# Evaluating IPv6 Support in Kubernetes

Vojdan Kjorveziroski
Faculty of Computer Science and Engineering, Ss. Cyril and Methodius University
Skopje, North Macedonia
vojdan.kjorveziroski@finki.ukim.mk

Anastas Mishev
Faculty of Computer Science and Engineering, Ss. Cyril and Methodius University
Skopje, North Macedonia
anastas.mishev@finki.ukim.mk

Sonja Filiposka
Faculty of Computer Science and Engineering, Ss. Cyril and Methodius University
Skopje, North Macedonia
sonja.filiposka@finki.ukim.mk

***Abstract* — Kubernetes is one of the most popular container orchestrators today, but IPv4 address scarcity and the lucrative reseller market introduce problems for cluster administrators who would like to deploy publicly available applications, without resorting to NAT. We discuss the extent of IPv6 support in recent versions of Kubernetes and offer solutions for IPv6 dual-stack implementation. We also provide a reference dual-stack IPv6 Kubernetes network architecture that can be employed in existing or new clusters.**

***Keywords — cloud computing, IPv6, Kubernetes, containers, overlay networks***

## I. INTRODUCTION

The rapid progress achieved in the last few decades in both compute capacity and network performance has led to a paradigm shift, from static bare-metal servers, through long-lived virtual machines to ephemeral containers that have gained a noticeable popularity in recent years. Even though the ultimate goal is to make a more efficient use of the underlying infrastructure both in terms of performance and security, nonetheless various issues have arisen during the quest for running as many applications as possible on the same hardware.

To facilitate easier management of large fleets of ephemeral containers distributed among different computing nodes, various container orchestrators have been developed, some of which started their lives in the laboratories of popular internet companies, others as community-based open-source efforts. One such example which enjoys a large user-base today is Kubernetes [1], [2], a project whose roots can be traced back to Google and later open-sourced and transformed into a community led effort. All container orchestrators, regardless of their underlying architecture, aim to ease the process of deploying new containers, dealing with scheduling, resource limitations, security, fault-tolerance, and networking. However, a persistent issue that poses challenges both for administrators and end-users alike is the ephemerality of the containers and the effects that this behavior has on network connections, especially when multiple containers are backing a single user-facing application. Since containers can be started and stopped at any point in time, additional helper components are deployed to ensure the routing of network packets only to those instances that are currently active and healthy. Publishing of such containerized applications, allowing public access to the wider world instead of only to the containers that are part of the same container network interface (CNI), is also a challenge. The various mitigation techniques that have been employed to alleviate the IPv4 address shortages, such as network address translation (NAT) further complicate this, breaking end-to-end connectivity, reducing transparency, and making troubleshooting harder.

While customers of the major cloud providers offering managed Kubernetes services have the option of effortlessly leasing IPv4 addresses through complementary load-balancing products, on-premise deployments of container orchestrators must either acquire a suitably large IPv4 block of addresses, or find alternatives for publicly exposing the hosted applications. Unfortunately, the dramatic growth in IPv4 prices as a result of their increasing scarcity and a thriving reseller market [3] has made managing IPv4 address blocks infeasible for many organizations that do not already possess them. Of course, the best solution to this problem is to simply adopt IPv6, thus overcoming any addresses shortages, eliminating the need for workarounds that break end-to-end connectivity, such as NAT. Unfortunately, the current state of IPv6 adoption is less than satisfactory, and according to Google's statistics only 34.33% of its users use IPv6 for accessing the site [4]. The low adoption rate is further confirmed by the fact that many popular web sites do not support IPv6 at all, mandating the use of dual-stack solutions for coexistence of IPv4 and IPv6 for the foreseeable future. Frustrated with the state of things, enthusiast groups have even published monitoring pages that track IPv6 support across the most popular sites [5], in an effort to publicize the problem. Cloud providers have also shown varying levels of support for this new protocol, adding support only recently [6], handing out very limited address spaces [7], or limiting the availability across different product families, thus forcing users to again use mitigation techniques [8].

