Kube-OVN Document

v1.14.4

Kube-OVN Team

Table of contents

1. Kube-OVN

1.1 What is Kube-OVN?

Kube-OVN is an enterprise-level cloud-native network orchestration system under CNCF that combines the capabilities of SDN with cloud-native technologies, providing the most functions, extreme performance and the easiest operation.

Kube-OVN uses Open Virtual Network (OVN) and Open vSwitch at the underlying layer to implement network orchestration and exposes its rich capabilities to Kubernetes networking. OVN and OVS have a long history, having emerged long before Kubernetes was born, and have become the de facto standard in the SDN field. Kube-OVN brings them into Kubernetes, significantly enhancing Kubernetes' networking capabilities.

1.2 Why Kube-OVN?

As the workloads running on Kubernetes become more diverse and the scenarios increase, the demand for networking also grows. As long-established networking components, OVN and OVS provide all the functionalities you need.

If you need to run KubeVirt on Kubernetes or have multi-tenant networking requirements, you will find that Kube-OVN's capabilities perfectly match your scenarios. Kube-OVN combines SDN capabilities with cloud-native technologies, offering the most functions, extreme performance and the easiest operation.

Most Functions:

If you miss the rich networking capabilities of the SDN age but are struggling to find them in the cloud-native age, Kube-OVN should be your best choice.

Leveraging the proven capabilities of OVS/OVN in the SDN, Kube-OVN brings the rich capabilities of network virtualization to the cloud-native space. It currently supports Subnet Management, Static IP Allocation, Distributed/Centralized Gateways, Underlay/Overlay Hybrid Networks, VPC Multi-Tenant Networks, Cross-Cluster Interconnect, QoS Management, Multi-NIC Management, ACL, Traffic Mirroring, ARM Support, and many more.

Extreme Performance:

If you're concerned about the additional performance loss associated with container networks, then take a look at How Kube-OVN is doing everything it can to optimize performance.

In the data plane, through a series of carefully optimized flow and kernel optimizations, and with emerging technologies such as eBPF, DPDK and SmartNIC Offload, Kube-OVN can approximate or exceed host network performance in terms of latency and throughput.

In the control plane, Kube-OVN can support large-scale clusters of thousands of nodes and tens of thousands of Pods through the tailoring of OVN upstream flow tables and the use and tuning of various caching techniques.

In addition, Kube-OVN is continuously optimizing the usage of resources such as CPU and memory to accommodate resource-limited scenarios such as the edge.

Easiest Operation:

If you're worried about container network operations, Kube-OVN has a number of built-in tools to help you simplify your operations.

Kube-OVN provides one-click installation scripts to help users quickly build production-ready container networks. Also built-in rich monitoring metrics and Grafana dashboard help users to quickly set up monitoring system.

Powerful command line tools simplify daily operations and maintenance for users. By combining with Cilium, users can enhance the observability of their networks with eBPF capabilities. In addition, the ability to mirror traffic makes it easy to customize traffic monitoring and interface with traditional NPM systems.

1.3 CNI Selection Recommendations

The Kubernetes community offers many excellent CNI projects, which can make selection difficult for users. We recommend first identifying your actual requirements, then evaluating how different projects address those needs - rather than comparing all products first and then deciding which one fits. This approach makes sense for two reasons:

1. Project maintainers primarily focus on their own projects and solving their community's problems - not tracking what other projects are doing or understanding their implementation details. Therefore, maintainers can't provide accurate comparison charts, and it's even harder for outsiders to do this.
2. For end users, understanding your internal needs is far more important than understanding the differences between external projects.

Creating a comparison chart under the Kube-OVN project that recommends Kube-OVN would inevitably be subjective and potentially inaccurate. Instead, we'll list scenarios where you **SHOULD NOT** choose Kube-OVN and provide our recommendations.

1.3.1 When You Need an eBPF Solution

Choose Cilium or Calico eBPF.

Kube-OVN uses Open vSwitch as its data plane, which is a relatively older network virtualization technology.

1.3.2 When You Need an All-in-One Solution (CNI, Ingress, Service Mesh, and Observability)

Choose Cilium.

Kube-OVN primarily focuses on CNI-level networking capabilities, requiring you to combine it with other ecosystem projects for these additional features.

1.3.3 When Running on OpenShift

Choose ovn-kubernetes.

Using third-party CNIs on OpenShift requires adapting to the Cluster Network Operator specifications, which Kube-OVN currently doesn't plan to support. Additionally, third-party network plugins won't receive official Red Hat support, and since networking is critical in Kubernetes, you'd need to coordinate between multiple vendors for solution design and troubleshooting.

1.3.4 When Using Public Cloud Kubernetes (EKS/AKS/GKE, etc.)

Choose the default CNI provided by your Kubernetes vendor, for the same reasons as above.

1.3.5 When Running AI Training and Inference Workloads

Use Hostnetwork or host-device to assign physical devices directly to containers.

AI workloads demand extremely low network latency, making any additional container network operations unnecessary.

1.4 Concepts Clarification: OVN/ovn-kubernetes/Kube-OVN

Due to the similarity of these terms and some abbreviations, confusion often arises in communication. Here's a brief clarification:

1.4.1 OVN

OVN is a virtual network controller maintained by the Open vSwitch community, providing foundational abstractions for virtual networking. It is platform-agnostic and can offer networking services to multiple CMS (Cloud Management Systems) such as OpenStack and Kubernetes. Both *ovn-kubernetes* and *Kube-OVN* rely on OVN's networking capabilities to provide network functionality for Kubernetes.

1.4.2 ovn-kubernetes

ovn-kubernetes was initially a project launched by OVN maintainers to provide CNI networking capabilities for Kubernetes using OVN. It is now the default network for OpenShift and is widely used in OpenShift environments. It offers advanced features such as:

- UDN (User-Defined Networks)
- Multihoming
- Hardware Acceleration

1.4.3 Kube-OVN

Kube-OVN was originally developed to address issues like static IP allocation, namespace-based address space assignment, and centralized gateways by building on OVN. In its early stages, it heavily borrowed design principles and architecture from *ovn-kubernetes*, such as:

- Using annotations to pass Pod network information.
- Leveraging *join* networks to bridge container and host networks.

With community contributions, it has evolved to support advanced features like Underlay networking, VPC, and KubeVirt integration.

PDF **Slack** **Support**

July 30, 2025

May 20, 2022

GitHub

1.5 Comments

2. Getting Started

2.1 Prerequisites

Kube-OVN is a CNI-compliant network system that depends on the Kubernetes environment and the corresponding kernel network module for its operation. Below are the operating system and software versions tested, the environment configuration and the ports that need to be opened.

2.1.1 Software Version

- Kubernetes >= 1.29.
- Docker >= 1.12.6, Containerd >= 1.3.4.
- OS: CentOS 7/8, Ubuntu 16.04/18.04/20.04.
- For other Linux distributions, please make sure `geneve`, `openvswitch`, `ip_tables` and `iptable_nat` kernel modules exist.

Attention:

1. For CentOS kernel version 3.10.0-862 bug exists in `netfilter` modules that lead Kube-OVN embed nat and lb failure.Please update kernel and check Floating IPs broken after kernel upgrade to Centos/RHEL 7.5 - DNAT not working.
2. Kernel version 4.18.0-372.9.1.el8.x86_64 in Rocky Linux 8.6 has a TCP connection problem TCP connection failed in Rocky Linux 8.6,please update kernel to 4.18.0-372.13.1.el8_6.x86_64 or later.
3. For kernel version 4.4, the related `openvswitch` module has some issues for ct, please update kernel version or manually compile `openvswitch` kernel module.
4. When building Geneve tunnel IPv6 in kernel should be enabled, check the kernel bootstrap options with `cat /proc/cmdline`.Check Geneve tunnels don't work when ipv6 is disabled for the detail bug info.

2.1.2 Environment Setup

- Kernel should enable IPv6, if kernel bootstrap options contain `ipv6.disable=1`, it should be set to `0`.
- `kube-proxy` works, Kube-OVN can visit `kube-apiserver` from Service ClusterIP.
- Make sure kubelet enabled `CNI` and find cni-bin and cni-conf in default directories, kubelet bootstrap options should contain `--network-plugin=cni --cni-bin-dir=/opt/cni/bin --cni-conf-dir=/etc/cni/net.d`.
- Make sure no other CNI installed or has been removed, check if any config files still exist in `/etc/cni/net.d/`.

2.1.3 Ports Need Open

Component	Port	Usage
ovn-central	6641/tcp	ovn nb db server listen ports
ovn-central	6642/tcp	ovn sb db server listen ports
ovn-central	6643/tcp	ovn northd server listen ports
ovn-central	6644/tcp	ovn raft server listen ports
ovn-ic	6645/tcp	ovn ic nb db server listen ports
ovn-ic	6646/tcp	ovn ic sb db server listen ports
ovs-ovn	Geneve 6081/udp, STT 7471/tcp, Vxlan 4789/udp	tunnel ports
kube-ovn-controller	10660/tcp	metrics port
kube-ovn-daemon	10665/tcp	metrics port
kube-ovn-monitor	10661/tcp	metrics port

If you are running firewalld on nodes, you need also to enable packet forwarding and masquerade:

```
# enable packet forwarding
firewall-cmd --add-forward --permanent
# enable IPv4 masquerade
firewall-cmd --add-masquerade --permanent
# enable IPv6 masquerade for the Kube-OVN IPv6/DualStack subnet (adjust if your subnet differs)
firewall-cmd --permanent --add-rich-rule 'rule family="ipv6" source address="fd00:10:16::/112" masquerade'

firewall-cmd --reload
```

PDF Slack Support

July 30, 2025

June 30, 2022

GitHub

2.1.4 Comments

2.2 One-Click Installation

Kube-OVN provides a one-click installation script and charts repo to help you quickly install a highly available, production-ready Kube-OVN container network with Overlay networking by default.

If you need Underlay/Vlan networking as the default container network, please read Underlay Installation

Before installation please read Prerequisites first to make sure the environment is ready. If you want to completely remove Kube-OVN, please refer to Uninstallation.

2.2.1 Script Installation

Download the installation script

We recommend using the stable release version for production environments, please use the following command to download:

```
wget https://raw.githubusercontent.com/kubeovn/kube-ovn/refs/tags/v1.14.4/dist/images/install.sh
```

If you are interested in the latest features of the master branch, please use the following command to download:

```
wget https://raw.githubusercontent.com/kubeovn/kube-ovn/master/dist/images/install.sh
```

Modify Configuration Options

Open the script using the editor and change the following variables to the expected:

```
REGISTRY="kubeovn"                      # Image Repo
VERSION="v1.14.4"                       # Image Tag
POD_CIDR="10.16.0.0/16"                 # Default subnet CIDR don't overlay with SVC/NODE/JOIN CIDR
SVC_CIDR="10.96.0.0/12"                 # Be consistent with apiserver's service-cluster-ip-range
JOIN_CIDR="100.64.0.0/16"               # Pod/Host communication Subnet CIDR, don't overlay with SVC/NODE/POD CIDR
LABEL="node-role.kubernetes.io/master"  # The node label to deploy OVN DB
IFACE=""                                # The name of the host NIC used by the container network, or if empty use the NIC that host Node IP in Kubernetes
TUNNEL_TYPE="geneve"                    # Tunnel protocol, available options: geneve, vxlan or stt. stt requires compilation of ovs kernel module
```

You can also use regular expression to match NIC names, such as `IFACE=enp6s0f0,eth.*`.

Run the Script

The script needs to be executed with root permission

`bash install.sh`

Wait Kube-OVN ready.

Upgrade

When using this script to upgrade Kube-OVN, please pay attention to the following points:

1. The script's `[Step 4/6]` restarts all container network Pods. During an upgrade, this step should be **skipped or commented out from the script** to avoid unintended restarts.
2. **Important:** If any parameters were adjusted during Kube-OVN operation, these changes **must be updated in the script before upgrading**. Otherwise, previous parameter adjustments **will be reverted**.

2.2.2 Helm Chart Installation

Since the installation of Kube-OVN requires setting some parameters, to install Kube-OVN using Helm, you need to follow the steps below.

View the node IP address

```
# kubectl get node -o wide
NAME                      STATUS     ROLES           AGE    VERSION    INTERNAL-IP    EXTERNAL-IP    OS-IMAGE            KERNEL-VERSION      CONTAINER-RUNTIME
kube-ovn-control-plane    NotReady   control-plane   20h    v1.26.0    172.18.0.3     <none>         Ubuntu 22.04.1 LTS  5.10.104-linuxkit   containerd://1.6.9
kube-ovn-worker           NotReady   <none>          20h    v1.26.0    172.18.0.2     <none>         Ubuntu 22.04.1 LTS  5.10.104-linuxkit   containerd://1.6.9
```

Add label to node

```
# kubectl label node -lbeta.kubernetes.io/os=linux kubernetes.io/os=linux --overwrite
node/kube-ovn-control-plane not labeled
node/kube-ovn-worker not labeled

# kubectl label node -lnode-role.kubernetes.io/control-plane kube-ovn/role=master --overwrite
node/kube-ovn-control-plane labeled

# The following labels are used for the installation of dpdk images and can be ignored in non-dpdk cases
# kubectl label node -lovn.kubernetes.io/ovs_dp_type!=userspace ovn.kubernetes.io/ovs_dp_type=kernel --overwrite
node/kube-ovn-control-plane labeled
node/kube-ovn-worker labeled
```

Add Helm Repo information

```
# helm repo add kubeovn https://kubeovn.github.io/kube-ovn/
"kubeovn" has been added to your repositories

$ helm repo list
NAME            URL
kubeovn         https://kubeovn.github.io/kube-ovn/

# helm repo update kubeovn
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "kubeovn" chart repository
Update Complete. ⎈Happy Helming!⎈

# helm search repo kubeovn
NAME                    CHART VERSION   APP VERSION     DESCRIPTION
kubeovn/kube-ovn        v1.14.4        v1.14.4         Helm chart for Kube-OVN
```

Install Kube-OVN with Helm

You can refer to the variable definitions in the `values.yaml` file for available parameters.

```
# helm install kube-ovn kubeovn/kube-ovn --wait -n kube-system --version v1.14.4
NAME: kube-ovn
LAST DEPLOYED: Thu Apr 24 08:30:13 2025
NAMESPACE: kube-system
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

Upgrade

Important: Similar to script-based upgrades, ensure all parameter adjustments are updated in the `values.yaml` file **before upgrading with Helm**. Otherwise, previous parameter adjustments **will be reverted**.

