Towards Fog-Aware Kubernetes

Ali J. Fahs
Univ. Rennes, Inria, CNRS, IRISA
ali.fahs@irisa.fr

Guillaume Pierre
Univ. Rennes, Inria, CNRS, IRISA
guillaume.pierre@irisa.fr

Abstract

Kubernetes is a mature and lightweight container management platform that easily scales to large-size clusters. However, we argue that in its current form it is essentially location-unaware, which makes it unsuitable to handle fog computing workloads. We therefore propose a roadmap of extensions to fill the gap between Kubernetes and the fog.

1 Introduction

Fog computing extends the concept of cloud computing with additional resources which are organized in a very different manner: when cloud platforms aim to concentrate huge amounts of computing resources in a small number of data centers, fog computing rather aims to distribute resources as broadly as possible across some defined area so some resources are always located in the immediate vicinity of every end user's devices. Proximity between end user's devices and the fog computing resources serving them is essential for applications which require ultra-low network latencies (e.g., augmented reality applications) or where it is desirable to process data where input data was produced and processing results will be used (e.g., IoT data analytics).

An essential requirement for a fog computing platform is therefore to take the location of fog resources and end-user devices into account to ensure proximity between the two. However, most platforms available today such as OpenStack, Docker Swarm, Mesos and Kubernetes were initially designed with data center deployments in mind and therefore often do not provide location awareness in their resource scheduling and/or load balancing algorithms.

In this paper we focus on understanding and addressing the limitations of Kubernetes in a fog computing scenario. Kubernetes matches many of the requirements of a fog platform: it is highly scalable and it can exploit even very limited machines thanks to its usage of lightweight containers rather than VMs. It is also a very open and mature platform that is now familiar to a large community of application developers. We discuss the limitations of the current implementation and propose a roadmap of the different improvements that remain necessary to make Kubernetes an equally attractive platform for fog computing scenarios as it is for cloud computing scenarios.

This paper is organized as follows. In Section 2, we discuss the state of the art. In Section 3, we introduce Kubernetes as a platform, discuss its advantages and disadvantages in the context of fog computing platform. In Section 4, we suggest the next steps through a roadmap. Finally, in Section 5 we conclude.

2 State of the art

A fog computing platform is in charge of managing compute, storage and networking resources that are physically distributed across a geographical area such as a building, a neighborhood and a city [1, 5]. It relies on virtualization techniques to support many independent end users and applications making use of these resources. For resource-efficiency reasons most fog computing platforms rely on lightweight container technologies rather than virtual machines, enabling one to envisage fog computing platforms making use of resources as limited as Raspberry Pis for example [11].

OpenStack++ is a set of OpenStack extensions that implements cloudlets: small-scale cloud data centers that are dispersed at the edge of the Internet [6]. A set of cloudlets may arguably be used to implement a fog platform. However, OpenStack++ bases itself on VM technologies, which does not match our requirements for resource efficiency in the presence of large numbers of independent applications sharing the resources of modest computing devices.

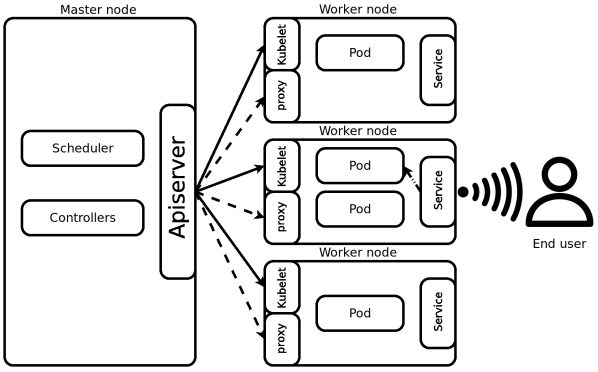

Figure 1: Kubernetes architecture.

PiCasso [8] is a container orchestration platform that specifically targets edge clouds with a focus on lightness and platform automation. However, it is still under development and not publicly available.

The most prominent container orchestration engines are Docker Swarm [4], Apache Mesos [7] and Kubernetes [2], which all provide advanced features for container scheduling, cluster management, and support for large-scale clusters [9]. They are currently the most mature systems to support the deployment of a fog/cloud platform. However, they lack a number of functionalities that are essential for large-scale fog computing scenarios, as we discuss in the next section in the specific case of Kubernetes.

3 Kubernetes and fog computing

We have chosen to base our efforts on Kubernetes. In particular, it offers many advanced functionalities such as the concepts of pods and services, auto-scaling, and a clean separation between the applications and the underlying platform. It has also attracted a very large community of users and developers, which positions it as a highly mature platform.

