Fog Computing out of the Box with FogDEFT Framework: A Case Study

Satish Narayana Srirama and Suvam Basak
School of Computer and Information Sciences,
University of Hyderabad, India.
satish.srirama@uohyd.ac.in

***Abstract*—Fog computing is the key technology to overcome the limitation of cloud computing in the domain of IoT applications. The service placement in the nearest fog devices drastically reduces the network delay, connectivity, and reliability issues and delivers real-time capabilities as an extension, reduces energy consumption and network overhead in the case of large sensor networks. However, the adoption rate of fog computing is not in proportion with the performance it proposes because, resource constraints, heterogeneity, and lack of standardization, require application-specific proprietary solutions. Therefore, we propose a framework that extends OASIS - Topology and Orchestration Specification for Cloud Applications (TOSCA) standard for modeling IoT applications and uses containerization technology to handle platform independence and interoperability, creating seamless coordination and cooperation across fog devices. The framework abstracts all the heterogeneity and complexities and offers a user-friendly paradigm to model and dynamically deploy fog services, on-demand, on the fly, from a remote system. The framework is demonstrated with a case study of the dynamic deployment of climate control service on the fog prototype.**

***Index Terms*—Fog computing, Internet of Things, Dynamic deployment, TOSCA, Docker.**

I. Introduction

The beginning of the era of IoT applications was cloud-centric. All the diverse data collected through the sensors from the different fields are stored in the cloud. The cloud has virtually infinite computational power for processing that data, generating an actuation signal to send back to the actuators. This architecture has some drawbacks. First, the cloud is physically placed thousands of kilometers away from the sensors and actuators in most cases, which adds a significant network delay between sensing and actuation. That is undesirable for some real-time applications. Second, this architecture needs stable connectivity to the cloud for continuously streaming the data. However, the IoT devices may have to handle unstable connectivity in the real world, leading to the reliability issues of the IoT applications. Third, with the exponential increase of IoT devices data generation rate has also increased rapidly. It is becoming an overhead for the network to move vast volumes of data to the cloud. Therefore, the concept of Edge and Fog computing comes into the picture to address these problems [1]. In fog computing, the utilization of gateway devices acts as an alternative or supportive computational equipment of the cloud. Therefore, service placement closer to the IoT devices reduces delays, network overhead, and reliability issues [2][3].

With the adoption of Fog computing, deployment of the services of an IoT application on fog nodes opens a nontrivial area of research. Several automation tools have been developed with the rise of Infrastructure as Code (IaC) to manage, configure, and provision servers and data centers. None of them primarily targets the deployment of fog services. Unlike conventional computing systems, fog devices are heterogeneous and have a completely different hardware architecture that may run on different software and be optimized to perform specific tasks. However, the main requirement of a fog federation is to have seamless cooperation and interoperability across all the fog devices inside the network. Therefore, the dynamic deployment of a service on fog devices imposes a challenge of handling platform independence, interoperability, and portability with resource constraints.

This problem is similar to the vendor lock-in and portability of a composite cloud application across different cloud service providers [4]. The issue of the cloud community has been addressed by OASIS - Topology and Orchestration Specification for Cloud Applications (TOSCA) [5]. In a nutshell, TOSCA is a modeling language. Application developers can describe the platform-agnostic blueprint of a composite cloud application in a YAML[1] file. That makes the application portable across the different cloud service providers with minimal effort [6]. The fog federation framework: FogDEFT Framework implements similar ideas for fog computing[2]. The FogDEFT Framework extends the TOSCA standards for portable fog services and hides the heterogeneity of the fog hardware. The framework provides user-friendly development paradigms and enables on-the-fly service deployment capabilities.

This paper demonstrates our earlier developed TOSCA standard-based FogDEFT Framework's potential in realizing the dynamic deployment of on-demand fog services. We have taken a use case scenario of dynamic deployment of fog services and demonstrated the application's system design and modeling, followed by deployment of fog services on the fly.

The main **contribution** of this paper is:

1) Design and modeling of a domain-specific IoT application with TOSCA.
2) Deployment of the fog service on-demand, on the fly, on resource-constrained fog devices.

[1] https://yaml.org
[2] https://github.com/cloud-and-smart-labs/fog-service-orchestration

The rest of the paper is organized as follows. Section II gives an overview of all related work in the domain. Section III gives an overview of the FogDEFT Framework. Section IV shows the Case study on climate control application with the FogDEFT Framework. Section V discusses the resource utilization of the orchestration process on the system running orchestrator and fog devices. Finally, section VI concludes with future potential applications of TOSCA in IoT.

II. Related work

Fog computing resolves many problems of cloud-centric IoT applications. However, fog computing comes with its challenges that significantly slow down the adoption rate of fog computing. On-demand deployment of the service on the fog node is one of these challenges. Therefore, application/service deployment for fog computing has been studied extensively.

A. Dynamic deployment of applications

In [7], N Ferry et al. carried out a case study of GeneSIS in smart buildings. They showed that GeneSIS could support security by design from the development (via deployment) to the operation of IoT systems and keep up security and adapt to evolving conditions and threats while maintaining their trustworthiness.

In [8], HA Hassan and RP Qasha propose a new approach to generate a deployable model for the distributed IoT systems based on a simplified, user-friendly declarative description of the smart devices' communication, configuration, installation, and computation with the IoT system parts with Ansible-based YAML description. The work minimizes the efforts for the deployment of the distributed IoT applications on various infrastructures, including the cloud.