```
helm upgrade -f values.yaml kube-ovn kubeovn/kube-ovn --wait -n kube-system --version v1.14.4
```

PDF Slack Support

July 30, 2025

May 20, 2022

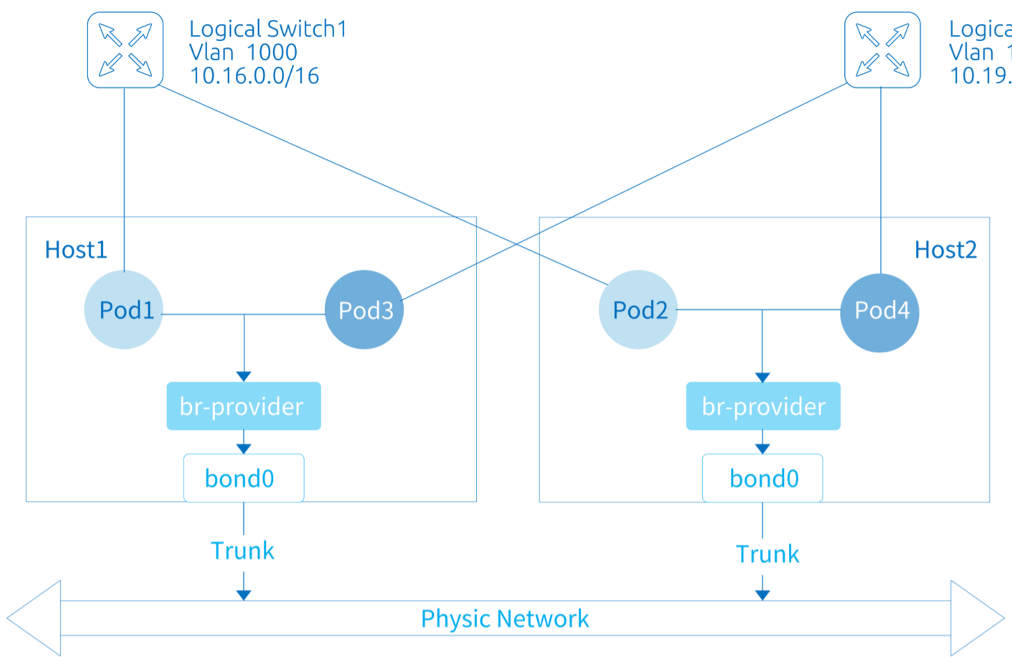
GitHub

2.2.3 Comments

2.3 Underlay Installation

By default, the default subnet uses Geneve to encapsulate cross-host traffic, and build an overlay network on top of the infrastructure.

For the case that you want the container network to use the physical network address directly, you can set the default subnet of Kube-OVN to work in Underlay mode, which can directly assign the address resources in the physical network to the containers, achieving better performance and connectivity with the physical network.



2.3.1 Limitation

Since the container network in this mode uses physical network directly for L2 packet forwarding, L3 functions such as SNAT/EIP, distributed gateway/centralized gateway in Overlay mode cannot be used. VPC level isolation is also not available for underlay subnet.

2.3.2 Comparison with Macvlan

The Underlay mode of Kube-OVN is very similar to the Macvlan, with the following major differences in functionality and performance:

1. Macvlan performs better in terms of throughput and latency performance metrics due to its shorter kernel path and the fact that it does not require OVS for packet processing.
2. Kube-OVN provides arp-proxy functionality through flow tables to mitigate the risk of arp broadcast storms on large-scale networks.
3. Since Macvlan works at the bottom of the kernel and bypasses the host netfilter, Service and NetworkPolicy functionality requires additional development. Kube-OVN provides Service and NetworkPolicy capabilities through the OVS flow table.
4. Kube-OVN Underlay mode provides additional features such as address management, fixed IP and QoS compared to Macvlan.

2.3.3 Environment Requirements

In Underlay mode, the OVS will bridge a node NIC to the OVS bridge and send packets directly through that node NIC, relying on the underlying network devices for L2/L3 level forwarding capabilities. You need to configure the corresponding gateway, Vlan and security policy in the underlying network device in advance.

1. For OpenStack VM environments, you need to turn off `PortSecurity` on the corresponding network port.
2. For VMware vSwitch networks, `MAC Address Changes`, `Forged Transmits` and `Promiscuous Mode Operation` should be set to `allow`.
3. For Hyper-V virtualization, `MAC Address Spoofing` should be enabled in VM nic advanced features.
4. Public clouds, such as AWS, GCE, AliCloud, etc., do not support user-defined Mac, so they cannot support Underlay mode network. In this scenario, if you want to use Underlay, it is recommended to use the VPC-CNI provided by the corresponding public cloud vendor..
5. The network interface that is bridged into ovs can not be type of Linux Bridge.

For management and container networks using the same NIC, Kube-OVN will transfer the NIC's Mac address, IP address, route, and MTU to the corresponding OVS Bridge to support single NIC deployment of Underlay networks. OVS Bridge name format is `br-PROVIDER_NAME`, `PROVIDER_NAME` is the name of `ProviderNetwork` (Default: provider).

2.3.4 Specify Network Mode When Deploying

This deployment mode sets the default subnet to Underlay mode, and all Pods with no subnet specified will run in the Underlay network by default.

Download Script

```
wget https://raw.githubusercontent.com/kubeovn/kube-ovn/release-1.14/dist/images/install.sh
```

MODIFY CONFIGURATION OPTIONS

```
ENABLE_ARP_DETECT_IP_CONFLICT # disable vlan arp conflict detection if necessary
NETWORK_TYPE                  # set to vlan
VLAN_INTERFACE_NAME           # set to the NIC that carries the Underlay traffic, e.g. eth1
VLAN_ID                       # The VLAN Tag need to be added, if set 0 no vlan tag will be added
POD_CIDR                      # The Underlay network CIDR, e.g. 192.168.1.0/24
POD_GATEWAY                   # Underlay physic gateway address, e.g. 192.168.1.1
EXCLUDE_IPS                   # Exclude ranges to avoid conflicts between container network and IPs already in use on the physical network, e.g.
192.168.1.1..192.168.1.100
ENABLE_LB                     # If Underlay Subnet needs to visit Service set it to true
EXCHANGE_LINK_NAME            # If swap the names of the OVS bridge and the bridge interface under the default provider-network. Default to false.
LS_DNAT_MOD_DL_DST            # If DNAT translate MAC addresses to accelerate service access. Default to true.
```

Run the Script

```
bash install.sh
```

2.3.5 Dynamically Create Underlay Networks via CRD

This approach dynamically creates an Underlay subnet that Pod can use after installation.

Create ProviderNetwork

ProviderNetwork provides the abstraction of host NIC to physical network mapping, unifies the management of NICs belonging to the same network, and solves the configuration problems in complex environments with multiple NICs on the same machine, inconsistent NIC names and inconsistent corresponding Underlay networks.

Create ProviderNetwork as below:

```
apiVersion: kubeovn.io/v1
kind: ProviderNetwork
metadata:
  name: net1
spec:
  defaultInterface: eth1
  customInterfaces:
    - interface: eth2
      nodes:
        - node1
  excludeNodes:
    - node2
```

Note: The length of the ProviderNetwork resource name must not exceed 12.

- `defaultInterface` : The default node NIC name. When the ProviderNetwork is successfully created, an OVS bridge named br-net1 (in the format `br-NAME`) is created in each node (except excludeNodes) and the specified node NIC is bridged to this bridge.
- `customInterfaces` : Optionally, you can specify the NIC to be used for a specific node.
- `excludeNodes` : Optional, to specify nodes that do not bridge the NIC. Nodes in this list will be added with the `net1.provider-network.ovn.kubernetes.io/exclude=true` tag.

Other nodes will be added with the following tags:

Key	Value	Description
net1.provider-network.ovn.kubernetes.io/ready	true	bridge work finished, ProviderNetwork is ready on this node
net1.provider-network.ovn.kubernetes.io/interface	eth1	The name of the bridged NIC in the node.
net1.provider-network.ovn.kubernetes.io/mtu	1500	MTU of bridged NIC in node

If an IP has been configured on the node NIC, the IP address and the route on the NIC are transferred to the corresponding OVS bridge.

Create VLAN

Vlan provides an abstraction to bind Vlan Tag and ProviderNetwork.

Create a VLAN as below:

```
apiVersion: kubeovn.io/v1
kind: Vlan
metadata:
  name: vlan1
spec:
  id: 0
  provider: net1
```

- `id` : VLAN ID/Tag, Kube-OVN will add this Vlan tag to traffic, if set 0, no tag is added. the vlan tag applies to a localnet port.
- `provider` : The name of ProviderNetwork. Multiple VLAN can use a same ProviderNetwork.

Create Subnet

Bind Vlan to a Subnet as below:

```
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
   name: subnet1
spec:
   protocol: IPv4
   cidrBlock: 172.17.0.0/16
   gateway: 172.17.0.1
   vlan: vlan1
   disableGatewayCheck: false
```

- `vlan` : The VLAN name to be used. Multiple subnets can reference the same VLAN.
- `disableGatewayCheck` : If the gateway in the underlying network does not exist, set this field to `true` to disable gateway detection.

2.3.6 Create Pod

You can create containers in the normal way, check whether the container IP is in the specified range and whether the container can interoperate with the physical network.

For fixed IP requirements, please refer to Fixed Addresses

2.3.7 Logical Gateway

For cases where no gateway exists in the physical network, Kube-OVN supports the use of logical gateways configured in the subnet in Underlay mode. To use this feature, set `spec.logicalGateway` to `true` for the subnet:

```yaml
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
   name: subnet1
spec:
   protocol: IPv4
   cidrBlock: 172.17.0.0/16
   gateway: 172.17.0.1
   vlan: vlan1
   logicalGateway: true
```

When this feature is turned on, the Pod does not use an external gateway, but a Logical Router created by Kube-OVN to forward cross-subnet communication.

2.3.8 Interconnection of Underlay and Overlay Networks

If a cluster has both Underlay and Overlay subnets, by default, Pods in the Overlay subnet can access the Pod IPs in the Underlay subnet via a gateway using NAT. From the perspective of Pods in the Underlay subnet, the addresses in the Overlay subnet are external, and require the underlying physical device to forward, but the underlying physical device does not know the addresses in the Overlay subnet and cannot forward. Therefore, Pods in the Underlay subnet cannot access Pods in the Overlay subnet directly via Pod IPs.

If you need to enable communication between Underlay and Overlay networks, you need to set the `u2oInterconnection` of the subnet to `true`. In this case, Kube-OVN will use an additional Underlay IP to connect the Underlay subnet and the `ovn-cluster` logical router, and set the corresponding routing rules to enable communication. Unlike the logical gateway, this solution only connects the Underlay and Overlay subnets within Kube-OVN, and other traffic accessing the Internet will still be forwarded through the physical gateway.

Specify logical gateway IP

After the interworking function is enabled, an IP from the subnet will be randomly selected as the logical gateway. If you need to specify the logical gateway of the Underlay Subnet, you can specify the field `u2oInterconnectionIP`.

Specify custom VPC for Underlay Subnet connection

By default, the Underlay Subnet will communicate with the Overlay Subnet on the default VPC. If you want to specify to communicate with a certain VPC, after setting `u2oInterconnection` to `true`, specify the `subnet.spec.vpc` field as the name of the VPC.

2.3.9 Notice

If you have an IP address configured on the network card of the node you are using, and the operating system configures the network using Netplan (such as Ubuntu), it is recommended that you set the renderer of Netplan to NetworkManager and configure a static IP address for the node's network card (disable DHCP).

```yaml
network:
  renderer: NetworkManager
  ethernets:
    eth0:
      dhcp4: no
      addresses:
        - 172.16.143.129/24
  version: 2
```

If you want to modify the IP or routing configuration of the network card, you need to execute the following commands after modifying the Netplan configuration:

```
netplan generate

nmcli connection reload netplan-eth0
nmcli device set eth0 managed yes
```

After executing the above commands, Kube-OVN will transfer the IP and routing from the network card to the OVS bridge.

If your operating system manages the network using NetworkManager (such as CentOS), you need to execute the following command after modifying the network card configuration:

```
nmcli connection reload eth0
nmcli device set eth0 managed yes
nmcli -t -f GENERAL.STATE device show eth0 | grep -qw unmanaged || nmcli device reapply eth0
```

Notice: If the host nic's MAC is changed, Kube-OVN will not change the OVS bridge's MAC unless kube-ovn-cni is restarted.

2.3.10 Known Issues

When the physical network is enabled with hairpin, Pod network is abnormal

When physical networks enable hairpin or similar behaviors, problems such as gateway check failure when creating Pods and abnormal network communication of Pods may occur. This is because the default MAC learning function of OVS bridge does not support this kind of network environment.

To solve this problem, it is necessary to turn off hairpin (or modify the relevant configuration of physical network), or update the Kube-OVN version.

When there are a large number of Pods, gateway check for new Pods fails

If there are a large number of Pods running on the same node (more than 300), it may cause packet loss due to the OVS flow table resubmit times exceeding the upper limit of ARP broadcast packets.

```
2022-11-13T08:43:46.782Z|00222|ofproto_dpif_upcall(handler5)|WARN|Flow: arp,in_port=331,vlan_tci=0x0000,dl_src=00:00:00:25:eb:
39,dl_dst=ff:ff:ff:ff:ff:ff,arp_spa=10.213.131.240,arp_tpa=10.213.159.254,arp_op=1,arp_sha=00:00:00:25:eb:39,arp_tha=ff:ff:ff:ff:ff:ff

bridge("br-int")
----------------
 0. No match.
     >>>> received packet on unknown port 331 <<<<
    drop

Final flow: unchanged
Megaflow: recirc_id=0,eth,arp,in_port=331,dl_src=00:00:00:25:eb:39
Datapath actions: drop
2022-11-13T08:44:34.077Z|00224|ofproto_dpif_xlate(handler5)|WARN|over 4096 resubmit actions on bridge br-int while processing
arp,in_port=13483,vlan_tci=0x0000,dl_src=00:00:00:59:ef:
13,dl_dst=ff:ff:ff:ff:ff:ff,arp_spa=10.213.152.3,arp_tpa=10.213.159.254,arp_op=1,arp_sha=00:00:00:59:ef:13,arp_tha=ff:ff:ff:ff:ff:ff
```

To solve this issue, modify the OVN NB option `bcast_arp_req_flood` to `false`:

```
kubectl ko nbctl set NB_Global . options:bcast_arp_req_flood=false
```

PDF   Slack   Support

July 30, 2025

May 20, 2022

GitHub

2.3.11 Comments

2.4 Install on Talos

Talos Linux is a modern Linux distribution built for Kubernetes.

2.4.1 Deploy Kube-OVN via Helm Chart

You can deploy Kube-OVN on Talos Linux clusters with the following command:

```
helm install kube-ovn kubeovn/kube-ovn --wait \
    -n kube-system \
    --version v1.14.4 \
    --set OVN_DIR=/var/lib/ovn \
    --set OPENVSWITCH_DIR=/var/lib/openvswitch \
    --set DISABLE_MODULES_MANAGEMENT=true \
    --set cni_conf.MOUNT_LOCAL_BIN_DIR=false
```

If you want to use underlay as the default network, you need to pass the relevant chart values. Here is an example:

```
helm install kubeovn kubeovn/kube-ovn --wait \
    -n kube-system \
    --version v1.14.4 \
    --set OVN_DIR=/var/lib/ovn \
    --set OPENVSWITCH_DIR=/var/lib/openvswitch \
    --set DISABLE_MODULES_MANAGEMENT=true \
    --set cni_conf.MOUNT_LOCAL_BIN_DIR=false \
    --set networking.NETWORK_TYPE=vlan \
    --set networking.vlan.VLAN_INTERFACE_NAME=enp0s5f1 \
    --set networking.vlan.VLAN_ID=0 \
    --set networking.NET_STACK=ipv4 \
    --set-json networking.EXCLUDE_IPS='"172.99.99.11..172.99.99.99"' \
    --set-json ipv4.POD_CIDR='"172.99.99.0/24"' \
    --set-json ipv4.POD_GATEWAY='"172.99.99.1"'
```

Note

Logical network interfaces, such as VLAN, Bond, and Bridge, cannot be used as provider interfaces for Underlay networks. Physical interfaces used for the Underlay network **MUST** be configured with `ignore=true` in the Talos machine configuration. Here is an example:

```yaml
machine:
  network:
    interfaces:
      - interface: enp0s5f1
        ignore: true
```

PDF Slack Support

July 30, 2025

April 16, 2025

GitHub

2.4.2 Comments

2.5 Uninstall

If you need to remove the Kube-OVN and replace it with another network plugin, please follow the steps below to remove all the corresponding Kube-OVN component and OVS configuration to avoid interference with other network plugins.

Feel free to contact us with an Issue to give us feedback on why you don't use Kube-OVN to help us improve it.

2.5.1 Delete Resource in Kubernetes

Please choose the uninstall command based on your installation method:

Script Uninstall **Helm Uninstall**

```
wget https://raw.githubusercontent.com/kubeovn/kube-ovn/release-1.14/dist/images/cleanup.sh
bash cleanup.sh

helm uninstall kube-ovn -n kube-system
```

2.5.2 Cleanup Config and Logs on Every Node

Run the following commands on each node to clean up the configuration retained by ovsdb and openvswitch:

```
rm -rf /var/run/openvswitch
rm -rf /var/run/ovn
rm -rf /etc/origin/openvswitch/
rm -rf /etc/origin/ovn/
rm -rf /etc/cni/net.d/00-kube-ovn.conflist
rm -rf /etc/cni/net.d/01-kube-ovn.conflist
rm -rf /var/log/openvswitch
rm -rf /var/log/ovn
rm -fr /var/log/kube-ovn
```

2.5.3 Reboot Node

Reboot the machine to ensure that the corresponding NIC information and iptable/ipset rules are cleared to avoid the interference with other network plugins:

```
reboot
```

PDF **Slack** **Support**

July 22, 2025

May 24, 2022

GitHub

2.5.4 Comments

3. User Guide

3.1 Installation and Configuration Options

In One-Click Installation we use the default configuration for installation. Kube-OVN also supports more custom configurations, which can be configured in the installation script, or later by changing the parameters of individual components. This document will describe what these customization options do, and how to configure them.

3.1.1 Built-in Network Settings

Kube-OVN will configure two built-in Subnets during installation:

1. `default` Subnet, as the default subnet used by the Pod to assign IPs, with a default CIDR of `10.16.0.0/16` and a gateway of `10.16.0.1`.
2. The `join` subnet, as a special subnet for network communication between the Node and Pod, has a default CIDR of `100.64.0.0/16` and a gateway of `100.64.0.1`.

The configuration of these two subnets can be changed during installation via the installation scripts variables:

```
POD_CIDR="10.16.0.0/16"
POD_GATEWAY="10.16.0.1"
JOIN_CIDR="100.64.0.0/16"
EXCLUDE_IPS=""
```

`EXCLUDE_IP` sets the address range for which `kube-ovn-controller` will not automatically assign from it, the format is: `192.168.10.20..192.168.10.30`.

Note that in the Overlay case these two Subnets CIDRs cannot conflict with existing host networks and Service CIDRs.

You can change the address range of both Subnets after installation by referring to Change Subnet CIDR and Change Join Subnet CIDR.

3.1.2 Config Service CIDR

Since some of the iptables and routing rules set by `kube-proxy` will conflict with the rules set by Kube-OVN, Kube-OVN needs to know the CIDR of the service to set the corresponding rules correctly.

This can be done by modifying the installation script:

```
SVC_CIDR="10.96.0.0/12"
```

You can also modify the args of the `kube-ovn-controller` Deployment after installation:

```
args:
- --service-cluster-ip-range=10.96.0.0/12
```

3.1.3 Overlay NIC Selection

In the case of multiple NICs on a node, Kube-OVN will select the NIC corresponding to the Kubernetes Node IP as the NIC for cross-node communication between containers and establish the corresponding tunnel.

If you need to select another NIC to create a container tunnel, you can change it in the installation script:

```
IFACE=eth1
```

This option supports regular expressions separated by commas, e.g. 'ens[a-z0-9],*eth[a-z0-9]*'.

It can also be adjusted after installation by modifying the args of the `kube-ovn-cni` DaemonSet:

```
args:
- --iface=eth1
```

If each machine has a different NIC name and there is no fixed pattern, you can use the node annotation `ovn.kubernetes.io/tunnel_interface` to configure each node one by one. This annotation will override the configuration of `iface`.

```
kubectl annotate node no1 ovn.kubernetes.io/tunnel_interface=ethx
```

3.1.4 Config MTU

Since Overlay encapsulation requires additional space, Kube-OVN will adjust the MTU of the container NIC based on the MTU of the selected NIC when creating the container NIC. By default, the Pod NIC MTU is the host NIC MTU - 100 on the Overlay Subnet, and the Pod NIC and host NIC have the same MTU on the Underlay Subnet.

If you need to adjust the size of the MTU under the Overlay subnet, you can modify the parameters of the `kube-ovn-cni` DaemonSet:

```
args:
- --mtu=1333
```

3.1.5 Global Traffic Mirroring Setting

When global traffic mirroring is enabled, Kube-OVN will create a `mirror0` virtual NIC on each node and copy all container network traffic from the current machine to that NIC, Users can perform traffic analysis with tcpdump and other tools. This function can be enabled in the installation script:

```
ENABLE_MIRROR=true
```

It can also be adjusted after installation by modifying the args of the `kube-ovn-cni` DaemonSet:

```
args:
- --enable-mirror=true
```

The ability to mirror traffic is disabled in the default installation, if you need fine-grained traffic mirroring or need to mirror traffic to additional NICs please refer to Traffic Mirror.

3.1.6 LB Settings

Kube-OVN uses L2 LB in OVN to implement service forwarding. In Overlay scenarios, users can choose to use `kube-proxy` for service traffic forwarding, in which case the LB function of Kube-OVN can be disabled to achieve better performance on the control plane and data plane.

This feature can be configured in the installation script:

```
ENABLE_LB=false
```

It can also be configured after installation by changing the args of the `kube-ovn-controller` Deployment:

```
args:
- --enable-lb=false
```

The LB feature is enabled in the default installation.

The spec field `enableLb` has been added to the subnet crd definition since Kube-OVN v1.12.0 to migrate the LB function of Kube-OVN to the subnet level. You can set whether to enable the LB function based on different subnets. The `enable-lb` parameter in the `kube-ovn-controller` deployment is used as a global switch to control whether to create a load-balancer record. The `enableLb` parameter added in the subnet is used to control whether the subnet is associated with a load-balancer record. After the previous version is upgraded to v1.12.0, the `enableLb` parameter of the subnet will automatically inherit the value of the original global switch parameter.

3.1.7 NetworkPolicy Settings

Kube-OVN uses ACLs in OVN to implement NetworkPolicy. Users can choose to disable the NetworkPolicy feature or use the Cilium Chain approach to implement NetworkPolicy using eBPF. In this case, the NetworkPolicy feature of Kube-OVN can be disabled to achieve better performance on the control plane and data plane.

This feature can be configured in the installation script:

```
ENABLE_NP=false
```

It can also be configured after installation by changing the args of the `kube-ovn-controller` Deployment:

```
args:
- --enable-np=false
```

NetworkPolicy is enabled by default.

3.1.8 EIP and SNAT Settings

If the EIP and SNAT capabilities are not required on the default VPC, users can choose to disable them to reduce the performance overhead of `kube-ovn-controller` in large scale cluster environments and improve processing speed.

This feature can be configured in the installation script:

```
ENABLE_EIP_SNAT=false
```

It can also be configured after installation by changing the args of the `kube-ovn-controller` Deployment:

```
args:
- --enable-eip-snat=false
```

EIP and SNAT is enabled by default. More information can refer to EIP and SNAT.

3.1.9 Centralized Gateway ECMP Settings

The centralized gateway supports two mode of high availability, primary-backup and ECMP. If you want to enable ECMP mode, you need to change the args of `kube-ovn-controller` Deployment:

```
args:
- --enable-ecmp=true
```

Centralized gateway default installation under the primary-backup mode, more gateway-related content please refer to Config Subnet.

The spec field `enableEcmp` has been added to the subnet crd definition since Kube-OVN v1.12.0 to migrate the ECMP switch to the subnet level. You can set whether to enable ECMP mode based on different subnets. The `enable-ecmp` parameter in the `kube-ovn-controller` deployment is no longer used. After the previous version is upgraded to v1.12.0, the subnet switch will automatically inherit the value of the original global switch parameter.

3.1.10 Kubevirt VM Fixed Address Settings

For VM instances created by Kubevirt, `kube-ovn-controller` can assign and manage IP addresses in a similar way to the StatefulSet Pod. This allows VM instances address fixed during start-up, shutdown, upgrade, migration, and other operations throughout their lifecycle, making them more compatible with the actual virtualization user experience.

This feature is enabled by default after v1.10.6. To disable this feature, you need to change the following args in the `kube-ovn-controller` Deployment:

```
args:
- --keep-vm-ip=false
```

3.1.11 CNI Settings

By default, Kube-OVN installs the CNI binary in the `/opt/cni/bin` directory and the CNI configuration file `01-kube-ovn.conflist` in the `/etc/cni/net.d` directory. If you need to change the installation location and the priority of the CNI configuration file, you can modify the following parameters of the installation script.

```
CNI_CONF_DIR="/etc/cni/net.d"
CNI_BIN_DIR="/opt/cni/bin"
CNI_CONFIG_PRIORITY="01"
```

Or change the Volume mount and args of the `kube-ovn-cni` DaemonSet after installation:

```
volumes:
- name: cni-conf
  hostPath:
    path: "/etc/cni/net.d"
- name: cni-bin
  hostPath:
    path:"/opt/cni/bin"
...
args:
- --cni-conf-name=01-kube-ovn.conflist
```

3.1.12 Tunnel Type Settings

The default encapsulation mode of Kube-OVN Overlay is Geneve, if you want to change it to Vxlan or STT, please adjust the following parameters in the installation script:

```
TUNNEL_TYPE="vxlan"
```

Or change the environment variables of `ovs-ovn` DaemonSet after installation:

```
env:
- name: TUNNEL_TYPE
  value: "vxlan"
```

If you need to use the STT tunnel and need to compile additional kernel modules for ovs, please refer to Performance Tuning.

Please refer to Tunneling Protocol Selection for the differences between the different protocols in practice.

3.1.13 SSL Settings

The OVN DB API interface supports SSL encryption to secure the connection. To enable it, adjust the following parameters in the installation script:

```
ENABLE_SSL=true
```

The SSL is disabled by default.

PDF **Slack** **Support**

July 30, 2025

May 24, 2022

GitHub

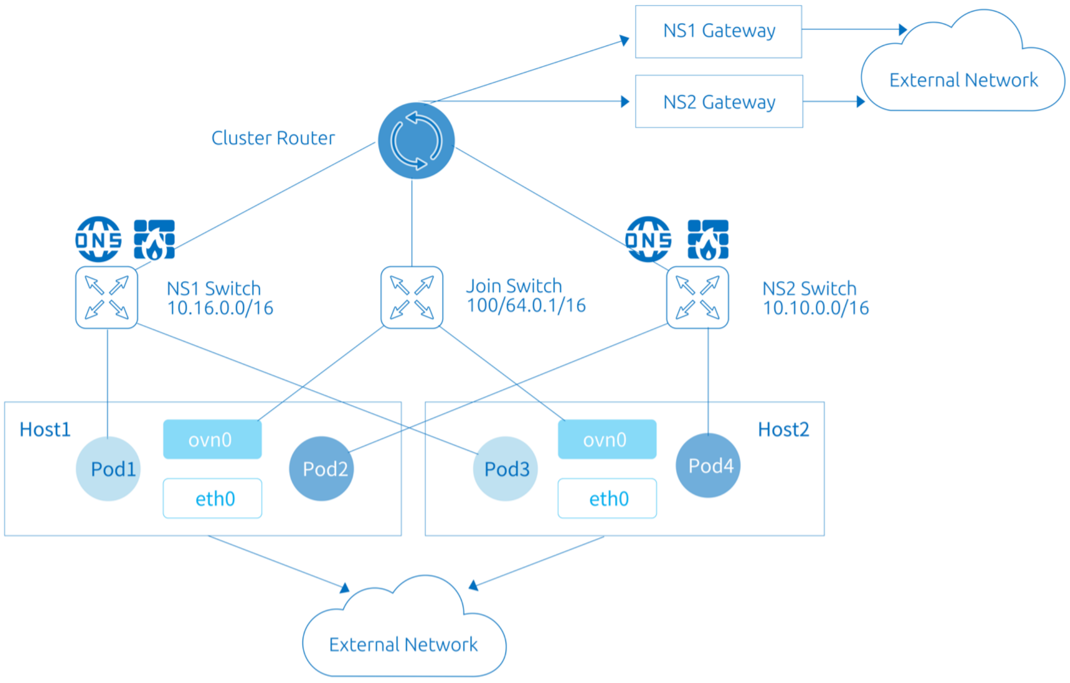
3.1.14 Comments

3.2 Config Subnet

Subnet is a core concept and basic unit of use in Kube-OVN, and Kube-OVN organizes IP and network configuration in terms of Subnet. Each Namespace can belong to a specific Subnet, and Pods under the Namespace automatically obtain IPs from the Subnet they belong to and share the network configuration (CIDR, gateway type, access control, NAT control, etc.).

Unlike other CNI implementations where each node is bound to a subnet, in Kube-OVN the Subnet is a global level virtual network configuration, and the addresses of one Subnet can be distributed on any node.

Note: Different subnets under the same VPC cannot contain the same IP, and different subnets connected to each other based on VPC peering or VPN cannot contain the same IP.



There are some differences in the usage and configuration of Overlay and Underlay Subnets, and this document will describe the common configurations and differentiated features of the different types of Subnets.

3.2.1 Default Subnet

To make it easier for users to get started quickly, Kube-OVN has a built-in default Subnet, all Namespaces that do not explicitly declare subnet affiliation are automatically assigned IPs from the default subnet and the network information. The configuration of this Subnet is specified at installation time, you can refer to Built-in Network Settings for more details. To change the CIDR of the default Subnet after installation please refer to Change Subnet CIDR.

In Overlay mode, the default Subnet uses a distributed gateway and NAT translation for outbound traffic, which behaves much the same as the Flannel's default behavior, allowing users to use most of the network features without additional configuration.

In Underlay mode, the default Subnet uses the physical gateway as the outgoing gateway and enables arping to check network connectivity.

Check the Default Subnet

The `default` field in the default Subnet spec is set to `true`, and there is only one default Subnet in a cluster, named `ovn-default`.

