Live Demonstration of Service Function Chaining allocation in Fog Computing

José Santos*, Tim Wauters*, Bruno Volckaert* and Filip De Turck*
* Ghent University - imec, IDLab, Department of Information Technology
Technologiepark-Zwijnaarde 126, 9052 Gent, Belgium
Email: josepedro.pereiradossantos@UGent.be

Abstract*— In recent years, cloud computing is evolving towards a distributed paradigm called Fog Computing, aiming to provide a distributed infrastructure by placing computational resources close to end-users. To fully leverage on Fog Computing, proper resource allocation is needed to cope with the demanding constraints introduced by IoT (e.g. low latency, high mobility). One of the main challenges that remain is Service Function Chaining (SFC). Services must be connected in a specific order forming an SFC allowing providers to benefit from the high flexibility and low operational costs introduced by network softwarization. In the demonstration, an SFC controller able to optimize the placement of service chains in Fog-cloud environments will be presented. The SFC controller has been implemented on the Kubernetes platform, an open-source orchestrator for the automatic deployment of micro-services. Our approach allows Kubernetes to deploy micro-services based on up-to-date information on the current status of the network infrastructure. The demonstration will show how application developers could use our approach to set up service chains for their services. Then, performance outcomes of our SFC controller will be shown, especially in terms of container deployment times.

***Index Terms*—Resource Provisioning, Fog Computing, Service Function Chaining, IoT, Kubernetes**

I. Introduction

Smart Cities [1] powered by the Internet of Things (IoT) have become quite popular in the past decade. Cloud computing is also evolving towards a distributed paradigm called Fog Computing [2], aiming to provide resources at the edges of the network to solve the demanding constraints introduced by IoT (e.g. low latency, high mobility). However, several challenges remain to fully benefit from Fog Computing technologies. One important challenge is called Service Function Chaining (SFC) [3], [4]. Services must be connected in a specific order forming an SFC that each user has to traverse to achieve a particular Network Service (NS). SFC allows cloud providers to dynamically reconfigure softwarized NSs without having to implement changes at the hardware level. Thus, providing a flexible and cost-effective alternative to today's static network environment. Furthermore, containers are currently revolutionizing the way developers build their software applications [5]. An application is decomposed in a set of small micro-services, which can be deployed across a large number of servers instead of the single monolithic application. In fact, containers are a tremendous alternative to the traditional Virtual Machines (VMs), due to their low overhead, high efficiency and increased portability.

In this paper, we present an SFC controller [1] to optimize the allocation of container-based service chains in Fog-cloud environments. The proposed approach has been accepted for publication in [6], where further details about SFC allocation in Fog-cloud environments are discussed. The current paper focuses on the testbed implementation and how developers could use our SFC controller in their experiments. In the live demonstration, we will set up a Kubernetes cluster with multiple Raspberry Pis. Then, we will explain how our SFC controller can be deployed in the Kubernetes platform and how to write configuration files to deploy container-based service chains based on our approach. Finally, we will analyze the chosen allocation schemes for each of the individual containers in the service chain and see the performance outcomes of our SFC controller in terms of container deployment times, expected service latency and load balancing. The proposed demo shows how SFC can be applied in a Fog-cloud environment while optimizing the resource allocation scheme in terms of low latency and bandwidth conservation.

II. The SFC Controller - Design and Implementation Overview

The SFC controller has been implemented as an extension to the Kubernetes platform. To fully understand our approach, two concepts coming from Kubernetes are essential, *Pods* and *Services* [7]. On one hand, a pod represents the collection of containers and storage (volumes) running in the same execution environment, meaning that micro-services in Kubernetes are often coupled together forming a group of containers. On the other hand, a *Service* is an abstract way to define a logical set of Pods and expose the applications running on them [8]. By using these two abstractions, there is no need to use a service discovery mechanism since pods have their own IP address, which makes load-balancing a straightforward process across them. The rationale behind these abstractions comes from the pods' volatility as they may be terminated, meaning that pods running at a certain moment may be different than the ones which are providing the service a few days later. Thus, users must not need to be aware that pods have been terminated and new ones have been deployed. An example of how container-based service chains can be deployed in Kubernetes based on these two abstractions is shown in Fig. 1. The