For system administrators to fully implement IPv6, all intermediate components need to support it, in tandem with the underlying core infrastructure. One of the most popular container orchestrators today, Kubernetes offers beta level support for dual-stack networking, a feature enabled by default since version 1.21, released in the spring of 2021. However, additional tools are required to allow seamless integration with existing network infrastructure and hosting of public facing applications within Kubernetes clusters.

The aim of this paper is to evaluate the level of support that the Kubernetes project offers for IPv6 networking, identify major challenges for implementing IPv6 Kubernetes connectivity in existing infrastructures and offer a sample architecture which can be reused to achieve this goal. The rest of this paper is organized as follows: in section II we introduce the necessary Kubernetes components and networking concepts, reflecting on existing work that has been published in this field, as well as offer ways in which IPv6 connectivity can be achieved even without support from upstream internet service providers. In section III we describe the current state of IPv6 support in Kubernetes, discussing additional third-party components relevant for on-premise installations that ease the process of hosting public facing applications. Taking into account the various characteristics and maturity levels of these addons, in section IV we propose a sample Kubernetes

978-1-6654-2585-8/21/$31.00 ©2021 IEEE

IPv6 architecture which can be used both in current and new deployments, allowing seamless integration with existing infrastructure, and automatic assignment of routable IPv4 and IPv6 addresses to applications instantiated in the cluster. We conclude the paper with section V, outlining plans for future work, and areas for improvement.

## II. Background and Related Work

Kubernetes as one of the most popular container orchestrators today has a complex architecture comprised of different components running on nodes with specific roles [9]. In the subsections that follow we first briefly explain the necessary concepts, before continuing with an overview of existing literature dealing with IPv6 migration techniques and Kubernetes networking.

### A. Essential Kubernetes Components

Kubernetes is a container orchestration software that allows container deployment among different nodes joined together in a cluster. The atomic unit of Kubernetes is called a pod, which represents at least one, possibly more containers that share an execution context. The master nodes are responsible for storing the cluster information, listening for state changes, and deciding which worker nodes to execute the instantiated pods. On the other hand, workers communicate with the masters and using a container runtime execute the scheduled workloads.

### B. Kubernetes Use-Cases and Networking

Apart from the traditional use-case for deploying microservice based applications, Kubernetes has been successfully used as an underlying system for many Platform as a Service (PaaS) and Software as a Service (SaaS) offerings in the cloud [10]. An interesting development is the increasing popularity of Kubernetes for edge-based [11] workloads where resources are severely constrained, requiring alternative approaches and development of more lightweight Kubernetes distributions [12]. Nonetheless, the core features remain unchanged and the requirement for external access to deployed workloads is still present.

Kubernetes supports different types of network addresses [13], regulating access to the deployed applications and the underlying backing pods, as well as providing higher level abstractions such as load-balancers with an associated rotation strategy, in an effort to deal with the inherent ephemerality of the spawned containers. Access to a pool of publicly routable IP addresses is a requirement for effortless and automatic publishing of deployed applications, without the need for any manual infrastructure configuration for port forwarding or NAT rules definition. Unfortunately, the scarcity of IPv4 address space and low adoption rates of IPv6, combined with the limited support for IPv6 by the various Kubernetes addons needlessly complicate such efforts.

### C. Related Work

The scarcity of IPv4 space has been a known problem for decades, and efforts to mitigate it have preceded its exhaustion. Richter et al. [14] describe the limited availability of IPv4 address space and offer possible mitigation techniques. One such popular option today with users located where IPv6 adoption levels are low among internet service providers (ISPs) is the use of tunnel brokers [15] which can allocate a dedicated IPv6 space for free, upon request. However, in such scenarios other issues arise such as questions relating to performance and security [16], since the tunnel broker has access to all the network traffic in addition to the ISP. Sookun et al. [17] and Kumar et al. [18] analyze the performance aspects of such IPv4 to IPv6 migration techniques using network simulation. In the Kubernetes camp, the authors of [19] benchmark various CNI plugins that allow the creation of overlay networks, facilitating internal and external communication between and towards pods. These analyses are focused only on IPv4 connectivity.

