UnBound: Multi-Tenancy Management in Scalable Fog Meta-Federations

Chih-Kai Huang, Guillaume Pierre

UnBound: Multi-Tenancy Management in Scalable Fog Meta-Federations

Chih-Kai Huang
Univ Rennes, Inria, CNRS, IRISA
chih-kai.huang@irisa.fr

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

***Abstract*—Designing large-scale fog computing platforms requires the aggregation of large numbers of servers in numerous locations covering a full country or even a continent. However, it would be difficult for a single organization to deploy enough resources while attracting sufficient workloads to generate high resource utilization. Instead, we propose the concept of *meta-federations*, where many independent local resource providers may lease their resources to multiple fog federations. We propose UnBound, a scalable meta-federations framework that specifically addresses the difficult multi-tenancy issue in this context to isolate workloads belonging to different users and fog providers. Extensive evaluations with federations of up to 500 geo-distributed Kubernetes *clusters* demonstrate that UnBound maintains comparable application deployment times to the state of the art in a single member cluster scenario, avoids increasing cross-cluster network traffic, keeps resource consumption within acceptable limits, and exhibits stability and scalability, making it a suitable solution for large-scale fog computing deployments.**

***Index Terms*—Fog computing, Kubernetes, Geo-Distributed clusters, Federations, Multi-Tenancy**

I. Introduction

Fog and edge computing have established themselves as extensions of traditional cloud platforms with additional nodes located next to the data sources. Processing incoming data in the near proximity of their sources enables lower response times, reduced need for long-distance communications, and better control of the end-to-end data processing pipelines [25].

However, in most cases, each new usage of fog computing technologies requires the creation of a new dedicated hardware and software infrastructure in the appropriate location [1]. Assigning a specific set of devices to a single use case negates the economies of scale delivered by the multi-tenancy and statistical multiplexing principles of cloud computing, as it essentially brings these services back to a pre-cloud era where every application had to be provisioned individually with its own dedicated hardware. Additionally, this architecture generates high costs for setting up each application-specific infrastructure while limiting its geographical scope to the set of locations where suitable devices have been deployed.

We argue that enabling fog platforms to embrace the full benefits of cloud computing principles requires the design of large-scale, public, multi-tenant, 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 paper proposes the design of scalable fog *meta-federations* to address this challenge. We define meta-federations 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. We discuss the concept of meta-federations in Section II.

An important and difficult challenge of fog meta-federations is multi-tenancy. In these systems, the same group of servers may be used to host workloads belonging to multiple tenants who are customers of different providers. This scenario requires the system to guarantee isolation at two different levels. First, two tenants of the same provider should not be able to see or interfere with each other's workloads. Second, two fog providers that have established access to the same fog cluster should also not be able to see or interfere with one another. To our best knowledge, this two-level multi-tenancy challenge is unique to federated environments where the same cluster may be used by multiple independent federations.

The second challenge is that of scalability. Building a fog computing infrastructure at the scale of a country or even a continent requires one to aggregate resources in thousands of different locations. A single "management cluster" therefore needs to be able to control access to thousands of "member clusters." Conversely, to maintain high resource utilization, each member cluster may decide to join numerous independent federations, each with its own management cluster.

We propose UnBound, a scalable fog meta-federations framework that addresses the complex multi-tenancy and scalability challenges introduced by meta-federations, where individual fog clusters may lease their resources to multiple administrative domains. UnBound uses Kubernetes to orchestrate resources within individual fog clusters [17] and Open Cluster Management (OCM) to federate multiple member clusters under the authority of a management cluster [23]. OCM is a multi-cluster orchestration tool that manages clusters and distributes workloads across them. We address the issue of multi-tenancy management by isolating federations within a single member cluster using the Virtual Kubernetes Clusters (vCluster) [9] project to create isolated logical sub-clusters within the member clusters. Each vCluster[1] has its own

[1]We use the term "vCluster project" to refer to the entire vCluster framework, and the term "vCluster" to refer to a virtual cluster created for isolation between different management clusters in a member cluster.

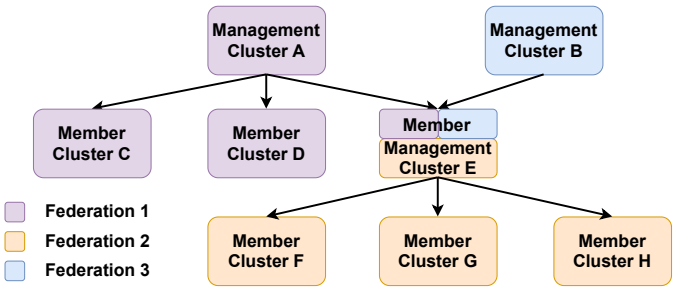


Fig. 1. An example of meta-federations.

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 deployments in a large-scale distributed cluster and show 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.

Section II of this paper discusses the motivation behind this work, then Section III provides an overview of the background and related work. In Section IV, we describe the design and components of UnBound. We evaluate our solution in Section V and summarize the conclusions in Section VI.

II. Motivation

Multi-cluster federations are a standard technique to aggregate multiple independent Kubernetes clusters into a single logical entity [19]. This concept enables users to gain seamless access to a large computing infrastructure in multiple geographical locations. Federation users submit workload deployment requests to a single *management cluster*, which subsequently forwards requests to one or several of their *member clusters* according to the job's metadata and some pre-defined scheduling policies. However, standard federation frameworks such as KubeFed [12] assume that all the hardware resources in a federation belong to a single administrative domain, which limits federations to scenarios where the organizations that own the resources have total trust in each other. Furthermore, they follow a strict hierarchical organization where a given cluster should be either a management cluster or a member of a single federation. More complex organizations (e.g., a cluster being a member of two different federations while also being a manager of a third one) are usually not supported.

We propose a different model where multiple clusters belonging to different organizations may freely establish or revoke peering relationships with one another. Figure 1 shows an example where three federations co-exist with varying types of relationships between the clusters. Federation 1 (in purple color) is a geo-distributed federation where management cluster A has established access rights to member clusters C, D, and E. Cluster E has multiple roles because it is also a member of Federation 3 while being in charge of managing Federation 2. Federation 2 also expands to multiple other member clusters. This example demonstrates the versatility of meta-federations capabilities, accommodating various configurations, including the one-to-many and many-to-one relationships showcased here.

