Networking Analysis and Performance Comparison of Kubernetes CNI Plugins

Ritik Kumar and Munesh Chandra Trivedi

Abstract Containerisation, in recent world, has proved to be a better aspect to deploy large-scale applications in comparison with virtualisation. Containers provide a small and compact environment, containing all libraries and dependencies, to run an application. It has also come to acknowledgement that application deployment on a multi-node cluster has proved to be more efficient in terms of cost, maintenance and fault tolerance in comparison with single-server application deployment. Kubernetes play a vital role in container orchestration, deployment, configuration, scalability and load balancing. Kubernetes networking enable container-based virtualisation. In this paper, we have discussed the Kubernetes networking in detail and have tried to give a in-depth view of the communication that takes place internally. We have tried to provide a detailed analysis of all the aspects of Kubernetes networking including pods, deployment, services, ingress, network plugins and multi-host communication. We have also tried to provide in detail comparison of various container network interface(CNI) plugins. We have also compared the results of benchmark tests conducted on various network plugins keeping performance under consideration (Ducastel, Benchmark results of Kubernetes network plugins (CNI) over 10 Gbit/s network [1]).

Keywords Networking analysis · Performance comparison · Kubernetes · CNI · Docker · Plugins · Containers · Pod · Virtualisation · Cluster

R. Kumar (✉) · M. C. Trivedi
Department of Computer Science and Engineering,
National Institute of Technology, Agartala, Agartala, India
e-mail: ritikkumarrrp@gmail.com

M. C. Trivedi
e-mail: drmunesh.nita@gmail.com
URL: https://bit.ly/2Zz4pB2

S. K. Bhatia et al. (eds.), *Advances in Computer, Communication and Computational Sciences*, Advances in Intelligent Systems and Computing 1158, https://doi.org/10.1007/978-981-15-4409-5_9

1 Introduction

With increasing traffic to applications online scalability, load balancing, fault tolerance, configuration, rolling updates and maintenance remain an area of concern. Traditional method of solving these issue makes virtualisation come into play. Creating virtual machines requires hardware resources, processing power and memory same as an operating system would require. Requiring huge amount of resources limits the number of virtual machines that can be created. Moreover, the overhead of running virtual machines remains an area of deep concern. Although virtual machines are still widely used in the Infrastructure as a Service (IaaS) space, we see Linux containers dominating the platform as a Service (PaaS) landscape [2].

Containerisation comes to the rescue considering such huge disadvantages of virtualisation. In a container-based virtualisation, we create small compact environment comprising of all the libraries and dependencies to run the application. Various research studies have shown that containers incur negligible overhead and have performance at par with native deployment [3] and take much less space [4].

In real-world scenario, a large-scale application deployment as a single service is not feasible rather it should is fragmented into small microservices and deployed in a multi-node cluster inside containers using container orchestration tools such as Kubernetes and docker swarm. Kubernetes is most portable, configurable and modular and is widely supported across different public clouds like Amazon AWS, IBM Cloud, Microsoft Azure, and Google Cloud [5]. Deployment on multi-node cluster not only reduces the overhead but also make the deployment and maintenance economically viable.For instance, large-scale applications such as eBay and pokemon go are deployed on Kubernetes.

Containers can be launched in a few seconds of duration and can be shortlived. Google Search launches about 7,000 containers every second [6] and a survey of 8 million container usage shows that 27% of containers have life span less than 5 min and 11% less than 1 min [7]. Container networking becomes very important when containers are created and destroyed so often [8]. Systems like Kubernetes are designed to provide all the primitives necessary for microservice architectures, and for these the network is vital [9]. Using Kubernetes, we can create a strongly bind cluster with one master node and multiple slave nodes.

In this chapter, we have discussed in detail about all the aspects of Kubernetes networking, various third-party plugins that kubernetes support, container to container communication, container to host communication and host-to-host communication. At the same time, we have compared the available CNI plugins on various parameters to provide an in-depth analysis Sect. 4.

