COLLEGE MATHS, TELECOMS
DOCTORAL INFORMATIQUE, SIGNAL
BRETAGNE SYSTEMES, ELECTRONIQUE

Université de Rennes

THÈSE DE DOCTORAT DE

L'UNIVERSITÉ DE RENNES

ÉCOLE DOCTORALE N° 601
Mathématiques, Télécommunications, Informatique, Signal, Systèmes, Électronique
Spécialité : *Informatique*

Par
Chih-Kai HUANG

Scalability of Geo-Distributed Fog Computing Federations

Thèse présentée et soutenue à Rennes, le 08 October 2024
Unité de recherche : IRISA (UMR 6074)

Rapporteurs avant soutenance :

Prénom NOM Fonction et établissement d'exercice
Prénom NOM Fonction et établissement d'exercice
Prénom NOM Fonction et établissement d'exercice

Composition du Jury :

Attention, en cas d'absence d'un des membres du Jury le jour de la soutenance, la composition du jury doit être revue pour s'assurer qu'elle est conforme et devra être répercutée sur la couverture de thèse

Président :	Prénom NOM	Fonction et établissement d'exercice *(à préciser après la soutenance)*
Examinateurs :	Prénom NOM	Fonction et établissement d'exercice
	Prénom NOM	Fonction et établissement d'exercice
	Prénom NOM	Fonction et établissement d'exercice
	Prénom NOM	Fonction et établissement d'exercice
Dir. de thèse :	Prénom NOM	Fonction et établissement d'exercice
Co-dir. de thèse :	Prénom NOM	Fonction et établissement d'exercice *(si pertinent)*

Invité(s) :

Prénom NOM Fonction et établissement d'exercice

Experiments presented in Chapter ?? and Chapter ?? were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see https://www.grid5000.fr).

Acknowledgement

Firstly, I would like to express my sincerest gratitude to my supervisor, Professor Guillaume Pierre, for his continue guidance and strong support in both my research and life throughout my entire Ph.D. journey in France. Without his patience and understanding, this journey would not incredibly happy and smooth. It is an great honor to be supervised by a fantastic professor like you. I will continue to help others just as you were helped me. I am also want to give big thanks to Shadi Ibrahim for his help and effort on this thesis.

I would like to thank the jury members: Professor xxx, Professor xxx, Professor xxx, Professor xxx, and Professor xxx. Thank all of the members for taking the time to review and evaluate my thesis and give many valuable comments and suggestion. I am also deeply grateful to Professor Erik Elmroth and Professor Cédric Tedeschi for being members of the CSID committee. Their invaluable feedback and support over three years, which success my Ph.D. study.

Many thanks to all the entire Magellan team for their support, chats, and discussions. In particular, I would like to express my appreciation to Emmanuel, Govind, Matthieu Simonin, and Stéphanie for their help in all aspects. A special thanks also to CMI Rennes for providing great service and take care all my complex France administrative procedures. My sincere gratitude to many wonderful people I met and knew in France, especially Zhilei, Kai Gu.

My deepest gratitude to my incredible family for their invincible support, care, and love make this dream happen. This achievement is not only to me but also to my family. Seeing my family proud would be my greatest success.

Finally, I want to give me a big thumb up to myself for all the hard work in my Ph.D. studies over the past three years. I am proud of myself, and I am excited to see what is the next stop for me.

Attitude defines your reach, details ensure your success

RÉSUMÉ

French version here

Première contribution :

Deuxième contribution :

Abstract

English version here

Contribution 1 :

Contribution 2 :

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Cloud computing as one of the successful computing paradigms makes a fresh start for enterprises in developing software and deploying applications. Traditionally, enterprises invest a lot of money in building their server rooms with Information Technology (IT) equipment such as servers, switches, and firewalls. Nowadays, by using cloud resources with high performance, flexibility, on-demand, reliability, and scalability from public cloud providers, enterprises can focus their major business and other priorities without worrying about IT-related installation and maintenance [1]. Therefore, numerous enterprises have deployed most or all of their applications and data to cloud data centers [2].

Each public cloud provider typically builds and maintains their cloud data centers, which are composed of a vast number of computing, network, and storage resources. Virtualization technologies enable Cloud Service Providers (CSPs) to abstract these resources and share them with multiple cloud users without letting inference each other. For example, hardware virtualization can run multiple Virtual Machines (VMs) in a single physical machine [3]. By doing so, it improves the computing resource utilization [4]. However, the number of data centers is typically small. Figure 1.1 shows the locations of Google Cloud data centers, which have 40 available regions in the world and plan to launch new regions such as Mexico [5]. This situation may remain these data centers physically distant from the end users, which results in the high access latency [6].

According to a survey from Enterprise Strategy Group, there is a strong trend for multi-cloud deployment. The findings indicated that 85% of organizations leverage two or more CSPs for application deployment [7]. A key reason for this trend is that these applications can deploy and distribute to different data center locations to gain the advantages of multi-cloud deployment such as reducing the latency [8].

However, the emergence of latency-sensitive applications requires a low end-to-end latency, which may not be compatible with traditional cloud computing deployment [9]. Some head-tracking applications, such as virtual reality and 360-degree video streaming, demand network transmission with application processing of less than 20 milliseconds to

Figure 1.1 – A map of Google Cloud regions around the world. Blue dot show current regions. Triangle represents future regions [5].

avoid motion sickness [10]. Moreover, the Internet of Things (IoT) industry is experiencing rapid growth that have 127 new devices connected to the Internet every second [11]. Another report shows that the number of connected IoT devices worldwide in 2030 is estimated to be more than 29 billion [12]. Typically, these IoT devices generate data and must be sent to a cloud data center for real-time data processing due to the limited computing capabilities of these devices [13]. Long-distance data transmission of such large amounts of data over the network may eventually saturate the existing network links [14].

To address the limitations of centralized cloud computing, the obvious solution is to place the computing resources close to the users. Fog computing is one of the concepts for this ideal. It deploys computing resources in the network edge, close to the end users and the location of data source generated by IoT devices [15]. This design enables computational tasks to be performed close to the data source, reducing end-to-end latency to improve the user Quality-of-Experience (QoE) and the volume of data transmitted from IoT devices to data centers for analysis. While fog computing brings significant benefits, there remain challenges that need to be tackled to fully realize the fog computing potential.

Firstly, the major differences between fog computing and cloud computing are related to computing paradigms. The computing resource of fog computing is geographically dis-

tributed with potentially weak servers, small storage, and unstable networks. For example, Raspberry Pi single-board computers are popular choices for prototyping fog computing nodes due to their affordability, and sustainability [16], [17]. On the other hand, cloud data centers are large and centralized with many powerful servers massive volumes of storage, and stable networks.

Although VMs have been widely used in cloud data centers, deploying applications through traditional VMs on these resource-constrained fog nodes presents significant challenges. In such cases, containerization is seen as a suitable technology for fog computing. The main reason is the lightweight nature of containers, which do not contain a full guest Operating System (OS). This design keeps resource consumption low, which significantly reduces the overhead. Moreover, without containing the OS, the size of the images can be greatly reduced. This reduction of the size allows containers to have faster launching time. Generally, a container can initialize and reach a ready state to process user requests in a few seconds or less. At the same time, several works leverage containers to improve resource utilization and reduce service latency in fog computing environments [16], [18]–[20].

As the number of containers increases, resulting in management difficulties. To handle these containers, it is desired a robust orchestrator to manage the need for autonomic deployment, scaling, and self-healing. Therefore, several container orchestration solutions have been developed, including Docker Swarm [21], Apache Mesos [22], and Kubernetes [23]. Between these orchestration solutions, Kubernetes has emerged as the most widely adopted platform [24]–[26].

Kubernetes is an open-source container orchestrator that automates the deployment, scaling, and management of containerized applications in hybrid or public cloud infrastructures. In 2018, the Cloud Native Computing Foundation (CNCF) accepted Kubernetes as the "graduated" maturity level, which shows Kubernetes is a stable and production-ready platform [27]. While Kubernetes handles the management of a large number of containers, it needs to rely on container runtime such as containerd [1], CRI-O [2] on each node to run the containers in a Kubernetes cluster. Kubernetes has been adopted in many academic works for fog computing environment [28]–[32]

The second challenge for fog computing is scalability. Building a fog computing platform at the scale of a country or even a continent, a single Kubernetes cluster may not be

1. containerd - https://reurl.cc/orWqkg
2. CRI-O - https://reurl.cc/bD0yQE

able to handle this size of requirement. For example, the Kubernetes official recommends that a Kubernetes cluster of the worker nodes should not exceed 5,000 nodes and each node should be no more than 110 Pods with a total of 150,000 Pods [33].

The limitation of scalability for a single Kubernetes cluster makes it hard to fulfill the requirement that the fog platform may cover a country or a continent. To overcome this issue, deploying a large number of individual Kubernetes clusters in different strategic locations has emerged as a viable solution. This design allows individual clusters to manage their own resources, thereby increasing the overall number of worker nodes and Pods to a significantly greater extent.

The Kubernetes community has recognized the need for multiple cluster deployment. Therefore they have established a dedicated Special Interest Group (SIG) [3] focused on managing these independent clusters. This SIG proposes a federation solution called Kubernetes Cluster Federation (KubeFed) to automate application deployment and resource management in multi-cluster environments [34]. In a cluster federation, a "management cluster" is in charge of deciding which of the "member clusters" will be in charge of handling each newly deployed application. Each member cluster manages its workloads and worker nodes while being overseen by a management cluster. To enable fog platform administrators to treat these independent geo-distributed clusters as a single homogeneous cluster, it is beneficial to utilize the federation framework to manage multiple clusters and distribute the workloads from a management cluster.

To build the desired fog computing platform, this thesis applies the above concepts of launching multiple Kubernetes clusters and assume each of which can be deployed in different strategic locations to serve the end users. In this context, the goals of this thesis is to investigate and address the scalability issues of fog computing within geo-distributed cluster federation environment.

The first scalability issue addressed in this thesis is the service coverage of fog resource providers. Deploying a sufficient number of geo-distributed fog clusters with the federation framework to cover a country or even an entire continent for a single company is a big challenge, especially in terms of capital and operational expenses. Moreover, it may be difficult for a fog computing platform built by a single company to attract sufficient workloads to generate high resource utilization. This lack of utilization results in increased average operating costs.

3. Multicluster Special Interest Group - https://reurl.cc/2zkaD6

The second scalability issue addressed in this thesis is the management of geo-distributed fog computing federations. A federation may have a large number of users and clusters, requiring a robust federation framework for ensuring isolation between users and the scalability of the platform. However, traditional federations framework, such as KubeFed, is designed for environments where all users trust each other and make the underlying assumption that all resources in a federation belong to a single administrative domain. Therefore, KubeFed only supports a limited multi-tenancy solution, which follows a standard Kubernetes design to create namespaces to isolate the tenants.

In terms of scalability for the federation framework, fog computing in a multi-cluster approach deploys a large number of distributed small clusters in strategic locations to provide services. Therefore, a single management cluster must demonstrate the ability to effectively manage a vast number of member clusters to support scalability in fog computing. However, KubeFed relies on a centralized control plane to manage member clusters, making it challenging to scale the number of member clusters and applications to a higher degree. Moreover, KubeFed uses the *Push* method to deploy applications across member clusters, which puts all of the control stress in a centralized management cluster. Using *Push* method can make the federation unstable in the presence of transient network failures [35].

The third scalability issue addressed in this thesis is the monitoring of geo-distributed fog computing federations. Monitoring plays a critical role in resource/service usage tracking, resource scheduling, and performance monitoring in such environments, especially in potentially resource-constrained and unstable fog cluster environments [36], [37]. For example, by using monitoring data, the scheduler can efficiently place applications on a set of available geo-distributed clusters [38]. However, monitoring a large cluster federation is a very challenging task. Compared to traditional centralized cloud computing, these fog clusters are located in different places. The volume of monitoring data transmitted in the federation grows with the number of clusters, which may eventually saturate the existing inter-cluster network links and represent the majority of the system management traffic [39]. In addition, the trend toward microservices deployment enables the design of new application architecture in which the number of components communicating with each other increases by orders of magnitude. This situation raises the number of monitored components, which further increases the number of metrics that need to be reported to the monitoring system.

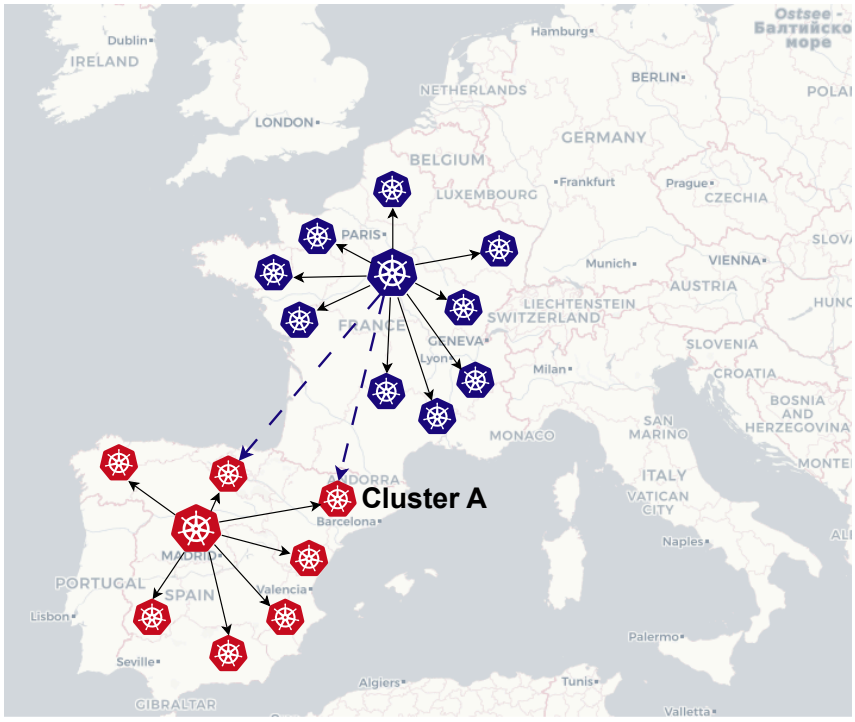


Figure 1.2 – An example of a fog provider with its federation from France wants to expand the service region to Spain. The Kubernetes logo represents the Kubernetes cluster, and the larger Kubernetes logo means a management cluster, while others are member clusters. The dashed line is the France fog provider's desired cluster to expand the service region. The base image is from OpenStreetMap [40].