While there is extensive literature dealing with the adoption aspects of IPv6 on one hand, and network optimization of Kubernetes clusters on the other, unfortunately, to the best of our knowledge, no comprehensive analysis tackling these subjects in relation to one another exists yet.

## III. Kubernetes IPv6 Support and Service Publishing

When an application is deployed in a Kubernetes cluster, there are three primary, production-ready, ways in which it can be exposed publicly so that it can be accessed outside the internal cluster network. Since all approaches have their benefits and drawbacks, an appropriate choice needs to be made, depending on the network requirements of the application.

For web-based applications that use HTTP as their transport protocol, an Ingress object can be instantiated, acting as a publicly reachable reverse proxy with access to the internal cluster network and thus the deployed applications as well. However, this approach is limited when it comes to applications that use other application-level protocols. In such cases, the use of either a LoadBalancer service, or a ClusterIP service with the ExternalIP parameter needs to be set [20], so that direct access to the instantiated backing pods is provided. While these two approaches are similar, when using external IPs with a ClusterIP type service, the allocation must be made manually, and explicitly set by the administrator, due to the lack of support for automatic IP address management (IPAM). Contrary to this, the LoadBalancer service type offers support for automatic provisioning and IP allocation, provided that the underlying infrastructure is appropriately configured as well. Most cloud providers today integrate the Kubernetes load balancing behavior with their standalone load balancing products, which are often marketed as separate features, and billed independently. Users of on-premise Kubernetes clusters must install a separate load-balancing plugin which can perform the task of IP address allocation and appropriate routing within the internal network, in cooperation with the CNI.

In cases when many applications hosted within the cluster need to be publicly accessible, the shortage of IPv4 addresses and their ever-increasing price can make such efforts completely impossible or prohibitively expensive. Kubernetes has been improving IPv6 support with each new version since 1.9, where IPv6-only support was introduced as an alpha feature, before graduating to beta in 1.18. However, IPv6 dual-stack support was introduced later, first as an alpha feature in 1.16, before graduating to beta only recently, in version 1.21, and being enabled by default for new deployments [21]. Moreover, to implement an IPv6 ready cluster, explicit support from the underlying CNI plugin responsible for establishment of the container network is required as well. Currently only a handful of such plugins support dual-stack operation, with different limitations and behaviors, depending

on how they are implemented, and whether they act on layer-2 or layer-3.

Even though the use of an IPv6 compliant CNI plugin would indeed offer IPv6 reachability over the internet, explicit support for IPv6 load-balancer addresses is necessary as well, both when it comes to cloud-based or on-premise cluster setups. For self-managed installations, this requires the installation of a load-balancing addon, which would be capable of allocating routable addresses on demand. There are multiple approaches that can be taken, the most popular ones being:

- Allocating additional IPs to the existing network interfaces of the Kubernetes nodes, with the network plugin responding to ARP requests of IPv4 services, and NDP requests of IPv6 services.
- Establishment of a peering session with a border router, advertising the allocated IPs of the created services.

While there are multiple load-balancing plugins in the CNCF portfolio [22] that are capable of supporting the creation of LoadBalancer Kubernetes objects, the most widely used in practice is MetalLB, which traces its roots back to Google as well, similar to Kubernetes itself. MetalLB supports both modes of operation described above for IPv4, but only the first one for IPv6, while support for direct peering sessions for purposes of IPv6 advertisements is currently under development [23].

## IV. Sample IPv6 Kubernetes Architecture

In this section we present a sample IPv6 Kubernetes cluster architecture by combining a compliant CNI plugin with a load-balancing addon that offers IPv6 support. By directly integrating with the existing network infrastructure and reusing devices that are already in-place we ensure high-performance and a familiar management process.

### A. Calico as a CNI Plugin

Calico is a layer-3 implementation of a CNI plugin which by default uses BGP full-mesh topology to peer Kubernetes nodes between themselves and distribute network routes within the cluster. Additionally, full support for Kubernetes network policy objects is provided, allowing the creation of inbound and outbound filtering rules which are automatically translated to the appropriate IPTables representation by the Calico daemon running on each affected node. Unlike the majority of other alternatives, Calico also offers full IPv6 dual-stack support [24], the reason why we have selected it for our sample architecture.