```
# kubectl get subnet ovn-default -o yaml
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
  creationTimestamp: "2019-08-06T09:33:43Z"
  generation: 1
  name: ovn-default
  resourceVersion: "1571334"
  selfLink: /apis/kubeovn.io/v1/subnets/ovn-default
  uid: 7e2451f8-fb44-4f7f-b3e0-cfd27f6fd5d6
spec:
  cidrBlock: 10.16.0.0/16
  default: true
  excludeIps:
  - 10.16.0.1
  gateway: 10.16.0.1
  gatewayType: distributed
  natOutgoing: true
  private: false
  protocol: IPv4
```

3.2.2 Join Subnet

In the Kubernetes network specification, it is required that Nodes can communicate directly with all Pods. To achieve this in Overlay network mode, Kube-OVN creates a `join` Subnet and creates a virtual NIC `ovn0` at each node that connect to the `join` subnet, through which the nodes and Pods can communicate with each other.

All network communication between Pods and Nodes will go through the `ovn0` network interface. When a Node accesses a Pod, it enters the virtual network via the `ovn0` interface, and the virtual network then connects to the host network through the `ovn0` interface.

The configuration of this Subnet is specified at installation time, you can refer to Built-in Network Settings for more details. To change the CIDR of the Join Subnet after installation please refer to Change Join CIDR.

Check the Join Subnet

The default name of this subnet is `join`. There is generally no need to make changes to the network configuration except the CIDR.

```
# kubectl get subnet join -o yaml
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
  creationTimestamp: "2019-08-06T09:33:43Z"
  generation: 1
  name: join
  resourceVersion: "1571333"
  selfLink: /apis/kubeovn.io/v1/subnets/join
  uid: 9c744810-c678-4d50-8a7d-b8ec12ef91b8
spec:
  cidrBlock: 100.64.0.0/16
  default: false
  excludeIps:
  - 100.64.0.1
  gateway: 100.64.0.1
  gatewayNode: ""
  gatewayType: ""
  natOutgoing: false
  private: false
  protocol: IPv4
```

Check the ovn0 NIC at the node:

```
# ifconfig ovn0
ovn0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1420
        inet 100.64.0.4  netmask 255.255.0.0  broadcast 100.64.255.255
        inet6 fe80::800:ff:fe40:5  prefixlen 64  scopeid 0x20<link>
        ether 0a:00:00:40:00:05  txqueuelen 1000  (Ethernet)
        RX packets 18  bytes 1428 (1.3 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 19  bytes 1810 (1.7 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

3.2.3 Create Custom Subnets

Here we describe the basic operation of how to create a Subnet and associate it with a Namespace, for more advanced configuration, please refer to the subsequent content.

Create Subnet

```
cat <<EOF | kubectl create -f -
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
  name: subnet1
spec:
  protocol: IPv4
  cidrBlock: 10.66.0.0/16
  excludeIps:
  - 10.66.0.1..10.66.0.10
  - 10.66.0.101..10.66.0.151
  gateway: 10.66.0.1
  gatewayType: distributed
  natOutgoing: true
  routeTable: ""
  namespaces:
  - ns1
  - ns2
EOF
```

- `cidrBlock` : Subnet CIDR range, different Subnet CIDRs under the same VPC cannot overlap.
- `excludeIps` : The address list is reserved so that the container network will not automatically assign addresses in the list, which can be used as a fixed IP address assignment segment or to avoid conflicts with existing devices in the physical network in Underlay mode.
- `gateway` : For this subnet gateway address, Kube-OVN will automatically assign the corresponding logical gateway in Overlay mode, and the address should be the underlying physical gateway address in Underlay mode.
- `namespaces` : Bind the list of Namespace for this Subnet. Pods under the Namespace will be assigned addresses from the current Subnet after binding.
- `routeTable` : Associate the route table, default is main table, route table definition please defer to Static Routes

Create Pod in the Subnet

```
# kubectl create ns ns1
namespace/ns1 created

# kubectl run nginx --image=docker.io/library/nginx:alpine -n ns1
deployment.apps/nginx created

# kubectl get pod -n ns1 -o wide
NAME                     READY   STATUS    RESTARTS   AGE   IP           NODE    NOMINATED NODE   READINESS GATES
nginx-74d5899f46-n8wtg   1/1     Running   0          10s   10.66.0.11   node1   <none>           <none>
```

Workload Subnet Binding

By default, Pods will be assigned IP addresses from the subnet belonging to the Namespace. If a specific subnet needs to be specified for a Workload, it can be achieved by setting the Pod's annotation `ovn.kubernetes.io/logical_switch` :

```
apiVersion: v1
kind: Pod
metadata:
  name: another-subnet
  annotations:
    ovn.kubernetes.io/logical_switch: subnet1
spec:
  containers:
  - name: another-subnet
    image: docker.io/library/nginx:alpine
```

If you need to bind a subnet to a Workload type resource such as Deployment or StatefulSet, you need to set the `ovn.kubernetes.io/logical_switch` Annotation in `spec.template.metadata.annotations` .

3.2.4 Overlay Subnet Gateway Settings

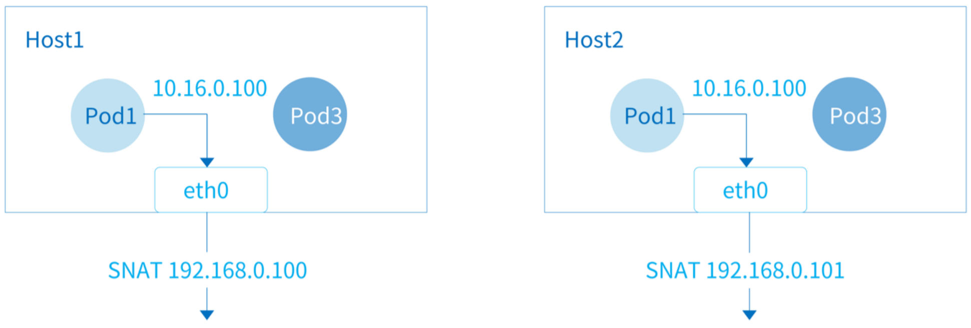
This feature only works for Overlay mode Subnets, Underlay type Subnets need to use the underlying physical gateway to access the external network.

Pods under the Overlay Subnet need to access the external network through a gateway, and Kube-OVN currently supports two types of gateways: distributed gateway and centralized gateway which can be changed in the Subnet spec.

Both types of gateways support the `natOutgoing` setting, which allows the user to choose whether snat is required when the Pod accesses the external network.

Distributed Gateway

The default type of gateway for the Subnet, each node will act as a gateway for the pod on the current node to access the external network. The packets from container will flow into the host network stack from the local `ovn0` NIC, and then forwarding the network according to the host's routing rules. When `natOutgoing` is `true`, the Pod will use the IP of the current host when accessing the external network.

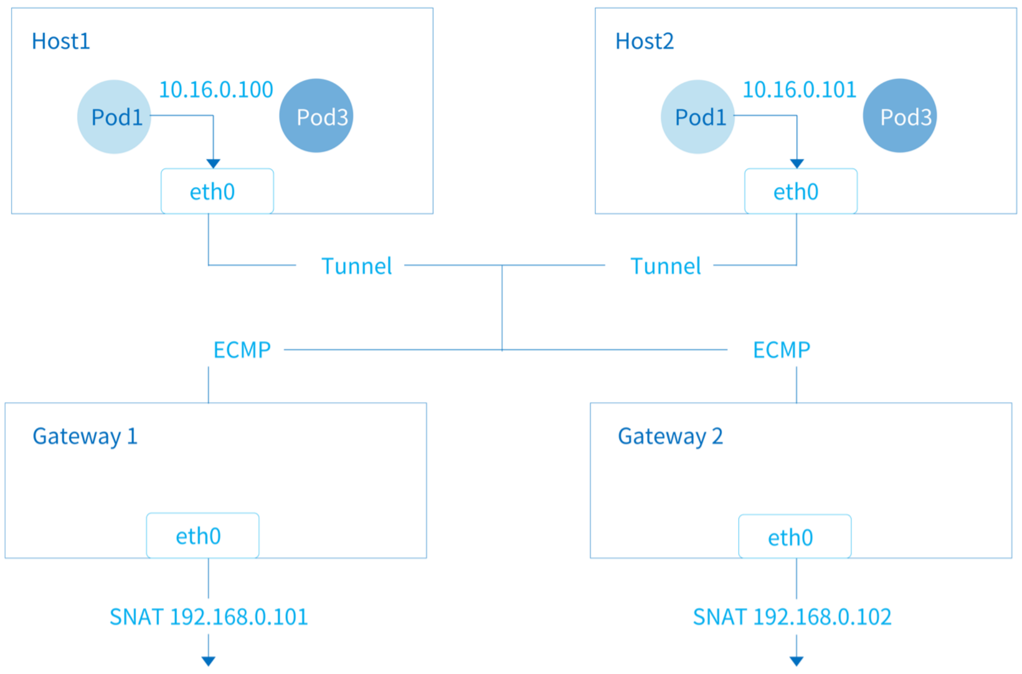


Example of a Subnet, where the `gatewayType` field is `distributed`:

```
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
  name: distributed
spec:
  protocol: IPv4
  cidrBlock: 10.166.0.0/16
  default: false
  excludeIps:
  - 10.166.0.1
  gateway: 10.166.0.1
  gatewayType: distributed
  natOutgoing: true
```

Centralized Gateway

Note: Pods under a centralized subnet cannot be accessed through `hostport` or a NodePort type Service with `externalTrafficPolicy: Local`.



If you want traffic within the Subnet to access the external network using a fixed IP for security operations such as auditing and whitelisting, you can set the gateway type in the Subnet to centralized. In centralized gateway mode, packets from Pods accessing the external network are first routed to the `ovn0` NIC of a specific nodes, and then outbound through the host's routing rules. When `natOutgoing` is `true`, the Pod will use the IP of a specific nodes when accessing the external network.

The centralized gateway example is as follows, where the `gatewayType` field is `centralized` and `gatewayNode` is the NodeName of the particular machine in Kubernetes.

```yaml
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
  name: centralized
spec:
  protocol: IPv4
  cidrBlock: 10.166.0.0/16
  default: false
  excludeIps:
  - 10.166.0.1
  gateway: 10.166.0.1
  gatewayType: centralized
  gatewayNode: "node1,node2"
  natOutgoing: true
```

- If a centralized gateway wants to specify a specific NIC of a machine for outbound networking, `gatewayNode` format can be changed to `kube-ovn-worker:172.18.0.2, kube-ovn-control-plane:172.18.0.3`.
- The centralized gateway defaults to primary-backup mode, with only the primary node performing traffic forwarding. If you need to switch to ECMP mode, please refer to ECMP Settings.
- The spec field `enableEcmp` has been added to the subnet crd definition since Kube-OVN v1.12.0 to migrate the ECMP switch to the subnet level. You can set whether to enable ECMP mode based on different subnets. The `enable-ecmp` parameter in the `kube-ovn-controller` deployment is no longer used. After the previous version is upgraded to v1.12.0, the subnet switch will automatically inherit the value of the original global switch parameter.

3.2.5 Subnet ACL

For scenarios with fine-grained ACL control, Subnet of Kube-OVN provides ACL to enable fine-grained rules.

The ACL rules in Subnet are the same as the ACL rules in OVN, and you can refer to ovn-nb ACL Table for more details. The supported filed in `match` can refer to ovn-sb Logical Flow Table.

Example of an ACL rule that allows Pods with IP address `10.10.0.2` to access all addresses, but does not allow other addresses to access itself, is as follows:

```yaml
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
  name: acl
spec:
  allowEWTraffic: false
  acls:
    - action: drop
      direction: to-lport
      match: ip4.dst == 10.10.0.2 && ip
      priority: 1002
    - action: allow-related
      direction: from-lport
      match: ip4.src == 10.10.0.2 && ip
      priority: 1002
  cidrBlock: 10.10.0.0/24
```

In some scenarios, users hope that the internal traffic of the subnet configured with ACL rules will not be affected, which can be achieved by configuring `allowEWTraffic: true`.

3.2.6 Subnet Isolation

The function of Subnet ACL can cover the function of Subnet isolation with better flexibility, we recommend using Subnet ACL to do the corresponding configuration.

By default the Subnets created by Kube-OVN can communicate with each other, and Pods can also access external networks through the gateway.

To control access between Subnets, set `private` to true in the subnet spec, and the Subnet will be isolated from other Subnets and external networks and can only communicate within the Subnet. If you want to open a whitelist, you can set it by `allowSubnets`. The CIDRs in `allowSubnets` can access the Subnet bidirectionally.

Enable Subnet Isolation Examples

```yaml
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
  name: private
spec:
  protocol: IPv4
  default: false
  namespaces:
  - ns1
  - ns2
  cidrBlock: 10.69.0.0/16
  private: true
  allowSubnets:
  - 10.16.0.0/16
  - 10.18.0.0/16
```

3.2.7 Underlay Settings

This part of the feature is only available for Underlay type Subnets.

- `vlan`: If an Underlay network is used, this field is used to control which Vlan CR the Subnet is bound to. This option defaults to the empty string, meaning that the Underlay network is not used.
- `logicalGateway`: Some Underlay environments are pure Layer 2 networks, with no physical Layer 3 gateway. In this case a virtual gateway can be set up with the OVN to connect the Underlay and Overlay networks. The default value is: `false`.

3.2.8 Gateway Check Settings

By default `kube-ovn-cni` will request the gateway using ICMP or ARP protocol after starting the Pod and wait for the return to verify that the network is working properly. Some Underlay environment gateways cannot respond to ICMP requests, or scenarios that do not require external connectivity, the checking can be disabled .

```yaml
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
  name: disable-gw-check
spec:
  disableGatewayCheck: true
```

3.2.9 Multicast-Snoop Setting

By default, if a Pod in a subnet sends a multicast packet, OVN's default behavior is to broadcast the multicast packet to all Pods in the subnet. If turned on the subnet's multicast snoop switch, OVN will forward based on the multicast table `Multicast_Group` in the `South Database` instead of broadcasting.

```yaml
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
  name: sample1
spec:
  enableMulticastSnoop: true
```

3.2.10 Subnet MTU Setting

Configure the MTU of the Pod under Subnet. After configuration, you need to restart the Pod under Subnet to take effect.

```yaml
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
  name: sample1
spec:
  mtu: 1300
```

3.2.11 Other Advanced Settings

- Configure IPPool
- Default VPC NAT Policy Rule
- Manage QoS
- Manage Multiple Interface
- DHCP
- External Gateway
- Cluster Inter-Connection with OVN-IC
- VIP Reservation

PDF **Slack** **Support**

July 30, 2025

May 24, 2022

GitHub +2

3.2.12 Comments

3.3 DualStack

Different subnets in Kube-OVN can support different IP protocols. IPv4, IPv6 and dual-stack types of subnets can exist within one cluster. However, it is recommended to use a uniform protocol type within a cluster to simplify usage and maintenance.

In order to support dual-stack, the host network needs to meet the dual-stack requirements, and the Kubernetes-related parameters need to be adjusted, please refer to official guide to dual-stack.

3.3.1 Create dual-stack Subnet

When configuring a dual stack Subnet, you only need to set the corresponding subnet CIDR format as `cidr=<IPv4 CIDR>,<IPv6 CIDR>`.