In meta-federations, each cluster may be owned by a different entity. Peering relationships between clusters may be established via a legal contract where one company leases some of its hardware resources to another. For example, Cluster C and Cluster D may belong to two separate companies in different locations. Both of them have established a peering contract with Federation 1 so that Federation 1 can use their computing resources and expand its range of service locations following a classical cloud-like pay-as-you-go business model. Using the meta-federation concept potentially allows one to build large-scale geo-distributed fog platforms at the scale of a country or even a continent.

Realizing the vision of a large meta-federations ecosystem 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.

III. Background and Related Work

Fog/edge computing [25] aims to extend the cloud computing concept with computing resources closer to the end-users. To support large-scale fog platforms, the rise of multi-cluster federations has spurred the development of various solutions to manage and orchestrate federated Kubernetes environments. We discuss these works in Section III-A. Section III-B then turns to work aiming to address multi-tenancy related challenges. We summarize the main approaches in Table I.

A. Multi-Cluster Federation Control Plane

Kubernetes Cluster Federation (KubeFed) provides application deployment and resource management in multiple Kubernetes cluster environments [12]. KubeFed extends Kubernetes with Custom Resource Definitions (CRDs) to provide abstractions such as FederatedNamespaces and FederatedDeployments for managing multi-cluster applications. Users can manage multiple Kubernetes clusters from a single host cluster with the KubeFed control plane installed. However, KubeFed has limited support for automated policy-based scheduling and propagates resources to target clusters without any prior checks

TABLE I
COMPARISON OF MULTI-CLUSTER FEDERATION CONTROL PLANES AND MULTI-TENANCY FRAMEWORKS.

	Meta-Federations	Sync Modes	Multi-Tenancy Methods	Isolation Levels	Certification
Multi-Cluster Federation Control Planes					
KubeFed [12]	+/-	Push	✖	✖	✖
mck8s [14]	+/-	Push	✖	✖	✖
OCM [23]	+/-	Pull	✖	✖	✔ CNCF
Karmada [16]	+/- (Push Mode) ✖ (Pull Mode)	Push/Pull	✖	✖	✔ CNCF
Liqo [5]	✔	Push	Flat Namespace	Soft	✖
Multi-Tenancy Frameworks					
Kiosk [8]	✖	✖	Flat Namespace	Soft	✖
HNC [6]	✖	✖	Hierarchical Namespaces	Soft	✖
EdgeNet [13]	✔	Push	Hierarchical Namespaces	Soft	✖
vCluster project [9]	✖	✖	Separate Control Plane	Hard	✔ Certified Kubernetes Distribution
Our contribution					
UnBound	✔	Pull	Separate Control Plane	Hard	+/- (Certified Components)

✔: the item is fully implemented/addressed. +/-: the item is partially implemented/addressed. ✖: the item is not implemented/addressed.

on resource availability in the chosen clusters. The KubeFed project is now retired and no longer under active development.

Multi-cluster Kubernetes (mck8s) addresses KubeFed's limitations and proposes resource-based automated placement, multi-cluster horizontal Pod[2] auto-scaling and bursting of container-based applications in geo-distributed cluster federations [14]. mck8s also provides additional functionality such as multi-cluster network discovery, global load balancing, and monitoring. However, 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 [11]. Pull-based methods are considered more robust, scalable, and secure as the main management tasks are handled by components in each member cluster, which reduces the control stress in the management cluster [15].

Open Cluster Management (OCM) presents a different management model inspired by the original principles of Kubernetes [23]. OCM separates multi-cluster operations into two phases: computation/decision (performed in the management cluster) and execution (performed in the member clusters). 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. 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 offers solutions for managing applications across multiple Kubernetes clusters [16]. In contrast to other solutions, 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 to provide a single point of control. Karmada supports both *Push* and *Pull* approaches to manage multiple clusters. 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 [15].

An important remark is that the four 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 only federation framework that supports some form of multi-tenancy is Liqo [5]. Liqo creates virtual node resources in the management cluster using a so-called Virtual Kubelet [24]. 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 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 a member cluster to isolate resources created by each management cluster. It however limits itself to flat Namespaces, which makes it impossible to 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.

B. Multi-Tenancy Frameworks

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

[2]A Pod is the smallest execution unit in Kubernetes. It is a group of one or more containers that are scheduled and managed together. Pods share the network and file system and are isolated from other Pods.

security policies to specific Namespaces [21]. 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 different techniques, as described next.

The Kiosk multi-tenancy framework uses Namespaces to provide isolated execution environments for tenant applications [8]. Each tenant maps to an *account*, and each account is scoped to a single Namespace called a *space* in Kiosk. Contrary to regular Kubernetes Namespaces, each Kiosk user can only see or interact with resources in their own space. However, this flat Namespace approach may make it difficult for cluster administrators to manage these Namespaces.

Extending the flat Kubernetes Namespace structure may be realized using the Hierarchical Namespace Controller (HNC) [6]. Hierarchical Namespaces may for example allow one to apply similar policies to multiple Namespaces. By default, Kubernetes 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 [13] 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.

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 [21]. However, these simple solutions contradict our objective of maximizing fog resource utilization by sharing the available servers between large numbers of tenants.

Finally, the Virtual Kubernetes Clusters (vCluster) project provides a fully functional virtual cluster that runs on top of another Kubernetes cluster [9]. 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 the vCluster project as a compliant Kubernetes distribution [10]. We therefore exploit the vCluster project to handle multi-tenancy management in UnBound meta-federations.

IV. System Design

In this work, we present the concept of *meta-federations* and propose UnBound, a scalable fog meta-federations framework that considers different levels of multi-tenancy to support clusters from different organizations in multiple Kubernetes cluster environments. We now discuss meta-federations design and present the details of UnBound's components.