1.1 Contributions

The goal of this thesis is to investigate scalability issues in the geo-distributed fog computing federation environment. Therefore, three contributions are proposed to address the above challenges. Our contributions are based on Kubernetes, Prometheus and their ecosystem due to their open source, widespread use and maturity. However, the concepts and algorithms proposed in this thesis can be easily applied and integrated with other current or future container orchestrators and monitoring solutions. The principal contributions of this work are as follows:

(1) **The concept of *meta-federation***
Designing large-scale fog computing platforms requires the aggregation of large numbers of clusters in numerous locations covering a country or even a continent. It would be difficult for a single organization to deploy enough resources while attract-

ing sufficient workloads to generate high resource utilization. As a result, there will be many fog providers in a country or a continent, and each provider is responsible for a specific area of fog resources. An independent fog resource provider may want to expand their geographical span in locations where they do not own resources themselves, giving its customers more choices of where to deploy applications. In this way, the fog provider can attract more new customers to increase the size of the business. As shown in Figure 1.2, a fog provider from France may want to use the fog resources from Spain.

Enabling fog platforms to embrace the full benefits of cloud computing principles requires the design of large-scale shared geo-distributed fog computing platforms that any application may make use of and where statistical multiplexing of large numbers of independent workloads can help guarantee high resource utilization. This thesis proposes the design of scalable fog *meta-federation* to address this challenge. We define a meta-federation as a complex ecosystem composed of many independent fog resource providers that may set up business agreements with one another to allow access to their computing resources and thereby expand their geographical span in locations where they do not own resources themselves.

(2) **UnBound: Multi-Tenancy Management in Scalable Fog Meta-Federations** Realizing the vision of a large meta-federations ecosystem that can span a country or a continent with thousands of local providers renting their computing resources to hundreds of independent federations requires one to address two main challenges: (i) Multi-tenancy: Workloads submitted by multiple federations to the same member cluster should be strictly isolated from each other, and multiple users from any single federation should also benefit from similar isolation guarantees. (ii) Scalability: Each management cluster must effectively control a large number of member clusters, while each member cluster must be able to lease its resources to a large number of management clusters.

In this thesis, we propose UnBound, a scalable fog meta-federations platform that specifically addresses the multi-tenancy scenario, where individual fog clusters may lease their resources to multiple administrative domains. UnBound relies on Kubernetes to orchestrate resources within individual fog clusters and Open Cluster Management (OCM) to federate multiple member clusters under the authority of

a management cluster [23], [41]. OCM is an open-source multi-cluster orchestration tool that manages clusters, distributes workloads across them, and takes into account the scalability of a federation. We address the issue of multi-tenancy management by isolating federations within a single member cluster using the Virtual Kubernetes Clusters (vCluster) project to create isolated logical sub-clusters within the member clusters [42]. Each vCluster [4] has its own API server and data store, which provides stronger isolation guarantees than simple Kubernetes namespaces to ensure that different federations do not interfere with one another.

We conduct extensive evaluations through real-world deployment in the Grid'5000 testbed and demonstrate that UnBound achieves inter-user and inter-federation isolation while maintaining comparable application creation time to the original Open Cluster Management and avoiding increasing cross-cluster network traffic between management and member clusters. Moreover, the resource consumption of UnBound components remains within acceptable limits. Finally, we demonstrate the stability and scalability of UnBound using federations with up to 500 member clusters and a member cluster belonging to up to 100 independent federations.

(3) **Efficient Monitoring in geo-distributed fog computing federations**
To enable accurate scheduling decisions, it is necessary to have information on the resource usage status of each member cluster in geo-distributed fog computing federations [38]. This requires a robust monitoring framework that can provide resource utilization data, such as Prometheus [43] and its extension Prometheus Federation [44]. However, periodically with the fixed frequency reporting the precise status of each available server is both unnecessary to allow accurate scheduling, unscalable when the number of servers grows and may waste network bandwidth in the federation.

In this thesis, we presents two frameworks, Acala and AdapPF, to balance between cross-cluster network traffic and accuracy of monitoring data in a federation. Acala three techniques, metrics aggregation, metrics deduplication, and dynamically adjust the collection frequency, to reduce cross-cluster network traffic between management cluster and member clusters.

4. In this thesis, the term "vCluster project" refers to the vCluster framework, and the term "vCluster" refers to a virtual cluster created for isolation between different management clusters in a member cluster.

1.2 Published papers

The following manuscripts are currently published or under review as part of this thesis:

Journal article(s)

(1) "Aggregate Monitoring for Geo-Distributed Kubernetes Cluster Federations", **Chih-Kai Huang** and Guillaume Pierre, in IEEE Transactions on Cloud Computing, Under review.

Conference paper(s)

(1) "UnBound: Multi-Tenancy Management in Scalable Fog Meta-Federations", **Chih-Kai Huang** and Guillaume Pierre, in ACM/IFIP International Middleware Conference, Under review.

(2) "AdapPF: Self-Adaptive Scrape Interval for Monitoring in Geo-Distributed Cluster Federations", **Chih-Kai Huang** and Guillaume Pierre, in Proceedings of the 28th IEEE Symposium on Computers and Communications, Tunis, Tunisia, Jul 2023.

(3) "Acala: Aggregate Monitoring for Geo-Distributed Cluster Federations", **Chih-Kai Huang** and Guillaume Pierre, in Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing, Tallinn, Estonia, Mar 2023.

1.3 Organization of the thesis

Chapter 2

Background

In this chapter, we first explore the cloud computing concept with its characteristics, architecture, service model and limitations, and then shift to geo-distributed fog computing. After that, we discuss the evolution of virtualization technology from Virtual Machines (VMs) to containers. We also review Kubernetes and the concept of the Kubernetes Federation.

2.1 Cloud computing

Cloud computing has been one of the most significant technological concepts over the past twenty years [45]. The history of cloud computing can be traced back to 1961 when John McCarthy introduced the first utility computing concept at the Massachusetts Institute of Technology [46]. Afterward, the Compaq Computer Corporation started to bring computing into the business aspect in 1996 [47]. It took almost a decade of evolution for three giants, Amazon, Microsoft, and Google, to roll out their own "cloud" services or platforms successfully. In 2006, Amazon released its cloud computing services, which included Amazon Simple Storage Service (S3) and Amazon Elastic Compute Cloud (EC2) [48], [49]. Then, Microsoft Windows Azure [50] and Google App Engine [51] were also announced and started sharing the cloud market. As of the fourth quarter of 2023, the three major Cloud Service Providers (CSPs) shared a total of around 66% of the cloud market [52].

Cloud computing is changing the way individuals and enterprises develop software and deploy applications. A Eurostat survey of 161,000 EU enterprises with different sizes shows that 45.2% of EU enterprises used cloud computing services in 2023. Compared to 2021, it is 4.2 percentage points greater, representing a growing trend in the business fields [53].

The rising trend of cloud computing can be attributed to the gradual maturation of software. Virtualization technologies such as Virtual Machines (VMs) [54] and contain-

ers [55] enable cloud computing as CSPs can leverage them to abstract the computing resources on a physical server and share each server with multiple users without letting them interfere with each other. Using virtualization technology in cloud platforms brings additional benefits, including improving the overall resource utilization in each server, reducing costs, and providing functions such as high availability and auto-scaling [56]. We will discuss virtualization technologies in Section 2.3.

For users, there are several advantages to using cloud computing such as reducing their investment cost of Information Technology (IT) infrastructures and software. The users can directly utilize the services provided by CSPs without needing to deploy or maintain their own servers and storage systems [57]. Cloud computing can also enhance the flexibility of deployed services such as dynamically scaling capacity of IT services up and down according to the demand [1]. For instance, a shopping website may experience greater workloads during the weekend, requiring the administrator to scale up the cloud resources to handle the load. After the peak has ended, they can scale resources down to maintain cost-effectiveness. One more advantage that users can benefit from using cloud computing is efficiency. Enterprise users only need to focus on their major business and do not need to care too much about IT-related work. Moreover, applications and data hosted in the cloud can be accessed from almost any device connected to the internet [58].

2.1.1 Cloud computing characteristics

Cloud computing relies on numerous servers, networks, and storage resources being placed together in the same location to build a data center that can provide services to cloud users over the Internet. According to the National Institute of Standards and Technology (NIST) definition, the basic characteristics of cloud computing are listed below [59]:

(1) **On-demand self-service:** CSPs provide cloud users with services or computing resources such as applications, data storage, and infrastructure. It can automatically allocate resources according to user requirements without system administrator intervention.

(2) **Broad network access:** Cloud users are able to access the cloud services with different types of devices such as mobile phones, laptops, and desktop computers, anytime and anywhere through the Internet.

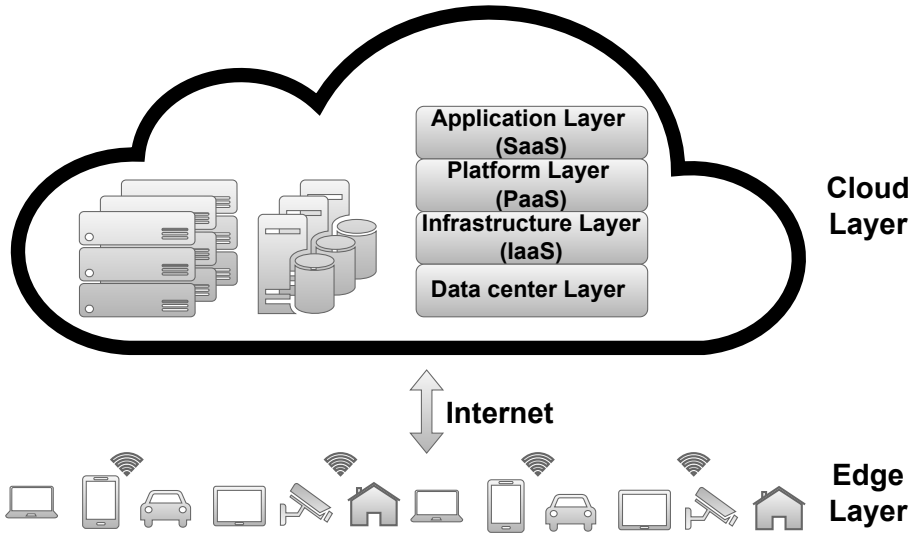


Figure 2.1 – High-level architecture of cloud computing.

(3) **Resource pooling:** The resources managed by CSPs are aggregated into a shared massive computing pool. By using virtualization technologies, CSPs can share the resources or provide services to the end users through a multi-tenancy model.

(4) **Rapid elasticity:** The size of services can quickly scale up and down to adapt to user needs.

(5) **Measured service:** CSPs can monitor the service usage of cloud users and are billed based on the pay-per-use method. Moreover, the CSPs can leverage monitoring data to manage the resources in a data center.

In addition to the above basic characteristics, there are some common characteristics, including massive scale, resilient computing, and geographic distribution service orientation [60]. To sum up these characteristics, cloud computing is the integration and development of distributed computing, Internet technology, and large-scale resource management.

2.1.2 Cloud computing architecture and business models

Figure 2.1 presents the relationship between cloud computing and its end users. The cloud layer aggregates computing resources into one or more data centers and provides services to the end users. The edge layer is composed of end users, including their Internet-of-Things (IoT) devices, mobile phones, and computers. These devices produce the data

and requests and send them through the Internet to the cloud data centers for analysis or processing.

The architecture of a data center can be divided into four layers, which are application, platform, infrastructure, and data center layers [57], [61]. NIST also classifies the service models into three levels: Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS) [59]. SaaS is oriented towards end users who only use complete applications. It provides Internet-based on-demand software services without requiring users to install or maintain them. Platform as a Service (PaaS) targets application developers, in which PaaS delivers a platform that contains software development and management frameworks. Developers only need to upload code and data to use the platform without having to worry about the issues related to the underlying network, storage, and operating system. IaaS is designed for users who require complete control over their computing infrastructure, including servers for computation, storage, and networking. Each service model can be mapped to the different architecture layers, and we discuss the architecture from the down to the top layer of cloud computing.

(1) **Data center layer:** CSPs operate the data centers to offer the services to the cloud users. Each data center includes thousands or more physical machines, such as servers, switches, and routers, packed into racks, and connected by a high-bandwidth internal network. There are several challenges in this layer, including the configuration of hardware components, ensuring fault tolerance, and keeping energy efficiency.

(2) **Infrastructure layer:** This layer, also called the virtualization layer, leverages virtualization technologies to share computing resources with different cloud users. The infrastructure layer belongs to the IaaS model, providing computing resources such as servers, storage, and networking. Some well-known IaaS products are Amazon Elastic Compute Cloud (Amazon EC2) [62], Google Cloud Storage [63], and Azure Virtual Network [64].

(3) **Platform layer:** The platform layer stands in the PaaS model and is built using the infrastructure layer. It is responsible for providing application frameworks to the software developers. It can relieve developers from the burden of managing servers and deployment settings. Google App Engine [65] is one of the products in this layer.