In [9], O Tomarchio et al. proposed a TOSCA-based framework, 'TORCH,' for deploying and orchestrating classical and containerized cloud applications on multiple cloud providers. The main benefit of the framework is the possibility to add support to any cloud platform at a very low implementation cost, and it allows deployment management through a simple web tool.

In [10], R Dautov et al. propose a hierarchical architecture for provisioning software updates from the cloud to terminal devices via edge gateways in a targeted manner through a last-mile deployment agent, placed on edge gateways via the centralized cloud in the form of containerized microservices, that receive firmware updates from the cloud and install them on connected IoT devices at the edge.

In [11], H Song et al. describe joint research on an industrial use case with a Smart Healthcare application provider on a model-based approach for automatically assigning multiple software deployments to hundreds of Edge gateways (fleet) and uses a set of hard and soft constraints to achieve correct, even distribution of software variants.

B. Deployment of fog applications

B Donassolo et al. in [12] proposed another orchestration framework called 'FITOR,' an automated deployment and microservice migration solution for IoT applications. The framework uses Optimized Fog Service Provisioning (O-FSP) based on a greedy approach that outperforms other relevant strategies in terms of i) acceptance rate, ii) provisioning cost, and iii) CPU usage.

In [13], N Ferry et al. developed a framework for continuous deployment for decentralized processing across heterogeneous IoT, edge, and cloud infrastructures called Generation and Deployment of Smart IoT Systems (GeneSIS). GeneSIS provides (i) a domain-specific modeling language to model the orchestration and deployment of Smart IoT Systems and (ii) an execution engine to support automatic deployment across IoT, edge, and cloud infrastructure resources.

In [14], S Venticinque and A Amato proposed a new fog service placement methodology. The methodology's effectiveness is demonstrated in the energy domain with smart grid.

In [15], G Davoli et al. developed a modular orchestration system called 'FORCH.' The orchestrator is aware of different service models (SaaS/PaaS/IaaS) and dynamically deploys services and manages resources on the fog nodes.

In [16], H Sami and A Mourad proposed a new framework for deploying fog service on-demand on the fly based on Kubeadm and Docker with the presence of volunteering devices. Moreover, the framework optimizes the container placement problem with an Evolutionary Memetic Algorithm (MA) that uses heuristics to make decisions.

In [17], H Sami et al. proposed an efficient resource and context-aware approach for deploying containerized microservices on-demand called Vehicular-OBUs-As-On-Demand-Fogs. The scheme embeds adaptable networking architecture combining cellular technologies and the vehicular ad-hoc wireless network (802.11p) and a Kubeadm-based approach for clustering with docker container-based microservices deployment. The solution provides an on-demand fog and service placement solution on vehicles based on an Evolutionary Memetic Algorithm.

C. Dynamic deployment of Fog applications

In [18], S Hoque et al. have carried out a technical evaluation of docker container and container orchestration tools, their capability, limitations, and how containerization can impact application performance. The result shows that significant adjustments are required to meet the fog environment needs, and they have proposed a framework based on the docker swarm to address issues with the help of 'OpenIoTFog' toolkits.

In 2013 F Li et al. put the first effort to use TOSCA, the new cloud standard, for IoT applications and demonstrated the feasibility of modeling IoT components gateways and drivers for building Air Handling Unit (AHU) with the first edition of TOSCA [19].

ACF da Silva et al. in [20] automatically deployed an IoT application with OpenTOSCA based on Mosquitto Message Broker running on the cloud, and the publishers and subscribers were running in two different raspberry pis. Later, in [21], they automatically deployed an IoT application out of the box where a python script on raspberry pi pushes the

data to the message broker on a cloud, and another virtual machine is hosting a web-based dashboard to present the sensor data. They validated this deployment with three case studies of emerging middleware (i) Eclipse Mosquitto, (ii) FIWARE Orion Context Broker, and (iii) OpenMTC.

In [22], A Tsagkaropoulos et al. presented TOSCA extensions for modeling applications relying on any combination of technologies and discussed semantic enhancements, optimization aspects, and methodology that should be followed for edge and fog deployment support. Furthermore, added a comparison with other cloud application deployment approaches.

In [23], HE Solayman and RP Qasha, used TOSCA for deploying IoT applications for Intensive Care Unit (ICU) based on Docker Containers. The demonstration shows automation of IoT application provisioning in heterogeneous environments consisting of hardware components and cloud instance message broker for network communication between components containerized with docker containers.

Table I summarizes the work already done in the domain of service deployment of fog computing.

However, in this paper, the proposed framework FogDEFT extends the de-facto standard for portable cloud applications for fog applications and widely used most popular container orchestration technologies Docker and Docker swarm for the deployment of fog services on demand. Moreover, the framework minimizes cloud involvement and uses on-premises infrastructure primarily, as J Delsing et al. argued in [24] about open internet automation limitations and discussed the idea of local cloud in IoT automation. Since this automation is physically and geographically local, hence local cloud meets system requirements of (1) interoperability of a wide range of IoT and legacy devices, (2) real-time capabilities as latency guarantee required for automation system, (3) scalability of enormous scale, (4) security fence from the external network of automation system and (5) ease of application engineering with integrity, and agility. The concept is verified in climate control applications.