A. System Model and Meta-Federations

We assume that many small or medium-sized fog resource providers may choose to cover only a limited region or set of regions with their own cluster federation. A fog cluster federation may be composed of multiple Kubernetes clusters that are deployed close to the end users. Within each cluster, we assume that all servers are located in the same geographical location and that all clusters and nodes can communicate with each other through the network infrastructure. We further assume that each Kubernetes cluster has enough computing resources to run UnBound's components.

In meta-federations, fog resource providers can establish business agreements to share their computing resources with one or more federations. These agreements allow Kubernetes federations to expand their geographical reach and serve users in different areas by using computing resources from other Kubernetes clusters. Different organizations may operate their own federations and host multiple users in their federations.

Users can run their applications on a specific Kubernetes cluster or distribute them among member clusters of a given federation. As a result, two separate isolation scenarios arise: inter-user isolation and inter-federation isolation. Inter-user isolation requires isolation between different users within the same Kubernetes cluster in a non-federated environment. Once a Kubernetes cluster takes the role of management cluster of a federation, the users in this management cluster can deploy applications across all member clusters in the federation. Similar to the case of a single cluster, the isolation of the users in the member clusters must also be considered. For inter-federation isolation, each federation, which may belong to different fog providers, should also not be able to see or interfere with one another and should be isolated when these federations access the same Kubernetes clusters.

We expect fog computing platforms to be geographically distributed, with their member clusters spanning a country or even a continent. In this environment, the scalability of meta-federations is a critical challenge. Each management cluster should be able to manage a large number of member clusters simultaneously, and each member cluster should be able to join multiple management clusters from different organizations to maintain high resource utilization. As a result, meta-federations support both *one* management cluster to *many* member clusters and many-to-one configurations. Therefore, each Kubernetes cluster can concurrently operate as both a management cluster and a member cluster.

B. System Architecture

To achieve the vision of meta-federations, UnBound relies on Kubernetes to orchestrate resources within individual fog clusters. Then, we build UnBound meta-federations based on two open-source projects, Open Cluster Management (OCM) and Virtual Kubernetes Clusters (vCluster). OCM is responsible for managing federated clusters and distributing the work-

loads across these clusters, where the *Pull* model of OCM is well-suited for large-scale federations to address the scalability challenge of meta-federations. To ensure isolation between federations that may operated by different organizations in the same member cluster, we select the vCluster project because of its hard isolation guarantees and low performance overhead.

Figure 2 illustrates the UnBound architecture and its components. A management cluster can accommodate multiple users who may deploy workloads in the host cluster or member clusters via the `kubectl` or `clusteradm`[3] tools. After a member cluster joins the management cluster, the system provisions a dedicated Kubernetes Namespace to represent the member cluster. Users can manually create ManifestWorks (we discuss ManifestWork in Section IV-C1) to this Namespace in the management cluster, and the system then deploys the workload, which is described in ManifestWorks, to the corresponding member cluster.

Within each member cluster, UnBound creates a specific Kubernetes Namespace for each corresponding management cluster. It then installs the OCM components and k3s-based[4] vCluster components in this Namespace. The agent in the member cluster continuously monitors the Namespace from the management cluster, pulling the latest ManifestWorks states, synchronizing them with the respective vCluster through its own API server, and pushing the current status of workloads to the management cluster. The metadata of workloads from the management cluster are stored in the vCluster's own data store. Based on these metadata, vCluster creates corresponding Pods or related Kubernetes resources in the underlying host Kubernetes cluster in the same Namespace of OCM components and vCluster. This design isolates all UnBound components and workloads from a management cluster within a dedicated Namespace in a member cluster, preventing interference from other management clusters. Additionally, this approach facilitates the enforcement of resource quotas in the Namespace for each management cluster within the member cluster [20]. We leave the topic of dynamically setting the resource quotas of different Namespaces and possibly allowing quota oversubscription for future work.

C. Components of UnBound

1) Open Cluster Management: Open Cluster Management (OCM) [23] simplifies the management of multiple Kubernetes clusters by decomposing multi-cluster operations in two phases: computation/decision and execution. Consequently, a federation is composed of two different roles: management cluster (hub) and member cluster (agent).

A federation's management cluster is responsible for managing and controlling multiple member clusters. It also makes placement decisions to distribute the workloads across the member clusters. On the other hand, each member cluster is responsible for carrying out the management cluster's instructions and running the workloads that were assigned to it.

[3]The `clusteradm` command-line interface allows users to interact with Open Cluster Management clusters.

[4]k3s is a Kubernetes distribution designed for fog/edge computing.

To achieve greater scalability for the federation platform, OCM employs the "hub-agent" architecture, which mirrors the original "hub-kubelet" pattern from Kubernetes. This architecture utilizes a *Pull* mechanism to retrieve the latest prescriptions from the management cluster and to continuously reconcile the workloads to the desired state in the member cluster. OCM introduces a Custom Resource (CR) called ManifestWork [18], [22]. A ManifestWork defines a group of Kubernetes resources to be deployed across member clusters in a federation. Users deploy ManifestWorks in a particular Namespace in the management cluster, and the work-agent in the member cluster subsequently monitors the contents of ManifestWorks in that Namespace to keep the status of workloads in sync with it. Note that only workloads that use ManifestWork will be deployed to the member clusters.

In each management cluster, UnBound exploits four main components from OCM: cluster-manager, placement-controller, registration-controller, and work-webhook.

- **Cluster-manager:** The cluster-manager is an operator in charge of creating and managing other controllers within the management cluster.
- **Placement-controller:** This controller enables users to schedule their workloads to a set of member clusters automatically. The OCM scheduling framework is based on the Kubernetes scheduling architecture and is organized in two steps: predicate and prioritize. The predicate handles hard requirements, such as label selection and taints/tolerations. After selecting clusters that satisfy the mandatory hard requirements, the prioritize phase evaluates the clusters identified in the predicate step based on soft requirements such as the number of clusters and resource status of clusters, to determine a suitable subset of member clusters. Moreover, users can expand the multi-cluster scheduling functionality through the OCM add-on framework.
- **Registration-controller:** Two main tasks for the registration-controller are for member cluster registration and receiving the health status of member clusters.
- **Work-webhook:** The work-webhook is an admission webhook running in the management cluster to validate the content of ManifestWorks.