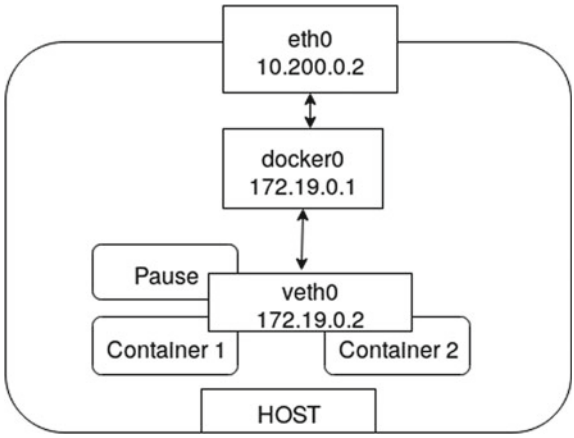

Fig. 1 Pod network

2 Components of Kubernetes Networking

2.1 Pod Network

In Kubernetes networking, pod is the most fundamental unit similar to what atom is to matter or a cell is to human body. A pod is a small environment consisting of one or more containers, an infrastructure (Pause) container and volumes. The idea of pod has been included in Kubernetes to reduce the overhead when two containers need to communicate very frequently. The infrastructure container is created to hold the network and inter-process communication namespaces shared by all the containers in the pod [10]. Inside a pod all the containers communicate through a shared network stack. Figure 1 depicts veth0 as a shared network stack between two containers inside a pod. Both the containers are addressable with IP 172.19.0.2 from outside the cluster. As both the containers have the same IP address, they are differentiated by the ports they listen to. Apart from this, each pod has a unique IP using which is referenced by other pods in cluster and data packets are transmitted and received. The shared network stack veth0 is connected to the eth0 via the cbr0 bridge (a custom bridge network) for communication with other nodes in the cluster. An overall address space is assigned for bridges in each node and then each bridge is assigned an address within this space depending on the node. The concept of custom bridge network is adopted instead of choosing docker0 bridge network to avoid IP address conflict between bridges in different nodes or namespaces.

Whenever a packet is transmitted, it goes via the router or external gateway which is connected to each of the nodes in cluster. Figure 2 shows two nodes connected via the router or external gateway. When a pod sends a data packet to a pod in different node in the network, it goes via the router. On receiving a data packet at the gateway, routing rules are applied. The rules specify how data packets are destined for each bridge and how they should be routed. This combination of virtual network interface,

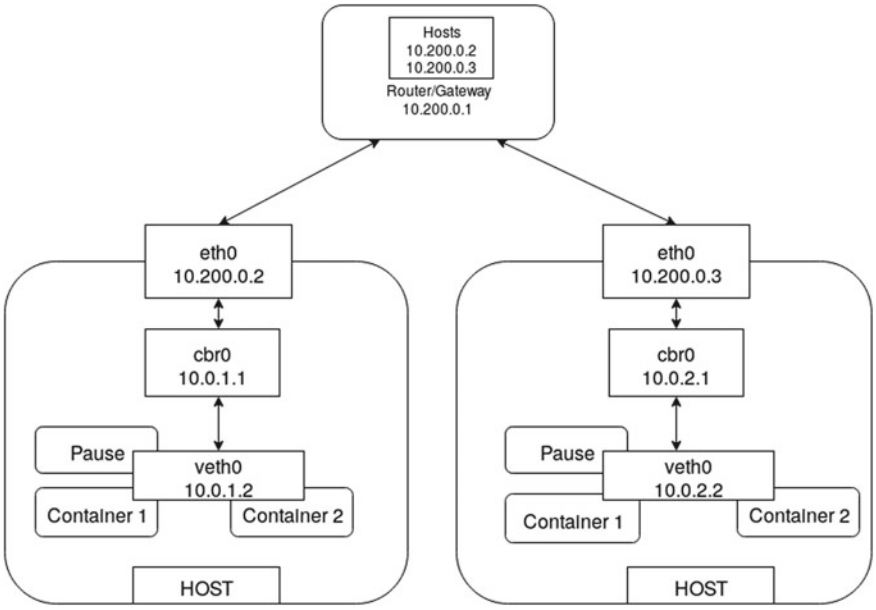

Fig. 2 Pod network

bridges and routing rules is known as overlay network. In Kubernetes, it is known as a pod network as they enable pods on different nodes to communicate with each other.

2.2 Service Networking

The pod networking is robust but not durable as the pods are not long lasting. The pods IP address cannot be taken as an endpoint for sending packets in the cluster as the IP changes everytime a new pod is created. Kubernetes solves this problem by creating services. A service is a collection of pods. It acts as a reverse-proxy load balancer [11]. The service contains a list of pods to which it redirects the client requests. Kubernetes also provides additional tools to check for health of pods. Service also enables load balancing, i.e. the external traffic is equally distributed among all the pods. There are four types of services in Kubernetes ClusterIP, NodePort, LoadBalancer and ExternalName. Similar to the pod network, the service network is also virtual [12].

Each service in Kubernetes has an virtual IP address. Kubernetes also provides an internal cluster DNS which maps service IP address to service name. The service IP is reachable from anywhere inside the cluster. Each node in cluster has a kube-proxy running on it. The kube-proxy is a network proxy that passes traffic between the

client and the servers. It redirects client requests to server. The kube-proxy supports three modes of proxy, i.e. userspace, IPTables and IPVS.