In recent years, significant work has been published regarding smart IoT-based solutions for greenhouse production and farming [25][26][27][28]. And some work in Cold Storage as well [29]. The proposed framework FogDEFT can realize such use cases to achieve the highest level of convenience and can change the climate of greenhouse or cold storage for specific crops and products, respectively.

III. FogDEFT Framework

The **FogDEFT** (**Fog** computing out of the box: **D**ynamic d**E**ployment of **F**og service containers with **T**OSCA) Framework is a fog federation framework built on the extension of the TOSCA standard, which is the de-facto standard for modeling cloud applications. The framework enables seamless cooperation and coordination between fog devices in the network, hides the heterogeneity, offers a development paradigm for the custom or user-developed fog application, and realizes dynamic deployment of the fog services on the fly on demand. The framework maintains three layers of abstraction as follows:

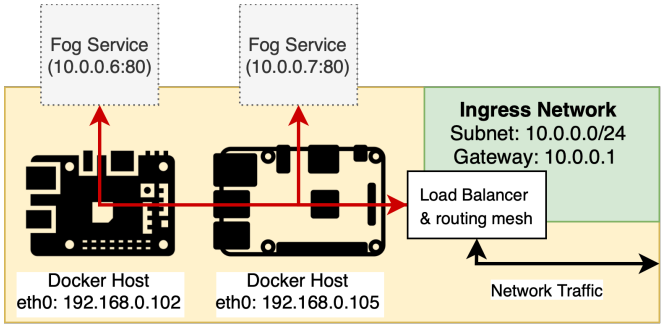


Fig. 1. Interoperability with Docker swarm

A. Platform independence

All fog devices are mostly network equipment and gateway devices (routers, network switches, drones) or conventional computational devices (sometimes). These devices consist of different hardware architectures and operating systems. Therefore, the first layer of the abstraction of the fog federation framework is to handle platform independence through virtualization. However, hardware virtualization is costly and resource-intensive. Therefore, it is not a feasible solution for resource-constrained fog devices. However, all these fog devices are network devices, and all network devices run on some form of Linux system. Therefore, these fog devices are running Linux kernels, making containerization or OS-level virtualization a feasible solution since containers take kernel support from the host machine and run in isolation without interfering with the host system or other containers. Therefore, multi-architecture builds[3] of docker images should be available on the Docker registry. The docker image of a specific processor architecture will be pulled on-demand during fog service orchestration.

B. Interoperability

The second essential requirement of fog computing is interoperability which ensures seamless cooperation and coordination between services running across fog nodes. Docker provides the native container orchestration tool called Docker Swarm[4]. Since the fog services are running in docker containers, anything that runs well in standalone containers runs equally well in swarm mode.

Docker Swarm creates clusters of fog devices with an Overlay Network[5] called Ingress Network[6] (`10.0.0.1/24`), illustrated in Figure 1. This Ingress Network has an inbuilt load balancer and routing mesh. The load balancer distributes the traffic across multiple containers when more than one replica (`10.0.0.6:80` and `10.0.0.7:80`) of a service is running.

[3]CLI tool Buildx by docker to build docker image of different CPU architecture using the QEMU emulation support from Linux Kernel.

[4]https://docs.docker.com/engine/swarm/

[5]https://docs.docker.com/network/overlay/

[6]https://docs.docker.com/engine/swarm/ingress/

TABLE I
A SUMMARY OF RELATED WORK AND THEIR FOCUS

Work	Architecture Framework	Performance	Virtualization	Agent	Deployment/Placement /Cost Algorithm	Service Placement	TOSCA
N Ferry et al.[7]	✓		✓	✓			
HA Hassan and RP Qasha [8]	✓	✓					
O Tomarchio et al. [9]	✓	✓	✓				✓
R Dautov et al. [10]			✓	✓			
H Song et al. [11]	✓	✓	✓	✓			
B Donassolo et al. [12]	✓	✓			✓	✓	
N Ferry et al. [13]	✓		✓	✓			
S Venticinque and A Amato [14]						✓	
G Davoli et al. [15]	✓						
H Sami and A Mourad. [16]			✓		✓	✓	
H Sami et al. [17]	✓		✓		✓	✓	
S Hoque et al. [18]	✓	✓	✓				
F Li et al.[19]							✓
ACF da Silva et al. [20]							✓
ACF da Silva et al. [21]							✓
A Tsagkaropoulos et al. [22]							✓
HE Solayman and RP Qasha [23]			✓				✓
FogDEFT	✓	✓	✓				✓

The routing mesh redirects the traffic from any docker host in the swarm to a specific docker host where the service container is running. Therefore, any fog service running in only one replica (either `10.0.0.6:80` or `10.0.0.7:80`) also becomes available through all the IP addresses of fog devices in the swarm through the same port number because of the routing mesh. Therefore, any fog service running in swarm mode or standalone mode can communicate with other services running in swarm mode without knowing the container's placement in the fog federation.

Therefore, this second layer of abstraction handles interoperability with the Docker Swarm and enables seamless coordination and cooperation in the fog federation. However, not all services can run in swarm mode. Any service that needs to deal with a specific hardware component (sensors/actuators) has to be placed into that gateway device connected to that component. Any service that handles processing or communication (message broker or server) between sensors and actuators should be placed in swarm mode.