However, Kubernetes was designed with cloud computing scenarios in mind. We briefly present its main architecture, and discuss its strengths and current limitations when trying to use it as the basis of a fog computing platform.

3.1 Kubernetes architecture

Kubernetes is an open-source container-based platform which can manage the deployment and orchestration of service instances over a large-scale computing infrastructure. As presented in Figure 1, a Kubernetes cluster consists of one master node and multiple worker nodes. The master node is responsible for monitoring and managing the cluster via a variety of control components such as the resource scheduler and the deployment controllers. Conversely, the worker nodes actually execute user's applications inside containers according to instructions given by the master node.

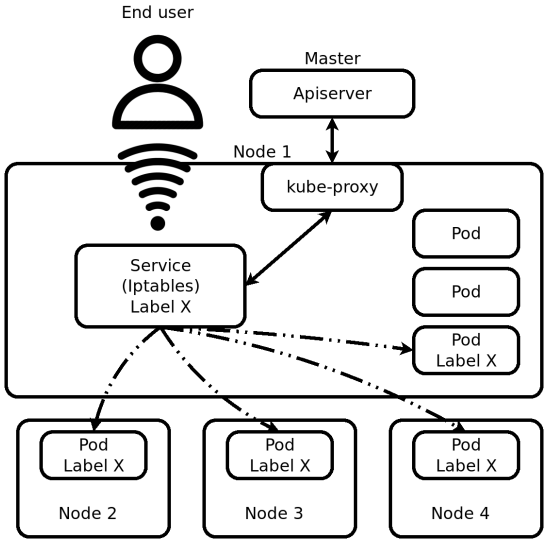

Figure 2: Organization of a Kubernetes service.

Kubernetes organizes containers in the form of *Pods*, where a pod is a tight group of logically-related containers running in a single worker node. Containers which belong to a single pod can communicate with each other using an isolated private network. Pods are not usually managed directly by the users. Rather, users are expected to create a *Deployment Controller*, which is in charge of the creation and management of one or more identical pods providing the expected service. Such a set of pods can be made publicly accessible from external end users by creating a *Service* which acts as a front end for the group of pods.

The master may schedule each pod in any worker node, by default chosen as the worker node with the lowest load in the system. Since pods may be placed in different nodes, communication is established as follows:

- Pod-to-pod communication is enabled using the Container Network Interface[1], which enables internal connectivity within the Kubernetes platform.
- Communication between the pods and the external world is ensured by the Service using a component called *kube-proxy* which relies on *iptables* to establish the appropriate routes.

As illustrated in Figure 2, the main purpose of a Kubernetes service is to connect end users to the exposed

[1] `https://github.com/containernetworking/cni`

pods. Whenever a pod is exposed, it takes a virtual IP address (called endpoint) and receives a label (Label X in the figure). The Service then acts as a load balancer between all the registered pods with the chosen label. The selection of the pod an end user request gets redirected to is driven by a load-balancing policy such as random, round-robin, source hashing, and shortest expected delay.

In the next sections we discuss the strengths and weaknesses of this architecture in the specific case where one chooses to use Kubernetes as the basis for a fog computing platform.

3.2 Advantages

Kubernetes has a number of properties that make it a great platform in cloud as well as fog computing environments.

Scalability. A fog architecture relies on wide distribution of marge numbers of points of presence, spread over a potentially large geographical area. Kubernetes supports large-scale clusters up to 5000 nodes and 150,000 pods, which largely matches the scalability requirements of most fog computing scenarios.

Simple management. Managing a Kubernetes cluster is very simple and robust. For example, adding new nodes is as simple as running a join command, and the master node will take care of everything, without the need of manual intervention.

Containerization. In a fog computing environment we expect to simultaneously execute large numbers of pods on modest-sized machines. It is therefore important to minimize the overheads due to the virtualization technologies. The fact that Kubernetes relies on containers rather than virtual machines significantly reduces this overhead, in particular with respect to memory usage.

Robustness. Kubernetes relies on control-theory principles where the system continuously monitors itself in order to react to any discrepancy between the actual and the desired system state. This makes Kubernetes extremely robust in face of various types of failures and misbehaviors that are due to be even more frequent in a decentralized fog architecture than a datacenter-based cloud.

Community. Kubernetes is surrounded by a large-scale community of thousands of contributors from the industrial and academic fields, meeting and events, and forms to file issues and ask questions. The support provided by this community eases the development of the software.