UnBound leverages three OCM components for each management cluster in a member cluster:

- **Klusterlet:** The task of Klusterlet is similar to cluster-manager, which is a bootstrap application to create and manage other agents in the member cluster.
- **Registration-agent:** The registration-agent operates in the member cluster and handles the registration process. After registration, it keeps regularly sending heartbeats to the controller and checking the certificate of registration validation.
- **Work-agent:** The work-agent monitors the ManifestWorks in a Namespace that represents the work-agent hosted member cluster in the management cluster. When the work-agent detects a change in the Namespace in the management cluster, it applies or changes the Kubernetes resources included in the ManifestWorks to the member cluster.

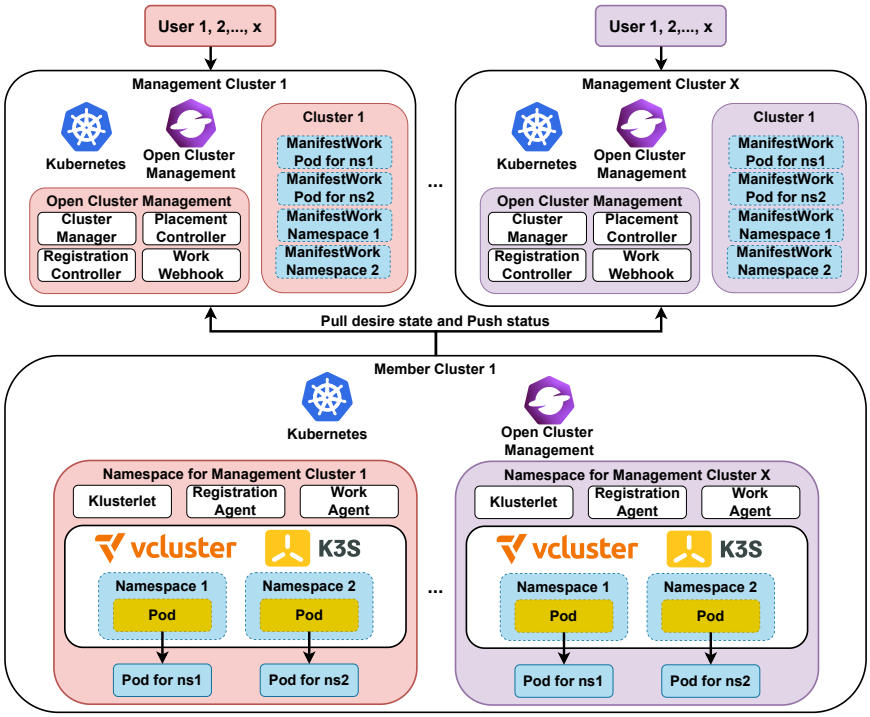


Fig. 2. UnBound architecture where a member cluster shares its resources with two management clusters. The "ns" abbreviation stands for Namespace.

2) Virtual Kubernetes Clusters: The vCluster project aims to run a full Kubernetes cluster as an application and host it in another Kubernetes cluster to provide multi-tenancy functionality [9]. It supports different Kubernetes distributions, including vanilla Kubernetes, K3s, and K0s. Each vCluster has its own control plane, providing better tenant isolation than the original Kubernetes Namespaces. Moreover, vCluster has its own storage backend and supports different databases, such as SQLite, MySQL, PostgreSQL, and etcd. vCluster relies on its host underlying cluster to provide its own worker node pool and networking resources.

The vCluster project separates Kubernetes resources using two levels: high-level and low-level. High-level resources, such as Deployment, StatefulSet, and Custom Resource Definitions (CRDs), are purely virtual. These resources only reach the vCluster API server and get stored in vCluster's data store to avoid using the API server and data store from the underlying Kubernetes cluster. Since vCluster does not have actual worker nodes and networking, some "low-level" resources such as Pods and Services need to be synchronized to the underlying Kubernetes cluster and in the same Namespace as the vCluster.

UnBound leverages vCluster to isolate different management clusters which may make simultaneous usage of the same member cluster. UnBound therefore deploys vCluster only in the member cluster. Each vCluster has two components:

- **Control plane:** vCluster's control plane includes the Kubernetes API server, data store, controller manager, and optional scheduler. In UnBound meta-federations, the Kubernetes API server handles the requests from the work-agent to create, update, or delete the workloads in the vCluster. The data store is the database where the API server stores metadata of all resources. The controller manager monitors the status of the entire vCluster and ensures that the vCluster is in the expected working state. The administrator can enable the scheduler inside the vCluster to provide scheduling with custom requirements, such as affinity and topology spreading in the host Kubernetes cluster.
- **Syncer:** vCluster uses a syncer to create low-level resources, such as Pods and Configmaps, in the underlying host cluster. To schedule low-level workloads in the host cluster, vCluster reuses the host cluster's scheduler to place workloads by default. As Pods are scheduled directly in the underlying host cluster, they experience no performance degradation. Similar to the work-agent component of OCM, after deploying low-level resources, vCluster's syncer keeps periodically synchronizing the status between vCluster's control plane and the underlying host cluster.

3) Multi-Tenancy Management: UnBound supports two levels of isolation: inter-user and inter-federation. We further divide inter-user isolation into two cases: users within a cluster and users within a federation.

- **Inter-user multi-tenancy within a Kubernetes cluster:** Administrators can use Kubernetes *Namespaces* to isolate workloads from each other. To limit resource usage in different Namespaces, the administrators can also leverage the resource quotas function [20] to divide cluster computing resources between multiple users in each Namespace.
- **Inter-user multi-tenancy within an UnBound meta-federation:** When a Kubernetes cluster becomes the man-

agement cluster of an UnBound meta-federation, its administrator can create Kubernetes *Namespaces* in the vCluster using Manifestworks (see Figure 2). The capability to isolate different users within the federation stems from the vCluster functionality which allows users to create cluster-scoped resources such as Namespaces. Furthermore, unlike the original OCM, which directly accesses Kubernetes clusters where the control plane manages the entire cluster, the completely separate control plane of vCluster offers users access to vCluster for detailed management or debugging tasks through the vCluster's kubeconfig file without affecting other vClusters or the host Kubernetes cluster.