C. Standardization

To enable the portability of a fog application from one fog federation to another requires some standardization. It is a similar type of problem faced by the cloud communities while porting a composite cloud application from one cloud service provider to another one. OASIS addressed the problem with Topology and Orchestration Specification for Cloud Applications (TOSCA), which standardizes a composite cloud application description. This description is the conceptual structure of an application with two basic building blocks: nodes and relationships. These nodes are the infrastructure and software components (server, virtual machine, runtime environments), and relationships define relationships between nodes (hosted on, depends on, connected to). Nodes consist of attributes, properties, capabilities, and requirements. Node's requirements and capabilities are counterparts to each other. If a node has a requirement for something, there should be another node with that capability to fulfill the requirement. Each node and relationship has node types and relationship types where attributes, properties, capabilities, and requirements are declared. TOSCA comes with normative nodes and relationship types. Any custom nodes and relationship types can be created to construct a TOSCA Service Template for a composite application by extending these normative nodes and relationship types. TOSCA Service Template contains a topology template. Inside the topology template conceptual topology of a composite application is designed with a set of node and relationship templates. This TOSCA Service Template is a standard description of an application in a YAML file that makes it portable across different platforms.

Interestingly, TOSCA is platform agnostic and provides language extension mechanisms. Therefore, TOSCA has been extended to standardize (i) serverless computing or Function as a Services (FaaS) in EU H2020 RADON project[7] and (ii) TOSCAData for cloud data pipeline [30]. Similarly, this FogDEFT Framework enables the portability of fog applications by extending TOSCA to describe the blueprint of fog application as a third layer of abstraction.

Previous extensions of the TOSCA in the RADON project and TOSCAData created new node types for the Service Template that describes FaaS and data pipeline applications. These node types are open source and available in online reposito-

[7]https://radon-h2020.eu

TABLE II
LIST OF NODE TYPES FOR FOG APPLICATIONS

Nodes Types	Description
`docker_containers`	Pull a `docker-compose.yaml` file from the given URL and deploy/undeploy on the host fog node
`docker_services`	Pull a `docker-compose.yaml` file from the given URL and deploy/undeploy on the Docker Swarm from Docker Leader node
`swarm_leader`	Initiates Docker Swarm on the host node and the node becomes Swarm Manager
`swarm_worker`	Host node joins the Docker Swarm as Worker node

TABLE III
LIST OF RELATIONSHIP TYPES FOR FOG APPLICATIONS

Relationship Types	Description
`token_transfer`	Relationship (dependency) between Swarm Leader and Swarm Worker

ries[8]. However, the extension of TOSCA for describing fog applications necessitates significant modifications to available node types. Table II and III gives details of newly created node[9] and relationship[10] types in FogDEFT Framework for creating TOSCA Service Templates of a fog application.

D. Dynamic deployment

The idea of the dynamic deployment of a fog application is to be able to deploy/undeploy services on these fog nodes on-demand with the help of a single command. The FogDEFT Framework will cover all the complexity and heterogeneity of fog devices without any human intervention. That necessitates the adoption of an orchestration tool.

The deployment of the services requires the adoption of a TOSCA complaint orchestrator. It is important to note that TOSCA is just a standard. It gives a string then it is up to the orchestrator to make sense of it. A TOSCA-compliant orchestrator consists of a TOSCA processor that can parse and interpret TOSCA templates and instantiate the service. All TOSCA node and relationship types include interface operations. Each operation is associated with implementation scripts (the type of the scripts depends on the orchestrators). These scripts get executed to instantiate the services during deployment.

The FogDEFT Framework adopted the xOpera orchestrator to deploy the services on the fog nodes. xOpera[11] is an open-source lightweight orchestrator currently compliant with the TOSCA Simple Profile in YAML Version 1.3[12]. It uses

[8]https://github.com/radon-h2020

[9]https://github.com/cloud-and-smart-labs/climate-control/tree/main/tosca/nodetypes

[10]https://github.com/cloud-and-smart-labs/climate-control/tree/main/tosca/relationshiptypes/token_transfer

[11]https://xlab-si.github.io/xopera-docs/02-cli.html

[12]https://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.3/TOSCA-Simple-Profile-YAML-v1.3.html

```yaml
interfaces:
  Standard:
    type: tosca.interfaces.node.lifecycle.Standard

    inputs:
      name:
        value: { get_property: [SELF, name] }
        type: string
      url:
        value: { get_property: [SELF, url] }
        type: string

    operations:
      create: playbooks/create.yaml
      delete: playbooks/delete.yaml
```

Listing 1: TOSCA node's interface operations

ansible[13] automation tools for the implementation of TOSCA standards. Therefore, all the nodes and relationship types listed in Tables II and III have associated ansible-playbook scripts (`create.yaml`, `delete.yaml`) for interface operations, as shown in Listing 1.

At the time of deployment, the xOpera orchestrator executes ansible-playbook scripts in a specific order to make this deployment happen. This specific order is one of the topological sorts of the application topology defined in the topology template inside TOSCA service templates. For the deployment, the order follows the transpose graph of the application topology.

IV. CASE STUDY

This paper demonstrates the dynamic deployment, by realizing a case study of the climate control system of the convention center. A convention center inside a city usually hosts diverse events like conferences, exhibitions, and cultural events. Probably a storage area for off times or a hospital isolation ward in times of pandemic, and the past two years made it clear. Therefore, all these events in different seasons require different climate conditions inside a convention center. For example, a cultural event needs different lighting requirements and intensity than an international conference. Even those requirements will differ from daytime to nighttime as well. The weather and season will play a significant role in climate control. A summer event requires lower temperature, a winter event higher temperature, and a monsoon needs lower humidity and temperature. The number of guests is also a factor in controlling temperature and humidity. Altogether, the automation of these climate control systems is one of the ideal scenarios for the dynamic deployment of fog services.