Now, whenever a packet is sent from client to server it passes the bridge at the client node and then moves to the router. The kube-proxy running on the nodes determines the path in which packet is to be routed. The packet is then forwarded to its destination. The IPTables rules are applied and the packet is redirected to its destination.

Thus, this makes up the service network. Service proxy system is durable and provides effective routing to packets in the network. The kube-proxy runs on the each node and is restarted whenever it fails. The service network ensures a durable and secures means of transmission of data packets within the cluster.

2.3 Ingress and Egress

The pod network and service network are complete within itself for communication between the pods inside the cluster. However, the services cannot be discovered by the external traffic. In order to access the services using HTTP or HTTPS requests, ingress controller is used. The traffic routing is controlled by the rules defined by the ingress resource. Figure 3 depicts how external requests are directed to the services via ingress.

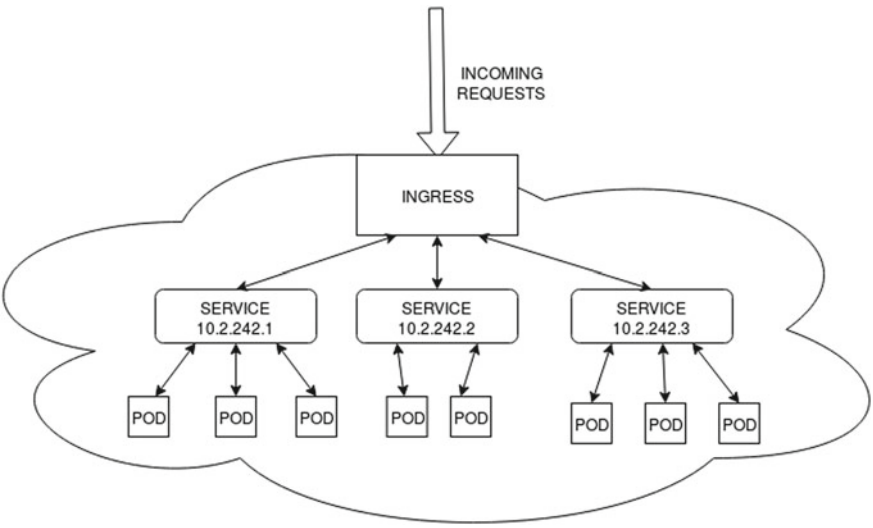

Fig. 3 Ingress

Egress network helps pods communicate to outside world. Egress network helps to establish communication path between the pods in the network to other services outside the cluster on the top of network policies.

3 Kubernetes Architecture

A Kubernetes cluster comprises of a master node and one or more worker nodes. The master node runs the kube-api server, kube-control manager, kube-scheduler and etcd [11]. The kube-api server acts as an interface between the user and kubernetes engine. It creates, reads, update and delete pods in the cluster. It also provides a shared frontend through which all the components interact. The control manager watches the state of the cluster and shift the state to the desired state by maintaining the desired number of pods in the cluster. The scheduler is responsible scheduling pods to be most appropriate nodes in the cluster. The etcd is key-value database which stores configuration data of the cluster. It also represents the state of the cluster. Each worker node has a kubelet running on it. Kubelet is a Kubernetes agent running on each node which is responsible for communication between the master and the worker nodes. The kubelet executes commands received from the master node on the worker nodes.

4 Container Network Interfaces (CNI)

Kubernetes offers a lot of flexibility in terms of kind of networking we want to establish. Instead of providing its own networking, Kubernetes allow us to create a plugin-based generic approach for networking of containers. CNI consists of a specification and libraries for writing plugins to configure network interfaces in Linux containers, along with a number of supported plugins [13]. The CNI is responsible for IP address assignment in the cluster and providing routes for communication inside the cluster.

CNI is capable of addressing containers IP addresses without resorting to network address translation. The various netwrok plugins built on top of CNI which can be used with Kubernetes are Flannel, Calico, Weavenet, Contiv, Cilium, Kube-router and Romana. The detailed comparison of the plugins is shown in below [14–20].