(4) **Application layer:** This layer can map to the SaaS model and aims to offer different cloud software services to cloud users, such as E-mail services and document editor. Typical examples are Google Workspace [66], which include Google Docs and Gmail.

2.1.3 Cloud computing deployment models

Cloud users can choose between different deployment models for their applications based on their requirements such as locations and policies. Cloud data centers, managed by CSPs such as Microsoft Azure, Google Cloud and Amazon Web Services, provide the services mentioned in the previous section in the form of public cloud, where cloud users do not need to maintain their own infrastructure and software. For cloud users who want a higher degree of performance, reliability and security, an alternative is to select a model named private cloud where cloud users build their own data centers to fulfill the needs of control or privacy.

Some users need their applications to run in specific geographic locations for performance or legal reasons, while others wish to avoid a single vendor lock-in. For them, multi-cloud is a viable solution for geo-distributed application deployment [8], [67]. The administrators of applications can launch multiple applications by using different public cloud service providers in various locations to serve end users. For example, Google Cloud and Microsoft Azure operate data centers in Taiwan. However, AWS does not have one. If AWS customers want to deploy an application to serve end users closer to Taiwan, they need to leverage other cloud providers, ending up with a multi-cloud deployment. Another geo-distributed deployment model is the hybrid cloud. The idea of hybrid cloud deployment is to combine the resources from one or more private data centers with the public cloud, which brings benefits from both sides. The users can keep sensitive workloads or data in a private data center and utilize the scalability and flexibility of the public cloud to run a larger number of applications and less sensitive workloads.

2.1.4 Cloud computing limitations

Cloud computing brings many benefits but it also presents challenges in different aspects. The first major issue for cloud computing is its energy consumption. A report from the International Energy Agency (IEA) shows that the electricity use for cloud data centers and transmission networks each is estimated up to 1.5% of the global use [68]. Moreover, the growing trend of data centers is driving an increase in energy usage by Artificial Intelligence (AI) [69]. Security and privacy are also significant topics that people care about. According to a survey from Cloud Security Alliance (CSA), the top threats are data breaches, weak identity, credential and access management, and insecure Application

Programming Interfaces (APIs) [70]. Another report shows that some of the threats are growing over the years, such as data breaches [71].

The enhancement of network accessibility and bandwidth, combined with the widespread proliferation of cloud data centers in different locations worldwide, has significantly reduced end user to cloud service latency. As a result, popular cloud services such as Facebook can be accessed within as little as 40 milliseconds round-trip latency [72]. The latency between end users and cloud services is an important topic for CSPs because lower latency brings a better user experience and thereby it impacts the profit of application owners. Amazon discovered that every extra 100 milliseconds of delay resulted in a 1% loss in sales [73]. Meanwhile, another study also demonstrated a similar outcome that an increase of 0.5 seconds in generating search results causes a 20% decrease in traffic [74]. Reducing network delays also enables the development of new latency-sensitive applications. For example, virtual reality and 360-degree video streaming require the total end-to-end latency, which includes both network transmission and application processing delays, to remain within 20 milliseconds [10].

The rapid development of the Internet of Things (IoT) enables the creation of smart home and improved urban services with the smart city. A forecast shows that the volume of data generated by IoT devices will reach 79.4 zettabytes by 2025 [75]. Meanwhile, to monitor and analyze the data collected from various IoT devices and sensors, the data must be sent to a cloud data center for real-time data processing. Long-distance data transmission of such large amounts of data over the network may eventually saturate the existing network links [14].

Although cloud users can take advantage of multi-cloud deployments to execute their applications in different data centers with multiple CSPs to reduce the latency and the volume of long-distance data transmission over the network, the centralization of large data centers means that they may remain physically distant from the end users [6].

As these limitations of cloud computing become more recognized, fog computing emerges as a solution to address their limitations.

2.2 Fog computing

Fog computing was proposed by Cisco in 2012 [15] as a widely distributed cloud-liked infrastructure to address the limitations of centralized cloud computing. The aim was to bridge the gap between end users/IoT devices and traditional cloud computing data

Table 2.1 – Differences between fog computing and cloud computing.

Characteristics	Fog computing	Cloud computing
Architecture	Decentralized fog nodes or clusters	Centralized data centers
Latency between users and nearest servers	Low	High
Distance from users	Close	Far
Bandwidth	Low	High
Computing capacity	Intermediate	High
Storage capacity	Intermediate	High
Use cases	Latency sensitive or IoT applications	General applications

centers by providing resources, including computing, storage, and networking services, that are closer to them. This design can fulfill the characteristics of IoT applications, such as geography distribution and low latency.

In 2017, the OpenFog Consortium Architecture Working Group published a white paper called "OpenFog Reference Architecture for Fog Computing" [76] to further consolidate the definition of fog computing. Then, in 2018, the IEEE adopted this standard [77]. In this standard, the authors state that "*Fog computing is a horizontal, system-level architecture that distributes computing, storage, control, and networking functions closer to the users along a cloud-to-thing continuum.*" Based on this definition, fog computing is seen as an extension of cloud computing where the computing resources are spread in different locations close to the data producers and users within a large geographical coverage. Fog computing should have all the characteristics of cloud computing, such as virtualization, service models, and efficiency. To conclude these two definitions, they share a similar concept: computing resources are provided between end users and cloud data centers to reduce the end-to-end latency of applications. We compare the main differences between fog computing and cloud computing in Table 2.1.

The main ideas of fog computing and edge computing are similar in that both of them address the latency issues between end users and cloud data centers. However, there are still two key differences between these two concepts. First, fog computing includes cloud computing to define a complete computing continuum, whereas edge computing excludes cloud computing as an independent architecture. Second, the structure of fog computing is hierarchical, combined with cloud, fog, and edge devices, while edge computing usually involves a flat structure with fewer layers [76].

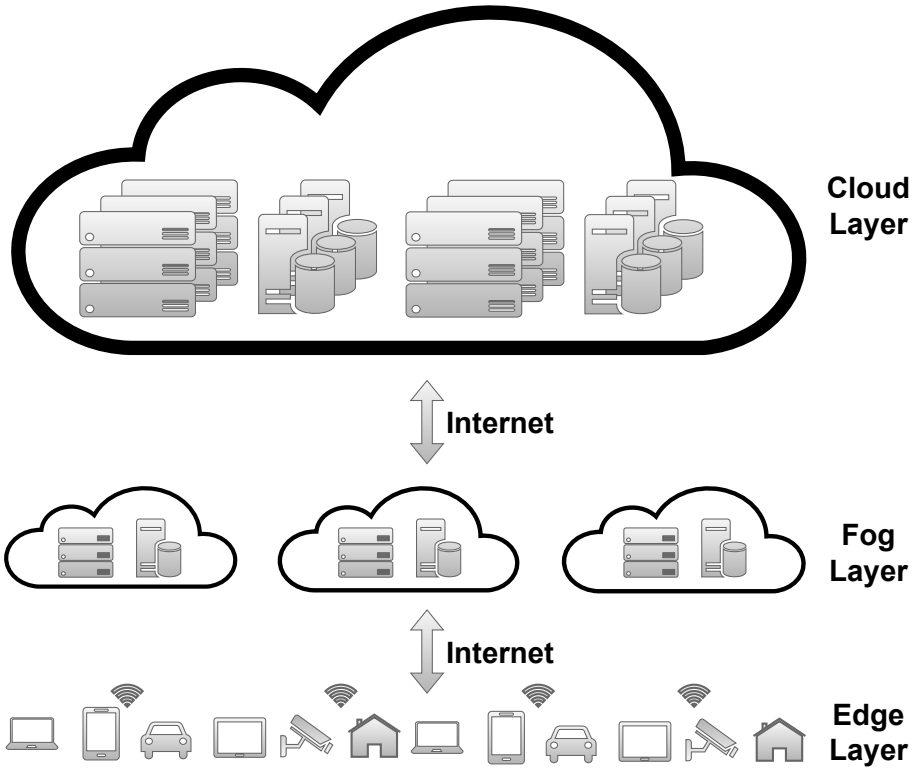


Figure 2.2 – High-level architecture of fog computing.

2.2.1 Fog computing architecture

Figure 2.2 illustrates a typical high-level fog computing architecture, which includes edge, fog, and cloud layers.

(1) **Edge layer:** This layer, also called the device layer, is the bottom layer in fog computing. The edge layer contains different types of devices, such as IoT devices, sensors, mobile phones, smart vehicles, and other endpoints that can connect to the Internet. These devices generate or collect data and send them to the upper layer for additional processing, such as analysis and decision-making. To send data, the methods of accessing the network are often wireless using protocols such as Wi-Fi, cellular network (LTE, 5G), and Long Range Wide Area Network (LoRaWAN) [78], [79].

(2) **Fog layer:** This layer is composed of computing resources located near the data sources outside traditional cloud data centers. These resources may potentially cover a very large region such as a country or a continent. Because each fog service is ex-

pected to serve a small number of users, fog servers often have limited computational power. For example, Raspberry Pi single-board computers are often used to build fog clusters [28], [80]–[82]. Moreover, additional devices can join the system and provide computational power, such as drones [83], and vehicles [84], which can also serve as part of the fog infrastructures. Applications which want to execute close to their end users will deploy in this layer.

(3) **Cloud layer:** This layer is made up of one or more powerful cloud data centers. Each data center consists of high-performance servers, high-speed network connections, and high-capacity storage. These computing resources can be used for applications that require reliability and high performance. Since these cloud data centers are physically far from end users, this layer can deploy the non-latency-sensitive part of fog applications.

2.2.2 Fog computing applications

The emergence of fog computing paradigms presents new opportunities to serve end users in close proximity and process data from sources outside traditional on-premise cloud data centers. The main idea of this design is to improve the user's Quality of Experience (QoE), especially for those applications that are not compatible with traditional cloud computing deployment [9].

Latency-sensitive applications require low end-to-end latency between users and applications. For example, humans have a low tolerance for delays or inconsistencies. Therefore, applications such as virtual reality and 360-degree video streaming need an end-to-end round-trip latency under 20 milliseconds [10]. Using a distributed fog infrastructure can reduce latency to meet the needs of latency-sensitive applications.

Fog computing resources located close to end-users can also bring benefits to applications such as video surveillance. These applications produce large volume of data and require broad network bandwidth to transmit these data to cloud data centers for processing. By using fog computing, processing can take place in closer infrastructures, which effectively minimizes the volume of data transmitted to the cloud.

Although web applications usually do not require ultra-low latency between end users and applications, as mentioned above, excessive latency for end users may reduce not only the profit but also the traffic. Improving the user experience is one of the main goals of fog computing. These fog infrastructures can be used for web content delivery and caching,

such as static items web pages, images and videos as well as application services [85], [86] to reduce end-to-end latency.

Several works have applied this distributed computing paradigm to different fields, including the fields of transportation [84], [87], smart city [88], [89], agriculture [90], [91], and entertainment [92].

2.2.3 Fog computing challenges

Fog computing addresses the limitations of cloud computing such as high latency and long-distance network transmission. However, to fully realize the fog computing potential, there remain challenges that need to be tackled. We discuss each point as follows:

(1) **Computing resource constraints:** Traditional cloud data centers are composed of many powerful servers, massive volumes of storage, and stable networks. Instead, fog infrastructures are equipped with potentially weak servers, small storage, and unstable networks. This challenge requires methods to handle workloads efficiently within these limits, making sure that users can still have a similar user experience to cloud computing. We discuss this constraint further in Section 2.3.

(2) **Scalability challenge:** Fog computing widely distributes fog infrastructures to strategic locations near the end users and data sources. These infrastructures may cover a very large region, such as a city, a country, or even a continent and may therefore be composed of a very large number of computing nodes. Maintaining a scalable fog platform demands a robust framework with a strong orchestrator to govern the fog resources as well as handle many functions such as deployment, scheduling, and monitoring, which are utilized by different users. The network traffic for management with a large number of infrastructures in the platform is also an issue that needs to be taken into account. Additionally, a fog platform may need to handle a large number of users or administrators. Therefore, it is crucial to deal with the multi-tenancy challenge that users may have from different departments or even organizations. We discuss this challenge in Section 2.4.

(3) **Security and privacy issues:** Cloud data centers implement different security measures to protect cloud users. Their centralized design makes it easier for CSPs to build their security ecosystem, which includes physical and network security since all the computing resources are in the same location. In centralized data centers, physical infrastructures can be secured by surveillance cameras and security guards,

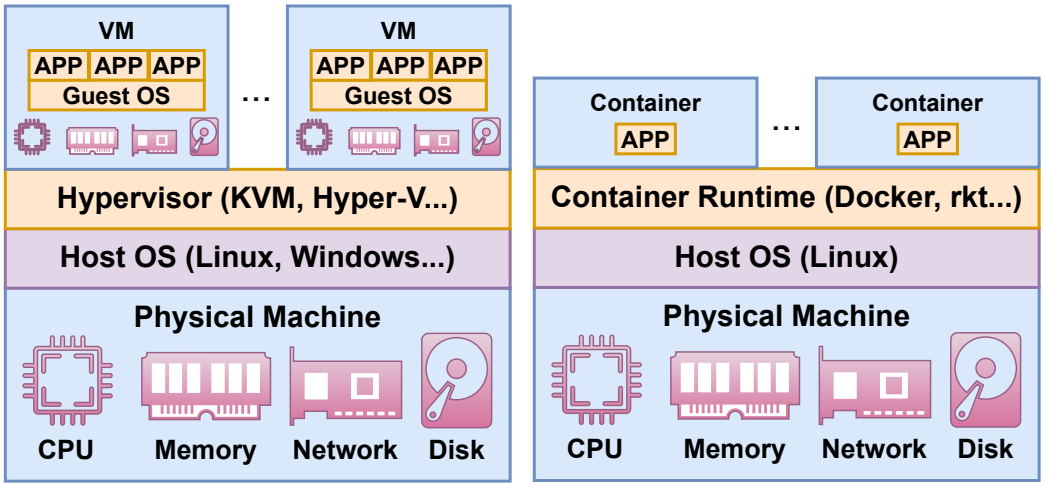
and the network can be protected by specifically designed machines, such as network firewalls and Intrusion Prevention Systems (IPSs) [93], [94]. In contrast, fog computing is a geographically distributed architecture, which makes security and privacy issues more challenging. Distributed fog infrastructures may be vulnerable to physical tampering or theft. Network protection in fog computing may require the use of distributed firewalls or IPSs rather than a single machine. To ensure data privacy, a centralized data center makes it easier to precisely locate data and computation. However, distributed fog resources within a single fog platform may be located in different countries with different data compliance regulations.