The CIDR order requires IPv4 to come first and IPv6 to come second, as follows.

```
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
  name: ovn-test
spec:
  cidrBlock: 10.16.0.0/16,fd00:10:16::/64
  excludeIps:
  - 10.16.0.1
  - fd00:10:16::1
  gateway: 10.16.0.1,fd00:10:16::1
```

If you need to use a dual stack for the default subnet during installation, you need to change the following parameters in the installation script:

```
POD_CIDR="10.16.0.0/16,fd00:10:16::/64"
JOIN_CIDR="100.64.0.0/16,fd00:100:64::/64"
```

3.3.2 Check Pod Address

Pods configured for dual-stack networks will be assigned both IPv4 and IPv6 addresses from that subnet, and the results will be displayed in the annotation of the Pod:

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    ovn.kubernetes.io/allocated: "true"
    ovn.kubernetes.io/cidr: 10.16.0.0/16,fd00:10:16::/64
    ovn.kubernetes.io/gateway: 10.16.0.1,fd00:10:16::1
    ovn.kubernetes.io/ip_address: 10.16.0.9,fd00:10:16::9
    ovn.kubernetes.io/logical_switch: ovn-default
    ovn.kubernetes.io/mac_address: 00:00:00:14:88:09
    ovn.kubernetes.io/network_types: geneve
    ovn.kubernetes.io/routed: "true"
...
podIP: 10.16.0.9
  podIPs:
  - ip: 10.16.0.9
  - ip: fd00:10:16::9
```

PDF Slack Support

February 15, 2023

May 24, 2022

GitHub

3.3.3 Comments

3.4 Fixed Addresses

By default, Kube-OVN randomly assigns IPs and Macs based on the Subnet to which the Pod's Namespace belongs. For workloads that require fixed addresses, Kube-OVN provides multiple methods of fixing addresses depending on the scenario.

- Single Pod fixed IP/Mac.
- Workload IP Pool to specify fixed addresses.
- StatefulSet fixed address.
- KubeVirt VM fixed address.

3.4.1 Single Pod Fixed IP/Mac

You can specify the IP/Mac required for the Pod by annotation when creating the Pod. The `kube-ovn-controller` will skip the address random assignment phase and use the specified address directly after conflict detection, as follows:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ippool
  labels:
    app: ippool
spec:
  replicas: 2
  selector:
    matchLabels:
      app: ippool
  template:
    metadata:
      labels:
        app: ippool
      annotations:
        ovn.kubernetes.io/ip_pool: 10.16.0.15,10.16.0.16,10.16.0.17 // for dualstack ippool use semicolon to separate addresses 10.16.0.15,fd00:10:16::000E;
10.16.0.16,fd00:10:16::0
    spec:
      containers:
        - name: ippool
          image: docker.io/library/nginx:alpine
```

The following points need to be noted when using annotation.

1. The IP/Mac used cannot conflict with an existing IP/Mac.
2. The IP must be in the CIDR range of the Subnet it belongs to.
3. You can specify only IP or Mac. When you specify only one, the other one will be assigned randomly.

3.4.2 Workload IP Pool

Kube-OVN supports setting fixed IPs for Workloads (Deployment/StatefulSet/DaemonSet/Job/CronJob) via annotation `ovn.kubernetes.io/ip_pool`. `kube-ovn-controller` will automatically select the IP specified in `ovn.kubernetes.io/ip_pool` and perform conflict detection.

The Annotation of the IP Pool needs to be added to the `annotation` field in the `template`. In addition to Kubernetes built-in workload types, other user-defined workloads can also be assigned fixed addresses using the same approach.

Deployment With Fixed IPs

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  namespace: ls1
  name: starter-backend
  labels:
    app: starter-backend
spec:
  replicas: 2
  selector:
    matchLabels:
      app: starter-backend
  template:
    metadata:
      labels:
        app: starter-backend
      annotations:
        ovn.kubernetes.io/ip_pool: 10.16.0.15,10.16.0.16,10.16.0.17 // for dualstack ippool use semicolon to separate addresses 10.16.0.15,fd00:10:16::000E;
10.16.0.16,fd00:10:16::000F;10.16.0.17,fd00:10:16::0010
    spec:
      containers:
      - name: backend
        image: docker.io/library/nginx:alpine
```

Using a fixed IP for Workload requires the following:

1. The IP in `ovn.kubernetes.io/ip_pool` should belong to the CIDR of the Subnet.
2. The IP in `ovn.kubernetes.io/ip_pool` cannot conflict with an IP already in use.
3. When the number of IPs in `ovn.kubernetes.io/ip_pool` is less than the number of replicas, the extra Pods will not be created. You need to adjust the number of IPs in `ovn.kubernetes.io/ip_pool` according to the update policy of the workload and the scaling plan.

3.4.3 StatefulSet Fixed Address

StatefulSet supports fixed IP by default, and like other Workload, you can use `ovn.kubernetes.io/ip_pool` to specify the range of IP used by a Pod.

Since StatefulSet is mostly used for stateful services, which have higher requirements for fixed addresses, Kube-OVN has made special enhancements:

1. Pods are assigned IPs in `ovn.kubernetes.io/ip_pool` in order. For example, if the name of the StatefulSet is web, web-0 will use the first IP in `ovn.kubernetes.io/ip_pool`, web-1 will use the second IP, and so on.
2. The logical_switch_port in the OVN is not deleted during update or deletion of the StatefulSet Pod, and the newly generated Pod directly reuses the old logical port information. Pods can therefore reuse IP/Mac and other network information to achieve similar state retention as StatefulSet Volumes.
3. Based on the capabilities of 2, for StatefulSet without the `ovn.kubernetes.io/ip_pool` annotation, a Pod is randomly assigned an IP/Mac when it is first generated, and then the network information remains fixed for the lifetime of the StatefulSet.

StatefulSet Example

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: web
spec:
  serviceName: "nginx"
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: docker.io/library/nginx:alpine
        ports:
        - containerPort: 80
          name: web
```

You can try to delete the Pod under StatefulSet to observe if the Pod IP changes.

Updating StatefulSet Pod IPs

Since the IPs of StatefulSet Pods are bound to their lifecycle along with the Pod names, directly updating the `ovn.kubernetes.io/ip_pool` annotation in the StatefulSet will not update the Pod IPs.

If you need to update the IPs of StatefulSet Pods, first scale the StatefulSet replicas down to 0. Then, update the annotation and restore the StatefulSet replicas afterward.

3.4.4 KubeVirt VM Fixed Address

For VM instances created by KubeVirt, `kube-ovn-controller` can assign and manage IP addresses in a similar way to the StatefulSet Pod. This allows VM instances address fixed during start-up, shutdown, upgrade, migration, and other operations throughout their lifecycle, making them more compatible with the actual virtualization user experience.

PDF **Slack** **Support**

June 13, 2025

May 20, 2022

GitHub

3.4.5 Comments

3.5 Reserved IP for Specific Resources

IP is used to maintain the IP address of Pod or VirtualMachine (VM). The lifecycle maintenance of IP includes the following business scenarios:

- IP is created with Pod and deleted with Pod.
- VM IP is retained by configuring ENABLE_KEEP_VM_IP. This type of IP is created with VM Pod, but not deleted with VM Pod.
- Statefulset Pod IP will automatically decide whether to retain Pod IP based on the capacity of Statefulset and the sequence number of Pod.

In actual business use, it is often necessary to reserve IP resources in advance. The business scenarios for reserving IP include the following two types:

- Pod or VM has been created and needs to reserve IP
- Pod or VM has not been created yet and needs to reserve IP

In the above scenarios, the naming correspondence between IP and Pod remains consistent:

- Naming format of Pod IP: Pod-name.Pod-namespace(.subnet-provider)
- Naming format of VM Pod IP: vm-name.Pod-namespace.(subnet-provider)

If you are unsure about these parameters and just want to simply reserve IP, please use IP Pool.

Specifically, this function is to reserve IP for specific Pod or VM. In the creation process of reserved IP, it is necessary to specify resource name, resource type, namespace, subnet and other necessary parameters. For fixed IP reservation, it is necessary to specify IP address and MAC address (if necessary).

Note: The previous implementation of Pod using vip to occupy IP is deprecated. (These two functions overlap)

3.5.1 1. Create Reserved IP

- Pod or VM has been created and needs to reserve IP
- Pod or VM has not been created yet and needs to reserve IP

Reserving IP is just an extended function, which supports Pod to use the reserved IP, but the usage method, naming rules and business logic of IP created with Pod are consistent. Therefore, when creating a reserved IP, it is necessary to clearly know what resources will use this IP in the future, and the type, Pod name or VM name, namespace, subnet and other information must be accurately filled in. When using this IP, the business needs to verify whether the Pod and VM bound to the IP are consistent with the attributes of the IP itself, otherwise the Pod or VM cannot use this IP.

The creation process of IP CR controller only handles the business scenario of reserving IP, and does not handle the IP resources created with Pod. In the process of IP resources created with Pod, the creation of LSP is before the creation of IP CR, so it can be judged based on whether LSP exists. In the processing process of IP CR controller, it will first judge whether LSP exists. If it exists, it will not handle this business logic: the business logic of IP created with Pod. The creation of reserved IP supports automatic allocation of IP and manual specification of IP. The creation process of IP will only implement IP occupation, but will not create LSP. The creation of LSP is still maintained in the process of Pod creation. The creation process of IP CR is just to reserve IP. This kind of IP will automatically add a keep-ip label, indicating that it is permanently reserved and will not be cleaned up with the deletion of Pod. This kind of reserved IP needs to be managed by the business or administrator, and GC will not automatically handle this IP.

1.1 Auto Allocate Address for Reserved IP

If you just want to reserve some IPs and have no requirements for the IP address itself, you can use the following yaml to create:

```
# cat 01-dynamic.yaml
```

```
apiVersion: kubeovn.io/v1
kind: IP
metadata:
  name: vm-dynamic-01.default
spec:
  subnet: ovn-default
  podType: "VirtualMachine"
  namespace: default
  podName: vm-dynamic-01
```

- `subnet` : The IP address is reserved from the Subnet.
- `podType` : Used to specify the Owner type of the Pod: `StatefulSet` , `VirtualMachine` .
- `podName` : Pod name or VirtualMachine name.
- `namespace` : Used to specify the namespace where the IP resource resides, Pod namespace or VirtualMachine namespace.

Note: These IP properties are not allowed to change

Query the IP address after the IP address is created:

```
# kubectl get subnet ovn-default
NAME          PROVIDER   VPC           PROTOCOL   CIDR           PRIVATE   NAT    DEFAULT   GATEWAYTYPE   V4USED   V4AVAILABLE   V6USED   V6AVAILABLE
EXCLUDEIPS      U2OINTERCONNECTIONIP
ovn-default   ovn        ovn-cluster   IPv4       10.16.0.0/16   false     true   true      distributed   7        65526         0        0
["10.16.0.1"]

# kubectl get ip vm-dynamic-01.default -o yaml
apiVersion: kubeovn.io/v1
kind: IP
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"kubeovn.io/v1","kind":"IP","metadata":{"annotations":{},"name":"vm-dynamic-01.default"},"spec":{"namespace":"default","podName":"vm-
dynamic-01","podType":"VirtualMachine","subnet":"ovn-default"}}
  creationTimestamp: "2024-01-29T03:05:40Z"
  finalizers:
  - kube-ovn-controller
  generation: 2
  labels:
    ovn.kubernetes.io/ip_reserved: "true" # reserved ip
    ovn.kubernetes.io/node-name: ""
    ovn.kubernetes.io/subnet: ovn-default
  name: vm-dynamic-01.default
  resourceVersion: "1571"
  uid: 89d05a26-294a-450b-ab63-1eaa957984d7
spec:
  attachIps: []
  attachMacs: []
  attachSubnets: []
  containerID: ""
  ipAddress: 10.16.0.13
  macAddress: 00:00:00:86:C6:36
  namespace: default
  nodeName: ""
  podName: vm-dynamic-01
  podType: VirtualMachine
  subnet: ovn-default
  v4IpAddress: 10.16.0.13
  v6IpAddress: ""

# kubectl ko nbctl show ovn-default | grep vm-dynamic-01.default
# The reserved IP address is assigned only in the IPAM, and the LSP is not created. Therefore, you cannot view the IP address
```

1.2 Specifies the reserved IP address

If there is a need for the IP address of the reserved IP, the following yaml can be used for fixed allocation:

```
# cat  02-static.yaml

apiVersion: kubeovn.io/v1
kind: IP
metadata:
  name: pod-static-01.default
spec:
  subnet: ovn-default
  podType: ""
  namespace: default
  podName: pod-static-01
  v4IpAddress: 10.16.0.3
  v6IpAddress:

# kubectl get ip pod-static-01.default -o yaml
apiVersion: kubeovn.io/v1
kind: IP
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"kubeovn.io/v1","kind":"IP","metadata":{"annotations":{},"name":"pod-static-01.default"},"spec":{"namespace":"default","podName":"pod-
static-01","podType":"","subnet":"ovn-default","v4IpAddress":"10.16.0.3","v6IpAddress":null}}
  creationTimestamp: "2024-01-29T03:08:28Z"
  finalizers:
  - kube-ovn-controller
  generation: 2
  labels:
    ovn.kubernetes.io/ip_reserved: "true"
    ovn.kubernetes.io/node-name: ""
    ovn.kubernetes.io/subnet: ovn-default
  name: pod-static-01.default
  resourceVersion: "1864"
  uid: 11fc767d-f57d-4520-89f9-448f9b272bca
spec:
  attachIps: []
  attachMacs: []
  attachSubnets: []
  containerID: ""
  ipAddress: 10.16.0.3
  macAddress: 00:00:00:4D:B4:36
  namespace: default
  nodeName: ""
  podName: pod-static-01
  podType: ""
  subnet: ovn-default
  v4IpAddress: 10.16.0.3
  v6IpAddress: ""
```

- `v4IpAddress` : Specify an IPv4 address that is within the CIDR range of the subnet.
- `v6IpAddress` : Specify an IPv6 address that is within the CIDR range of the subnet.

Pod use reserved IP

Note: The Pod(VMS) name and namespace must be the same as the reserved IP address, otherwise the Pod(VMS) cannot use the IP address.

After a Pod or VM is deleted, the IP CR remains.

```
# kubectl get po -n default -o wide
NAME            READY   STATUS    RESTARTS   AGE   IP          NODE              NOMINATED NODE   READINESS GATES
pod-static-01   1/1     Running   0          30s   10.16.0.3   kube-ovn-worker   <none>           <none>
```

3.5.2 2. Delete

The kube-ovn-controller GC process does not clean up individual IP resources. To clear an IP address and its LSPS, delete the IP CR resource.

The IP deletion process formats the ipam key and LSP name based on the podName, namespace, and subnet provider in the IP attribute, releases the IPAM slot, deletes the LSP, and clears the Finalizer of the IP.

PDF **Slack** **Support**

July 30, 2025

January 25, 2024

GitHub

3.5.3 Comments

3.6 Configure IPPool

IPPool is a more granular IPAM management unit than Subnet. You can subdivide the subnet segment into multiple units through IPPool, and each unit is bound to one or more namespaces.