[1]https://github.com/jpedro1992/sfc-controller

component that makes decisions in terms of pod allocations in Kubernetes is called Kube–Scheduler (KS). The KS is the default scheduling feature in the Kubernetes platform, which is responsible for deciding on which nodes pods should be allocated. Our SFC controller has been implemented as a "scheduler extender" process that the KS calls out as a final step when a scheduling decision is needed. The scheduling decision process in Kubernetes by applying our SFC controller is shown in Fig. 2. Essentially, every pod requiring allocation is added to a waiting queue, which is continuously monitored by the KS. If a pod is added to the waiting queue, the KS searches for an adequate node for the placement. Firstly, KS executes the node filtering operation, where KS verifies which nodes are capable of running the pod by applying a set of filters. Inadequate nodes are already removed from the list of candidate nodes by applying these filters. Then, KS calls out the SFC controller to make the final decision on which cluster node the service must be provisioned based on the remaining set of nodes. The SFC controller gathers allocation information through labels defined on the pod configuration file. These labels are listed in Table I. In Fig. 3, an example of a pod configuration file with SFC information is shown. The SFC controller makes use of these labels to choose the best candidate node from the filtered ones to the desired scheduling policy. The provisioning algorithm is then selected based on the *Policy* label. Two policies are currently supported: *Latency-aware* and *Location-aware*. On one hand, if *Latency-aware* is preferred, the SFC controller selects the best candidate node based on the calculation of Dijkstra's shortest path algorithm. Provisioning records are kept of the previously allocated pods based on the *Network Service Header* label. If any of those corresponds to the same NS, the shortest paths will be calculated for each of the possible nodes. Thus, the node with the lowest combined shortest paths will be selected. On the other hand, if the *Location-aware* policy is chosen, the node selection is based on minimizing latency depending on the *Target Location* label, since certain pods may be preferred to be deployed on a certain Fog location or even in the Cloud, as they require a high amount of resources. Regarding bandwidth, for both strategies each candidate node is checked to confirm that it has enough bandwidth to support the given pod based on the *Min Bandwidth* label. After completion of each request, pod information is stored as a provisioning record to be consulted in further requests and the node's available bandwidth is updated. Thus, the SFC controller knows exactly the available bandwidth between requests, which allows it to make informed decisions based on latency and bandwidth information. If no suitable node is found after policy execution, link costs are calculated. The node with the maximum residual bandwidth link adequate to support the expected minimum bandwidth is selected to allocate the pod. Otherwise, it is not possible to allocate the service without compromising bandwidth and, thus, an event is triggered due to the failed pod deployment. Further details on how the SFC controller has been implemented and on its respective provisioning algorithms have been explained in [6].

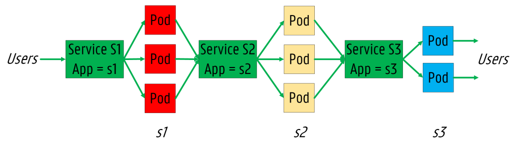
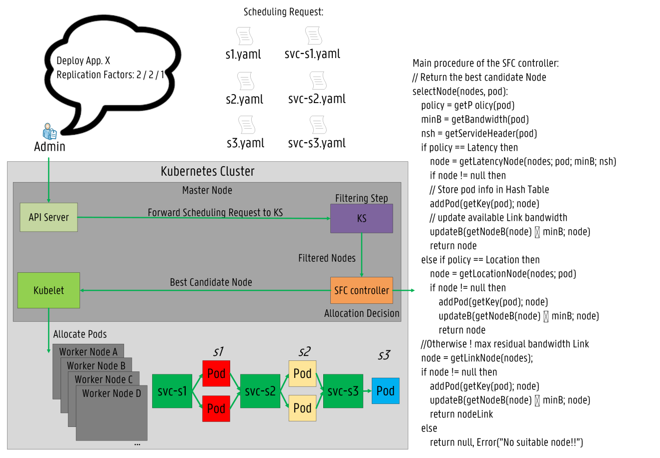


Fig. 1: An example of a container-based Service Function Chain deployment in Kubernetes.