For the development of this system, the main challenges are (i) Designing a generic system (mostly hardware platform) equipped with sensors and actuators that can host some fog services. (ii) Develop a set of fog services based on the FogDEFT framework that will dynamically deploy on the fly based on what event has been organized. These services will be running on on-premises infrastructure with minimal

[13]https://www.ansible.com/

TABLE IV
DEVICE LIST

Device Name	Processor Architecture	Memory Size	Operating System
Arduino Uno	Microcontroller ATmega328P	32 KB	-
Arduino Nano 33 BLE Sense	ARM® Cortex®-M4	1 MB	-
Raspberry Pi 4	ARMv7l 32-Bit	4 GB	Raspberry Pi OS 10
Raspberry Pi 4	ARMv8 64-Bit	4GB	Raspberry Pi OS 11
Workstation	Intel Xeon 16 Core	32 GB	Ubuntu 20.04

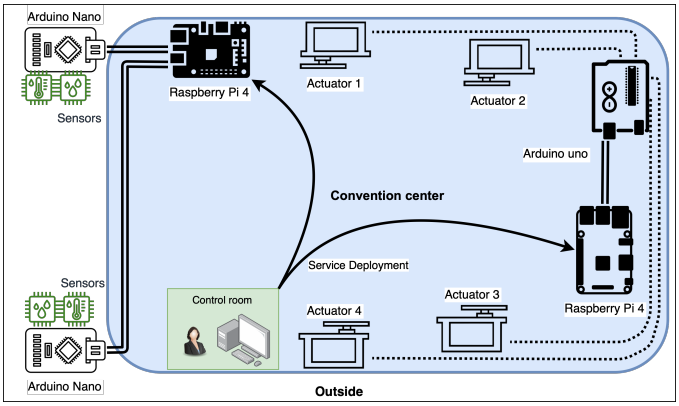


Fig. 2. Hardware design

internet usage and cloud involvement. One centralized system can control the deployment/undeployment of the services.

A. System design

The design of an IoT-driven system consists of two parts: hardware and software. The section described the requirement of a generic hardware platform with sensors and actuators capable of hosting some fog services. The following subsections illustrate the hardware prototype followed by the software architecture with FogDEFT Framework on the hardware prototype, creating a fog federation for deploying the case study services.

1) Hardware design: The prototype for the demonstration of dynamic deployment of fog services uses a handful of IoT devices listed in Table IV.

Figure 2 shows the illustration of the hardware design of the prototype. Two Arduino Nano 33 BLE Sense boards are placed outside the convention center and connected to a Raspberry Pi 4 through serial ports. These two Arduino boards have inbuilt sensors (temperature, humidity, light, barometric pressure, proximity, and microphone) to sense the outside environment. Another Arduino Uno is connected to four actuators (servo motors) through Pulse Width Modulation (PWM) pins and one Raspberry Pi 4 through a serial port. All the Raspberry Pis and the Workstation inside the control room are connected to the network and have internet connectivity.

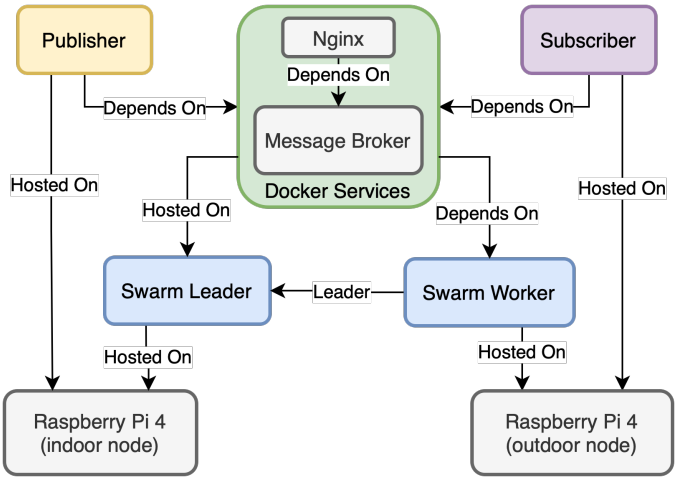


Fig. 3. Service design blueprint

2) Software design: After designing the generic hardware platform, it is up to the job of the software to create the platform for fog federation with these on-premises gateway devices (Raspberry Pis). The devices listed in Table IV come under two different categories.

First, Arduinos come under the category of microcontrollers. These are programmable chips. Whenever these devices are connected to power, they execute the same program, whatever is loaded into it. We have three Arduinos here of two different categories: Nano 33 BLE Sense and Uno. The first one is for sensing the outside environments. Therefore, these two Arduino were programmed to collect the sensor dataset and send it through the serial port of that connected Raspberry Pi 4. The second one is for actuation to control the climate inside the convention center. Therefore, this is programmed to perform actuation on connected actuators. These actuation parameters are retrieved from the serial port connected to the Raspberry Pi 4.

Second, Raspberry Pis and the system inside the control room come under the microprocessor category. Unlike Arduinos, these are typical computers with operating systems that can load programs and execute processes and services. Therefore, these devices will be treated as fog nodes. Here, the FogDEFT Framework comes into the picture to ease the development and deployment of fog services on the fog nodes.