This thesis aims to address the second point mentioned above, which is the scalability challenge in geo-distributed fog computing. Scalability is a key concern in this environment that enables the fog platform to efficiently handle increasing infrastructures, workloads, users, and management traffic. As a result, designing frameworks to enable the evolution of scalable fog computing platforms is the main objective of this thesis.

2.3 Virtualization technology: virtual machines and containers

Virtualization is a key technology which enables multiple users to share computing resources in a physical server without interfering with each other. By doing this, virtualization improves the computing resource utilization and facilitates functions such as on-demand scaling [4]. There are two main types of virtualization for computing: hardware virtualization and operating system virtualization [3]. While numerous virtualization technologies also exist for network or storage, they fall outside the scope of this thesis and will not be discussed.

Hardware virtualization, also called hypervisor-based virtualization, is a technology to run multiple Virtual Machines (VMs) in a single physical machine [3], [95]. As shown in Figure 2.3(a), each of the VMs has its own virtual computing resources, operating system and applications, which provide the same user experience as a physical server. For example, users can install any applications in a VM or use any kind of operating system. To manage the computing resources of VMs, a hypervisor or Virtual Machine Monitor (VMM) is required. The hypervisor allocates resources to each VM and manages the scheduling between VM resources and the physical hardware. Although a VM has its



(a) Hardware virtualization (b) OS virtualization

Figure 2.3 – Comparison of hardware virtualization (a) and OS virtualization (b) architectures.

own virtual computing resources, the execution of computing tasks is carried out by the physical hardware. Well-known hypervisors include Microsoft Hyper-V [1], VMware ESXi [2], Xen [3], and VirtualBox [4].

Another type of virtualization is Operating System (OS) virtualization, also known as containerization. It is an approach that encapsulates an application, along with its essential libraries, dependencies, and execution environment, within a container image [3], [95]. Containers present the characteristic of portability and isolation. By using the same image, applications can have a consistent execution in different computing environments. Different from VMs, containers rely on a shared image registry, such as the Docker Registry [96], that allows users to easily download application images and run them in any environment equipped with the container runtime. Container isolation in the same execution environment is provided through the use of the Linux kernel features such as namespaces and control groups (cgroups) [97]–[99]. Well-known container technologies include Docker [5], LXC [6], Podman [7], and Containerd [8].

1. Microsoft Hyper-V - https://reurl.cc/dLYLAg
2. VMware ESXi - https://reurl.cc/1393qV
3. Xen - https://reurl.cc/g4d4EN
4. VirtualBox - https://reurl.cc/orjrAj
5. Docker - https://reurl.cc/eLrlXx
6. LXC - https://reurl.cc/RWNGvr
7. Podman - https://reurl.cc/VN9O8R
8. Containerd - https://reurl.cc/97lNVV

Although VMs have been the backbone of centralized cloud computing architecture, as discussed in section 2.2.2, the computing resources of fog infrastructures are often limited and geographically distributed compared to cloud data centers. Using VMs to deploy applications in this type of machine is very challenging. VM-based applications include applications related to software, dependencies, and data and contain an entire guest OS, which potentially consumes significant amounts of computing resources. This situation leads to high overhead for VMs, which might not be acceptable on constrained fog devices. Moreover, this overhead makes it hard to scale the number of VM-based applications per server. VM image sizes are usually large also due to the guest OS, often reaching into the gigabytes [85], [100]. Using VMs in this context may also cause an increase in the launch time of applications to serve users, which creates significant delays in fog computing, whereas the goal of fog is to provide a better user experience.

To effectively address the challenges related to performance and scalability encountered with VMs in fog computing environments, OS virtualization offers a more suitable solution. The main reason is the lightweight nature of containers, which do not need to contain a full guest OS, as illustrated in Figure 2.3(b). This characteristic ensures low consumption of computing resources, which is especially important in constrained fog infrastructures. Furthermore, the image sizes can be greatly reduced as they do not include a full OS. This reduction of the size allows containers to have faster startup time than VMs. Typically, it takes a matter of seconds or less for a container to start and turn ready status to handle user requests [85], [98]. Additionally, multiple solutions exist to further reduce the launch time of a container [81], [101], [102]. As a result, container-based applications have a fast launch time and are able to scale the number of applications to a higher degree across geo-distributed fog devices, which is crucial for adapting to the dynamic demands in fog computing scenarios. Many works use containers to enhance the resource utilization efficiency and reduce service latency within fog computing environments [16], [18], [19], [79], [103].

Because each physical machine may run large number of containers, fog infrastructures are requested to manage thousands or even tens of thousands of containers. Managing these amounts of containers spread across numerous machines in a fog computing environment requires a robust orchestrator to handle deployment, scaling, and networking seamlessly and efficiently. Various container orchestrators have been proposed such as Docker Swarm [21], Apache Mesos [22], and Kubernetes [23]. Among these orchestrators, Kubernetes has now become the most widely-used platform [24]–[26].

On the other hand, there is still no standardized platform for fog computing that can support all its specific requirements, especially regarding scalability. Therefore, this thesis considers containerization as a viable virtualization solution for geo-distributed fog computing environments. Owing to the fact that Kubernetes is currently the de-facto standard for cloud scenarios, we choose it as the preferred orchestrator for managing containers and clusters, aiming to explore ways to meet the scalability requirements of fog computing. Kubernetes has been adopted in many academic works for this type of projects [28], [29], [79], [104], [105].

2.4 Kubernetes

Kubernetes is an open-source container orchestrator, often called K8s, which was initially designed by Google. Later, Kubernetes was donated to the Cloud Native Computing Foundation (CNCF). In 2018, CNCF accepted Kubernetes at the graduated maturity level, which shows that Kubernetes is a stable and production-ready platform [27]. As an open-source project, it has attracted around 3 600 contributors and is very active in releasing new versions [106]. This level of activity can speed up bug fixes and feature deployment to accommodate the rapidly evolving needs of its users and can keep Kubernetes at the forefront of container orchestration technologies.

Kubernetes can be used to automate the deployment, scaling, and management of containerized applications in hybrid or public cloud infrastructures. It is deployed on a set of computing nodes that coordinate a cluster. Each Kubernetes cluster is composed of two roles: control plane and worker node. The control plane is in charge of managing worker nodes and containers. Typically, the control plane is deployed on a computing node in the Kubernetes cluster. To provide fault tolerance and high availability in a production environment, the platform administrator can also install multiple control planes that are distributed across several machines. Each cluster also needs at least one worker node to run the containers. Kubernetes relies on container runtimes such as containerd [9], CRI-O [10], Docker Engine [11], and Mirantis Container Runtime [12] on each node.

9. containerd - https://reurl.cc/orWqkg
10. CRI-O - https://reurl.cc/bD0yQE
11. Docker Engine - https://reurl.cc/G4Wl8A
12. Mirantis Container Runtime - https://reurl.cc/M4qGp3

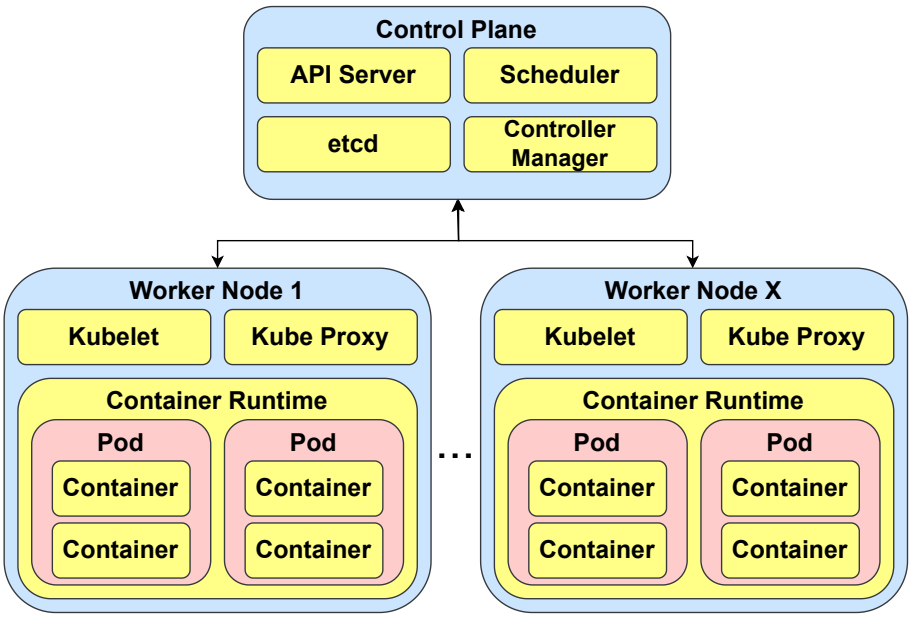


Figure 2.4 – Simplified Kubernetes Architecture.

2.4.1 Architecture

This section introduces the basic architecture of Kubernetes. It covers components, workloads, services, and the Custom Resource Definition (CRD).

Components

Figure 2.4 presents a simplified view of the Kubernetes architecture. The control plane includes four main components: API server, scheduler, etcd, controller manager.

(1) **API server:** The API server exposes the Kubernetes API for the system administrators and other components to send the requests and commands. Kubernetes API provides a standard interface to interact with the Kubernetes platform and perform tasks such as deploying and managing applications. In the Kubernetes API, a resource is an endpoint that stores API objects. For example, the built-in *Pod* resource has a collection of *Pod* objects. It is possible to launch multiple API servers and enable load balancing to distribute requests across these API servers in a Kubernetes cluster.

(2) **Scheduler:** The scheduler is responsible for monitoring newly initiated Pods (we discuss the concept of *Pod* in the next section) without an assigned worker node

and determining an appropriate worker node for their execution. A *pod* can be scheduled using different indicators such as affinity/anti-affinity specifications, policy constraints, and hardware requirements.

(3) **etcd:** This is a key-value data store with features such as strong consistency, distributed design, and high efficiency. The task for etcd is to store all of the objects from a Kubernetes cluster. In a production environment, etcd is usually deployed into an etcd cluster with an odd number of the etcd such as 3 or 5 to ensure high availability. This is because etcd is based on the Raft [107] algorithm to keep data consistent across multiple etcd, which requires a majority of members to accept updates to the etcd cluster status. Similar to Kubernetes, etcd is also a CNCF graduate project.

(4) **Controller manager:** The controller manager oversees the different controller processes in a Kubernetes cluster. Each controller monitors the status of their target Kubernetes resources through the API server and ensures that the actual status converges toward the desired status. The controller manager composes a series of controllers to manage different kinds of resources, including node controller, job controller, and deployment controller.

The second role in Kubernetes is worker nodes that are responsible for running the containerized applications that are placed by the scheduler in the control plane. Each worker node executes three main elements: Kubelet, Kube-proxy, and container runtime.

(1) **Kubelet:** The Kubelet is an agent in charge of manage the lifecycle of containers within a worker node. Kubelet continually *pulls* the latest information about desired status from the API server and ensures that the status of corresponding containers in the worker node match their PodSpec. A PodSpec is a definition of the intended behavior of the Pods. In addition to pulling information, Kubelet also periodically reports the status of the worker node and monitoring Pods to the API server.

(2) **Kube-proxy:** The kube-proxy is tasked with setting up network communication for Pods in the Kubernetes cluster. It runs in each worker node and manages the network rules to allow Pods to communicate with each other internally within the cluster, and with the external world.

(3) **Container runtime:** The container runtime is the component that allows Kubernetes to actually run containers. As motioned above, there are several runtimes for Kubernetes so that the platform administrator can choose their preferred runtime

to deploy in their cluster. Containerd is currently the more popular choice to run containers in Kubernetes [108].

Several add-ons can also be deployed in a Kubernetes cluster to provide cluster-level functionalities, including Domain Name System (DNS) services, network plugins, and comprehensive monitoring of container resources and worker nodes. The platform administrator can select from various projects CNCF hosts to fulfill specific needs. These projects can be installed easily through package managers such as Helm [13].

Workloads

In the Kubernetes ecosystem, the smallest execution unit is a Pod. A pod is defined as a group of one or more containers that are scheduled on the same worker node and managed by Kubernetes together. The most frequent use case uses a single container per pod [109]. In the case of containers that need to work together intensively, the users can run multiple containers in a Pod. The containers in a Pod share computing resources, dependencies, and volumes and can communicate with each other through a local host with different numbers.

To enable the both high availability and application scalability. Kubernetes offers different types of built-in resources for managing Pod replication efficiently, including Deployment, StatefulSet, and DaemonSet.

The *Deployment* resource is designed to manage stateless applications, such as web servers and API backends, that do not need to store the data generated by end users. The administrators describe the desired state of Pods in a YAML or JSON file. The Deployment controller then watches the running application state and continually ensures that the status between the desired and the observed status are consistent. For example, the administrators can change the desired number of Pods at any time. The Deployment Controller will handle the related operations, such as creating new Pods or deleting Pods. This can be used to quickly scale the number of replicas up or down.