- **Inter-federation multi-tenancy within a single member cluster:** Management clusters belonging to different organizations may concurrently use the resources of the same member cluster. To enable multi-tenancy management in this scenario, UnBound employs vCluster to isolate workloads associated with different management clusters. Each vCluster uses its own control plane, which effectively isolates workloads from the different management clusters. Moreover, the metadata of the Kubernetes resources in vCluster are stored in its own data store. As a result, requests and metadata created by management clusters cannot reach the underlying Kubernetes cluster nor other vClusters, which provides a stronger form of isolation than Kubernetes Namespaces. Note that the owner of a member cluster can still access information about all workloads in their cluster. Therefore, while the owner lends its clusters to others, it can still enforce the terms of the resource leasing contract.

V. Performance Evaluation

We evaluate UnBound using four sets of experiments: (1) multi-cluster application creation in a member cluster; (2) application stability despite a vCluster failure; (3) one management cluster with multiple member clusters; and (4) multiple management clusters with one member cluster. We run the experiments in both cloud and fog networking environments.

A. Experimental Setup

Our prototype implementation uses modified code from the Open Cluster Management project to automatically create vCluster in the particular Namespace and connect the work-agent to vCluster. We also fine-tuned the vCluster configurations to suit UnBound meta-federations. For data collection, we use tcpdump to collect the cross-cluster network traffic and Kubernetes Metrics Server to measure resource consumption, including CPU and memory usage [7]. We then compare our solution to the original Open Cluster Management.

To ensure realistic experiments, we run our evaluations in the large-scale Grid'5000 distributed cluster testbed [2]. We emulate a fog networking environment using netem [4] and introduce a 50 ms delay with 5 ms jitter to both the network ingress and egress, resulting in a total network latency of 100 ms with 10 ms jitter.

B. Multi-Cluster Application Creation in a Member Cluster

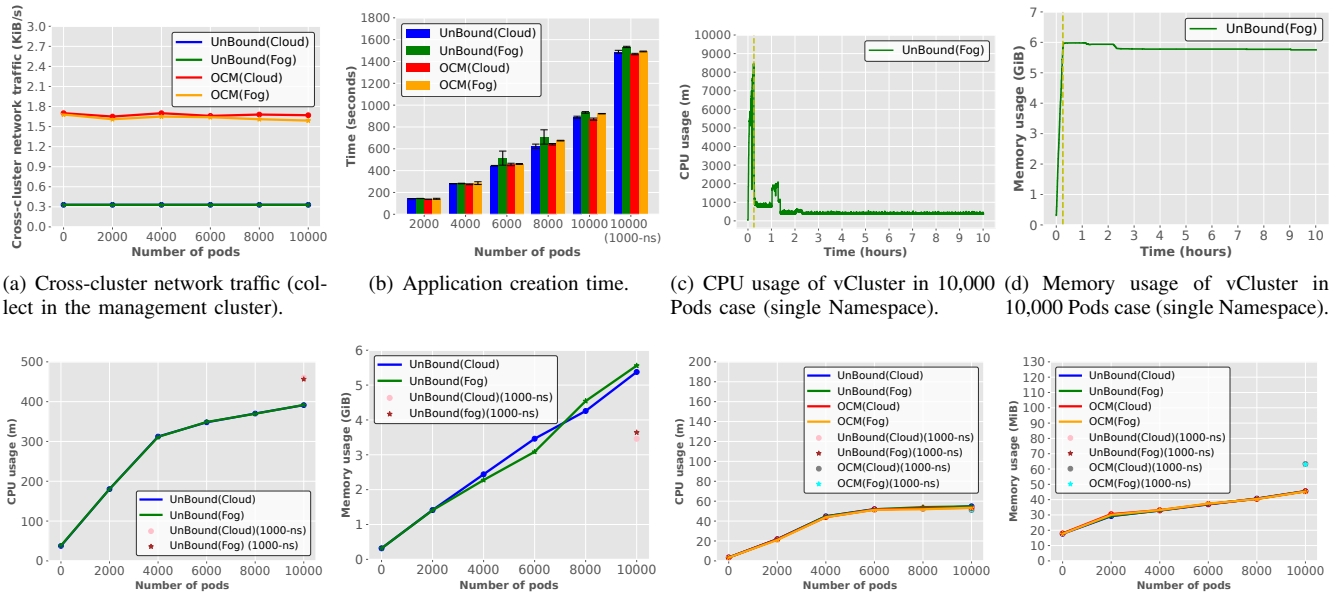
In this experiment, we launch two Kubernetes clusters: a management cluster which uses a single Virtual Machine (VM) with 16 cores and 32 GiB of memory; and a member cluster which uses 101 VMs, including one control plane node with 16 cores and 32 GiB of memory and 100 worker nodes with 2 cores and 4 GiB of memory each. We deploy up to 1000 ManifestWorks in the management cluster to create Kubernetes Deployments in the member cluster. Each Deployment creates 10 nginx Pods, resulting in up to 10,000 Pods in total within the same Namespaces. Moreover, we also create a scenario with 10,000 Pods where each Deployment runs in its own Namespace in the member cluster so there are a total of 1000 Namespaces, and each Namespace contains 10 nginx Pods to simulate inter-user isolation in UnBound. We refer to this scenario as 1000-ns in the following section.

We run each experiment three times and report the average results across them. The only exceptions are Figures 3(c) and 3(d) for long-term collection experiments, which show the results of a single run (repeating these experiments draws similar results). For other figures, we skip the first 9000 seconds of data and average the remaining 7200 seconds as the results for a round. We do this because we found that vCluster optimizes deployed resources and uses slightly more computing resources during the first 9000 seconds, as we can see in Figures 3(c) and 3(d). Therefore, the data during this first period does not represent normal resource usage.