### B. Avoiding Source NAT

During the deployment process of every Kubernetes cluster, the administrator should choose appropriate IPv4 and IPv6 subnets which will be reserved for pod and Service IP address allocation, respectively. The pod IP range is further subdivided into equal smaller subnets, each one allocated to a specific node within the cluster and advertised to the others using the previously established BGP session. In this manner, each pod deployed on a given node is assigned an IP address from the dedicated subnet for that given cluster member. Care must be taken to select sufficiently large subnets, since this can limit the number of pods that can be scheduled on a given node. A graphical representation of this behavior is shown in Fig. 1. However, since by default BGP sessions are

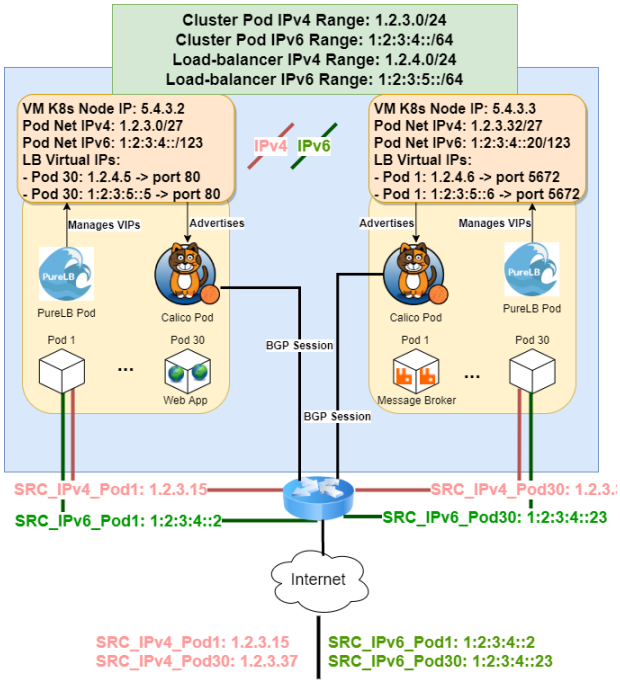


Fig. 1. IPv4 and IPv6 pod communication without NAT

established only between the cluster nodes, other devices are not aware of the existence of these routes, and communication with hosts located outside the cluster requires source NAT, where the source IP will be the routable IP address of the Kubernetes node hosting the pod [25]. As discussed previously, this breaks end-to-end connectivity and NAT is something that we would like to avoid. Fortunately, Calico has an option of peering with an external router as well [26], giving the option of advertising the allocated IP ranges, disabling NAT, and establishing true end-to-end connectivity. This is reflected in Fig. 1 where the source IP of a request sent from within the pods does not change as it traverses network boundaries. However, if not enough IPv4 addresses are available, then source NAT will have to be performed by the gateway, but at least this will be transparent within the local network.

It is not expected that every deployment of Kubernetes will have access to a core router speaking BGP to advertise the assigned prefixes, but there are free workarounds that can be implemented. For example, there are many open-source firewall solutions that support the installation of a BGP speaker, such as PfSense, OPNSense, VyOS. In case no native IPv6 connectivity is offered by the ISP, or only a limited number of addresses are provided, tunnel brokers can be utilized, some of which offer /48 subnets upon request.

### C. Optimizing Performance

By utilizing a BGP full-mesh topology, a significant overhead will be placed on each node in case of a large Kubernetes cluster, since each member will have to maintain BGP sessions with all others. The solution to this old and well-known problem is to utilize a route reflector, or multiple route reflectors for redundancy, which would in turn redistribute the routes to the Kubernetes nodes. In this manner, each node only maintains BGP sessions with the route reflectors, significantly lowering the computing overhead. The Calico CNI plugin by default supports peering with external route reflectors and can even convert an

existing Calico BGP speaker to a route reflector [27]. In terms of on-premise deployments, many of the open-source BGP implementations do support acting as a route reflector as well.