The *statefulSet* resources are utilized to handle stateful applications, such as databases and data stream processing systems that require a stable identity or persistent storage for saving data. Statefulsets are similar to Deployment in that the administrators provide a Yaml or JSON files with desired specifications. Different from a Deployment, each pod in a StatefulSet can be deployed and scaled in a strict sequential order. This characteristic is particularly crucial for micro-services that have a specific process order.

13. Helm - https://reurl.cc/dLRKek

The *DaemonSet* is usually used when administrators want to deploy applications to all node, including control planes and worker nodes. The DaemonSet ensures these nodes in a Kubernetes cluster run the application, which is for instance suited for log collection and monitoring applications.

In addition to the standard built-in resources, Custom Resource Definitions (CRDs) can be used to create new types of resources, as we discuss next.

Custom Resource Definitions (CRDs) and custom Kubernetes controllers

Custom Resource Definition (CRD) is a powerful mechanism for extending the Kubernetes API. It allows administrators to create and manage new Custom Resources (CRs) that beyond the default of built-in resources such as Pods and Deployment. When new CRDs are deployed in a Kubernetes cluster, the API Server creates new RESTful paths and handles the whole lifecycle for these CRs. Users can interact with them in the same way as with built-in resources. CRs are increasingly being used to implement core functionalities within the Kubernetes framework [110].

The controller pattern in Kubernetes is to run a control loop and repeatedly track the current status of objects. The controller will continually make sure its actual state is the same as the desired status specified by the user. Without the custom controller designed to execute actual logic for the CR, the CR only stores in the Kubernetes cluster and only uses it to store and retrieve structured data. At this point, it is necessary to run a custom controller for managing the CR and to continuously synchronize and update the status of it.

The custom controller, also called operator, can be designed and similar to the pattern of Kubernetes by using Monitor, Analyze, Plan, Execute over a shared Knowledge (MAPE-K) principle or any domain-specific logic [111], [112]. This can be done by frameworks in different coding language such as Kubernetes Operator Pythonic Framework (Kopf) [14], Java Operator SDK [15], and kubebuilder [16].

2.4.2 Multi-tenancy

Kubernetes is designed for environments where all users trust each other. Similar to sharing the server with virtualization technology, a Kubernetes cluster can have many dif-

14. Kpof - https://reurl.cc/kr4aNx
15. Java Operator SDK - https://reurl.cc/YVM0Wn
16. kubebuilder - https://reurl.cc/OG2jpr

ferent users to deploy their applications and services thereby saving costs and simplifying administration.

Supporting multi-tenancy and isolating, the workloads of multiple tenants can however be realized by making each user deploy applications in a separate namespace. Then, using Role-Based Access Controls (RBAC) to restrict each user in their namespace and to scope security policies to specific namespaces [113]. This method is considered a "soft" form of isolation as all tenants share the same control plane, and appropriate configurations are required to isolate their data planes.

On the other hand, "hard" tenant isolation is often obtained by creating a separate Kubernetes cluster. It can be down by creating new Kubernetes clusters for each tenant. Both of control and data plane are totally separated from each others. However, the cost of launching multiple clusters is high and may hard to manage these clusters.

Another method for hard isolation is using virtual control plane in a single Kubernetes, each tenant can have their own control plane. For example, each tenant is able to stores metadata in their own etcd database, which prevents data leakage between tenants. The isolation of data plane in this case, there are two different design. First is these tenants share the same data plane and the applications are isolated by original container technology. Second is each virtual control plane have its own worker node in a Kubernetes cluster.

It can only isolate the data plane by reserving specific servers within a Kubernetes cluster for specific tenants. Each tenant use the same control plane and deploy the applications to their own worker nodes. We discuss multi-tenancy frameworks further in the section 3.2.

2.4.3 Monitoring

Monitoring is an essential functionality for modern computing systems to keep the system healthy and improve resource utilization. The demand for monitoring become higher due to the increasing complexity of systems, which monitor from only bare metal to have more objects such as VMs and containers.

Kubernetes is a complex system that is composed of many components, such as worker nodes, Kubernetes resources, networking, and controllers. Monitoring these objects not only allows real-time understanding of their status but also traces the history data for debugging or risk prediction. Moreover, monitoring data can be used to efficiently schedule applications on a set of available resources. However, monitoring the greater number of

components makes it more challenging. For example, it produces a huge amount of data and needs to be stored in the cluster, which requires large storage space. This amount of data makes it difficult for the administrator to find the right data and may waste the network bandwidth to transfer them in the Kubernetes cluster.

Prometheus

Prometheus is an open-source monitoring and alerting software. Prometheus is a graduate project, as announced by the CNCF. The graduated project maturity level shows that Prometheus is stable for production and has great potential to integrate with modern orchestrators such as Kubernetes. Prometheus has also become the monitoring solution for many research works [114]–[116].

Three main components in the Prometheus ecosystem, Prometheus server, exporters, and alertmanager. The Prometheus server is responsible for scraping, monitoring data, and storing them in a time-series database. The term of scrape is the action from Prometheus to fetch metrics from targets. The administrator can set a scrape interval to periodically pull the monitoring data. Scrape interval refers to the time duration between two consecutive data pulls from the designated targets or endpoints. For example, the default scrape interval is 60 seconds, which means the Prometheus server scrapes the metrics every 60 seconds. The administrator can query these stored metrics through Prometheus Query Language (PromQL). PromQL is designed for time series data that can do mathematical operations and includes data aggregation functions. Prometheus can also be integrated with visualization tools such as Grafana [17].

In the Prometheus design, it can only use HTTP to pull the metrics. For those applications or services that do not have native Prometheus metrics endpoint, the exporters are required to deploy for converting metrics from target systems into a format that Prometheus can pull. Prometheus officially maintains several exporters such as node-exporter [18].

Alarms have always been an important part of the monitoring system. In Prometheus, the tasks of scraping the data and sending the alarms are separated into two components. The administrator set the alerting rules in the Prometheus server and the server periodically evaluate the rules. Once the conditions of alerting rules are met, the Prometheus

17. Grafana - https://reurl.cc/eL0Y1L
18. Node-exporter - https://reurl.cc/rrzWbO

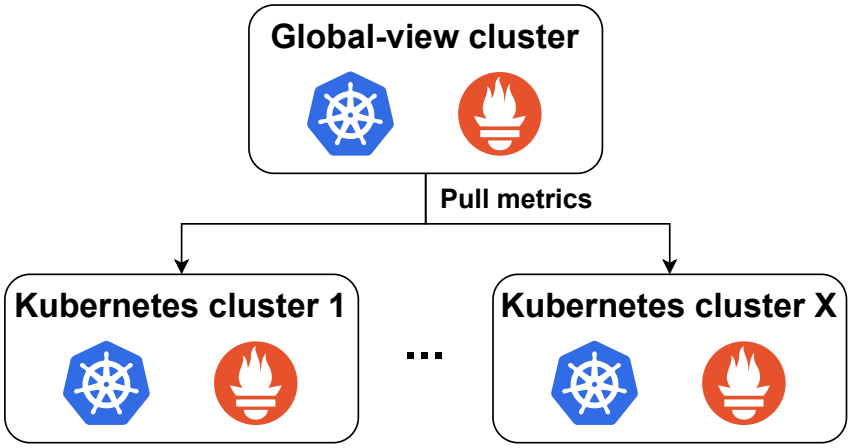


Figure 2.5 – Architecture of Prometheus Federation.

server will push alerts to the alertmanager. Then, alertmanager handle these alerts and send the alarm messages to the users.

In addition to deploy Prometheus in a single Kubernetes clusters, Prometheus also provides a function called Federation. As shown in Figure 2.5, this feature allows a Prometheus server to gather monitoring data from other Prometheus servers, which can therefore build a global-view cluster and scale up to monitor multiple Kubernetes clusters. By querying the monitoring metrics from the global-view cluster, the administrators can easily monitor the status of other Kubernetes clusters instead of accessing each cluster individually.

In section 3.3, we explore the limitation of Prometheus Federation for monitoring the multiple Kubernetes clusters in detailed.

2.4.4 Scalability

As mentioned before, a single Kubernetes cluster contains many components such as worker nodes and Pods. It will be very challenging for the control plane to directly operate all Kubernetes resources in a cluster [117]. To address this issue, Kubernetes chose the pull model to manage the cluster, where deploys Kubelet in each worker node as an agent. Kubelet gets the desired state of the pods and makes sure that the corresponding pods are the same as desired. These resources do not need the control plane to maintain and manage, which makes Kubernetes more scalable.

Nevertheless, a single Kubernetes cluster still has its size limitation to keep performance and stability. The Kubernetes official suggests that the size of a single Kubernetes cluster should not exceed a specified number limit according to the following criteria. For the number of Pods in a Kubernetes cluster, each worker node should not run over 110 Pods and 150,000 Pods in total across the whole cluster. Moreover, the number of worker nodes should be no more than 5,000 nodes [33]. This scalability of a single Kubernetes cluster may not be able to handle the size of a fog computing platform.

To overcome issues related to the scalability of a single Kubernetes cluster, deploying multiple clusters has emerged as a viable solution. The platform can be scaled by launching several clusters and managing them together. However, managing these clusters efficiently and easily is a difficult challenge. Therefore, it quickly becomes desirable to organize the platform as a "federation" of multiple independent clusters, each of which is in charge of its own resources and components. By doing so, the administrators are able to manage the resources of multiple independent clusters as a single homogeneous cluster.

2.4.5 Federation

The emergence of the Multicluster Special Interest Group (SIG) [19] from the Kubernetes community is to solve the issues of multi-cluster administration and application management in multiple Kubernetes clusters environment. The SIG proposes different Application Programming Interfaces (APIs) to deal with the challenges in this environment. For instance, the goal of the About API is to enable the identification of clusters within a *ClusterSet*(a group of clusters), and the purpose of the Work API is to deploy the workloads across different clusters within a *ClusterSet*.

In addition to the above APIs, Multicluster SIG presents a federation solution called Kubernetes Cluster Federation (KubeFed) [34], which provides application deployment and resource management in multiple Kubernetes cluster environments. The kubeFed platform is typically organized into one management cluster and several member clusters. The management cluster determines which member clusters will handle each newly deployed application. Users can manage multiple Kubernetes clusters from a single host cluster with the KubeFed control plane installed.

KubeFed extends Kubernetes with CRDs to offer abstractions such as *FederatedNamespaces* and *FederatedDeployments* for managing multi-cluster federated resources.

19. Multicluster Special Interest Group - https://reurl.cc/2zkaD6

It also introduces three concepts for these resources: *Template*, *Placement*, and *Override*. *Template* specifies the desired state of the federated resources across all member clusters. *Placement* defines the member cluster to be deployed for the federated resources. If this field is empty, KubeFed will not be distributed to any cluster. *Override* allows the user to customize the configuration for specific member clusters. For example, it can be used to change the number of replicas for a particular cluster. Based on these abstractions and concepts, the users can deploy their applications to different Kubernetes clusters with a number of Pods and where they should run that under KubeFed control.

However, the KubeFed project is now retired and is no longer maintained or under active development. Therefore, it is necessary to discover other federation solutions. We discuss this topic in section 3.1.

Chapter 3

State of the Art

This chapter discusses the state of the art related to this thesis. A single Kubernetes cluster is challenge to handle the fog computing platform that may cover a country or even across a continent. In this thesis, we apply the federation concept in Kubernetes clusters as a solution to build a scalable fog computing platform. Therefore, we first present the related work of the multi-cluster federation control plane in Section 3.1. Second, we review the multi-tenancy frameworks in Kubernetes orchestrator in Section 3.2. Finally, Section 3.3 explores the existing literature and tools in monitoring for fog computing environments.

Cloud computing, the prevailing computing paradigm, has received much attention from both academic and industrial communities worldwide. The researchers investigate across various critical topics, including resource utilization [118], pricing [119], and security [120]. To meet regional requirements for applications or to avoid a single vendor lock-in, multi-cloud is one of the viable solutions. The administrators of applications can launch multiple applications by using different cloud providers in various locations to serve end-users. However, the cloud data centers are often far from the end-users, which can lead to latency issues.

Fog computing extends the cloud computing concept with additional resources located closer to the end-users. It has received much attention from academia in the last few years. Many prior studies present different facets of fog/edge computing, including placement of jobs and services [121], service caching [85], [86], seamless application migration [122], and supporting data stream processing [123]. These works are based on a single distributed cluster, which will necessarily face the scalability problem. To handle this issue, we are now witnessing an increasing adoption of geo-distributed multi-cluster deployments. Some works focus on job scheduling [124], whereas others address resources management [125] and fault prediction [126].

The rise of multi-cluster federations for fog computing has spurred the development of various control plane solutions to manage and orchestrate federated Kubernetes environments.

3.1 Federation of multi-cluster Kubernetes control plane

xxxxx

In the KubeFed, the workloads can be propagated to the different clusters with two fields in a YAML file of a federated resource, *spec.placement.clusters* and *spec.placement.clusterSelector*. The administrators can select one or more clusters that the clusters should run the applications with the field of *spec.placement.clusters* in a YAML file of a federated resource. The users can also utilize the field of *spec.placement.clusterSelector* to choose the clusters that have a specific label. Note that if the field of *spec.placement.clusters* is provided, then the *spec.placement.clusterSelector* will be ignored to make the priority of manual selection higher than others such as to match the labels. Moreover, KubeFed propagates resources to target clusters without any prior checks or validations. This design presents significant challenges in automatically managing workloads in a large-scale environment. The KubeFed project is now retired and no longer maintained nor under active development.

In the context of Kubernetes Cluster Federation (KubeFed) for fog computing, a demonstration [127] shows that using federation with Kubernetes clusters for fog computing is better than a single Kubernetes. The authors create two scenarios: A single Kubernetes cluster manages the entire infrastructure with worker nodes across different regions. The second scenario is to separate these infrastructures into independent Kubernetes clusters and federate them together by utilizing the KubeFed. The results show that KubeFed in fog computing is stronger at handling network faults. In this demonstration, they also note that the deployment time for the applications increment is more than linear matter in a single cluster or federated multiple clusters. The federation framework is based on KubeFed, which therefore will have the same drawbacks as KubeFed.