Figure 3(a) demonstrates that UnBound effectively reduces cross-cluster network traffic in both cloud and fog environments for any number of Pods. The average network traffic across different numbers of Pods in UnBound is 0.33 KiB/s in both environments, while the average network traffic for OCM is 1.68 KiB/s in the cloud and 1.63 KiB/s in the fog. To enhance clarity, the figure excludes the results for 1000-ns experiments, which are essentially identical to those for 10,000 Pods within a single Namespace. These findings not only show that our solution does not introduce additional cross-cluster network traffic between the management and member clusters but also evidence the effectiveness of UnBound.

Figure 3(b) presents the necessary time to create applications until all Pods reach running status in the member cluster. The creation times for UnBound(Cloud), UnBound(Fog), OCM(Cloud), and OCM(Fog) are 1488, 1533, 1468 and 1492 seconds, respectively, for the 10,000 Pods case. The fog environment takes a little longer using either UnBound or OCM. This is due to network latency, which may cause a delay in pulling the latest prescription. Our solution takes 20 (Cloud) and 41 (Fog) seconds longer than OCM, which shows that the overhead of our approach is small. The 1000-ns scenario shows the same trend, but it takes longer to create the applications because the Namespaces for each Deployment must also be created, resulting in 2000 ManifestWorks instead of 1000 in the same Namespace case.

Figure 3(c) and Figure 3(d) represent the long-term collection of 10,000 Pods in the same Namespace in the member



(a) Cross-cluster network traffic (collect in the management cluster).

(b) Application creation time.

(c) CPU usage of vCluster in 10,000 Pods case (single Namespace).

(d) Memory usage of vCluster in 10,000 Pods case (single Namespace).

(e) CPU usage of vCluster.

(f) Memory usage of vCluster.

(g) CPU usage of work-agent.

(h) Memory usage of work-agent.

Fig. 3. Results of multi-cluster application creation experiment. 1000-ns represents the results of a scenario with 10,000 Pods that each Deployment runs in its own Namespace, resulting in 1000 Namespaces. The yellow dash lines in Figures (c) and (d) indicate the time when all Pods reach running status.

```
Command: kubectl get pods -n mgt-10-158-0-2-6443-klusterlet
NAME                                                     READY   STATUS    RESTARTS      AGE
adservice-7d857689bd-l9j7n-x-default-x-vcluster          1/1     Running   0             28m
cartservice-5d844fc8b7-2wt4n-x-default-x-vcluster        1/1     Running   0             28m
checkoutservice-84cb944764-22fx2-x-default-x-vcluster    1/1     Running   0             28m
coredns-f8bfc8497-cqnhb-x-kube-system-x-vcluster         1/1     Running   0             29m
currencyservice-76f9b766b4-f842p-x-default-x-vcluster    1/1     Running   0             28m
emailservice-767cd45966-nj8gm-x-default-x-vcluster       1/1     Running   0             28m
frontend-8475b5657d-kpsvd-x-default-x-vcluster           1/1     Running   0             28m
mgt-10-158-0-2-6443-klusterlet-76c679994b-7kn7z          1/1     Running   0             29m
mgt-10-158-0-2-6443-klusterlet-registration-agent-78c6ccdb7249w  1/1  Running  1 (28m ago)  29m
mgt-10-158-0-2-6443-klusterlet-work-agent-555896bddb-bdp2c  1/1  Running  0             28m
paymentservice-866fd4b98-p588p-x-default-x-vcluster      1/1     Running   0             28m
productcatalogservice-5b9df8d49b-nftmg-x-default-x-vcluster  1/1  Running  0             28m
recommendationservice-6ffb84bb94-t7bc4-x-default-x-vcluster  1/1  Running  0             28m
redis-cart-76b9545755-4zm2z-x-default-x-vcluster         1/1     Running   0             28m
shippingservice-648c56798-zpnmm-x-default-x-vcluster     1/1     Running   0             28m
vcluster-0                                               2/2     Running   0             8s
Command: kubectl get events -A --field-selector involvedObject.name=vcluster-0 --no-headers
 | grep Killing | grep Stopping.container.vcluster | wc -l
100
```

Fig. 4. Application stability despite a vCluster failure.

cluster. The yellow dash lines in the figures indicate the time when all Pods reach running status. We only show the results for the fog environment, as the results for the cloud environment are similar. Before 914 seconds, the member cluster is busy creating Pods, which causes CPU and memory usage to increase. Once all Pods are running, the vCluster spends approximately 2.5 hours optimizing the deployed workloads, which requires more CPU resources than the maintenance phase. Memory usage follows the same trend, stabilizing after the vCluster completes its optimization. The average CPU usage in the maintenance phase is 391 millicores with a standard deviation of 36 m[5], and the average memory usage is 5.9 GiB with a standard deviation of 9 MiB. These results show that the resource usage of vCluster is small and stable.

[5]The Kubernetes metrics server uses millicores (m) to measure CPU usage. One millicore equals 0.001 vCPU/core for cloud providers.

Figure 3(e) plots vCluster CPU usage with different numbers of Pods. The CPU usage trend is almost the same in clouds and fogs, up to 391 m in the 10,000 Pods case. For the 1000-ns scenario, vCluster consumes more CPU usage, 460 m (Cloud) and 456 m (Fog), because it maintains more resources, including Namespaces, Kubernetes Deployment, and Pods. As shown in Figure 3(f), the memory usage of vCluster experiences linear growth from 0.3 GiB (for 0 Pod), 1.4 GiB (2000 Pods), to 5.4 GiB (10,000 Pods). However, the 1000-ns scenario consumes less memory: 3.5 GiB (Cloud) and 3.6 GiB (Fog). We see that UnBound's CPU and memory usage remain low, even for a large number of Pods.

Figure 3(g) illustrates the work-agent resource usage. The CPU usage is similar regardless of the methods or environments. For 10,000 Pods, the CPU usage is between 51 m and 55 m. Figure 3(h) shows that the 1000-ns case uses more

memory, which is around 63 MiB, whereas the memory usage of 10,000 Pods in a single Namespace is 45 MiB.

Overall, these results demonstrate that UnBound performs comparably to the original Open Cluster Management in both cloud and fog network environments despite introducing additional components to provide inter-federation isolation.