3.6.1 Instructions

Below is an example:

```yaml
apiVersion: kubeovn.io/v1
kind: IPPool
metadata:
  name: pool-1
spec:
  subnet: ovn-default
  ips:
  - "10.16.0.201"
  - "10.16.0.210/30"
  - "10.16.0.220..10.16.0.230"
  namespaces:
  - ns-1
```

Field description:

Field	Usage	Comment
subnet	Specify the subnet to which it belongs	Required
ips	Specify IP ranges	Support three formats: , and ... Support IPv6.
namespaces	Specifies the bound namespaces	Optional

3.6.2 Precautions

1. To ensure compatibility with Workload Universal IP Pool Fixed Address, the name of the IP pool cannot be an IP address;
2. The `.spec.ips` of the IP pool can specify an IP address beyond the scope of the subnet, but the actual effective IP address is the intersection of `.spec.ips` and the CIDR of the subnet;
3. Different IP pools of the same subnet cannot contain the same (effective) IP address;
4. The `.spec.ips` of the IP pool can be modified dynamically;
5. The IP pool will inherit the reserved IP of the subnet. When randomly assigning an IP address from the IP pool, the reserved IP included in the IP pool will be skipped;
6. When randomly assigning an IP address from a subnet, it will only be assigned from a range other than all IP pools in the subnet.

PDF **Slack** **Support**

July 30, 2025

July 10, 2023

GitHub

3.6.3 Comments

3.7 Custom Routes

Custom routes can be configured via Pod's annotations. Here is an example:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: custom-routes
  annotations:
    ovn.kubernetes.io/routes: |
      [{
        "dst": "192.168.0.101/24",
        "gw": "10.16.0.254"
      }, {
        "gw": "10.16.0.254"
      }]
spec:
  containers:
  - name: nginx
    image: docker.io/library/nginx:alpine
```

Do not set the `dst` field if you want to configure the default route.

For workloads such as Deployment, DaemonSet and StatefulSet, custom routes must be configured via `.spec.template.metadata.annotations`:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: custom-routes
  labels:
    app: nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
      annotations:
        ovn.kubernetes.io/routes: |
          [{
            "dst": "192.168.0.101/24",
            "gw": "10.16.0.254"
          }, {
            "gw": "10.16.0.254"
          }]
    spec:
      containers:
      - name: nginx
        image: docker.io/library/nginx:alpine
```

PDF Slack Support

September 26, 2023

February 16, 2023

GitHub

3.7.1 Comments

3.8 EIP and SNAT

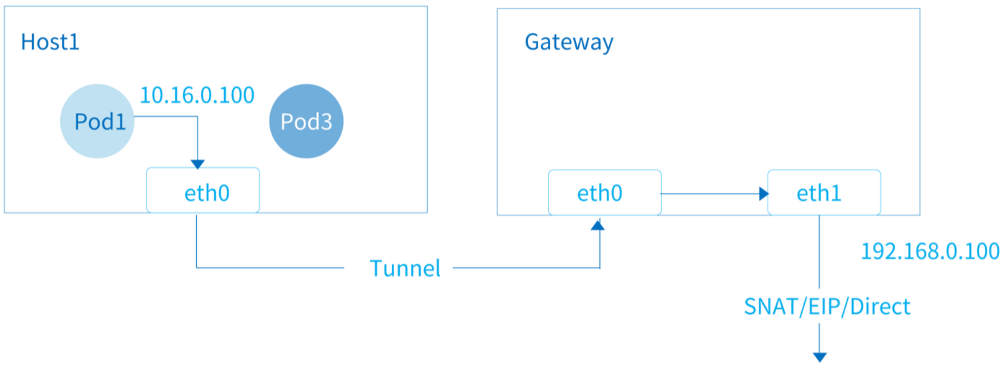
This configuration is for the network under the default VPC. User-defined VPCS support two types of NAT. Please refer to:

- VPC Iptables NAT Gateway
- VPC OVN NAT Gateway

Any VPC supports the use of any one or more external subnets, but some factors need to be considered:

- If the user only needs to use the OVN NAT function for subnets under the default VPC and uses it through the pod annotation method, please refer to the current documentation.
- If the subnets under any VPC of the user need to use the OVN NAT function, or wish to maintain one or more external networks through provider network, vlan, subnet CRD, as well as through OVN-EIP, OVN-DNAT, OVN-FIP, For maintaining EIP and NAT, please refer to ovn-snat CRD VPC OVN NAT Gateway.
- If the subnets under any VPC of the user need to use the Iptables NAT function, please refer to VPC Iptables NAT Gateway.

Kube-OVN supports SNAT and EIP functionality at the Pod level using the L3 Gateway feature in OVN. By using SNAT, a group of Pods can share an IP address for external access. With the EIP feature, a Pod can be directly associated with an external IP. External services can access the Pod directly through the EIP, and the Pod will also access external services through this EIP.



3.8.1 Preparation

- In order to use the OVN's L3 Gateway capability, a separate NIC must be bridged into the OVS bridge for overlay and underlay network communication. The host must have other NICs for management.
- Since packets passing through NAT will go directly to the Underlay network, it is important to confirm that such packets can pass safely on the current network architecture.
- Currently, there is no conflict detection for EIP and SNAT addresses, and an administrator needs to manually assign them to avoid address conflicts.

3.8.2 Create Config

Create ConfigMap `ovn-external-gw-config` in `kube-system` Namespace:

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: ovn-external-gw-config
  namespace: kube-system
data:
  enable-external-gw: "true"
  external-gw-nodes: "kube-ovn-worker"
  external-gw-nic: "eth1"
  external-gw-addr: "172.56.0.1/16"
  nic-ip: "172.56.0.254/16"
  nic-mac: "16:52:f3:13:6a:25"
```

- `enable-external-gw`: Whether to enable SNAT and EIP functions.
- `type`: `centralized` or `distributed`, Default is `centralized` If `distributed` is used, all nodes of the cluster need to have the same name NIC to perform the gateway function.
- `external-gw-nodes`: In `centralized` mode, The names of the node performing the gateway role, comma separated..
- `external-gw-nic`: The name of the NIC that performs the role of a gateway on the node.
- `external-gw-addr`: The IP and mask of the physical network gateway.
- `nic-ip`, `nic-mac`: The IP and Mac assigned to the logical gateway port needs to be an unoccupied IP and Mac for the physical subnet.

3.8.3 Confirm the Configuration Take Effect

Check the OVN-NB status to confirm that the `ovn-external` logical switch exists and that the correct address and chassis are bound to the `ovn-cluster-ovn-external` logical router port.

```
# kubectl ko nbctl show
switch 3de4cea7-1a71-43f3-8b62-435a57ef16a6 (external)
    port localnet.external
        type: localnet
        addresses: ["unknown"]
    port external-ovn-cluster
        type: router
        router-port: ovn-cluster-external
router e1eb83ad-34be-4ed5-9a02-fcc8b1d357c4 (ovn-cluster)
    port ovn-cluster-external
        mac: "ac:1f:6b:2d:33:f1"
        networks: ["172.56.0.100/16"]
        gateway chassis: [a5682814-2e2c-46dd-9c1c-6803ef0dab66]
```

Check the OVS status to confirm that the corresponding NIC is bridged into the `br-external` bridge:

```
# kubectl ko vsctl ${gateway node name} show
e7d81150-7743-4d6e-9e6f-5c688232e130
    Bridge br-external
        Port br-external
            Interface br-external
                type: internal
        Port eth1
            Interface eth1
        Port patch-localnet.external-to-br-int
            Interface patch-localnet.external-to-br-int
                type: patch
                options: {peer=patch-br-int-to-localnet.external}
```

3.8.4 Config EIP amd SNAT on Pod

SNAT and EIP can be configured by adding the `ovn.kubernetes.io/snat` or `ovn.kubernetes.io/eip` annotation to the Pod, respectively:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-snat
  annotations:
    ovn.kubernetes.io/snat: 172.56.0.200
spec:
  containers:
  - name: pod-snat
    image: docker.io/library/nginx:alpine
---
apiVersion: v1
kind: Pod
metadata:
  name: pod-eip
  annotations:
    ovn.kubernetes.io/eip: 172.56.0.233
spec:
  containers:
  - name: pod-eip
    image: docker.io/library/nginx:alpine
```

The EIP or SNAT rules configured by the Pod can be dynamically adjusted via kubectl or other tools, remember to remove the `ovn.kubernetes.io/routed` annotation to trigger the routing change.

```
kubectl annotate pod pod-gw ovn.kubernetes.io/eip=172.56.0.221 --overwrite
kubectl annotate pod pod-gw ovn.kubernetes.io/routed-
```

When the EIP or SNAT takes into effect, the `ovn.kubernetes.io/routed` annotation will be added back.

3.8.5 Advanced Configuration

Some args of `kube-ovn-controller` allow for advanced configuration of SNAT and EIP:

- `--external-gateway-config-ns` : The Namespace of Configmap `ovn-external-gw-config` , default is `kube-system` .
- `--external-gateway-net` : The name of the bridge to which the physical NIC is bridged, default is `external` .
- `--external-gateway-vlanid` : Physical network Vlan Tag number, default is 0, i.e. no Vlan is used.

PDF **Slack** **Support**

July 30, 2025

May 24, 2022

GitHub

3.8.6 Comments

3.9 Manage QoS

Kube-OVN supports two types of Pod level QoS:

- Maximum bandwidth limit QoS.
- `linux-netem`, QoS for simulating latency and packet loss that can be used for simulation testing.

Currently, only Pod level QoS is supported, and QoS restrictions at the Namespace or Subnet level are not supported.

3.9.1 Maximum Bandwidth Limit QoS

This type of QoS can be dynamically configured via Pod annotation and can be adjusted without restarting running Pod. Bandwidth speed limit unit is `Mbit/s`.

```
apiVersion: v1
kind: Pod
metadata:
  name: qos
  namespace: ls1
  annotations:
    ovn.kubernetes.io/ingress_rate: "3"
    ovn.kubernetes.io/egress_rate: "1"
spec:
  containers:
  - name: qos
    image: docker.io/library/nginx:alpine
```

Use annotation to dynamically adjust QoS:

```
kubectl annotate --overwrite  pod nginx-74d5899f46-d7qkn ovn.kubernetes.io/ingress_rate=3
```

Test QoS

Deploy the containers needed for performance testing:

```
kind: DaemonSet
apiVersion: apps/v1
metadata:
  name: perf
  namespace: ls1
  labels:
    app: perf
spec:
  selector:
    matchLabels:
      app: perf
  template:
    metadata:
      labels:
        app: perf
    spec:
      containers:
      - name: nginx
        image: docker.io/kubeovn/perf
```

Exec into one Pod and run iperf3 server:

```
# kubectl exec -it perf-4n4gt -n ls1 sh
# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Exec into the other Pod and run iperf3 client to connect above server address:

```
# kubectl exec -it perf-d4mqc -n ls1 sh
# iperf3 -c 10.66.0.12
Connecting to host 10.66.0.12, port 5201
[  4] local 10.66.0.14 port 51544 connected to 10.66.0.12 port 5201
[ ID] Interval           Transfer     Bandwidth       Retr  Cwnd
[  4]   0.00-1.00   sec  86.4 MBytes   725 Mbits/sec    3    350 KBytes
[  4]   1.00-2.00   sec  89.9 MBytes   754 Mbits/sec  118    473 KBytes
[  4]   2.00-3.00   sec   101 MBytes   848 Mbits/sec  184    586 KBytes
[  4]   3.00-4.00   sec   104 MBytes   875 Mbits/sec  217    671 KBytes
[  4]   4.00-5.00   sec   111 MBytes   935 Mbits/sec  175    772 KBytes
[  4]   5.00-6.00   sec   100 MBytes   840 Mbits/sec  658    598 KBytes
[  4]   6.00-7.00   sec   106 MBytes   890 Mbits/sec  742    668 KBytes
[  4]   7.00-8.00   sec   102 MBytes   857 Mbits/sec  764    724 KBytes
[  4]   8.00-9.00   sec  97.4 MBytes   817 Mbits/sec  1175    764 KBytes
[  4]   9.00-10.00  sec   111 MBytes   934 Mbits/sec  1083    838 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth       Retr
[  4]   0.00-10.00  sec  1010 MBytes   848 Mbits/sec  5119             sender
[  4]   0.00-10.00  sec  1008 MBytes   846 Mbits/sec                  receiver

iperf Done.
```

Modify the ingress bandwidth QoS for the first Pod:

```
kubectl annotate --overwrite  pod perf-4n4gt -n ls1 ovn.kubernetes.io/ingress_rate=30
```

Test the Pod bandwidth again from the second Pod:

```
# iperf3 -c 10.66.0.12
Connecting to host 10.66.0.12, port 5201
[  4] local 10.66.0.14 port 52372 connected to 10.66.0.12 port 5201
[ ID] Interval           Transfer     Bandwidth       Retr  Cwnd
[  4]   0.00-1.00   sec  3.66 MBytes  30.7 Mbits/sec    2   76.1 KBytes
[  4]   1.00-2.00   sec  3.43 MBytes  28.8 Mbits/sec    0    104 KBytes
[  4]   2.00-3.00   sec  3.50 MBytes  29.4 Mbits/sec    0    126 KBytes
[  4]   3.00-4.00   sec  3.50 MBytes  29.3 Mbits/sec    0    144 KBytes
[  4]   4.00-5.00   sec  3.43 MBytes  28.8 Mbits/sec    0    160 KBytes
[  4]   5.00-6.00   sec  3.43 MBytes  28.8 Mbits/sec    0    175 KBytes
[  4]   6.00-7.00   sec  3.50 MBytes  29.3 Mbits/sec    0    212 KBytes
[  4]   7.00-8.00   sec  3.68 MBytes  30.9 Mbits/sec    0    294 KBytes
[  4]   8.00-9.00   sec  3.74 MBytes  31.4 Mbits/sec    0    398 KBytes
[  4]   9.00-10.00  sec  3.80 MBytes  31.9 Mbits/sec    0    526 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth       Retr
[  4]   0.00-10.00  sec  35.7 MBytes  29.9 Mbits/sec    2             sender
[  4]   0.00-10.00  sec  34.5 MBytes  29.0 Mbits/sec                  receiver

iperf Done.
```

3.9.2 linux-netem QoS

Pod can use annotation below to config `linux-netem` type QoS: `ovn.kubernetes.io/latency` `ovn.kubernetes.io/limit` and `ovn.kubernetes.io/loss`.

To install netem related modules on RHEL series operating systems, follow these instructions: yum install -y kernel-modules-extra && modprobe sch_netem

- `ovn.kubernetes.io/latency`: Set the Pod traffic delay to an integer value in ms.
- `ovn.kubernetes.io/jitter`: Set the Pod traffic jitter to an integer value in ms.
- `ovn.kubernetes.io/limit`: Set the maximum number of packets that the `qdisc` queue can hold, and takes an integer value, such as 1000.
- `ovn.kubernetes.io/loss`: Set packet loss probability, the value is float type, for example, the value is 20, then it is set 20% packet loss probability.

PDF | Slack | Support

July 30, 2025

May 24, 2022

GitHub

3.9.3 Comments

3.10 Webhook

Using Webhook, you can verify CRD resources within Kube-OVN. Currently, Webhook mainly performs fixed IP address conflict detection and Subnet CIDR conflict detection, and prompts errors when such conflicts happen.

Since Webhook intercepts all Subnet and Pod creation requests, you need to deploy Kube-OVN first and Webhook later.

3.10.1 Install Cert-Manager

Webhook deployment requires certificate, we use cert-manager to generate the associated certificate, we need to deploy cert-manager before deploying Webhook.

You can use the following command to deploy cert-manager:

```
kubectl apply -f https://github.com/cert-manager/cert-manager/releases/download/v1.8.0/cert-manager.yaml
```

More cert-manager usage please refer to cert-manager document.

3.10.2 Install Webhook

Download Webhook yaml and install:

```
# kubectl apply -f https://raw.githubusercontent.com/kubeovn/kube-ovn/release-1.14/yamls/webhook.yaml
deployment.apps/kube-ovn-webhook created
service/kube-ovn-webhook created
validatingwebhookconfiguration.admissionregistration.k8s.io/kube-ovn-webhook created
certificate.cert-manager.io/kube-ovn-webhook-serving-cert created
issuer.cert-manager.io/kube-ovn-webhook-selfsigned-issuer created
```

3.10.3 Verify Webhook Take Effect

Check the running Pod and get the Pod IP `10.16.0.15`:

```
# kubectl get pod -o wide
NAME                      READY   STATUS    RESTARTS   AGE     IP           NODE              NOMINATED NODE   READINESS GATES
static-7584848b74-fw9dm   1/1     Running   0          2d13h   10.16.0.15   kube-ovn-worker   <none>
```

Write yaml to create a Pod with the same IP:

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    ovn.kubernetes.io/ip_address: 10.16.0.15
    ovn.kubernetes.io/mac_address: 00:00:00:53:6B:B6
  labels:
    app: static
  managedFields:
  name: staticip-pod
  namespace: default
spec:
  containers:
  - image: docker.io/library/nginx:alpine
    imagePullPolicy: IfNotPresent
    name: qatest
```

When using the above yaml to create a fixed address Pod, it prompts an IP address conflict:

```
# kubectl apply -f pod-static.yaml
Error from server (annotation ip address 10.16.0.15 is conflict with ip crd static-7584848b74-fw9dm.default 10.16.0.15): error when creating "pod-static.yaml": admission webhook "pod-ip-validaing.kube-ovn.io" denied the request: annotation ip address 10.16.0.15 is conflict with ip crd static-7584848b74-fw9dm.default 10.16.0.15
```

PDF Slack Support

July 30, 2025

May 24, 2022

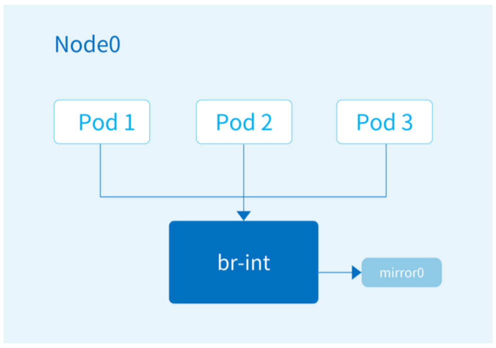
GitHub

3.10.4 Comments

3.11 Traffic Mirror

The traffic mirroring feature allows packets to and from the container network to be copied to a specific NIC of the host. Administrators or developers can listen to this NIC to get the complete container network traffic for further analysis, monitoring, security auditing and other operations. It can also be integrated with traditional NPM for more fine-grained traffic visibility.

The traffic mirroring feature introduces some performance loss, with an additional CPU consumption of 5% to 10% depending on CPU performance and traffic characteristics.



3.11.1 Global Traffic Mirroring Settings

The traffic mirroring is disabled by default, please modify the args of `kube-ovn-cni` DaemonSet to enable it:

- `--enable-mirror=true`: Whether to enable traffic mirroring.
- `--mirror-iface=mirror0`: The name of the NIC that the traffic mirror is copied to. This NIC can be a physical NIC that already exists on the host machine. At this point the NIC will be bridged into the br-int bridge and the mirrored traffic will go directly to the underlying switch. If the NIC name does not exist, Kube-OVN will automatically create a virtual NIC with the same name, through which the administrator or developer can access all traffic on the current node on the host. The default is `mirror0`.

Next, you can listen to the traffic on `mirror0` with tcpdump or other traffic analysis tools.

```
tcpdump -ni mirror0
```

3.11.2 Pod Level Mirroring Settings

If you only need to mirror some Pod traffic, you need to disable the global traffic mirroring and then add the `ovn.kubernetes.io/mirror` annotation on a specific Pod to enable Pod-level traffic mirroring.

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: mirror-pod
  namespace: ls1
  annotations:
    ovn.kubernetes.io/mirror: "true"
spec:
  containers:
  - name: mirror-pod
    image: docker.io/library/nginx:alpine
```

3.11.3 Performance Test

Test on the same environment with the traffic mirroring switch on and off, respectively

1. Pod to Pod in the same Nodes

ENABLE TRAFFIC MIRRORING

Size	TCP Latency	TCP Bandwidth	UDP Latency	UDP Lost Rate	UDP Bandwidth
64	12.7 us	289 Mbits/sec	12.6 us	(1.8%)	77.9 Mbits/sec
128	15.5 us	517 Mbits/sec	12.7 us	(0%)	155 Mbits/sec
512	12.2 us	1.64 Gbits/sec	12.4 us	(0%)	624 Mbits/sec
1k	13 us	2.96 Gbits/sec	11.4 us	(0.53%)	1.22 Gbits/sec
4k	18 us	7.67 Gbits/sec	25.7 us	(0.41%)	1.50 Gbits/sec

DISABLE TRAFFIC MIRRORING

Size	TCP Latency	TCP Bandwidth	UDP Latency	UDP Lost Rate	UDP Bandwidth
64	11.9 us	324 Mbits/sec	12.2 us	(0.22%)	102 Mbits/sec
128	10.5 us	582 Mbits/sec	9.5 us	(0.21%)	198 Mbits/sec
512	11.6 us	1.84 Gbits/sec	9.32 us	(0.091%)	827 Mbits/sec
1k	10.5 us	3.44 Gbits/sec	10 us	(1.2%)	1.52 Gbits/sec
4k	16.7 us	8.52 Gbits/sec	18.2 us	(1.3%)	2.42 Gbits/sec

2. Pod to Pod in the different Nodes

ENABLE TRAFFIC MIRRORING

Size	TCP Latency	TCP Bandwidth	UDP Latency	UDP Lost Rate	UDP Bandwidth
64	258 us	143 Mbits/sec	237 us	(61%)	28.5 Mbits/sec
128	240 us	252 Mbits/sec	231 us	(64%)	54.9 Mbits/sec
512	236 us	763 Mbits/sec	256 us	(68%)	194 Mbits/sec
1k	242 us	969 Mbits/sec	225 us	(62%)	449 Mbits/sec
4k	352 us	1.12 Gbits/sec	382 us	(0.71%)	21.4 Mbits/sec

DISABLE TRAFFIC MIRRORING

Size	TCP Latency	TCP Bandwidth	UDP Latency	UDP Lost Rate	UDP Bandwidth
64	278 us	140 Mbits/sec	227 us	(24%)	59.6 Mbits/sec
128	249 us	265 Mbits/sec	265 us	(23%)	114 Mbits/sec
512	233 us	914 Mbits/sec	235 us	(21%)	468 Mbits/sec
1k	238 us	1.14 Gbits/sec	240 us	(15%)	891 Mbits/sec
4k	370 us	1.25 Gbits/sec	361 us	(0.43%)	7.54 Mbits/sec

3. Node to Node

ENABLE TRAFFIC MIRRORING

Size	TCP Latency	TCP Bandwidth	UDP Latency	UDP Lost Rate	UDP Bandwidth
64	205 us	162 Mbits/sec	183 us	(11%)	74.2 Mbits/sec
128	222 us	280 Mbits/sec	206 us	(6.3%)	155 Mbits/sec
512	220 us	1.04 Gbits/sec	177 us	(20%)	503 Mbits/sec
1k	213 us	2.06 Gbits/sec	201 us	(8.6%)	1.14 Gbits/sec
4k	280 us	5.01 Gbits/sec	315 us	(37%)	1.20 Gbits/sec

DISABLE TRAFFIC MIRRORING

Size	TCP Latency	TCP Bandwidth	UDP Latency	UDP Lost Rate	UDP Bandwidth
64	204 us	157 Mbits/sec	204 us	(8.8%)	81.9 Mbits/sec
128	213 us	262 Mbits/sec	225 us	(19%)	136 Mbits/sec
512	220 us	1.02 Gbits/sec	227 us	(21%)	486 Mbits/sec
1k	217 us	1.79 Gbits/sec	218 us	(29%)	845 Mbits/sec
4k	275 us	5.27 Gbits/sec	336 us	(34%)	1.21 Gbits/sec

4. Pod to the Node where the Pod is located

ENABLE TRAFFIC MIRRORING

Size	TCP Latency	TCP Bandwidth	UDP Latency	UDP Lost Rate	UDP Bandwidth
64	12.2 us	295 Mbits/sec	12.7 us	(0.27%)	74.1 Mbits/sec
128	14.1 us	549 Mbits/sec	10.6 us	(0.41%)	153 Mbits/sec
512	13.5 us	1.83 Gbits/sec	12.7 us	(0.23%)	586 Mbits/sec
1k	12 us	2.69 Gbits/sec	13 us	(1%)	1.16 Gbits/sec
4k	18.9 us	4.51 Gbits/sec	21.8 us	(0.42%)	1.81 Gbits/sec

DISABLE TRAFFIC MIRRORING

Size	TCP Latency	TCP Bandwidth	UDP Latency	UDP Lost Rate	UDP Bandwidth
64	10.4 us	335 Mbits/sec	12.2 us	(0.75%)	95.4 Mbits/sec
128	12.1 us	561 Mbits/sec	11.3 us	(0.25%)	194 Mbits/sec
512	11.6 us	1.87 Gbits/sec	10.7 us	(0.66%)	745 Mbits/sec
1k	12.7 us	3.12 Gbits/sec	10.9 us	(1.2%)	1.46 Gbits/sec
4k	16.5 us	8.23 Gbits/sec	17.9 us	(1.5%)	2.51 Gbits/sec

5. Pod to the Node where the Pod is not located

ENABLE TRAFFIC MIRRORING

Size	TCP Latency	TCP Bandwidth	UDP Latency	UDP Lost Rate	UDP Bandwidth
64	234 us	153 Mbits/sec	232 us	(63%)	29.4 Mbits/sec
128	237 us	261 Mbits/sec	238 us	(49%)	76.1 Mbits/sec
512	231 us	701 Mbits/sec	238 us	(57%)	279 Mbits/sec
1k	256 us	1.05 Gbits/sec	228 us	(56%)	524 Mbits/sec
4k	330 us	1.08 Gbits/sec	359 us	(1.5%)	35.7 Mbits/sec

DISABLE TRAFFIC MIRRORING

Size	TCP Latency	TCP Bandwidth	UDP Latency	UDP Lost Rate	UDP Bandwidth
64	283 us	141 Mbits/sec	230 us	(26%)	55.8 Mbits/sec
128	234 us	255 Mbits/sec	234 us	(25%)	113 Mbits/sec
512	246 us	760 Mbits/sec	234 us	(22%)	458 Mbits/sec
1k	268 us	1.23 Gbits/sec	242 us	(20%)	879 Mbits/sec
4k	326 us	1.20 Gbits/sec	369 us	(0.5%)	7.87 Mbits/sec

6. Pod to the cluster ip service

ENABLE TRAFFIC MIRRORING

Size	TCP Latency	TCP Bandwidth	UDP Latency	UDP Lost Rate	UDP Bandwidth
64	237 us	133 Mbits/sec	213 us	(65%)	25.5 Mbits/sec
128	232 us	271 Mbits/sec	222 us	(62%)	54.8 Mbits/sec
512	266 us	800 Mbits/sec	234 us	(60%)	232 Mbits/sec
1k	248 us	986 Mbits/sec	239 us	(50%)	511 Mbits/sec
4k	314 us	1.03 Gbits/sec	367 us	(0.6%)	13.2 Mbits/sec

TCP-Conn-Number	QPS	Avg-Resp-Time	Stdev-Resp-Time	Max-Resp-Time
10	14305.17	0.87ms	1.48ms	24.46ms
100	29082.07	3.87ms	4.35ms	102.85ms

DISABLE TRAFFIC MIRRORING

Size	TCP Latency	TCP Bandwidth	UDP Latency	UDP Lost Rate	UDP Bandwidth
64	241 us	145 Mbits/sec	225 us	(19%)	60.2 Mbits/sec
128	245 us	261 Mbits/sec	212 us	(15%)	123 Mbits/sec
512	252 us	821 Mbits/sec	219 us	(14%)	499 Mbits/sec
1k	253 us	1.08 Gbits/sec	242 us	(16%)	852 Mbits/sec
4k	320 us	1.32 Gbits/sec	360 us	(0.47%)	6.70 Mbits/sec

TCP-Conn-Number	QPS	Avg-Resp-Time	Stdev-Resp-Time	Max-Resp-Time
10	13634.07	0.96ms	1.72ms	30.07ms
100	30215.23	3.59ms	3.20ms	77.56ms

7. Host to the Node port service where the Pod is not located on the target Node

ENABLE TRAFFIC MIRRORING

TCP-Conn-Number	QPS	Avg-Resp-Time	Stdev-Resp-Time	Max-Resp-Time
10	14802.73	0.88ms	1.66ms	31.49ms
100	29809.58	3.78ms	4.12ms	105.34ms

DISABLE TRAFFIC MIRRORING

TCP-Conn-Number	QPS	Avg-Resp-Time	Stdev-Resp-Time	Max-Resp-Time
10	14273.33	0.90ms	1.60ms	37.16ms
100	30757.81	3.62ms	3.41ms	59.78ms

8. Host to the Node port service where the Pod is located on the target Node

ENABLE TRAFFIC MIRRORING

TCP-Conn-Number	QPS	Avg-Resp-Time	Stdev-Resp-Time	Max-Resp-Time
10	15402.39	802.50us	1.42ms	30.91ms
100	29424.66	4.05ms	4.31ms	90.60ms

DISABLE TRAFFIC MIRRORING

TCP-Conn-Number	QPS	Avg-Resp-Time	Stdev-Resp-Time	Max-Resp-Time
10	14649.21	0.91ms	1.72ms	43.92ms
100	32143.61	3.66ms	3.76ms	67.02ms

PDF Slack Support

November 8, 2023

May 24, 2022

GitHub

3.11.4 Comments

3.12 NetworkPolicy Logging

NetworkPolicy is a interface provided by Kubernetes and implemented by Kube-OVN through OVN's ACLs. With NetworkPolicy, if the networks are down, it is difficult to determine whether it is caused by a network failure or a NetworkPolicy rule problem. Kube-OVN provides NetworkPolicy logging to help administrators quickly locate whether a NetworkPolicy drop rule has been hit, and to record the illegal accesses.

Once NetworkPolicy logging is turned on, logs need to be printed for every packet that hits a Drop rule, which introduces additional performance overhead. Under a malicious attack, a large number of logs in a short period of time may exhaust the CPU. We recommend turning off logging by default in production environments and dynamically turning it on when you need to troubleshoot problems.

3.12.1 Enable NetworkPolicy Logging

Add the annotation `ovn.kubernetes.io/enable_log` to the NetworkPolicy where logging needs to be enabled, as follows:

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny-ingress
  namespace: kube-system
  annotations:
    ovn.kubernetes.io/enable_log: "true"
spec:
  podSelector: {}
  policyTypes:
  - Ingress
```