Multi-cluster Kubernetes (mck8s) [38] addresses the limitations of KubeFed and proposes resource-based automated placement, multi-cluster horizontal pod auto-scaling, and bursting of container-based applications in geo-distributed cluster federations. The resource-based automated placement can reduce the pending pods from 65% to 6% compared to KubeFed with the real-world dataset, Google cluster trace [128]. To avoid com-

puting resources are not enough for a fixed number of managed clusters, mck8s also provides a Cluster Provisioner and Cluster Autoscaler (CPCA) to create k8s clusters from public cloud data centers and automatically join the federation. Moreover, mck8s integrate several open-source tools to provide additional functionality, such as multi-cluster network discovery, global load balancing, and monitoring. However, the authors note that as the quantity of clusters rises, the time required for deployment also expands. This situation makes mck8s face a scalability challenge for a large federation composed of many Kubernetes clusters.

Both KubeFed and mck8s rely on the *Push* method to deploy applications to member clusters, which can be unstable in the presence of transient network failures [35] particularly in fog computing scenarios where network reliability can be a concern. In contrast, the pull-based model offers an alternative approach. Each of the member clusters periodically checks from the API server of management for updates. Once member clusters identify the changes, such as increasing the number of replicas for application, these clusters independently apply the necessary updates. Therefore, pull-based methods are usually considered more robust and scalable.

To avoid a single point failure and address the scalability of KubeFed for edge cloud use cases, researchers introduce a vision of decentralized Kubernetes Federation Control Plane [129]. They leverage distributed federated databases with conflict-free replicated data types (CRDTs) to maintain the status of resources across Kubernetes clusters. To suit this design, they also propose a distributed algorithm to schedule the resources. However, a notable drawback of the approach is the complexity and potential overhead introduced by managing the distributed database while ensuring consistency across a large number of Kubernetes clusters for the reason of a flat architecture.

Open Cluster Management (OCM) [41] presents a different management model inspired by the original principles of Kubernetes. OCM separates multi-cluster operations into two phases: computation/decision (management cluster) and execution (member cluster). The management cluster stores prescriptions (i.e., the desired state of applications that users want to deploy in member clusters), whereas member clusters periodically actively *Pull* the latest prescriptions from the management cluster, ensuring that applications in the member cluster are consistent with the expected state. This design reduces the load on the management cluster and makes OCM more scalable than a push-based design. The *placement* module in OCM provides the ability to schedule the workloads to a set of member clusters dynamically. The process involves two main phases: Predicate

and Prioritize, where clusters are selected based on hard requirements and then ranked based on soft requirements such as computing resources. Moreover, OCM also has the addon framework for placement to extend the multi-cluster scheduling capabilities. For example, users can implement a customized score provider to rank the member clusters and schedule the workloads. In addition, OCM is a sandbox project of the Cloud Native Computing Foundation (CNCF), which shows great potential for managing multiple Kubernetes clusters.

The Karmada project [130] offers automation solutions for managing applications across multiple Kubernetes clusters. In contrast to other solutions by extending with CRDs, it simplifies multi-cluster application management by providing custom control plane components. It therefore introduces the Karmada API server, Karmada controller manager, and Karmada scheduler and other components to provide a single point of control. The detail information for main component is listed below:

— Karmada API server: This component can serve as both the interface of the control plane for Karmada and for its underlying Kubernetes. To provide an operation that is the same as the original Kubernetes cluster, they build it based on the implementation of the Kube API server.

— Karmada controller manager: This component includes various controllers for different proposes that Karmada needs. The goal of this controller is to monitor Karmada resource objects and communicate with the API servers of the underlying clusters to create standard Kubernetes resources. Users can, based on their requirements, deploy the related controllers since not every default controllers within the kube-controller-manager is essential for Karmada's operation.

— Karmada scheduler: This component is to schedule the Kubernetes resources to the member clusters. Karmada provides different propagation policies. For example, the users can deploy their workloads to specified set of target member clusters by label or cluster name. The users can also based on taints and tolerations of clusters

Karmada supports both *Push* and *Pull* approaches to manage multiple clusters and applications. According to the Karmada authors, the *Push* method is most suitable for deployment in public cloud environments, while the *Pull* model is better for private cloud and edge-related scenarios [131]. The main different points from this report between *Push* and *Pull* methods are performance and security. The performance can be improved because of the decentralized management by *Pull* agent within each member cluster to release the loading pressure of the centralized control plane. In terms of the security of

Pull approach, each *Pull* agent is isolated in different member clusters, which manage its resources independently, avoiding direct external manipulations. Karmada is an incubation project of CNCF.

An important remark is that the five solutions discussed above make the same underlying assumption that all resources in a federation belong to a single administrative domain. They are therefore not designed to support isolation between workloads being deployed by different management clusters. This results in the resources possibly conflicting if multiple management clusters deploy resources to the same member cluster with the same name.

The federation framework that supports some form of multi-tenancy is Liqo [132]. Liqo creates virtual node resources in the management cluster using a so-called Virtual Kubelet [133]. After a pod is scheduled to a virtual node, the corresponding virtual kubelet creates a twin-pod object in the member cluster for actual execution. This design ensures that all behaviors are identical to those in a single Kubernetes cluster when the administrator issues commands in the management cluster. However, Liqo relies on a *Push* model, which may limit its scalability. Liqo addresses multi-tenancy by using different namespaces in member clusters to isolate resources created by each management cluster. It however limits itself to flat namespaces to isolate the resources created by different management clusters, which makes it impossible to further isolate workloads produced by different users within a single management cluster. Moreover, all management clusters send their requests to a single shared API server in the member cluster and store the related data in a single shared database, representing only a soft version of workload isolation between federations.

3.2 Multi-Tenancy Frameworks in Kubernetes

By default, Kubernetes is designed for environments where all users trust each other. Supporting multi-tenancy and isolating the workloads of multiple tenants can however be realized by making each user deploy applications in a separate namespace and by using Role-Based Access Controls (RBAC) to restrict each user in their namespace and to scope security policies to specific namespaces [113]. This method is considered a soft form of isolation as all tenants share the same control plane, and appropriate configurations are required to isolate their data planes. Stronger forms of isolation require the different techniques described next.

The Kiosk open-source multi-tenancy framework is developed by Loft Labs that uses namespaces to provide isolated execution environments for tenant applications [134]. Kiosk defined different roles to interact with Kubernetes. *Cluster Admin* allows to operate and manage cluster-wide resources such as RBAC and others roles of Kiosk. Each tenant maps to an *account*, and each *account* is scoped to a single namespace called a *space* in Kiosk. This framework also provide *AccountQuota*, which is similar to ResourceQuotas [135] function in Kubernetes, to limit resources usage for *account*. Contrary to regular Kubernetes namespaces, each Kiosk user can only see or interact with resources in their own *space*. However, Kiosk adopt the flat namespace approach may make it difficult for cluster administrators to manage these namespaces. The project appears to be inactive development, with the most recent release was on 23, November, 2021 [1].

Another project also leverages flat namespaces as their multi-tenancy solution, named Capsule [136]. Capsule improves the original Kubernetes namespace management, which groups multiple Kubernetes namespaces into a *tenant*. This design allows cluster administrators to easier manage multiple namespaces at once, rather than setting policies, such as ResourceQuota and RBAC, for each namespace. Although this method addresses the difficult management issue, it is still a flat namespace design that makes it chanllenge to further scale to the next levels.

Extending the flat Kubernetes namespace structure may be realized using the Hierarchical Namespace Controller (HNC) **hnc**. Hierarchical namespaces may for example allow one to apply similar policies to multiple namespaces. However, role bindings operate at the namespace level, and each role binding must be created individually for each namespace. HNC proposes sub-namespaces to address this problem. The administrator can create child namespaces of another namespace, and the lifecycle of this sub-namespace is bound to its parent. This design can reduce the complexity of namespace management. However, this remains a soft form of tenant isolation.

EdgeNet **edgenet** proposes a hierarchical namespace architecture with a sub-namespace mechanism as a way to implement multi-tenancy in federations of multiple Kubernetes clusters. The system comprises a Federation Manager and a Custom Resource (CR) called Selective Deployment. However, the Federation Manager is still under development. The hierarchical namespaces use a single control plane and data store for all tenants, which, again, provides only soft isolation properties.

1. The date are checked on 21, March, 2024

Hard tenant isolation is often obtained by creating a separate Kubernetes cluster for each tenant or by reserving specific servers within a Kubernetes cluster for specific tenants [113]. However, these simple solutions contradict our objective of maximizing fog resource utilization by sharing the available servers between large numbers of tenants.

Finally, Virtual Kubernetes Clusters (vCluster) project is a fully functional virtual cluster that runs on top of other Kubernetes cluster **vcluster**. Each vCluster has its own control plane and schedules all workloads into a single namespace of the host cluster. Each vCluster has its own API server and data store, which provides a strong form of isolation between the tenants of different virtual clusters. Furthermore, the Kubernetes community has officially certified vCluster project as a compliant Kubernetes distribution **vcluster-certified**.

3.3 Monitoring for fog computing environments

The main purposes of geo-distributed resource monitoring are to track the resource usage of the computing nodes, especially in potentially resource-restricted and unstable environments. A number of open source and commercial monitoring tools such as DARGOS **DARGOS:2013**, Zabbix **Zabbix:2010**, PCMONS **PCMONS:2011**, JCatascopia **JCatascopia:2014**, and Nagios **Nagios:2022** were developed to suit cloud computing requirements. However, they are not considered appropriate for geo-distributed fog computing environments **COSTA:2022**, **Emon:2020**. On the other hand, some authors present monitoring solutions and architectures designed with the specific constraints of fog computing in mind.

PyMon **PyMon:2017** is an extension to the lightweight open-source software Monit **monit:2022**. PyMon provides a monitoring solution for fog environments specially designed to run on ARM-based single-board computers. However, its scalability was not evaluated, which may not suit large-scale scenarios **COSTA:2022**.

FMonE **fmone:2018** also aims to address in fog environments with an independent, stand-alone monitoring solution. The authors evaluate their work using up to 78 Virtual Machines (VMs). However, such a number remains very far from the scale at which global fog platforms are expected to operate.

FogMon **fogmon:2021** proposes a Peer to Peer (P2P) monitoring architecture which deploys an agent in each member node called “Follower.” Followers report the hardware-based metrics to the “Leader” node. Each Follower node is linked to a single Leader node

and runs in a classic client-server model. To reduce the network traffic between Followers and Leaders, FogMon adopts a solution similar to Acala's deduplication, where Followers only send data with the average or variance value greater than a threshold compared to the last sent. AdaptiveMon **p2pfog:2022** extends FogMon with two additional functions: Indicators Selection and Rate. Indicators Selection decreases the number of metrics, whereas Rate adjusts the frequency of metrics reported from Follower to Leader. However, these two solutions are not designed for cluster federation environments where nodes are not considered individually but cluster by cluster.

The most popular monitoring tool is Prometheus **prom:2022**. Since 2016, Prometheus has been accepted by the Cloud Native Computing Foundation (CNCF) as a "graduated" project, which shows its great potential in conjunction with Kubernetes. At the same time, many works use Prometheus as a basis for system monitoring **prom4:2022**, **prom5:2020**, **prom2:2022**, **prom:2021**, **prom6:2020**, **prom3:2021**.

Prometheus provides a function called "Federation" which allows a Prometheus server to collect the metrics from other Prometheus servers. A common use case is building a global-view Prometheus server which scrapes and stores the monitoring data from other Prometheus servers. Two levels of the federation are instance-level drill-down and job-level drill-down. In Prometheus terminology, an *instance* is an endpoint that the user can scrape from and a *job* is a collection of *instances* with the same purpose. Prometheus Federation is often used to monitor systems in geo-distributed environments **livingFog:2022**, **edgecloud:2021**, **mck8s:2021**, **promfed1:2018**, **HPC:2022**.

However, Prometheus Federation features three limitations that generate the high network traffic highlighted in Section **??** and make it unsuitable for our purposes. First, the highest scrape level of Prometheus Federation is job-level, and it uses the *match* mechanism to select the series of metrics. For example, the operator can write job="node-exporter" in a federation server's configuration file to scrape the metrics that match this label from the target Prometheus servers. It results in scraping the matching metrics that are all the nodes [2] in the target cluster when job="node-exporter" is set. This design is suitable for backing the metrics for high availability purposes but not fitting for the management cluster to manage the federated clusters. It wastes the network bandwidth to transmit and disk resources to save the same node metrics in the management cluster. Second, Prometheus Federation will append all original labels in each metric when a Prometheus server scrapes from the target Prometheus server to identify where the metric

2. We assume all nodes in all clusters have installed node-exporter and labeled job="node-exporter".

comes from. However, all original labels are unnecessary for recognition, and the scheduler may not need this detailed information to make the decision. Furthermore, the labels are attached before the metrics transmission, which increases the cross-cluster network traffic. The third point is that Prometheus Federation collects the monitoring data at a fixed periodicity. The system will therefore scrape all the metrics even if some of the metrics values did not change, which once again will waste network bandwidth.

Prometheus also supports a feature called "recording rules" which is similar to Acala's metrics aggregation. Using it, one can pre-aggregate selected metrics, store the results in member clusters, and scrape them from other Prometheus servers with appropriate labels. However, recording rules in Prometheus need to be defined manually for each metric in each member cluster, which is error-prone and may increase the deployment and configuration cost in large-scale environments. Moreover, Prometheus does not provide metrics deduplication so it reports data to the global view cluster periodically, regardless of whether the value has changed since the previous scraping period.