	Flannel	Calico	WeaveNet	Contiv	Cilium	Kube Router	Romana
Language	Go	Go	Go	Go	Go	Go	Bash
IP Version	IPV4	IPV4 IPV6	IPV4	IPV4 IPV6	IPV4 IPV6	IPV4	IPV4
Network Policy	None	Ingress Egress	Ingress Egress	Ingress Egress	Ingress Egress	Ingress	Ingress Egress
Encryption	None	None	NaCl crypto Libraries	None	Encryption with IPSec tunnels	None	None
Network model	Layer 2	Layer 3	Layer 2	Layer 2 Layer 3	Layer 2	Layer 3	Layer 3
Layer 2 Encapsulation	VXLAN	–	VXLAN	VXLAN	VXLAN Geneve	–	–
Layer 3 routing	IP tables	IP tables Kube-proxy	IPTables Kube-proxy	IP tables	BPF Kube-proxy	IP tables IPVS IP sets	IP tables
Layer 3 Encapsulation	–	IPIP	Sleeve	VLAN	–	IPVS/ LVS DR mode, GRE/IPIP	–
Layer 4 route distribution	–	BGP	–	BGP	–	BGP	BGP, OSPF
Load balancing	–	Available	Available	Available	Available	Available	Available
Database	K8s ETCD	Storing state in K8s API datastore	Storage in pods	K8s ETCD	K8s ETCD	K8s ETCD	K8s ETCD
Platforms	Windows Linux	Windows linux	Linux	Linux	Linux	Linux	Linux

5 Table Showing Comparison of CNI Plugins

Conclusions from the above table

1. All plugins use ingress and egress network policies except Flannel.
2. Only Weavenet and Cilium offer encryption, thus providing security while transmission. The encryption and decryption increase the transmission time for packets.
3. Weavenet and Contiv offers both Layer 2 as well as Layer 3 encapsulation.
4. Romana does not offer any encapsulation.
5. Flannel, WeaveNet and Cilium is not supported by any Layer 4 route.
6. All network plugins have inbuilt load balancer except flannel.

6 Performance Comparison

The main aim of all CNI plugins is to achieve container abstraction for container clustering tools. Availability of so many network plugins makes it difficult to choose one considering advantages and disadvantages. In this section, we have compared the CNI plugins, using benchmark results obtained on testing network plugins on Supermicro bare-metal servers connected through a supermicro 10 Gbit switch, keeping performance under consideration [1].

Testing Setup

1. The test was performed by Alexis Ducastel on three supermicro bare-metal servers connected through a supermicro 10 Gbit switch [1].
2. The Kubernetes (v1.14) cluster was set up on Ubuntu 18.04 LTS using docker 18.09.2.
3. The test compares Flannel (v0.11.0), Calico (v3.6), Canal (Flannel for networking + Calico for firewall) (v3.6), Cilium (v1.4.2), Kube-Router (v0.2.5) and Weavenet (v2.5.1).
4. The tests were conducted on configuring server and switch with jumbo frames activated (by setting MTU to 9000).
5. For each parameter, the tests were conducted thrice and the average of the three tests were noted.

6.1 Testing Parameters

1. The performance was tested on the basis of bandwidth offered by the plugins to different networking protocols such as TCP, UDP, HTTP, FTP and SCP.
2. The plugins were also compared on basis of the resource consumption such as RAM and CPU.

6.2 Test Results

6.2.1 Bandwidth Test for Different Network Protocol

In this test, the different network plugins were tested on the basis of various network protocol such as TCP, UDP, HTTP, FTP and SCP. The bandwidth offered by the plugins to each networking protocol was recorded. The table below summarises the result of the tests. All the test results are recorded at custom MTU of 9000.

Network Plugins	TCP	UDP	HTTP	FTP	SCP
Flannel	9814	9772	9010	9295	1653
Calico	9838	9830	9110	8976	1613
Canal	9812	9810	9211	9069	1612
WeaveNet	9803	9681	7920	8121	1591
Cilium	9678	9662	9131	8771	1634
Kube router	9762	9892	9375	9376	1681
Cilium encrypted	815	402	247	747	522
WeaveNet encrypted	1320	417	351	1196	705

Note: The bandwidth is measured in Mbit/s (Higher the better).

Conclusions from bandwidth tests

1. For transmission control protocol (TCP) calico and flannel stand out as best cnis.
2. For user datagram protocol (UDP) kube-router and calico offered higher bandwidth.
3. For hyper text transfer protocol (HTTP) kube-router and canal offered higher bandwidth.
4. For file transfer protocol (FTP) and secure copy protocol (SCP) kube-router and flannel offered higher bandwidth.
5. Among weavenet and cilium encrypted, weavenet offered higher bandwidth for all of the network protocols.

Network plugins	Bare metal (No plugins only K8s)	Flannel	Calico	Canal	Cilium	WeaveNet	Kube router	Cilium encrypted	WeaveNet encrypted
RAM	374	427	441	443	781	501	420	765	495
CPU usage	40	57	59	58	111	89	57	125	92

Note: The RAM (without cache) and CPU usage are measured in percentage (Lower the better).

