



 Star	898	 Fork	152	 Watch	52	 Follow @collabnix	0
--	-----	--	-----	---	----	--	---

Cluster Networking

Kubernetes - Beginners | Intermediate | Advanced

[View on GitHub](#) [Join Slack](#)

Cluster Networking

Kubernetes is a technology that helps you get the most out of your hardware. Containers are deployed on several nodes, making sure that every CPU cycle, every byte of memory, and every block of storage is not wasted. However, this is no easy task. Networking in Kubernetes is a central part of Kubernetes, but it can be challenging to understand exactly how it is expected to work. There are 4 distinct networking problems to address:

- Highly-coupled container-to-container communications: this is solved by pod
- Pod-to-Service communications: this is covered by services.
- External-to-Service communications: this is covered by services.
- Pod-to-Pod communications: this is the primary focus of this lab.

Kubernetes Networking Rules

Kubernetes is highly modular and open-source project. Several components were left to the community to develop. In particular, implementing a cluster-networking solution must conform to a set of high-level rules. They can be summarized as follows:

- Pods scheduled on the same node must be able to communicate with other pods
- All system daemons (background processes, for example, kubelet) running on the node must be able to communicate with all other pods on the node
- Pods that use the host network must be able to contact all other pods on the node

Cluster Networking

So, as you can see Kubernetes eliminates the need for NAT or link containers. In Kubernetes every Pod gets its own IP address. This means you do not need to explicitly create links between Pods and you almost never need to deal with mapping container ports to host ports. This creates a clean, backwards-compatible model where Pods can be treated much like VMs or physical hosts from the perspectives of port allocation, naming, service discovery, load balancing, application configuration, and migration.

There are a number of networking models that adhere to the above rules. In this article, we'll select some of them for discussion. But, before listing the different network plugin examples, let's have a quick overview of some important Kubernetes networking terms.

What Is An Overlay Network?

In general, we can define networks as underlay and overlay types:

Underlay network

Underlay network is closer to the physical layer or you can say that, the network without SDN capabilities. It includes switches, routers, VLANs and so on. It is the basis on which overlay networks are built. It tends to be less scalable due to technical limitations. However, since it's closer to the actual hardware, it is slightly faster than an overlay.

Overlay network

Overlay network refers to the virtual network layer (SDN). In this type, you'll hear terms like veth (virtual eth or virtual network interface), and VxLAN. It is designed to be highly scalable than the underlying network. For example, while VLANs in the underlying network support only 4096 identifiers, VxLAN can reach up to 16 million ones.

Kubernetes supports both networking models, so you can base your model of choice on

other factors than whether or not the cluster can handle it.

What is a Container Network Interface (CNI)?

A CNI is simply a link between the container runtime (like Docker or rkt) and the network plugin. The network plugin is nothing but the executable that handles the actual connection of the container to or from the network, according to a set of rules defined by the CNI. So, to put it simply, a CNI is a set of rules and Go libraries that aid in container/network-plugin integration.

All of the CNIs can be deployed by simply running a pod or a daemonset that launches and manages their daemons. Let's have a look now at the most well-known Kubernetes networking solutions

AWS VPC CNI for Kubernetes

The AWS VPC CNI offers integrated AWS Virtual Private Cloud (VPC) networking for Kubernetes clusters. This CNI plugin offers high throughput and availability, low latency, and minimal network jitter. Additionally, users can apply existing AWS VPC networking and security best practices for building Kubernetes clusters. This includes the ability to use VPC flow logs, VPC routing policies, and security groups for network traffic isolation.

Using this CNI plugin allows Kubernetes pods to have the same IP address inside the pod as they do on the VPC network. The CNI allocates AWS Elastic Networking Interfaces (ENIs) to each Kubernetes node and using the secondary IP range from each ENI for pods on the node. The CNI includes controls for pre-allocation of ENIs and IP addresses for fast pod startup times and enables large clusters of up to 2,000 nodes.

Additionally, the CNI can be run alongside Calico for network policy enforcement. The AWS VPC CNI project is open source with documentation on GitHub.

Azure CNI for Kubernetes

Azure CNI is an open source plugin that integrates Kubernetes Pods with an Azure Virtual Network (also known as VNet) providing network performance at par with VMs. Pods can connect to peered VNet and to on-premises over Express Route or site-to-site VPN and

are also directly reachable from these networks. Pods can access Azure services, such as storage and SQL, that are protected by Service Endpoints or Private Link. You can use VNet security policies and routing to filter Pod traffic. The plugin assigns VNet IPs to Pods by utilizing a pool of secondary IPs pre-configured on the Network Interface of a Kubernetes node.