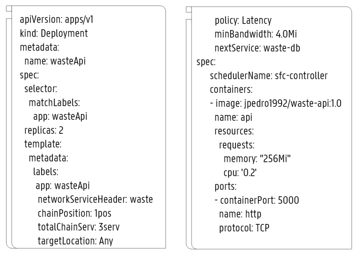


```
Main procedure of the SFC controller:
// Return the best candidate Node
selectNode(nodes, pod):
  policy = getP olicy(pod)
  minB = getBandwidth(pod)
  nsh = getServideHeader(pod)
  if policy == Latency then
    node = getLatencyNode(nodes; pod; minB; nsh)
    if node != null then
    // Store pod info in Hash Table
    addPod(getKey(pod); node)
    // update available Link bandwidth
    updateB(getNodeB(node) [illegible] minB; node)
    return node
  else if policy == Location then
    node = getLocationNode(nodes; pod)
    if node != null then
      addPod(getKey(pod); node)
      updateB(getNodeB(node) [illegible] minB; node)
      return node
  //Otherwise ! max residual bandwidth Link
  node = getLinkNode(nodes);
  if node != null then
    addPod(getKey(pod); node)
    updateB(getNodeB(node) [illegible] minB; node)
    return nodeLink
  else
    return null, Error("No suitable node!!")
```

Fig. 2: The scheduling decision process in Kubernetes by deploying the SFC controller.

TABLE I: Adding labels to the pod configuration file with SFC information.

Label	**Description**
Network Service Header	The specific SFC identifier (String).
Chain Position	The position of the given pod in the SFC.
Total Services	The total number of services in the SFC.
Target Location	The preferred location for the deployment.
Policy	The preferred allocation policy.
Min Bandwidth	The minimum expected bandwidth.
Prev Service	The previous service in the SFC.
Next Service	The next service in the SFC.



```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: wasteApi
spec:
  selector:
    matchLabels:
      app: wasteApi
  replicas: 2
  template:
    metadata:
      labels:
       app: wasteApi
        networkServiceHeader: waste
        chainPosition: 1pos
        totalChainServ: 3serv
        targetLocation: Any
```

```
        policy: Latency
        minBandwidth: 4.0Mi
        nextService: waste-db
spec:
      schedulerName: sfc-controller
      containers:
      - image: jpedro1992/waste-api:1.0
        name: api
        resources:
          requests:
            memory: "256Mi"
            cpu: '0.2'
        ports:
        - containerPort: 5000
          name: http
          protocol: TCP
```

Fig. 3: An example of a pod configuration file.

III. Demonstration Setup

In this paper, we propose a demonstration of an SFC controller running on a Kubernetes cluster set up using multiple Raspberry Pis with Weave Net [9] as a network overlay as illustrated in Fig. 4. In the live demonstration, the following features will be shown:

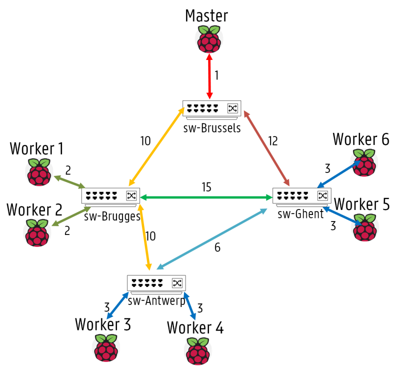


Fig. 4: The Kubernetes cluster used in the demo.

- **SFC controller installation:** The demo starts by showing how our SFC controller can be deployed in Kubernetes. Logs can be accessed to confirm its proper deployment.
- **Allocation requirements:** Then, explanations will be given on how to set up service requirements for each of the individual containers in the service chain and how the SFC controller will make use of that information in the scheduling process.
- **Service chain deployment:** Service chains based on the use cases presented in [6] will be deployed on the Kubernetes cluster. Several service requirements will be used (e.g. preferred location, minimum bandwidth, CPU and RAM requests). Two schedulers will be evaluated: the KS and the SFC controller.
- **Performance outcomes:** After chain deployment, performance outcomes will be analyzed to prove that our SFC controller is able to cope with all the predefined requirements. Results in terms of pod deployment times, expected service latency and load balancing will be compared with the ones obtained by KS as demonstrated in Fig. 5 and Fig. 6.

