting the Far-Edge. Projects like Akri[1] and KubeEdge[2] have connected Far-Edge devices to Kubernetes (k8s), the de facto tool for service orchestration, but they do not consider the possibility of deploying services directly onto these devices despite their growing computational power. To address this, we propose the Far-edge IoT device mAnagement (FITA) platform, which supports software deployment and management on Far-Edge devices and integrates their specific details into k8s.

This demo shows how we can leverage the FITA platform to deploy and manage a simple IoT application. It highlights the initial deployment of its services to devices with specific resources and shows how we can rely on the orchestration process of k8s to handle node malfunctions.

*Both authors contributed equally to this research.
[1]https://docs.akri.sh/
[2]https://kubeedge.io/

2 System Architecture

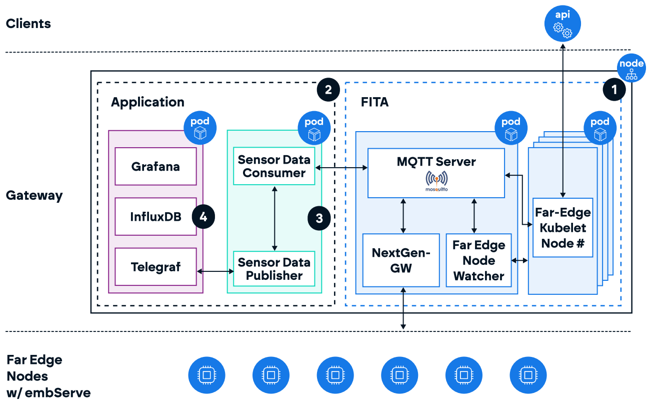

Figure 1: High-level system architecture of the FITA demo.

Figure 1 presents the architecture of the demo, which is composed of FITA (cf. 1) to manage the services on the Far-Edge and the Application (cf. 2) that consumes the data from the services that run on the Far-Edge.

FITA is composed of:

- embServe[4] framework that runs in the Far-Edge devices. It implements a service-oriented architecture for devices that do not support traditional techniques for software deployment (i.e., containers) by leveraging dynamic code loading;
- NextGenGW[5] that ensures interoperability regarding Far-Edge communication protocols by homogenising and exposing Far-Edge device properties and capabilities with the MQTT protocol and IETF SDF;
- Far-Edge Node Watcher that uses NextGenGW abstraction to know the available Far-Edge devices and keeps the k8s cluster updated by launching or removing Far-Edge Kubelet Pods as the devices connect/disconnect from NextGenGW;

- Far-Edge Kubelet built on top of the Virtual Kubelet[3] which provides a virtual representation of the Far-Edge device in the k8s cluster. It is responsible for the deployment of application components on the Far-Edge devices using the NextGenGW. One instance of this application is running per device.

The Application is composed of two Pods. The first Pod (cf. 3) contains a simple Python module that subscribes (Sensor Data Consumer) to the sensor data of the Far-Edge devices using the NextGenGW, converts it to a format compatible with the Telegraf socket listener input plugin, and publishes it (Sensor Data Publisher) to Telegraf. The second Pod (cf. 4) contains a traditional dashboard stack, where Telegraf inserts data into the InfuxDB database that Grafana reads and serves in a user-facing dashboard.

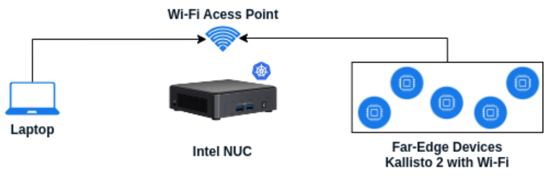

Figure 2: Overview of the FITA demo setup.

Figure 2 provides an overview of the demo setup. The gateway layer is implemented using an Intel NUC 13 Pro running the microk8s distribution (v1.29.4), where we deploy FITA and the Application. The NUC serves a Wi-Fi access point to provide connectivity to the laptop and to the Far-Edge devices. We use 5 Kallisto modules[4] with a Wi-Fi add-on as Far-Edge devices. Each module contains a different set of sensors where 3 of the modules are equipped with temperature sensors. The laptop shows the Grafana dashboard and sends the deployment commands to k8s.