Azure CNI is available natively in the Azure Kubernetes Service (AKS).

Calico

Calico is a scalable and secure networking plugin. It can be used to manage and secure network policies not only for Kubernetes, but also for containers, virtual machines, and even bare metal servers. Calico works on Layer 3 of the network stack. It works by implementing a vRouter (as opposed to a vSwitch) on each node. Since it is working on L3, it can easily use the Linux kernel's native forwarding functionality. The Felix agent is responsible for programming L3 Forwarding Information base with the IP addresses of the pods scheduled on the node where it is running.

Calico uses vRouters to allow pods to connect to each other across different nodes using the physical network (underlay). It does not use overlay, tunneling or VRF tables. Also, it does not require NAT since each pod can be assigned a public IP address that is accessible from anywhere as long as the security policy permits it.

Deployment differs based on the type of environment or the cloud provider where you'll be hosting your cluster. This document contains all the supported Calico deployment methods.

Cilium

Cilium uses layers 3, 4 (network), and layer 7 (application) to function. It brings a solution that is not only aware of the packets that pass through, but also the application and protocol (for example, HTTP) that those packets are using. Having such a level of inspection allows Cilium to control and enforce network and application security policies. Be aware, though that for this plugin to work, you must be using a Linux kernel that is equal to or higher than 4.8. That's because Cilium uses a new kernel feature Berkeley Packet Filter (BPF), which can replace iptables.

Cilium runs a daemon called cillium-agent on each node. It compiles the BPF filters and transfers them to the kernel for further processing.

Weave Net from WeaveWorks

Weave Net is an easy-to-use, resilient, and fast-growing network plugin that can be used for more than just container networking. When installed, Weave Net creates a virtual router on each host (called peer). Those routers start communicating with each other to establish protocol handshake and, later, learn the network topology. The plugin also creates a bridge interface on each host. All pods get attached to this interface, and they are assigned IP addresses and netmasks. Within the same node, Weave Net uses the kernel to move packets from one pod to another. This protocol is called the fast data path. When the packet is destined to a pod on another host, the plugin uses the sleeve protocol, in which UDP is used to contact the router on the destination host to transfer packets. Subsequently, those packets are captured by the kernel and passed to the target pod.

One way to install Weave Net on a Kubernetes cluster is to apply a daemonset which will automatically install the necessary containers for running the plugin on each node. Once up and running, all pods will use this network for their communication. The peers are self-configuring, so you can add more nodes to the cluster and they'll use the same network without further configuration from your side.

Flannel

Flannel is a networking plugin created by CoreOS. It implements cluster networking in Kubernetes by creating an overlay network. It starts a daemon called flanneld on each node. This daemon runs under a pod whose name starts with kube-flannel-ds-*. When assigning IP addresses, Flannel allocates a small subset of IPs of each host (by default, 10.244.X.0/24). This subset is brought from a larger, preconfigured address space. This subset is used to assign an IP address of each pod on the node.

Flannel uses Kubernetes API server or the cluster's etcd database directly to store information about the assigned subnets, network configuration, and the host IP address.

Packet forwarding among hosts is done through several protocols like UDP and VXLAN.

LAB - Weave Net Implementation

Open <https://labs.play-withk8s.com> and login with your Github / Docker account and launch at least 2 instances simultaneously. Once you have the instances created run the follow kubeadm command to create a kubernetes cluster:

```
kubeadm init --apiserver-advertise-address $(hostname -i)
```

Command completion may take some time, you will have following output once the command gets completed.

```
kubeadm join 192.168.0.18:6443 --token l6m3z8.qa8xj647tip4on1y \
  --discovery-token-ca-cert-hash sha256:10f8323fcd8c9f7f4921a8440776638494
```

Note:- Token and Certificate hash will vary.

You can use this output to join worker nodes to your kubernetes cluster, but make sure you have port 6443 opened for the VM in your Infra. Now lets check the CNI, our kubernetes is using at this moment.

```
[node1 ~]$ kubectl get po -A
```

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE
kube-system	coredns-6dcc67dcbc-jhpyl	1/1	Running	0	2m
kube-system	coredns-6dcc67dcbc-nbg7m	1/1	Running	0	2m
kube-system	etcd-node1	1/1	Running	0	9m
kube-system	kube-apiserver-node1	1/1	Running	0	1m
kube-system	kube-controller-manager-node1	1/1	Running	0	9m
kube-system	kube-proxy-kn8kn	1/1	Running	0	2m
kube-system	kube-scheduler-node1	1/1	Running	0	8m