3.3 Limitations

Despite its many qualities, Kubernetes was not designed with fog computing in mind. It therefore has a number of limitations that make it difficult to use out of the box for fog computing environments.

Centralization. A Kubernetes cluster is composed of one master node and a number of worker nodes. In this design, all important management decisions are made by the master. In particular, most of the cluster controllers are located there. In a fog computing environment where the worker nodes may be located far from the master this increases the latency of all the control operations performed by the master over some of the worker nodes.

Location unawareness. The purpose of a fog computing platform is to exploit the proximity between the platform resources and the end users [1]. However, Kubernetes was designed with the hypothesis that all available resources are equally suitable to serve any end user. Although this hypothesis is very sensible in cloud scenarios, it introduces significant limitations in a fog context.

- *Pod scheduling:* When creating a new pod, the Kubernetes scheduler does not take the end users' locations into account to determine which worker node should host the new pod. As a result, the pods which constitute a service may be located further away than necessary from the end users they are serving.
- *Load balancing:* Even though the end users may be located in close vicinity of a pod providing the requested service, simple load-balancing policies such as round-robin may direct user requests to other pods located at much greater distance. It is therefore necessary to also adjust the way user requests are routed to the appropriate pod.

4 Making Kubernetes location-aware

We believe that the most important limitation that needs to be addressed is to make Kubernetes location-aware since location-awareness is the prime goal of any fog computing platform. A longer-term objective will be to decentralize some of the platform's control processes.

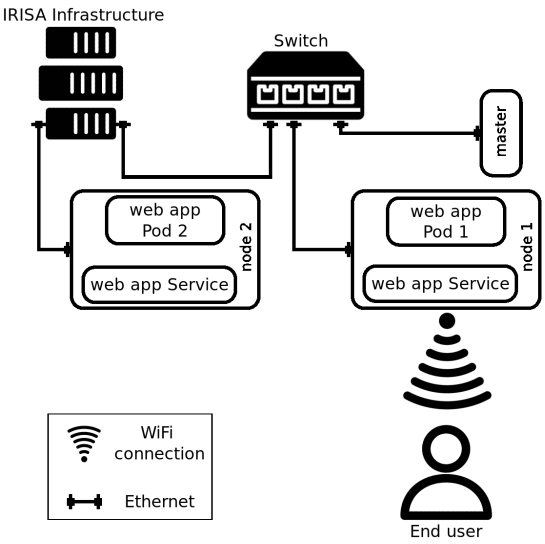

Figure 3: Testbed topology

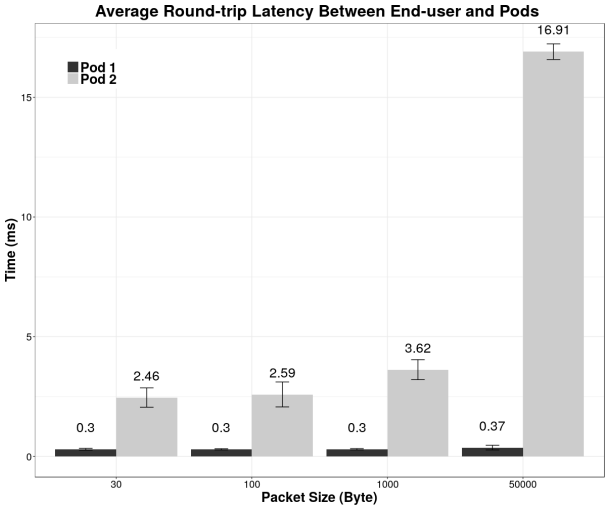

Figure 4: Round-trip latencies.

4.1 Illustrating the need for location awareness

To illustrate the need for location-awareness, we deployed a small testbed composed of three Raspberry Pi 3 nodes running Kubernetes v1.9. As illustrated in Figure 3, one of these nodes acts as the cluster's master node while the other two are considered as worker nodes. The worker nodes also act as Wi-Fi gateways so the end users may connect to the closest worker node with a 1-hop Wi-Fi connection. The master and worker nodes are connected to each other using a 1 Gbps Ethernet local-area network[2].

Figure 4 shows the network-level round-trip time between the end user and the two pods running identical Web applications:

- When accessing Pod 1, the packets need to traverse only one Wi-Fi network hop to reach the pod.
- When accessing Pod 2, the packets need to traverse one Wi-Fi network hop plus one extra Ethernet network hop.