### *D. Choosing a Load-Balancing Option*

The last remaining piece of the puzzle is choosing the load-balancer plugin which would be able to automatically allocate routable IP addresses to instantiated services and advertise these to the border router. Since the previously discussed MetalLB implementation still does not support BGP peering and advertisement of IPv6 prefixes, alternatives need to be found. Furthermore, even if it did support peering, this would complicate things, since multiple sessions would be required for each node, one for Calico and one for MetalLB. PureLB is another load-balancer implementation for Kubernetes which aims to solve the double peering problem by allocating addresses to a virtual interface, which in turn can be advertised by using a routing protocol. In our case we can leverage Calico to advertise these ranges as well, by utilizing the support for load balancer IP advertisements [28], present since version 3.18.

## V. Conclusion

Even after 10 years since the World IPv6 Day, IPv6 adoption is still low, and many residential ISPs and popular websites do not support this protocol. Faced with IPv4 address space exhaustion and the lucrative business developed around IPv4 trading, customers are faced with little choice than to embrace IPv6, especially new service providers whose goal is to serve large amounts of customers without employing mitigation techniques such as NAT.

In this paper we evaluated the state of IPv6 support in Kubernetes, provided a sample architecture using open-source components such as the Calico CNI and the PureLB load-balancing plugin, and optimized the performance by utilizing route-reflectors. We also discussed alternative ways of obtaining IPv6 support in cases where the ISP does not support this newer protocol, through tunnel brokers.

An area for future research is the performance comparison between various CNI plugins when it comes to IPv6, and how this differs with results obtained in existing literature relating to their support of IPv4.

## Acknowledgement

The work presented in this paper has received funding from the Faculty of Computer Science and Engineering under the “SCAP” project.

## References

[1] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, ‘Borg, Omega, and Kubernetes: Lessons learned from three container-management systems over a decade’, *Queue*, vol. 14, no. 1, pp. 70–93, Jan. 2016, doi: 10.1145/2898442.2898444.

[2] M. Lukša, *Kubernetes in action*. Shelter Island, NY: Manning Publications Co, 2018.

[3] I. Livadariu, A. Elmokashfi, and A. Dhamdhere, ‘On IPv4 transfer markets: Analyzing reported transfers and inferring transfers in the wild’, *Computer Communications*, vol. 111, pp. 105–119, Oct. 2017, doi: 10.1016/j.comcom.2017.07.012.

[4] ‘IPv6 – Google’. https://www.google.com/intl/en/ipv6/statistics.html (accessed Aug. 24, 2021).

[5] L. Haugen, ‘Why No IPv6? The World’s Largest Websites Lacking IPv6 Support’. https://whynoipv6.com/ (accessed Aug. 25, 2021).

[6] ‘Google Cloud - July 20 2021 Announcement’, *Google Cloud*. https://cloud.google.com/vpc/docs/release-notes (accessed Aug. 25, 2021).

[7] ‘Azure Public IP address prefix - Azure Virtual Network’. https://docs.microsoft.com/en-us/azure/virtual-network/public-ip-address-prefix (accessed Aug. 25, 2021).

[8] ‘Overview of IPv6 - Azure Load Balancer’. https://docs.microsoft.com/en-us/azure/load-balancer/load-balancer-ipv6-overview (accessed Aug. 25, 2021).

[9] T.-T. Nguyen, Y.-J. Yeom, T. Kim, D.-H. Park, and S. Kim, ‘Horizontal Pod Autoscaling in Kubernetes for Elastic Container Orchestration’, *Sensors*, vol. 20, no. 16, Art. no. 16, Jan. 2020, doi: 10.3390/s20164621.

[10] V. Medel, R. Tolosana-Calasanz, J. Á. Bañares, U. Arronategui, and O. F. Rana, ‘Characterising resource management performance in Kubernetes’, *Computers & Electrical Engineering*, vol. 68, pp. 286–297, May 2018, doi: 10.1016/j.compeleceng.2018.03.041.