3 Service Deployment

We configured the Grafana dashboard to show temperature data for two sources. By default, no service is available on the Far-Edge devices, so no data is shown on the dashboard. Therefore, we request the Deployment detailed on Listing 1 that asks the k8s scheduler to deploy two replicas of a Pod with a container named *temperature*, which contains the Far-Edge temperature service. The deployment requires a Node with a temperature sensor (*extra.resources.fhp/temperature_sensor*) and running embServe (*extra.resources.fhp/embserve*), as depicted by the Node selector (cf. lines 21 to 23). Naturally, the devices without temperature sensors are rejected, and the scheduler selects two of the three possible devices. Afterwards, the k8s scheduler sends the create Pod command to the Far-Edge Kubelet of the selected devices, which fetches the image *fhp/temperature_sensor:0.0.1* from the local OCI image registry and deploys it using NextGenGW and embServe. After deployment, the new instances of temperature sensors are reported by NextGenGW, the Sensor Data Consumer starts receiving the sensor data, and the Sensor Data Publisher forwards it to Telegraf.

[3]https://virtual-kubelet.io/
[4]https://sensry.net/products/

Listing 1: Temperature service Deployment in YAML.

```
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: temperature-deployment
5    labels:
6      app: temperature
7  spec:
8    replicas: 2
9    selector:
10     matchLabels:
11       app: temperature
12   template:
13     metadata:
14       labels:
15         app: temperature
16     spec:
17       containers:
18       - name: temperature
19         image: fhp/temperature_sensor:0.0.1
20         imagePullPolicy: IfNotPresent
21       nodeSelector:
22         extra.resources.fhp/embserve: "true"
23         extra.resources.fhp/temperature_sensor: "true"
```

4 Service Recovery

In the Deployment detailed by Listing 1, we set the number of replicas of the Pod to two. This is one of the significant advantages of k8s, which allows users to define the desired state of the cluster. The Kubernetes Deployment Controller automatically handles the necessary changes to enforce this state, so when a Deployment degrades (i.e., a Node abandons the cluster), the Controller will redeploy services as needed. Therefore, to simulate this scenario, we select one of the devices previously selected for deployment and remove its power supply, simulating a malfunction. The Far-Edge Node Watcher detects that the device is no longer connected to NextGenGW and deletes the Far-Edge Kubelet. This signals the k8s scheduler that the Node is no longer available, which triggers the deployment of a new instance of the Pod in the remaining device, ensuring minimal disruption in the Application.

Acknowledgments

This work has received funding from the Horizon Europe research and innovation programme of the European Union, under grant 101092912, project MLSysOps and by the Research Centre in Real-Time and Embedded Computing Systems – CISTER, funded by national funds through the FCT/MCTES (PIDDAC): Base Funding - UIDB/04234/2020.

References

[1] Knud Lasse Lueth. State of the iot 2020: 12 billion iot connections, surpassing non-iot for the first time. [Online] Available: https://iot-analytics.com/state-of-the-iot-2020-12-billion-iot-connections-surpassing-non-iot-for-the-first-time/, Accessed on: July, 2024.
[2] Keyan Cao, Yefan Liu, Gongjie Meng, and Qimeng Sun. An overview on edge computing research. *IEEE Access*, 8:85714–85728, 2020.
[3] Rustem Dautov and Hui Song. Towards agile management of containerised software at the edge. In *2020 IEEE Conference on Industrial Cyberphysical Systems (ICPS)*, volume 1, pages 263–268, 2020.
[4] João Oliveira, Filipe Sousa, and Luís Almeida. embserve: Embedded services for constrained devices. In *2023 IEEE 19th International Conference on Factory Communication Systems (WFCS)*, pages 1–8, 2023.
[5] Carlos Resende, Waldir Moreira, and Luís Almeida. Nextgengw: a software-based architecture targeting iot interoperability. In *2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–4, 2022.