Clearly, reducing the network path between the end user and the pod which provides him with services significantly reduces the round-trip time latencies. The difference grows even more when the packet size increases.

[2]The local-area network infrastructure guarantees a 1 Gbps capacity although the Ethernet interface of the Raspberry Pi 3 only supports 10/100 Mbps. This means that the network infrastructure will not impose additional bottlenecks besides the limitations of the Raspberry Pi hardware.

We believe that the main source of additional latency when routing a message through a 2-hop network path largely resides in the context switching and delays due to multiple layers of network virtualization in the Raspberry Pi which routes traffic between the end user and the service pod. In a larger fog computing environment where geographically-distributed nodes may be connected to various commodity networks, we expect these differences to be at least as large.

4.2 How Kubernetes routes requests to pods (and how to improve it)

Figure 5 details the way an external end user's requests get routed toward one of the pods providing the requested service. In a standard cloud scenario, the system administrator needs to setup an *Ingress* (on a public IP address usually registered using DNS) which exposes the service to the outside world. The Ingress merely forwards traffic to the appropriate Service on the same node, which in turn decides to which pod the request should be routed to.

In the fog computing scenario depicted on the bottom half of the figure, it is no longer necessary to rely on a specific Ingress. Instead, since the end users connect directly to one of the worker nodes, their requests immediately enter the Kubernetes cluster without the need to link the external Internet with the internal Kubernetes system. In Kubernetes, every worker node can access the service and make request-routing decisions locally.

The default load-balancing policy randomly distributes requests among pods. This is obviously sub-optimal in the case of fog computing, as for

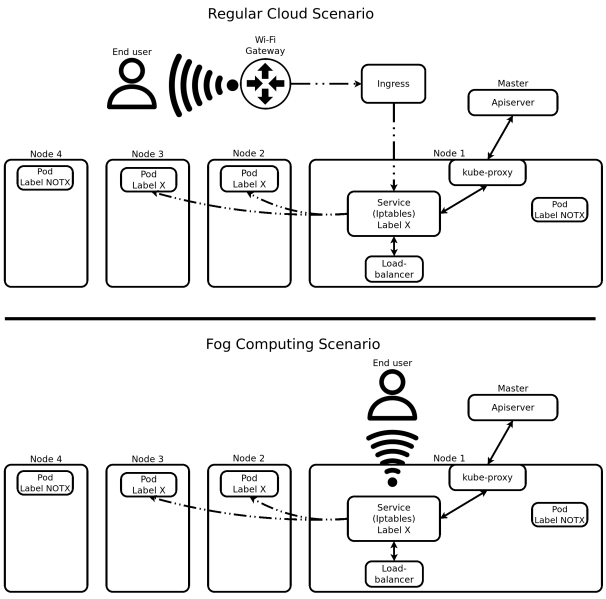

Figure 5: Communication between pods and external end users.

example a worker node which contains a pod of some service may redirect requests to another pod instead of servicing the request itself. We therefore plan to design new location-aware load-balancing policies that will aim at routing traffic to a nearby pod. The decision may be based on nodes' GPS coordinates (if we want to minimize the geographical distance) or Vivaldi coordinates [3] (if we want to minimize the network latency).

4.3 Location-aware scheduling

When creating a new pod, the Kubernetes scheduler should use knowledge about the end-users' locations as well as the available resources in each worker node to determine in which node the new pod will be most useful. Kubernetes already monitors the nodes' load, but it will be necessary to add user location monitoring. In our case a user's location may be approximated by the location of the Wi-Fi gateway it is using to access the system. If every worker node is attached to a Vivaldi coordinates, then determining the optimal pod placement can be reduced to a geometric problem [10].

However, we must also anticipate that users may be mobile and that the workload imposed on each pod will significantly vary over time. It will probably become necessary to adapt the Kubernetes auto-scaler or to design an entirely new controller to periodically estimate how efficient the current replica placement is, and to initiate adjustments when necessary.

A longer-term goal may also be to decentralize the scheduling process itself. In a fog computing environment, the resources to be scheduled are usually located very close from the process which requested their creation. In this context it might make sense to avoid using a centralized scheduling component, and to rather let the nearby worker nodes self-organize to allocate their available resources according to local demands.

5 Conclusion

Fog and cloud computing platforms share many identical concerns, which makes it attractive to use mature cloud computing platforms to build new fog computing platforms. However, this requires a number of adjustments in particular to support location-awareness. We have chosen Kubernetes as the basis for our work, and proposed a roadmap of the necessary adjustments before it can realistically be considered as the world's best fog computing platform.

References