[11] A. Palade, A. Kazmi, and S. Clarke, ‘An Evaluation of Open Source Serverless Computing Frameworks Support at the Edge’, in *2019 IEEE World Congress on Services (SERVICES)*, Milan, Italy, Jul. 2019, pp. 206–211. doi: 10.1109/SERVICES.2019.00057.

[12] S. Böhm and G. Wirtz, ‘Profiling Lightweight Container Platforms: MicroK8s and K3s in Comparison to Kubernetes’, presented at the 13th Central European Workshop on Services and their Composition, Bamberg, Germany, Mar. 2021.

[13] N. Nguyen and T. Kim, ‘Toward Highly Scalable Load Balancing in Kubernetes Clusters’, *IEEE Commun. Mag.*, vol. 58, no. 7, pp. 78–83, Jul. 2020, doi: 10.1109/MCOM.001.1900660.

[14] P. Richter, M. Allman, R. Bush, and V. Paxson, ‘A Primer on IPv4 Scarcity’, *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 2, pp. 21–31, Apr. 2015, doi: 10.1145/2766330.2766335.

[15] ‘Hurricane Electric Free IPv6 Tunnel Broker’. https://tunnelbroker.net/ (accessed Aug. 25, 2021).

[16] S. A. Abdulla, ‘Survey of security issues in IPv4 to IPv6 tunnel transition mechanisms’, *IJSN*, vol. 12, no. 2, p. 83, 2017, doi: 10.1504/IJSN.2017.083830.

[17] Y. Sookun and V. Bassoo, ‘Performance analysis of IPv4/IPv6 transition techniques’, in *2016 IEEE International Conference on Emerging Technologies and Innovative Business Practices for the Transformation of Societies (EmergiTech)*, Aug. 2016, pp. 188–193. doi: 10.1109/EmergiTech.2016.7737336.

[18] R. K. CV and H. Goyal, ‘IPv4 to IPv6 Migration and Performance Analysis using GNS3 and Wireshark’, in *2019 International Conference on Vision Towards Emerging Trends in Communication and Networking (ViTECoN)*, Mar. 2019, pp. 1–6. doi: 10.1109/ViTECoN.2019.8899746.

[19] R. Kumar and M. C. Trivedi, ‘Networking Analysis and Performance Comparison of Kubernetes CNI Plugins’, in *Advances in Computer, Communication and Computational Sciences*, Singapore, 2021, pp. 99–109. doi: 10.1007/978-981-15-4409-5_9.

[20] ‘Kubernetes Service External IPs’, *Kubernetes*. https://kubernetes.io/docs/concepts/services-networking/service/#external-ips (accessed Aug. 25, 2021).

[21] ‘Kubernetes IPv4/IPv6 dual-stack’, *Kubernetes*. https://kubernetes.io/docs/concepts/services-networking/dual-stack/ (accessed Aug. 25, 2021).

[22] ‘CNCF Cloud Native Interactive Landscape – Service Proxies’, *CNCF Cloud Native Interactive Landscape*. https://landscape.cncf.io (accessed Aug. 25, 2021).

[23] ‘Add support for MP BGP encoding for IPv4 and IPv6 by ykulazhenkov · Pull Request #590 · metallb/metallb’, *GitHub*. https://github.com/metallb/metallb/pull/590 (accessed Aug. 25, 2021).

[24] ‘Configure dual stack or IPv6 only’. https://docs.projectcalico.org/networking/ipv6 (accessed Aug. 25, 2021).

[25] ‘Kubernetes Cluster Networking’, *Kubernetes*. https://kubernetes.io/docs/concepts/cluster-administration/networking/ (accessed Aug. 25, 2021).

[26] ‘Configure BGP peering’. https://docs.projectcalico.org/networking/bgp (accessed Aug. 25, 2021).

[27] ‘Configuring Route Reflectors in Calico’, *Tigera*, Mar. 22, 2019. https://www.tigera.io/blog/configuring-route-reflectors-in-calico/ (accessed Aug. 25, 2021).

[28] ‘Advertise Kubernetes service IP addresses’. https://docs.projectcalico.org/archive/v3.18/networking/advertise-service-ips (accessed Aug. 25, 2021).