C. Application Stability despite a vCluster Failure

UnBound leverages vCluster to isolate different federations in a member cluster. However, vCluster may potentially crash and affect the workloads it manages. To demonstrate the resilience of UnBound in the face of vCluster failures, we launch two clusters: one management cluster and one member cluster. We create ManifestWorks for deploying the Online Boutique micro-service demo application [3] and wait for these applications to reach running status in the member cluster. We then delete the vCluster Pod 100 times and check the application status in the member cluster.

The first command in Figure 4 lists all Pods running in the particular Namespace to check the information about these Pods, including their status, the times of restarts, and their age. The second command displays the number of times the vCluster has been deleted in the log files. We can see in the figure that after 100 deletions, the age of the vCluster Pod is only 8 seconds, while the other micro-service Pods (whose names end with vCluster) are 28 minutes, which shows that the vCluster is indeed deleted by our script. Note that the vCluster has 0 restart because we delete the Pod of the vCluster instead of restarting it. Additionally, the registration-agent restarts one time upon completion of the registration process. The figure shows that all micro-service Pods restarted 0 times, which indicates that the applications managed by the vCluster are not affected if the vCluster crashes. The micro-service application continues to serve users even after the main component fails. This result demonstrates the stability of UnBound.

D. One Management Cluster with Multiple Member Clusters

We now explore the case of a single management cluster managing multiple member clusters. We deploy a management cluster with one VM that has 16 CPU cores and 32 GiB of memory. Then, we launch 100, 200, and up to 500 member clusters. Each member cluster uses only one VM with 2 CPU cores and 4 GiB of memory. We collect 2 hours of data for each round and average them to represent the results. We then run the experiment three times and present the average results of these three rounds in the figures. This experiment only shows the results of UnBound since the previous outcomes show a similar trend between our solution and OCM.

Figure 5(a) plots the cross-cluster network traffic between a management cluster and multiple member clusters. The cross-cluster network traffic grows as the number of member clusters increases, and is similar in cloud and fog scenarios. With 500 member clusters, the network traffic is around 166 KiB/sec in both cloud and fog cases. Figures 5(b) and 5(c) show the resource usage of the API server in the management cluster from where the work-agent in member clusters will pull the latest prescriptions. The CPU and memory usage also grow linearly with the number of member clusters. The CPU usage in the fog environment varies from 58 (0 member cluster), 123 (100 member clusters), to 338 m (500 member clusters), whereas the memory usage in the fog environment is from 309, 492, to 843 MiB.

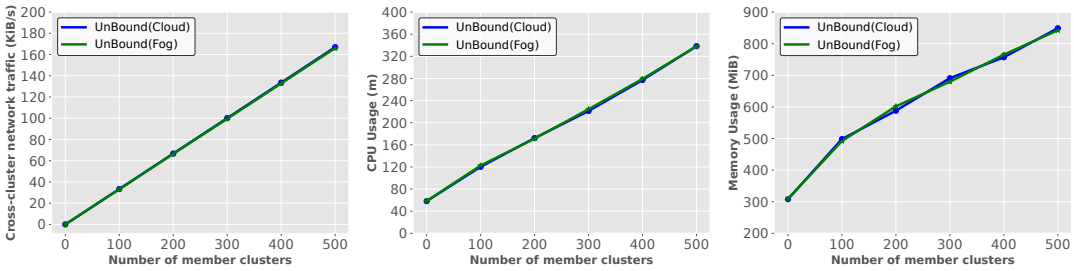
E. Multiple Management Clusters with One Member Cluster

We now study the scenario of multiple management clusters managing a single member cluster. We launch a member cluster consisting of one control plane node with 16 cores and 32 GiB of memory, and 100 worker nodes with 2 cores and 4 GiB of memory per worker. Next, we deploy 20, 40, and up to 100 management clusters, each with a single VM equipped with 2 CPU cores and 4 GiB of memory. Similar to Section V-D, we conduct three rounds of experiments; the outcome of each round is the average of 2 hours of data. The final results are the average of these three rounds.

Figures 6(a) and 6(b) illustrate the CPU and memory consumption of the API server in the member cluster, respectively. In the cloud/fog environment, the CPU usage of the API server exhibits superlinear growth, rising from 153 m/153 m (for 0 management cluster) to 251 m/251 m (for 20 clusters) and reaching 2308 m/2328 m (for 100 clusters). In contrast, memory usage growth is relatively slow compared to CPU usage, demonstrating a linear trend from 934 MiB to 1550 MiB in the cloud scenario. The growth in CPU usage results from the fact that agents of OCM in the member cluster send requests to the API server not only to the management cluster but also to the member cluster. Additionally, the components of vCluster also send requests to the API server. This surge in requests leads to increased processing time on the API server, potentially causing delays and timeouts that necessitate re-sending requests, further exacerbating the CPU usage. We however note that the absolute numbers remain reasonable, with a member cluster needing to allocate around 2 CPU cores and 1.5 GiB of memory when being a member of 100 federations simultaneously. One potential way to reduce these numbers could be to scale the number of API servers in the member cluster and enable load balancing to distribute requests across multiple API servers.

Figures 6(c) and 6(d) depict the utilization percentage of the entire member cluster, including control plane and worker nodes. The CPU usage percentages exhibit relatively flat growth until the member cluster is shared between 60 management clusters. Subsequently, CPU usage increases sharply due to the high load on the API server in the control plane and Kube-proxy in each node. At the same time, the percentages of memory cluster utilization rise steadily from 16% to 33%.

We conclude that, although not totally negligible, the performance overhead of UnBound for both the management and the member clusters remains reasonable. This demonstrates the feasibility of realizing our vision of fog meta-federations capable of spanning entire countries or even continents.