Currently we cannot see any CNI in the cluster. Now lets plugin weavnet CNI in our cluster. To do it run the following command:-

```
[node1 ~]$ kubectl apply -n kube-system -f \
> "https://cloud.weave.works/k8s/net?k8s-version=$(kubectl version | base64 | tr -d '\n')
serviceaccount/weave-net created
```

```
clusterrole.rbac.authorization.k8s.io/weave-net created
clusterrolebinding.rbac.authorization.k8s.io/weave-net created
role.rbac.authorization.k8s.io/weave-net created
rolebinding.rbac.authorization.k8s.io/weave-net created
daemonset.apps/weave-net created
```

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE
kube-system	coredns-6dcc67dcbc-jhpyl	0/1	Running	0	37s
kube-system	coredns-6dcc67dcbc-nbg7m	1/1	Running	0	37s
kube-system	kube-proxy-kn8kn	1/1	Running	0	37s
kube-system	weave-net-qlwp6	2/2	Running	0	33s

Now we can see the weave net CNI in our cluster. To install a different CNI plugin (Calico) run the following commands:

```
[node1 ~]$ kubectl delete -n kube-system -f "https://cloud.weave.works/
serviceaccount "weave-net" deleted
clusterrole.rbac.authorization.k8s.io "weave-net" deleted
clusterrolebinding.rbac.authorization.k8s.io "weave-net" deleted
role.rbac.authorization.k8s.io "weave-net" deleted
rolebinding.rbac.authorization.k8s.io "weave-net" deleted
daemonset.apps "weave-net" deleted
```

```
[node1 ~]$ kubectl apply -f https://docs.projectcalico.org/manifests/calico.
configmap/calico-config created
customresourcedefinition.apiextensions.k8s.io/felixconfigurations.crd.projectcalico.org
customresourcedefinition.apiextensions.k8s.io/ipamblocks.crd.projectcalico.org
customresourcedefinition.apiextensions.k8s.io/blockaffinities.crd.projectcalico.org
customresourcedefinition.apiextensions.k8s.io/ipamhandles.crd.projectcalico.org
customresourcedefinition.apiextensions.k8s.io/ipamconfigs.crd.projectcalico.org
customresourcedefinition.apiextensions.k8s.io/bgppeers.crd.projectcalico.org
customresourcedefinition.apiextensions.k8s.io/bgpconfigurations.crd.projectcalico.org
customresourcedefinition.apiextensions.k8s.io/ippools.crd.projectcalico.org
customresourcedefinition.apiextensions.k8s.io/hostendpoints.crd.projectcalico.org
customresourcedefinition.apiextensions.k8s.io/clusterinformations.crd.projectcalico.org
customresourcedefinition.apiextensions.k8s.io/globalnetworkpolicies.crd.projectcalico.org
customresourcedefinition.apiextensions.k8s.io/globalnetworksets.crd.projectcalico.org
customresourcedefinition.apiextensions.k8s.io/networkpolicies.crd.projectcalico.org
customresourcedefinition.apiextensions.k8s.io/networksets.crd.projectcalico.org
clusterrole.rbac.authorization.k8s.io/calico-kube-controllers created
```

```
clusterrolebinding.rbac.authorization.k8s.io/calico-kube-controllers created
clusterrole.rbac.authorization.k8s.io/calico-node created
clusterrolebinding.rbac.authorization.k8s.io/calico-node created
daemonset.apps/calico-node created
serviceaccount/calico-node created
deployment.apps/calico-kube-controllers created
serviceaccount/calico-kube-controllers created
```

```
[node1 ~]$ kubectl get po -A
```

NAMESPACE	NAME	READY	STATUS	RESTARTS
kube-system	calico-kube-controllers-6ff88bf6d4-tgtzb	1/1	Running	0
kube-system	calico-node-24h85	1/1	Running	0
kube-system	coredns-846jhw23g9-9af73	1/1	Running	0
kube-system	coredns-846jhw23g9-hmswk	1/1	Running	0
kube-system	etcd-jbaker-1	1/1	Running	0
kube-system	kube-apiserver-jbaker-1	1/1	Running	0
kube-system	kube-controller-manager-jbaker-1	1/1	Running	0
kube-system	kube-proxy-8fzp2	1/1	Running	0
kube-system	kube-scheduler-jbaker-1	1/1	Running	0


Don't forget to clean up.

Join KubeDaily

7 Members Online

Support


MEMBERS ONLINE

 h3ll_boy


 MEE6

 Nitinkashyap

Apex Legends

 ojaswa

 Parmeshwar

 prasad

 vikas027

Free voice chat from Discord

Connect

[Tweets by collabnix](#)

kubelabs is maintained by [collabnix](#).

This page was generated by [GitHub Pages](#).