The design of the prototype in Figure 2 consists of two fog nodes (Raspberry Pi 4). The fog services to maintain the climate will be deployed on these two fog nodes. Here, Figure 3 illustrates the topology design of fog services to be deployed on the fog nodes inside convention centers to IoT-driven climate control systems. In this illustration, boxes are nodes, and directed edges are relationships between nodes, as discussed in section III-C.

The bottom two gray nodes represent the fog nodes. The green node in the middle represents Docker Services (Message broker and web server to show the sensor data and state) running in the swarm mode (interservice dependencies are

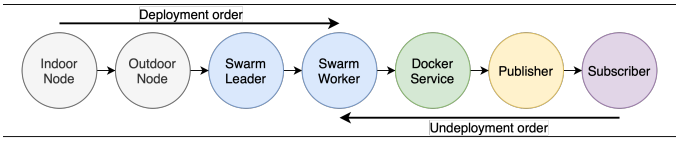


Fig. 4. One of the valid orders for deployment/undeployment

mentioned in the docker-compose file). This Docker Service node is hosted and depends on two blue nodes, Swarm Leader and Swarm Worker, respectively. The edge between two blue nodes is their relationship, and both are hosted on one fog node, respectively. The remaining two yellow and purple nodes are Publisher and Subscriber services, respectively. The Publisher node pushes the sensor data to the message broker. Therefore this node is a standalone container hosted on the fog node connected to the Arduino outside. Similarly, the Subscriber node receives the broadcast of each update from the message broker and makes adjustments to actuators. Therefore, this node is also a standalone container hosted on the fog node connected to the Arduino wired with actuators.

This service blueprint of the design indicates that at least four different microservices (Message broker, web viewer, sensor data publisher, and Subscriber with climate controller) are required. Interestingly, this design illustrated in Figure 3 requires only one URI change inside the purple node named Subscriber. That could dynamically change to utterly different climate conditions, probably requiring one from a specific event. Hence, this fog federation framework provides a versatile platform to deploy services on demand on the fly.

With these node and relationship types provided by the FogDEFT Framework the TOSCA Service Template[14] of the blueprint given in Figure 3 is shown in Listing 2.

V. Results and Discussion

We deployed the Service Template given in Listing 2 on the prototype Figure 2 from a remote system (corresponds to the control room) with the xOpera orchestrator. As discussed in section III-D, the deployment and undeployment order of the nodes follows topological sort of application topology shown in Figure 3. Therefore, one of the valid deployment and undeployment orders is given in Figure 4.

The resource utilization of each node (Indoor, Outdoor, and Workstation) is given in Table V. This resource usage only includes the resource consumption of these processes responsible for the orchestrations of the fog services. Resource consumption of fog services is out of the framework's scope. In our experiment, the deployment and undeployment took around 121.05s and 96.90s, respectively, from the workstation with a single thread. However, the xOpera orchestrator offers a multithreaded approach to deploy node templates parallelly based on the dependency defined on the topology template. However, the first-time service deployment may take longer for other factors like available bandwidth (puling, extraction of images).

[14] https://github.com/cloud-and-smart-labs/climate-control/blob/main/tosca/service.yaml

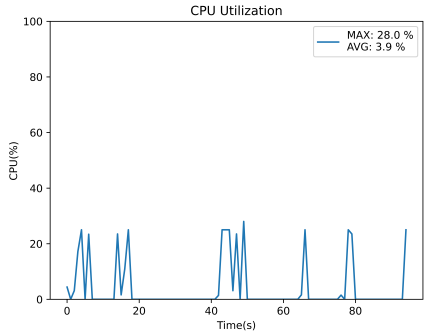
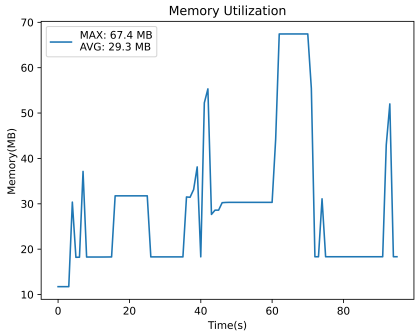
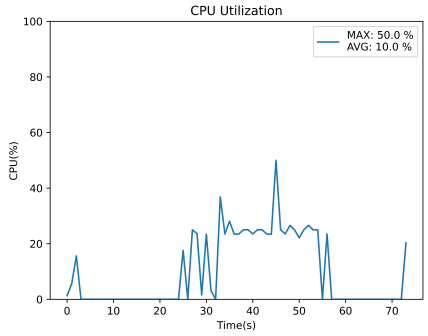
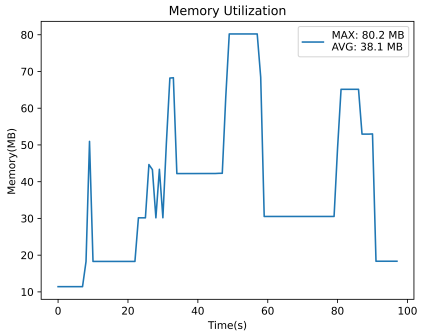
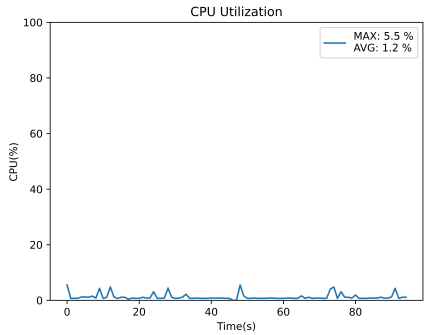
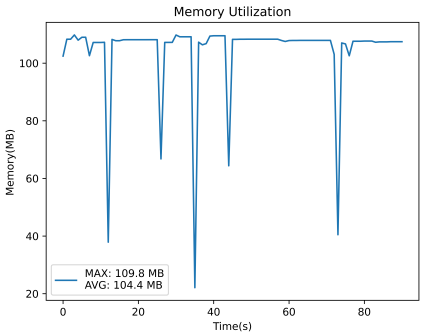
```yaml
topology_template:
  node_templates:
    outdoor-node:
      type: tosca.nodes.Compute
      attributes:
        private_address: 192.168.0.103
        public_address: 192.168.0.103

    indoor-node:
      type: tosca.nodes.Compute
      attributes:
        private_address: 192.168.0.105
        public_address: 192.168.0.105

    docker-swarm-leader:
      type: fog.docker.SwarmLeader
      requirements:
        - host: indoor-node

    docker-swarm-worker:
      type: fog.docker.SwarmWorker
      requirements:
        - host: outdoor-node
        - leader: docker-swarm-leader

    broker-service:
      type: fog.docker.Services
      properties:
        name: broker
        url: https://repo/brokr/docker-compose.yaml
      requirements:
        - host: docker-swarm-leader
        - dependency: docker-swarm-worker

    sensor-data-publisher:
      type: fog.docker.Containers
      properties:
        name: publisher
        url: https://repo/pub/docker-compose.yaml
      requirements:
        - host: outdoor-node
        - dependency: broker-service

    actuator-data-subscriber:
      type: fog.docker.Containers
      properties:
        name: subscriber
        url: https://repo/subs/docker-compose.yaml
      requirements:
        - host: indoor-node
        - dependency: broker-service
```