To overcome these monitoring challenges, we base our work on Prometheus and introduce Acala. Acala automatically aggregates the metrics whose metric name and labels are identical in different servers, which reduces the cross-cluster network traffic as well as the deployment and configuration cost. It also deduplicates metrics values and thereby avoids transferring unchanged values over and over again.

Chapter 4

Efficient Monitoring in Geo-Distributed Cluster Federations

4.1 rerer

Conclusion

Première section de l'intro

Bibliography

[1] IBM, *What Are the Benefits of Cloud Computing?*, https://reurl.cc/WR70R7, cited Mar 2024.

[2] M.-G. Avram, « Advantages and Challenges of Adopting Cloud Computing from an Enterprise Perspective », *Elsevier Procedia Technology*, vol. 12, 2014.

[3] O. Laadan and J. Nieh, « Operating System Virtualization: Practice and Experience », *in ACM Annual Haifa Experimental Systems Conference*, 2010.

[4] Y. Xing and Y. Zhan, « Virtualization and Cloud Computing », *in Springer Future Wireless Networks and Information Systems*, 2012.

[5] Google, *Cloud Locations*, https://reurl.cc/Gjd0LG, cited Apr 2024.

[6] D. Bermbach, F. Pallas, D. G. Pérez, *et al.*, « A Research Perspective on Fog Computing », *in Springer ICSOC Workshop on IoT Systems Provisioning and Management for Context-Aware Smart Cities*, 2018.

[7] F. Blair, *The Digital Forecast: 40-Plus Cloud Computing Stats and Trends to Know in 2023*, https://reurl.cc/8v30j7, cited Apr 2024.

[8] Z. Wu and H. V. Madhyastha, « Understanding the Latency Benefits of Multi-Cloud Webservice Deployments », *ACM SIGCOMM Computer Communication Review*, vol. 43, 2013.

[9] C. Avasalcai, I. Murturi, and S. Dustdar, « Edge and Fog: A Survey, Use Cases, and Future Challenges », *Wiley Fog Computing: Theory and Practice*, 2020.

[10] K. Mania, B. D. Adelstein, S. R. Ellis, and M. I. Hill, « Perceptual Sensitivity to Head Tracking Latency in Virtual Environments with Varying Degrees of Scene Complexity », *in ACM Symposium on Applied Perception in Graphics and Visualization*, 2004.

[11] P. Mark, S. Jason, and T. Christopher, *What's New With the Internet of Things?*, https://reurl.cc/va5Zdy, cited Apr 2024.

[12] L. S. Vailshery, *Number of Internet of Things (IoT) Connected Devices Worldwide from 2019 to 2030*, https://reurl.cc/2YrL69, cited Apr 2024.

[13] L. Hou, S. Zhao, X. Xiong, *et al.*, « Internet of Things Cloud: Architecture and Implementation », *IEEE Communications Magazine*, vol. 54, 2016.

[14] P. Bellavista, J. Berrocal, A. Corradi, *et al.*, « How Fog Computing Can Support Latency/Reliability-Sensitive IoT Applications: An Overview and a Taxonomy of State-of-the-Art Solutions », *Wiley Fog Computing: Theory and Practice*, 2020.

[15] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, « Fog Computing and Its Role in the Internet of Things », *in ACM SIGCOMM Workshop on Mobile Cloud Computing*, 2012.

[16] P. Bellavista and A. Zanni, « Feasibility of Fog Computing Deployment Based on Docker Containerization Over RaspberryPi », *in IEEE International Conference on Distributed Computing and Networking*, 2017.

[17] Raspberry Pi, *Raspberry Pi for Industry*, https://reurl.cc/oRgjn3, cited Apr 2024.

[18] J. Luo, L. Yin, J. Hu, *et al.*, « Container-Based Fog Computing Architecture and Energy-Balancing Scheduling Algorithm for Energy IoT », *Elsevier Future Generation Computer Systems*, vol. 97, 2019.

[19] A. Omar, B. Imen, S. M'hammed, B. Bouziane, and B. David, « Deployment of Fog Computing Platform for Cyber Physical Production System Based on Docker Technology », *in International Conference on Applied Automation and Industrial Diagnostics*, 2019.

[20] R. Mahmud and A. N. Toosi, « Con-Pi: A Distributed Container-Based Edge and Fog Computing Framework », *IEEE Internet of Things Journal*, vol. 9, 2021.

[21] Docker, *Swarm Mode Overview*, https://reurl.cc/XqdbOg, cited Mar 2024.

[22] B. Hindman, A. Konwinski, M. Zaharia, *et al.*, « Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center », *in USENIX Symposium on Networked Systems Design and Implementation*, 2011.

[23] The Kubernetes Authors, *Kubernetes*, https://reurl.cc/L48ovL, cited Mar 2024.

[24] S. Conway, *Survey Shows Kubernetes Leading As Orchestration Platform*, https://reurl.cc/77gA2b, cited Mar 2024.

[25] Datadog, *9-Insights On Real-World Container Use*, https://reurl.cc/G4W4Rv, cited Mar 2024.

[26] Cloud Native Computing Foundation, *CNCF Survey 2019*, https://reurl.cc/j3LY3p, cited Mar 2024.

[27] Cloud Native Computing Foundation, *Kubernetes Maturity Level*, https://reurl.cc/eL1LrW, cited Mar 2024.

[28] C. Wöbker, A. Seitz, H. Mueller, and B. Bruegge, « Fogernetes: Deployment and Management of Fog Computing Applications », *in IEEE/IFIP Network Operations and Management Symposium*, 2018.

[29] Z. Wan, Z. Zhang, R. Yin, and G. Yu, « KFIML: Kubernetes-Based Fog Computing IoT Platform for Online Machine Learning », *IEEE Internet of Things Journal*, vol. 9, 2022.

[30] J. Santos, T. Wauters, B. Volckaert, and F. De Turck, « Towards Network-Aware Resource Provisioning in Kubernetes for Fog Computing Applications », *in IEEE Conference on Network Softwarization*, 2019.

[31] J. Santos, T. Wauters, B. Volckaert, and F. De Turck, « Resource Provisioning in Fog Computing: From Theory to Practice », *MDPI Sensors*, vol. 19, 2019.

[32] A. Prountzos and E. G. Petrakis, « DeFog: Adaptive Microservice Scheduling on Kubernetes Clusters in Cloud-Edge-Fog Infrastructures », *in Springer International Conference on Advanced Information Networking and Applications*, 2024.

[33] The Kubernetes Authors, *Considerations for Large Clusters*, https://reurl.cc/ezvrYK, cited Apr 2024.

[34] The KubeFed Authors, *KubeFed: Kubernetes Cluster Federation*, https://reurl.cc/9RGa8x, cited Mar 2024.

[35] K. Manaouil and A. Lebre, « Kubernetes and the Edge? », Inria Rennes – Bretagne Atlantique, Tech. Rep. RR-9370, 2020.

[36] B. Costa, J. Bachiega Jr, L. R. Carvalho, M. Rosa, and A. Araujo, « Monitoring Fog Computing: A Review, Taxonomy and Open Challenges », *Elsevier Computer Networks*, vol. 215, 2022.

[37] S. K. Battula, S. Garg, J. Montgomery, and B. Kang, « An Efficient Resource Monitoring Service for Fog Computing Environments », *IEEE Transactions on Services Computing*, vol. 13, 2019.

[38] M. A. Tamiru, G. Pierre, J. Tordsson, and E. Elmroth, « mck8s: An Orchestration Platform for Geo-Distributed Multi-Cluster Environments », *in IEEE International Conference on Computer Communications and Networks*, 2021.

[39] C.-K. Huang and G. Pierre, « Acala: Aggregate Monitoring for Geo-Distributed Cluster Federations », *in ACM/SIGAPP Symposium on Applied Computing*, 2023.

[40] The OpenStreetMap Contributors, *OpenStreetMap Provides Map Data for Thousands of Websites, Mobile Apps, and Hardware Devices*, https://reurl.cc/5v5O56, cited Apr 2024.

[41] The Open Cluster Management Authors, *Open Cluster Management*, https://reurl.cc/qrAYxp, cited Mar 2024.

[42] Loft Labs, *vCluster*, https://reurl.cc/kOE19q, cited May 2024.

[43] The Prometheus Authors, *Overview*, https://reurl.cc/Vz1DgA, cited May 2024.

[44] The Prometheus Authors, *Federation*, https://reurl.cc/mM3o2W, cited May 2024.

[45] E. Shein, *The Most Important Cloud Advances of the Decade*, https://reurl.cc/A4zqO3, cited Mar 2024.

[46] J. Surbiryala and C. Rong, « Cloud Computing: History and Overview », *in IEEE Cloud Summit*, 2019.

[47] A. Regalado, *Who Coined Cloud Computing*, https://reurl.cc/E4amxg, cited Mar 2024.

[48] Amazon, *Announcing Amazon S3 - Simple Storage Service*, https://reurl.cc/prV8al, cited Mar 2024.

[49] Amazon, *Announcing Amazon Elastic Compute Cloud (Amazon EC2) - Beta*, https://reurl.cc/Z9z8Zp, cited Mar 2024.

[50] Microsoft, *Windows Azure General Availability*, https://reurl.cc/E4aWVa, cited Mar 2024.

[51] Google, *App Engine 1.6.0 Out of Preview Release*, https://reurl.cc/WRVyO5, cited Mar 2024.

[52] F. Richter, *Amazon Maintains Cloud Lead as Microsoft Edges Closer*, https://reurl.cc/Xqd44e, cited Mar 2024.

[53] Eurostat, *Cloud Computing - Statistics on the Use By Enterprises*, https://reurl.cc/RWE56Z, cited Mar 2024.

[54] J. E. Smith and R. Nair, « The Architecture of Virtual Machines », *IEEE Computer*, vol. 38, 2005.

[55] C. Pahl, A. Brogi, J. Soldani, and P. Jamshidi, « Cloud Container Technologies: A State-of-the-Art Review », *IEEE Transactions on Cloud Computing*, vol. 7, 2017.

[56] A. Badkar, *7 Astonishing Benefits of Virtualization in Cloud Computing*, https://reurl.cc/D4EqDe, cited Mar 2024.

[57] R. Beri and V. Behal, « Cloud computing: A survey on cloud Computing », *Foundation of Computer Science International Journal of Computer Applications*, vol. 111, 2015.

[58] GlobalDots, *13 Key Cloud Computing Benefits for Your Business*, https://reurl.cc/77mKY1, cited Mar 2024.

[59] P. Mell and T. Grance, « The NIST Definition of Cloud Computing », 2011.

[60] N. Subramanian and A. Jeyaraj, « Recent Security Challenges in Cloud Computing », *Elsevier Computers and Electrical Engineering*, vol. 71, 2018.

[61] Q. Zhang, L. Cheng, and R. Boutaba, « Cloud Computing: State-of-the-Art and Research Challenges », *Springer Journal of Internet Services and Applications*, vol. 1, 2010.

[62] Amazon, *Amazon EC2*, https://reurl.cc/WRoyoO, cited Mar 2024.

[63] Google, *Google Cloud Storage*, https://reurl.cc/Z9YVO3, cited Apr 2024.

[64] Azure, *Azure Virtual Network*, https://reurl.cc/YVZqla, cited Apr 2024.

[65] Google, *Google App Engine*, https://reurl.cc/zlko26, cited Mar 2024.

[66] Google, *Google Workspace*, https://reurl.cc/g4oWep, cited Mar 2024.

[67] J. Hong, T. Dreibholz, J. A. Schenkel, and J. A. Hu, « An Overview of Multi-Cloud Computing », *in Springer AINA Workshop on Multi-Clouds and Mobile Edge Computing*, 2019.

[68] I. E. Agency, *Data Centres and Data Transmission Networks*, https://reurl.cc/qr647R, cited Mar 2024.

[69] N. S. Malik and Bloomberg, *With AI Forcing Data Centers to Consume More Energy, Software That Hunts for Clean Electricity Across the Globe Gains Currency*, https://reurl.cc/WR7M55, cited Mar 2024.

[70] C. S. Alliance, *The Treacherous 12: Cloud Computing Top Threats in 2016*, https://reurl.cc/E4evD0, cited Mar 2024.

[71] A. Chaudhary, *Cloud Security Threats to Watch Out for in 2023: Predictions and Mitigation Strategies*, https://reurl.cc/g41olN, cited Mar 2024.

[72] B. Schlinker, I. Cunha, Y.-C. Chiu, S. Sundaresan, and E. Katz-Bassett, « Internet Performance from Facebook's Edge », *in ACM Internet Measurement Conference*, 2019.

[73] Gigspaces, *Amazon Found Every 100ms of Latency Cost Them 1% in Sales*, https://reurl.cc/E4Nzzk, cited Mar 2024.

[74] G. Linden, *Geeking with Greg: Marissa Mayer at Web 2.0.* https://reurl.cc/M4mdXp, cited Mar 2024.

[75] Statista, *Data Volume of Internet of Things (IoT) Connections Worldwide in 2019 and 2025*, https://reurl.cc/80Az0o, cited Mar 2024.

[76] OpenFog Consortium Architecture Working Group, *OpenFog Reference Architecture for Fog Computing*, https://reurl.cc/j3EQ8Z, cited Mar 2024.

[77] IEEE Standard Association, « IEEE Standard for Adoption of OpenFog Reference Architecture for Fog Computing », *IEEE Std 1934-2018*, 2018.

[78] M. Chiang and T. Zhang, « Fog and IoT: An Overview of Research Opportunities », *IEEE Internet of Things Journal*, vol. 3, 2016.

[79] D. Battulga, M. Farhadi, M. A. Tamiru, L. Wu, and G. Pierre, « LivingFog: Leveraging Fog Computing and LoRaWAN Technologies for Smart Marina Management (Experience Paper) », *in Conference on Innovation in Clouds, Internet and Networks*, 2022.