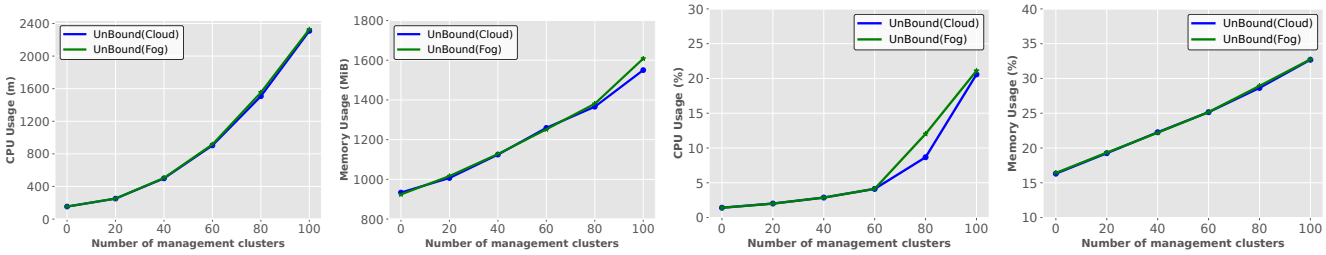


(a) Cross-cluster network traffic (collect in management cluster).

(b) CPU usage of API server in management cluster.

(c) Memory usage of API server in management cluster.

Fig. 5. Performance of UnBound with one management cluster managing multiple member clusters: Cross-cluster network traffic (a), and CPU and memory usage of the API server in the management cluster (b)(c).



(a) CPU usage of API server in member cluster.

(b) Memory usage of API server in member cluster.

(c) CPU utilization rate of member cluster.

(d) Memory utilization rate of member cluster.

Fig. 6. Performance of UnBound with multiple management clusters managing the same member cluster: CPU and memory usage of the API server in the member cluster (a)(b), and CPU and memory usage of the whole member cluster (c)(d).

VI. Conclusion

This paper presents meta-federations which enable fog clusters to federate their resources with one another in a very flexible way. We propose UnBound, a solution for meta-federations with scalability and different levels of multi-tenancy management in mind. Extensive experiments with deployments up to 500 *clusters* show that UnBound achieves inter-user and inter-federation isolation while maintaining performance comparable to the original Open Cluster Management and acceptable overhead levels. We have made our project's source code available under a liberal open-source license[6].

Acknowledgment

Experiments presented in this paper 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).

References

[1] Arif Ahmed, HamidReza Arkian, Davaadorj Battulga, Ali J. Fahs, Mozhdeh Farhadi, Dimitrios Giouroukis, Adrien Gougeon, Felipe Oliveira Gutierrez, Guillaume Pierre, Paulo R. Souza Jr, Mulugeta Ayalew Tamiru, and Li Wu. Fog Computing Applications: Taxonomy and Requirements. *CoRR*, 2019. http://arxiv.org/abs/1907.11621.

[2] Franck Cappello, E. Caron, M. Dayde, F. Desprez, Y. Jegou, P. Primet, E. Jeannot, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, B. Quetier, and O. Richard. Grid'5000: A Large Scale and Highly Reconfigurable Grid Experimental Testbed. In *Proc. IEEE/ACM Intl Workshop on Grid Computing*, 2005.

[3] Google LLC. Online Boutique. https://reurl.cc/z670ga. cited Sep 2024.

[4] Stephen Hemminger. Network Emulation with NetEm. In *Proc. Australia's National Linux Conference*, 2005.

[5] Marco Iorio, Fulvio Risso, Alex Palesandro, Leonardo Camiciotti, and Antonio Manzalini. Computing Without Borders: The Way Towards Liquid Computing. *IEEE Trans. on Cloud Computing*, 11(3), 2023.

[6] Kubernetes Multitenancy Working Group. The Hierarchical Namespace Controller. https://reurl.cc/QZV7AZ, Sep 2024.

[7] Kubernetes SIGs. Kubernetes Metrics Server. https://reurl.cc/RrmAGG. cited Sep 2024.

[8] Loft Labs, Inc. Kiosk. https://reurl.cc/GKn2qW. Sep 2024.

[9] Loft Labs, Inc. Virtual Kubernetes Clusters. https://reurl.cc/kOE19q. cited Sep 2024.

[10] Loft Labs, Inc. vCluster Becomes First Certified Kubernetes Distribution for Virtual Kubernetes Clusters. https://reurl.cc/4j1YpK, cited Sep 2024.

[11] Karim Manaouil and Adrien Lebre. Kubernetes and the Edge? Technical Report RR-9370, Inria Rennes – Bretagne Atlantique, 2020.

[12] Multicluster Special Interest Group. KubeFed: Kubernetes Cluster Federation. https://reurl.cc/6vdD5y, Sep 2024.

[13] Berat Can Şenel, Maxime Mouchet, Justin Cappos, Timur Friedman, Olivier Fourmaux, and Rick McGeer. Multitenant Containers as a Service (CaaS) for Clouds and Edge Clouds. *IEEE Access*, 11, 2023.

[14] Mulugeta Ayalew Tamiru, Guillaume Pierre, Johan Tordsson, and Erik Elmroth. mck8s: An Orchestration Platform for Geo-Distributed Multi-Cluster Environments. In *Proc. ICCCN*, 2021.

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

[16] The Karmada Authors. Karmada. https://reurl.cc/RyW9rz, Sep 2024.

[17] The Kubernetes Authors. Kubernetes. https://reurl.cc/L48ovL. Sep 2024.

[18] The Kubernetes Authors. Custom Resources. https://reurl.cc/q0Wmjy, cited Sep 2024.

[6]UnBound - https://github.com/CKHuang-public/UnBound

[19] The Kubernetes Authors. Multicluster Special Interest Group. https://reurl.cc/Xq4dx0, cited Sep 2024.
[20] The Kubernetes Authors. Resource Quotas. https://reurl.cc/WRaekL, cited Sep 2024.
[21] The Kubernetes Authors. Multi-Tenancy. https://reurl.cc/RWrRyg, Sep 2024.
[22] The Open Cluster Management Authors. ManifestWork. https://reurl.cc/YEkyQX, cited Sep 2024.
[23] The Open Cluster Management Authors. Open Cluster Management. https://reurl.cc/mMkE7W, cited Sep 2024.
[24] The Virtual Kubelet Authors. Virtual Kubelet. https://reurl.cc/m0RK1Y, cited Sep 2024.
[25] Shanhe Yi, Cheng Li, and Qun Li. A Survey of Fog Computing: Concepts, Applications and Issues. In *Proc. workshop on mobile big data*, 2015.