Listing 2: TOSCA Service Template

VI. Conclusion and Future Work

This paper demonstrated the dynamic deployment of fog service on-demand with a fog federation framework: FogDEFT. The framework abstracts the heterogeneity of fog devices and provides a standardized platform for deploying custom or user-developed applications on the fly. The xOpera orchestrator and ansible automation tool uses Secure Shell (SSH) infrastructure to push ansible modules for the deployment of the service. Therefore, this framework is secure, agentless, and works with out-of-the-box fog devices.

TABLE V
RESOURCE UTILIZATION AND PERFORMANCE

Device	CPU (%)	Memory (MB)
Raspberry Pi 4 Indoor Node		
Raspberry Pi 4 Outdoor Node		
Workstation Orchestrator		

The development of TOSCA primarily targets the standardization of cloud applications. However, this work demonstrates the potential of TOSCA in standardizing fog services for IoT-driven applications. The orchestrator used in this work was developed for the cloud application deployment in virtual machines. In IoT, non-IP-based networking is prevalent for M2M communication. Therefore, creating a lightweight IoT-focused orchestrator with non-IP-based networks support can unlock huge possibilities like M2M or drone to drone on-demand service deployment.

ACKNOWLEDGMENT

This research is supported by SERB, India, through grant CRG/2021/003888. We also thank financial support to UoH-IoE by MHRD, India (F11/9/2019-U3(A)).

REFERENCES

[1] L. Bittencourt, R. Immich, R. Sakellariou, N. Fonseca, E. Madeira, M. Curado, L. Villas, L. DaSilva, C. Lee, and O. Rana, "The internet of things, fog and cloud continuum: Integration and challenges," *Internet of Things*, vol. 3-4, pp. 134–155, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2542660518300635

[2] R. Buyya and S. N. Srirama, *Fog and edge computing: principles and paradigms*. John Wiley & Sons, 2019.

[3] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, ser. MCC '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 13–16. [Online]. Available: https://doi.org/10.1145/2342509.2342513

[4] J. Opara-Martins, R. Sahandi, and F. Tian, "Critical analysis of vendor lock-in and its impact on cloud computing migration: a business perspective," *Journal of Cloud Computing*, vol. 5, no. 1, pp. 1–18, 2016.

[5] T. Binz, G. Breiter, F. Leyman, and T. Spatzier, "Portable cloud services using tosca," *IEEE Internet Computing*, vol. 16, no. 3, pp. 80–85, 2012.

[6] T. Binz, U. Breitenbücher, O. Kopp, and F. Leymann, "Tosca: portable automated deployment and management of cloud applications," in *Advanced Web Services*. Springer, 2014, pp. 527–549.

[7] N. Ferry, P. H. Nguyen, H. Song, E. Rios, E. Iturbe, S. Martinez, A. Rego *et al.*, "Continuous deployment of trustworthy smart iot systems." *The Journal of Object Technology*, 2020.

[8] H. A. Hassan and R. P. Qasha, "Toward the generation of deployable distributed IoT system on the cloud," *IOP Conference Series: Materials Science and Engineering*, vol. 1088, no. 1, p. 012078, feb 2021. [Online]. Available: https://doi.org/10.1088/1757-899x/1088/1/012078

[9] O. Tomarchio, D. Calcaterra, G. Di Modica, and P. Mazzaglia, "Torch: a tosca-based orchestrator of multi-cloud containerised applications," *Journal of Grid Computing*, vol. 19, no. 1, pp. 1–25, 2021.