The demonstration highlights the importance of SFC issues in Fog-cloud environments. Several studies have been conducted in recent years, but most research has been focused on theoretical modeling and simulation studies, which limit their applicability to real deployments. Our demo will exhibit a practical testbed implementation of SFC provisioning in a real-world scenario by extending Kubernetes with network-aware mechanisms, enabling allocation decisions based on the current status of the network infrastructure.

IV. Conclusions

In this paper, we propose a demonstration of SFC allocation in Fog Computing. The presented SFC controller has been validated on the Kubernetes platform. Our approach allows application developers to set up requirements, especially in terms of bandwidth, which will help our SFC controller to allocate micro-services based on latency and location. In the live demonstration, we will explain how to deploy our SFC controller in the Kubernetes platform, how to write configuration files to deploy container-based service chains with our approach and, then, see performance outcomes in terms of container deployment time and allocation scheme. Our demo shows how SFC can be applied in a Fog-cloud environment while focusing on latency reduction and bandwidth conservation.

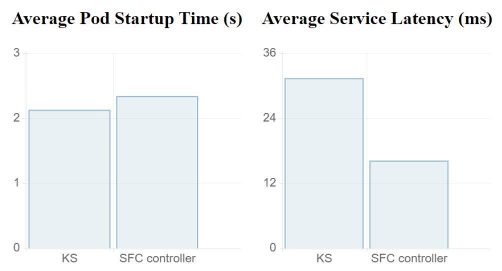


Fig. 5: Results in terms of pod deployment times and expected service latency.

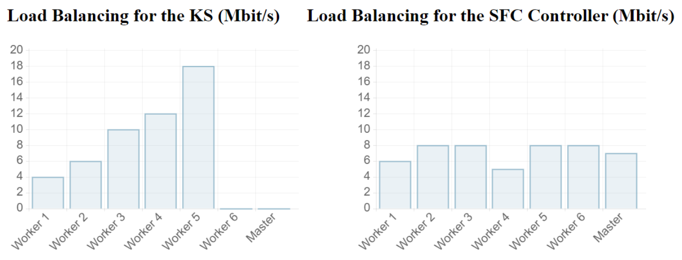


Fig. 6: Results in terms of load balancing.

Acknowledgment

This research was performed within the project ”Intelligent DEnse And Longe range IoT networks (IDEAL-IoT)” under Grant Agreement #S004017N, from the fund for Scientific Research-Flanders (FWO-V).

References

[1] H. Arasteh, V. Hosseinnezhad, V. Loia, A. Tommasetti, O. Troisi, M. Shafie-Khah, and P. Siano, “Iot-based smart cities: a survey,” in *2016 IEEE 16th International Conference on Environment and Electrical Engineering (EEEIC)*. IEEE, 2016, pp. 1–6.
[2] R. Mahmud, R. Kotagiri, and R. Buyya, “Fog computing: A taxonomy, survey and future directions,” in *Internet of everything*. Springer, 2018, pp. 103–130.
[3] D. Bhamare, R. Jain, M. Samaka, and A. Erbad, “A survey on service function chaining,” *Journal of Network and Computer Applications*, vol. 75, pp. 138–155, 2016.
[4] Y. Xie, Z. Liu, S. Wang, and Y. Wang, “Service function chaining resource allocation: A survey,” *arXiv preprint arXiv:1608.00095*, 2016.
[5] S. Newman, *Building microservices: designing fine-grained systems*. ” O’Reilly Media, Inc.”, 2015.
[6] J. Santos, P. Leroux, T. Wauters, B. Volckaert, and F. De Turck, “Towards delay-aware container-based service function chaining in fog computing,” in *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium. Accepted for publication*, April 2020.
[7] K. Hightower, B. Burns, and J. Beda, *Kubernetes: up and running: dive into the future of infrastructure*. ” O’Reilly Media, Inc.”, 2017.
[8] (2019) Kubernetes, automated container deployment, scaling, and management. [Online]. Available: https://kubernetes.io/
[9] (2020) Weave net, network containers across any environment. [Online]. Available: https://www.weave.works/oss/net/