Conclusions from resource utilisation tests

1. Kube Router and Flannel show a very good performance in resource utilisation consuming minimum percentage among all network plugins in terms of CPU and RAM.
2. Cilium consumes a lot of memory and processing power in comparison with other network plugins.
3. For encrypted network, plugins weavenet offers better performance in comparison with cilium.

7 Conclusion

The pod, services, ingress and CNIs collectively make the Kubernetes networking durable and robust. The Kubernetes architecture (including the pod, services, kube-apiserver, kube-scheduler, kube-control manager and etcd) manages the entire communication that takes place inside the cluster. The option of having CNI enables us to choose from a variety of plugins as per the requirement. From the comparison of the various plugins, none stood out efficent in all the parameters. The following conclusions can be drawn from the comparison results:

1. Flannel is easy to set up, auto-detects MTU, offers (on average) a good bandwidth for transmitting packets under all the protocols and has very less CPU and RAM utilisation. However, flannel does not support network policies and does not provide a load balancer.
2. Calico and kube-router on average offers a good bandwidth for transmitting packets, supports network policies, provides a load balancer and have a low resource utilisation. At the same time, both the networks do not auto-detect MTU.
3. Among the encrypted networks weavenet stand out better than cilium in most of the aspects including bandwidth offered to networking protocols and resource utilisation (CPU and RAM).

References

1. A. Ducastel, Benchmark results of Kubernetes network plugins (CNI) over 10 Gbit/s network. Available at https://itnext.io/benchmark-results-of-kubernetes-network-plugins-cni-over-10gbit-s-network-updated-april-2019-4a9886efe9c4. Accessed on Aug 2019
2. L.H. Ivan Melia, Linux containers: why they're in your future and what has to happen first, White paper. Available at http://www.cisco.com/c/dam/en/us/solutions/collateral/data-center-virtualization/openstack-at-cisco/linux-containers-white-paper-cisco-red-hat.pdf. Downloaded in Aug 2019
3. R.R.W. Felter, A. Ferreira, J. Rubio, An updated performance comparison of virtual machines and linux containers, in *Published at the International Symposium on Performance Analysis of Systems and Software (ISPASS)* (IEEE, Philadelphia, PA, 2015), pp. 171–172
4. A.R.R.R. Dua, D. Kakadia, Virtualization vs containerization to support paas, in *Published in the proceedings of IEEE IC2E* (2014)
5. W.F. Cong Xu, K. Rajamani, NBWGuard: realizing network QoS for Kubernetes, in *Published in proceeding Middleware '18 Proceedings of the 19th International Middleware Conference Industry*
6. A. Vasudeva, Containers: the future of virtualization & SDDC. Approved by SNIA Tutorial Storage Networking Industry Association (2015). Available at https://goo.gl/Mb3yFq. Downloaded in Aug 2019
7. K. McGuire, *The Truth about Docker Container Lifecycles*. Available at https://goo.gl/Wcj894. Downloaded in Aug 2019
8. *Official Docker Documentation*. Available at https://docs.docker.com/network/. Accessed on Aug 2019
9. R.J. Victor Marmol, T. Hockin, Networking in containers and container clusters. *Published in Proceedings of netdev 0.1* (Ottawa, Canada, 2015). Downloaded in Aug 2019

10. A. Kanso, H. Huang, A. Gherbi, Can linux containers clustering solutions offer high availability. Published by semantics scholar (2016)
11. Official Kubernetes Documentation Available at https://kubernetes.io/docs/concepts/. Accessed on Aug 2019
12. M. Hausenblas, *Container Networking from Docker to Kubernetes*
13. *Container Networking Interface*. Available at https://github.com/containernetworking/cni. Accessed on Aug 2019
14. Flannel. Available at https://github.com/coreos/flannel. Accessed on Aug 2019
15. Calico. Available at https://docs.projectcalico.org/v3.8/introduction/. Accessed on Aug 2019
16. WeaveNet. Available at https://www.weave.works/docs/net/latest/overview/. Accessed on Aug 2019
17. Contiv. Available at https://contiv.io/documents/networking/. Accessed on Aug 2019
18. Cilium. Available at https://cilium.readthedocs.io/en/v1.2/intro/. Accessed on Aug 2019
19. Kube Router. Available at https://github.com/cloudnativelabs/kube-router. Accessed on Aug 2019
20. Romana. Available at https://github.com/romana/romana. Accessed on Aug 2019