[10] R. Dautov, H. Song, and N. Ferry, "Towards a sustainable iot with last-mile software deployment," in *2021 IEEE Symposium on Computers and Communications (ISCC)*, 2021, pp. 1–6.

[11] H. Song, R. Dautov, N. Ferry, A. Solberg, and F. Fleurey, "Model-based fleet deployment in the iot–edge–cloud continuum," *Software and Systems Modeling*, pp. 1–26, 2022.

[12] B. Donassolo, I. Fajjari, A. Legrand, and P. Mertikopoulos, "Fog based framework for iot service provisioning," in *2019 16th IEEE Annual Consumer Communications Networking Conference (CCNC)*, 2019, pp. 1–6.

[13] N. Ferry, P. Nguyen, H. Song, P.-E. Novac, S. Lavirotte, J.-Y. Tigli, and A. Solberg, "Genesis: Continuous orchestration and deployment of smart iot systems," in *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, vol. 1, 2019, pp. 870–875.

[14] S. Venticinque and A. Amato, "A methodology for deployment of iot application in fog," *Journal of Ambient Intelligence and Humanized Computing*, vol. 10, no. 5, pp. 1955–1976, 2019.

[15] G. Davoli, D. Borsatti, D. Tarchi, and W. Cerroni, "Forch: An orchestrator for fog computing service deployment," in *2020 IFIP Networking Conference (Networking)*, 2020, pp. 677–678.

[16] H. Sami and A. Mourad, "Dynamic on-demand fog formation offering on-the-fly iot service deployment," *IEEE Transactions on Network and Service Management*, vol. 17, no. 2, pp. 1026–1039, 2020.

[17] H. Sami, A. Mourad, and W. El-Hajj, "Vehicular-obus-as-on-demand-fogs: Resource and context aware deployment of containerized micro-services," *IEEE/ACM Transactions on Networking*, vol. 28, no. 2, pp. 778–790, 2020.

[18] S. Hoque, M. S. De Brito, A. Willner, O. Keil, and T. Magedanz, "Towards container orchestration in fog computing infrastructures," in *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, vol. 2, 2017, pp. 294–299.

[19] F. Li, M. Vögler, M. Claeßens, and S. Dustdar, "Towards automated iot application deployment by a cloud-based approach," in *2013 IEEE 6th International Conference on Service-Oriented Computing and Applications*, 2013, pp. 61–68.

[20] A. C. F. da Silva, U. Breitenbücher, K. Képes, O. Kopp, and F. Leymann, "Opentosca for iot: Automating the deployment of iot applications based on the mosquitto message broker," in *Proceedings of the 6th International Conference on the Internet of Things*, ser. IoT'16. New York, NY, USA: Association for Computing Machinery, 2016, p. 181–182. [Online]. Available: https://doi.org/10.1145/2991561.2998464

[21] A. C. F. da Silva, U. Breitenbücher, P. Hirmer, K. Képes, O. Kopp, F. Leymann, B. Mitschang, and R. Steinke, "Internet of things out of the box: Using tosca for automating the deployment of iot environments." in *CLOSER*, 2017, pp. 330–339.

[22] A. Tsagkaropoulos, Y. Verginadis, M. Compastié, D. Apostolou, and G. Mentzas, "Extending tosca for edge and fog deployment support," *Electronics*, vol. 10, no. 6, 2021. [Online]. Available: https://www.mdpi.com/2079-9292/10/6/737

[23] H. E. Solayman and R. P. Qasha, "Portable modeling for icu iot-based application using tosca on the edge and cloud," in *2022 International Conference on Computer Science and Software Engineering (CSASE)*, 2022, pp. 301–305.

[24] J. Delsing, J. Eliasson, J. van Deventer, H. Derhamy, and P. Varga, "Enabling iot automation using local clouds," in *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, 2016, pp. 502–507.

[25] R. Rayhana, G. Xiao, and Z. Liu, "Internet of things empowered smart greenhouse farming," *IEEE Journal of Radio Frequency Identification*, vol. 4, no. 3, pp. 195–211, 2020.

[26] A. F. Subahi and K. E. Bouazza, "An intelligent iot-based system design for controlling and monitoring greenhouse temperature," *IEEE Access*, vol. 8, pp. 125 488–125 500, 2020.

[27] M. A. M. Ariffin, M. I. Ramli, M. N. M. Amin, M. Ismail, Z. Zainol, N. D. Ahmad, and N. Jamil, "Automatic climate control for mushroom cultivation using iot approach," in *2020 IEEE 10th International Conference on System Engineering and Technology (ICSET)*, 2020, pp. 123–128.

[28] M. Muñoz, J. L. Guzmán, J. A. Sánchez-Molina, F. Rodríguez, M. Torres, and M. Berenguel, "A new iot-based platform for greenhouse crop production," *IEEE Internet of Things Journal*, vol. 9, no. 9, pp. 6325–6334, 2022.

[29] H. Afreen and I. S. Bajwa, "An iot-based real-time intelligent monitoring and notification system of cold storage," *IEEE Access*, vol. 9, pp. 38 236–38 253, 2021.

[30] C. K. Dehury, P. Jakovits, S. N. Srirama, G. Giotis, and G. Garg, "Toscadata: Modeling data pipeline applications in tosca," *Journal of Systems and Software*, vol. 186, p. 111164, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121221002508