[80] A. Ahmed, H. Arkian, D. Battulga, *et al.*, « Fog Computing Applications: Taxonomy and Requirements », *arXiv preprint arXiv:1907.11621*, 2019.

[81] A. Ahmed and G. Pierre, « Docker Container Deployment in Fog Computing Infrastructures », *in IEEE International Conference on Edge Computing*, 2018.

[82] A. van Kempen, T. Crivat, B. Trubert, D. Roy, and G. Pierre, « MEC-ConPaaS: An Experimental Single-Board Based Mobile Edge Cloud », *in IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*, 2017.

[83] N. Mohamed, J. Al-Jaroodi, I. Jawhar, H. Noura, and S. Mahmoud, « UAVFog: A UAV-Based Fog Computing for Internet of Things », *in IEEE Smart World Congress*, 2017.

[84] C. Zhu, G. Pastor, Y. Xiao, and A. Ylajaaski, « Vehicular Fog Computing for Video Crowdsourcing: Applications, Feasibility, and Challenges », *IEEE Communications Magazine*, vol. 56, 2018.

[85] C.-K. Huang and S.-H. Shen, « Enabling Service Cache in Edge Clouds », *ACM Transactions on Internet of Things*, vol. 2, 2021.

[86] C.-K. Huang, S.-H. Shen, C.-Y. Huang, T.-L. Chin, and C.-A. Shen, « S-cache: Toward an Low Latency Service Caching for Edge Clouds », *in ACM MobiHoc Workshop on Pervasive Systems in the IoT Era*, 2019.

[87] A. S. Alfakeeh and M. A. Javed, « Intelligent Data-Enabled Task Offloading for Vehicular Fog Computing », *MDPI Applied Sciences*, vol. 13, 2023.

[88] B. Tang, Z. Chen, G. Hefferman, *et al.*, « Incorporating Intelligence in Fog Computing for Big Data Analysis in Smart Cities », *IEEE Transactions on Industrial Informatics*, vol. 13, 2017.

[89] N. Chen, Y. Chen, X. Ye, H. Ling, S. Song, and C.-T. Huang, « Smart City Surveillance in Fog Computing », *in* 2017.

[90] Y. Kalyani and R. Collier, « A Systematic Survey on the Role of Cloud, Fog, and Edge Computing Combination in Smart Agriculture », *MDPI Sensors*, vol. 21, 2021.

[91] H. A. Alharbi and M. Aldossary, « Energy-Efficient Edge-Fog-Cloud Architecture for IoT-Based Smart Agriculture Environment », *IEEE Access*, vol. 9, 2021.

[92] Y. Lin and H. Shen, « CloudFog: Leveraging Fog to Extend Cloud Gaming for Thin-Client MMOG with High Quality of Service », *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, 2016.

[93] Fortinet, *What Is A Data Center?*, https://reurl.cc/Dj14y6, cited Apr 2024.

[94] Fortinet, *Data Center Security*, https://reurl.cc/VzaN1Z, cited Apr 2024.

[95] A. Bhardwaj and C. R. Krishna, « Virtualization in Cloud Computing: Moving from Hypervisor to Containerization — A Survey », *Springer Arabian Journal for Science and Engineering*, vol. 46, 2021.

[96] Docker, *Registry*, https://reurl.cc/OGlLqX, cited Mar 2024.

[97] A. M. Joy, « Performance Comparison Between Linux Containers and Virtual Machines », *in International Conference on Advances in Computer Engineering and Applications*, 2015.

[98] Q. Zhang, L. Liu, C. Pu, Q. Dou, L. Wu, and W. Zhou, « A Comparative Study of Containers and Virtual Machines in Big Data Environment », *in IEEE International Conference on Cloud Computing*, 2018.

[99] C. Pahl, « Containerization and the PaaS Cloud », *IEEE Cloud Computing*, vol. 2, 2015.

[100] K. Jin and E. L. Miller, « The Effectiveness of Deduplication on Virtual Machine Disk Images », *in ACM The Israeli Experimental Systems Conference*, 2009.

[101] A. Ahmed, A. Mohan, G. Cooperman, and G. Pierre, « Docker Container Deployment in Distributed Fog Infrastructures with Checkpoint/Restart », *in IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*, 2020.

[102] A. Ahmed and G. Pierre, « Docker-pi: Docker Container Deployment in Fog Computing Infrastructures », *Inderscience International Journal of Cloud Computing*, vol. 9, 2020.

[103] E. Yigitoglu, M. Mohamed, L. Liu, and H. Ludwig, « Foggy: A Framework for Continuous Automated IoT Application Deployment in Fog Computing », *in IEEE International Conference on AI and Mobile Services*, 2017.

[104] W.-S. Zheng and L.-H. Yen, « Auto-Scaling in Kubernetes-Based Fog Computing Platform », *in Springer International Computer Symposium*, 2019.

[105] N. D. Nguyen, L.-A. Phan, D.-H. Park, S. Kim, and T. Kim, « ElasticFog: Elastic Resource Provisioning in Container-Based Fog Computing », *IEEE Access*, vol. 8, 2020.

[106] The Kubernetes Authors, *Kubernetes Releases and Contributors*, https://reurl.cc/97MydV, cited Apr 2024.

[107] D. Ongaro and J. Ousterhout, « In Search of an Understandable Consensus Algorithm », *in USENIX Annual Technical Conference*, 2014.

[108] Datadog, *10 Insights on Real-World Container Use*, https://reurl.cc/80QLEb, cited Apr 2024.

[109] The Kubernetes Authors, *What Is a Pod*, https://reurl.cc/G47z6A, cited Apr 2024.

[110] The Kubernetes Authors, *Custom resources*, https://reurl.cc/WRpnA5, cited Apr 2024.

[111] P. Arcaini, E. Riccobene, and P. Scandurra, « Modeling and Analyzing MAPE-K Feedback Loops for Self-Adaptation », *in IEEE/ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, 2015.

[112] J. Dobies and J. Wood, *Kubernetes Operators: Automating the Container Orchestration Platform*. O'Reilly Media, 2020.

[113] The Kubernetes Authors, *Multi-tenancy*, https://reurl.cc/RWrRyg, cited Mar 2024.

[114] G. Carcassi, J. Breen, L. Bryant, R. W. Gardner, S. McKee, and C. Weaver, « SLATE: Monitoring Distributed Kubernetes Clusters », *in ACM Practice and Experience in Advanced Research Computing*, 2020.

[115] T. Hu and Y. Wang, « A Kubernetes Autoscaler Based on Pod Replicas Prediction », *in Asia-Pacific Conference on Communications Technology and Computer Science*, 2021.

[116] L. Toka, G. Dobreff, B. Fodor, and B. Sonkoly, « Machine Learning-Based Scaling Management for Kubernetes Edge Clusters », *IEEE Transactions on Network and Service Management*, vol. 18, 2021.

[117] The Open Cluster Management Authors, *Hub-Spoke Architecture*, https://reurl.cc/0vo00x, cited Apr 2024.

[118] M. Borkowski, S. Schulte, and C. Hochreiner, « Predicting Cloud Resource Utilization », *in IEEE/ACM International Conference on Utility and Cloud Computing*, 2016.

[119] H. Xu and B. Li, « Dynamic Cloud Pricing for Revenue Maximization », *IEEE Transactions on Cloud Computing*, vol. 1, 2013.

[120] A. Singh and K. Chatterjee, « Cloud Security Issues and Challenges: A Survey », *Elsevier Journal of Network and Computer Applications*, vol. 79, 2017.

[121] A. J. Fahs and G. Pierre, « Tail-Latency-Aware Fog Application Replica Placement », *in Springer International Conference on Service-Oriented Computing*, 2020.

[122] P. Souza Junior, D. Miorandi, and G. Pierre, « Good Shepherds Care for Their Cattle: Seamless Pod Migration in Geo-Distributed Kubernetes », *in IEEE International Conference on Fog and Edge Computing*, 2022.

[123] H. Arkian, G. Pierre, J. Tordsson, and E. Elmroth, « Model-Based Stream Processing Auto-Scaling in Geo-Distributed Environments », *in IEEE International Conference on Computer Communications and Networks*, 2021.

[124] J. Huang, C. Xiao, and W. Wu, « Rlsk: A Job Scheduler for Federated Kubernetes Clusters Based on Reinforcement Learning », *in IEEE International Conference on Cloud Engineering*, 2020.

[125] D. Kim, H. Muhammad, E. Kim, S. Helal, and C. Lee, « TOSCA-Based and Federation-Aware Cloud Orchestration for Kubernetes Container Platform », *MDPI Applied Sciences*, vol. 9, 2019.

[126] B. Shayesteh, C. Fu, A. Ebrahimzadeh, and R. H. Glitho, « Automated Concept Drift Handling for Fault Prediction in Edge Clouds Using Reinforcement Learning », *IEEE Transactions on Network and Service Management*, vol. 19, 2022.

[127] F. Faticanti, D. Santoro, S. Cretti, and D. Siracusa, « An Application of Kubernetes Cluster Dederation in Fog Computing », *in Conference on Innovation in Clouds, Internet and Networks*, 2021.

[128] C. Reiss, J. Wilkes, and J. L. Hellerstein, « Google Cluster-Usage Traces: Format and Schema », *Google White Paper*, vol. 1, 2011.

[129] L. Larsson, H. Gustafsson, C. Klein, and E. Elmroth, « Decentralized Kubernetes Federation Control Plane », *in IEEE/ACM International Conference on Utility and Cloud Computing*, 2020.

[130] The Karmada Authors, *Karmada*, https://reurl.cc/RyW9rz, cited Mar 2024.

[131] The Karmada Authors, *Test Report on Karmada's Support for 100 Large-Scale Clusters*, https://reurl.cc/Wv8Emy, cited Mar 2024.

[132] M. Iorio, F. Risso, A. Palesandro, L. Camiciotti, and A. Manzalini, « Computing Without Borders: The Way Towards Liquid Computing », *IEEE Transactions on Cloud Computing*, vol. 11, 2023.

[133] The Virtual Kubelet Authors, *Virtual Kubelet*, https://reurl.cc/mORK1Y, cited Mar 2024.

[134] Loft Labs, *Kiosk*, https://reurl.cc/GKn2qW, cited Mar 2024.

[135] The Kubernetes Authors, *Resource Quotas*, https://reurl.cc/WRaekL, cited Mar 2024.

[136] The Capsule Authors, *Capsule*, https://reurl.cc/OGyOyR, cited Mar 2024.

COLLEGE DOCTORAL BRETAGNE
MATHS, TELECOMS
INFORMATIQUE, SIGNAL
SYSTEMES, ELECTRONIQUE

Université de Rennes

Titre : titre (en français).....

Mot clés : de 3 à 6 mots clefs

Résumé : Eius populus ab incunabulis primis ad usque pueritiae tempus extremum, quod annis circumcluditur fere trecentis, circummurana pertulit bella, deinde aetatem ingressus adultam post multiplices bellorum aerumnas Alpes transcendit et fretum, in iuvenem erectus et virum ex omni plaga quam orbis ambit inmensus, reportavit laureas et triumphos, iamque vergens in senium et nomine solo aliquotiens vincens ad tranquilliora vitae discessit. Hoc inmaturo interitu ipse quoque sui pertaesus excessit e vita aetatis nono anno atque vicensimo cum quadriennio imperasset. natus apud Tuscos in Massa Veternensi, patre Constantio Constantini fratre imperatoris, matreque Galla. Thalassius vero ea tempestate praefectus praetorio praesens ipse quoque adrogantis ingenii, considerans incitationem eius ad multorum augeri discrimina, non maturitate vel consiliis mitigabat, ut aliquotiens celsae potestates iras principum molliverunt, sed adversando iurgandoque cum parum congrueret, eum ad rabiem potius evibrabat, Augustum actus eius exaggerando creberrime docens, idque, incertum qua mente, ne lateret adfectans. quibus mox Caesar acrius efferatus, velut contumaciae quoddam vexillum altius erigens, sine respectu salutis alienae vel suae ad vertenda opposita instar rapidi fluminis irrevocabili impetu ferebatur. Hae duae provinciae bello quondam piratico catervis mixtae praedonum.

Title: titre (en anglais).....

Keywords: de 3 à 6 mots clefs

Abstract: Eius populus ab incunabulis primis ad usque pueritiae tempus extremum, quod annis circumcluditur fere trecentis, circummurana pertulit bella, deinde aetatem ingressus adultam post multiplices bellorum aerumnas Alpes transcendit et fretum, in iuvenem erectus et virum ex omni plaga quam orbis ambit inmensus, reportavit laureas et triumphos, iamque vergens in senium et nomine solo aliquotiens vincens ad tranquilliora vitae discessit. Hoc inmaturo interitu ipse quoque sui pertaesus excessit e vita aetatis nono anno atque vicensimo cum quadriennio imperasset. natus apud Tuscos in Massa Veternensi, patre Constantio Constantini fratre imperatoris, matreque Galla. Thalassius vero ea tempestate praefectus praetorio praesens ipse quoque adrogantis ingenii, considerans incitationem eius ad multorum augeri discrimina, non maturitate vel consiliis mitigabat, ut aliquotiens celsae potestates iras principum molliverunt, sed adversando iurgandoque cum parum congrueret, eum ad rabiem potius evibrabat, Augustum actus eius exaggerando creberrime docens, idque, incertum qua mente, ne lateret adfectans. quibus mox Caesar acrius efferatus, velut contumaciae quoddam vexillum altius erigens, sine respectu salutis alienae vel suae ad vertenda opposita instar rapidi fluminis irrevocabili impetu ferebatur. Hae duae provinciae bello quondam piratico catervis mixtae praedonum.