Next, you can observe the log of dropped packets in `/var/log/ovn/ovn-controller.log` on the host of the corresponding Pod:

```
# tail -f /var/log/ovn/ovn-controller.log
2022-07-20T05:55:03.229Z|00394|acl_log(ovn_pinctrl0)|INFO|name="<unnamed>", verdict=drop, severity=warning, direction=to-lport: udp,vlan_tci=0x0000,dl_src=00:
00:00:21:b7:d1,dl_dst=00:00:00:8d:0b:86,nw_src=10.16.0.10,nw_dst=10.16.0.7,nw_tos=0,nw_ecn=0,nw_ttl=63,tp_src=54343,tp_dst=53
2022-07-20T05:55:06.229Z|00395|acl_log(ovn_pinctrl0)|INFO|name="<unnamed>", verdict=drop, severity=warning, direction=to-lport: udp,vlan_tci=0x0000,dl_src=00:
00:00:21:b7:d1,dl_dst=00:00:00:8d:0b:86,nw_src=10.16.0.9,nw_dst=10.16.0.7,nw_tos=0,nw_ecn=0,nw_ttl=63,tp_src=44187,tp_dst=53
2022-07-20T05:55:08.230Z|00396|acl_log(ovn_pinctrl0)|INFO|name="<unnamed>", verdict=drop, severity=warning, direction=to-lport: udp,vlan_tci=0x0000,dl_src=00:
00:00:21:b7:d1,dl_dst=00:00:00:8d:0b:86,nw_src=10.16.0.10,nw_dst=10.16.0.7,nw_tos=0,nw_ecn=0,nw_ttl=63,tp_src=54274,tp_dst=53
2022-07-20T05:55:11.231Z|00397|acl_log(ovn_pinctrl0)|INFO|name="<unnamed>", verdict=drop, severity=warning, direction=to-lport: udp,vlan_tci=0x0000,dl_src=00:
00:00:21:b7:d1,dl_dst=00:00:00:8d:0b:86,nw_src=10.16.0.9,nw_dst=10.16.0.7,nw_tos=0,nw_ecn=0,nw_ttl=63,tp_src=32778,tp_dst=53
2022-07-20T05:55:11.231Z|00398|acl_log(ovn_pinctrl0)|INFO|name="<unnamed>", verdict=drop, severity=warning, direction=to-lport: udp,vlan_tci=0x0000,dl_src=00:
00:00:21:b7:d1,dl_dst=00:00:00:8d:0b:86,nw_src=10.16.0.9,nw_dst=10.16.0.7,nw_tos=0,nw_ecn=0,nw_ttl=63,tp_src=34188,tp_dst=53
2022-07-20T05:55:13.231Z|00399|acl_log(ovn_pinctrl0)|INFO|name="<unnamed>", verdict=drop, severity=warning, direction=to-lport: udp,vlan_tci=0x0000,dl_src=00:
00:00:21:b7:d1,dl_dst=00:00:00:8d:0b:86,nw_src=10.16.0.10,nw_dst=10.16.0.7,nw_tos=0,nw_ecn=0,nw_ttl=63,tp_src=43290,tp_dst=53
2022-07-20T05:55:22.096Z|00400|acl_log(ovn_pinctrl0)|INFO|name="<unnamed>", verdict=drop, severity=warning, direction=to-lport:
icmp,vlan_tci=0x0000,dl_src=00:00:00:6c:42:91,dl_dst=00:00:00:a5:d7:63,nw_src=10.16.0.9,nw_dst=10.16.0.10,nw_tos=0,nw_ecn=0,nw_ttl=64,icmp_type=8,icmp_code=0
2022-07-20T05:55:22.097Z|00401|acl_log(ovn_pinctrl0)|INFO|name="<unnamed>", verdict=drop, severity=warning, direction=to-lport:
icmp,vlan_tci=0x0000,dl_src=00:00:00:6c:42:91,dl_dst=00:00:00:a5:d7:63,nw_src=10.16.0.9,nw_dst=10.16.0.10,nw_tos=0,nw_ecn=0,nw_ttl=64,icmp_type=8,icmp_code=0
2022-07-20T05:55:22.098Z|00402|acl_log(ovn_pinctrl0)|INFO|name="<unnamed>", verdict=drop, severity=warning, direction=to-lport:
icmp,vlan_tci=0x0000,dl_src=00:00:00:6c:42:91,dl_dst=00:00:00:a5:d7:63,nw_src=10.16.0.9,nw_dst=10.16.0.10,nw_tos=0,nw_ecn=0,nw_ttl=64,icmp_type=8,icmp_code=0
```

3.12.2 Other NetworkPolicy Logging

By default, after setting the "ovn.kubernetes.io/enable_log" annotation, only logs matching the drop ACL rule can be printed. If you want to view logs matching other ACL rules, it is not supported.

Starting from Kube-OVN v1.13.0, a new annotation "ovn.kubernetes.io/log_acl_actions" is added to support logging that matches other ACL rules. The value of the annotation needs to be set to "allow".

Add annotation `ovn.kubernetes.io/log_acl_actions` to NetworkPolicy, as shown below:

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny-ingress
  namespace: kube-system
  annotations:
    ovn.kubernetes.io/enable_log: "true"
    ovn.kubernetes.io/log_acl_actions: "allow"
spec:
  podSelector: {}
  policyTypes:
  - Ingress
```

Access the test pod and check /var/log/ovn/ovn-controller.log of the host where the corresponding Pod is located. You can see the Allow ACL Rule log

```
2024-08-14T09:27:49.590Z|00004|acl_log(ovn_pinctrl0)|INFO|name="np/test.default/ingress/IPv4/0", verdict=allow, severity=info, direction=to-lport:
icmp,vlan_tci=0x0000,dl_src=96:7b:b 0:2f:a0:1a,dl_dst=a6:e5:1b:c2:1b:f8,nw_src=10.16.0.7,nw_dst=10.
16.0.12,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,icmp_type=8,icmp_code=0
```

3.12.3 Disable NetworkPolicy Logging

Set annotation `ovn.kubernetes.io/enable_log` in the corresponding NetworkPolicy to `false` to disable NetworkPolicy logging:

```
kubectl annotate networkpolicy -n kube-system default-deny-ingress ovn.kubernetes.io/enable_log=false --overwrite
```

3.12.4 AdminNetworkPolicy and BaselineAdminNetworkPolicy Logging

Starting from v1.13.0, Kube-OVN supports the AdminNetworkPolicy and BaselineAdminNetworkPolicy functions. For an introduction to AdminNetworkPolicy and BaselineAdminNetworkPolicy, see AdminNetworkPolicy.

For cluster network policies, you can also print logs that match ACL action rules by setting the "ovn.kubernetes.io/log_acl_actions" annotation. The annotation's value can be a combination of one or more of "allow,drop,pass".

Note that the "ovn.kubernetes.io/enable_log" annotation is only used when printing network policy logs. When printing cluster network policy logs, you do not need to set this annotation. You only need to set the "ovn.kubernetes.io/log_acl_actions" annotation.

PDF **Slack** **Support**

August 16, 2024

July 20, 2022

GitHub

3.12.5 Comments

3.13 LoadBalancer Type Service

Kube-OVN supports the implementation of VPC and VPC gateway. For specific configurations, please refer to the VPC configuration.

Due to the complexity of using VPC gateways, the implementation based on VPC gateways has been simplified. It supports creating LoadBalancer type Services in the default VPC, allowing access to Services in the default VPC through LoadBalancerIP.

First, make sure the following conditions are met in the environment:

1. Install `multus-cni` and `macvlan cni`.
2. LoadBalancer Service support relies on simplified implementation of VPC gateway code, still utilizing the vpc-nat-gw image and depending on macvlan for multi-interface functionality support.
3. Currently, it only supports configuration in the default VPC. Support for LoadBalancers in custom VPCs can be referred to in the VPC configuration.

3.13.1 Steps to Configure Default VPC LoadBalancer Service

Enable Feature Flag

Modify the deployment `kube-ovn-controller` under the kube-system namespace and add the parameter `--enable-lb-svc=true` to the `args` section to enable the feature (by default it's set to false).

```
containers:
- args:
  - /kube-ovn/start-controller.sh
  - --default-cidr=10.16.0.0/16
  - --default-gateway=10.16.0.1
  - --default-gateway-check=true
  - --enable-lb-svc=true                  // parameter is set to true
```

Create NetworkAttachmentDefinition CRD Resource

Refer to the following YAML and create the `net-attach-def` resource:

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: lb-svc-attachment
  namespace: kube-system
spec:
  config: '{
      "cniVersion": "0.3.0",
      "type": "macvlan",
      "master": "eth0",                            //Physical network card, configure according to the actual situation
      "mode": "bridge"
    }'
```

By default, the physical NIC `eth0` is used to implement the multi-interface functionality. If another physical NIC is needed, modify the `master` value to specify the name of the desired physical NIC.

Create Subnet

The created Subnet is used to allocate LoadBalancerIP for the LoadBalancer Service, which should normally be accessible from outside the cluster. An Underlay Subnet can be configured for address allocation.

Refer to the following YAML to create a new subnet:

```
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
  name: attach-subnet
spec:
  protocol: IPv4
  provider:
lb-svc-attachment.kube-system          //The provider format is fixed and consists of the Name.Namespace of the net-attach-def resource created in the
previous step
```

```
  cidrBlock: 172.18.0.0/16
  gateway: 172.18.0.1
  excludeIps:
  - 172.18.0.0..172.18.0.10
```

In the `provider` parameter of the Subnet, `ovn` or `.ovn` suffix is used to indicate that the subnet is managed by Kube-OVN and requires corresponding logical switch records to be created.

If `provider` is neither `ovn` nor ends with `.ovn`, Kube-OVN only provides the IPAM functionality to record IP address allocation without handling business logic for the subnet.

Create LoadBalancer Service

Refer to the following YAML to create a LoadBalancer Service:

```
apiVersion: v1
kind: Service
metadata:
   annotations:
     lb-svc-attachment.kube-system.kubernetes.io/logical_switch: attach-subnet #Optional
     ovn.kubernetes.io/attachmentprovider: lb-svc-attachment.kube-system #Required
   labels:
     app: dynamic
   name: test-service
   namespace: default
spec:
   loadBalancerIP: 172.18.0.18 #Optional
   ports:
     - name: test
       protocol: TCP
       port: 80
       targetPort: 80
   selector:
     app: dynamic
   sessionAffinity: None
   type: LoadBalancer
```

In the yaml, the annotation `ovn.kubernetes.io/attachmentprovider` is required, and its value is composed of the Name.Namespace of the `net-attach-def` resource created in the first step. This annotation is used to find the `net-attach-def` resources when creating Pods.

The subnet used for multi-interface address allocation can be specified through an annotation. The annotation key format is `net-attach-def` resource's `Name.Namespace.kubernetes.io/logical_switch`. This configuration is `optional` and if LoadBalancerIP address is not specified, addresses will be dynamically allocated from this subnet and filled into the LoadBalancerIP field.

If a static LoadBalancerIP address is required, the `spec.loadBalancerIP` field can be configured. The address must be within the specified subnet's address range.

After creating the Service using the YAML, you can see the Pod startup information in the same namespace as the Service:

```
# kubectl get pod
NAME READY STATUS RESTARTS AGE
lb-svc-test-service-6869d98dd8-cjvll 1/1 Running 0 107m
# kubectl get svc
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
test-service LoadBalancer 10.109.201.193 172.18.0.18 80:30056/TCP 107m
```

When specifying the `service.spec.loadBalancerIP` parameter, it will be assigned to the service's external IP field. If not specified, the parameter will be assigned a random value.

View the YAML output of the test Pod to see the assigned multi-interface addresses:

```
# kubectl get pod -o yaml lb-svc-test-service-6869d98dd8-cjvll
apiVersion: v1
kind: Pod
metadata:
   annotations:
     k8s.v1.cni.cncf.io/network-status: |-
       [{
           "name": "kube-ovn",
           "ips": [
               "10.16.0.2"
           ],
           "default": true,
           "dns": {}
       },{
           "name": "default/test-service",
           "interface": "net1",
           "mac": "ba:85:f7:02:9f:42",
           "dns": {}
       }]
     k8s.v1.cni.cncf.io/networks: default/test-service
     k8s.v1.cni.cncf.io/networks-status: |-
       [{
           "name": "kube-ovn",
           "ips": [
               "10.16.0.2"
           ],
           "default": true,
           "dns": {}
       },{
           "name": "default/test-service",
           "interface": "net1",
           "mac": "ba:85:f7:02:9f:42",
           "dns": {}
       }]
     ovn.kubernetes.io/allocated: "true"
     ovn.kubernetes.io/cidr: 10.16.0.0/16
     ovn.kubernetes.io/gateway: 10.16.0.1
     ovn.kubernetes.io/ip_address: 10.16.0.2
     ovn.kubernetes.io/logical_router: ovn-cluster
     ovn.kubernetes.io/logical_switch: ovn-default
     ovn.kubernetes.io/mac_address: 00:00:00:45:F4:29
     ovn.kubernetes.io/pod_nic_type: veth-pair
     ovn.kubernetes.io/routed: "true"
     test-service.default.kubernetes.io/allocated: "true"
     test-service.default.kubernetes.io/cidr: 172.18.0.0/16
     test-service.default.kubernetes.io/gateway: 172.18.0.1
     test-service.default.kubernetes.io/ip_address: 172.18.0.18
     test-service.default.kubernetes.io/logical_switch: attach-subnet
     test-service.default.kubernetes.io/mac_address: 00:00:00:AF:AA:BF
     test-service.default.kubernetes.io/pod_nic_type: veth-pair
```

Check the service information:

```
# kubectl get svc -o yaml test-service
apiVersion: v1
kind: Service
metadata:
   annotations:
     kubectl.kubernetes.io/last-applied-configuration: |
       {"apiVersion":"v1","kind":"Service","metadata":{"annotations":{"test-service.default.kubernetes.io/logical_switch":"attach-subnet"},"labels ":
{"app":"dynamic"},"name":"test-service","namespace":"default"},"spec":{"ports":[{"name":"test", "port":80,"protocol":"TCP","targetPort":80}],"selector":
{"app":"dynamic"},"sessionAffinity":"None","type":"LoadBalancer "}}
     ovn.kubernetes.io/vpc:ovn-cluster
     test-service.default.kubernetes.io/logical_switch: attach-subnet
   creationTimestamp: "2022-06-15T09:01:58Z"
   labels:
     app: dynamic
   name: test-service
   namespace: default
   resourceVersion: "38485"
   uid: 161edee1-7f6e-40f5-9e09-5a52c44267d0
spec:
   allocateLoadBalancerNodePorts: true
   clusterIP: 10.109.201.193
   clusterIPs:
   - 10.109.201.193
   externalTrafficPolicy: Cluster
   internalTrafficPolicy: Cluster
   ipFamilies:
   - IPv4
   ipFamilyPolicy: SingleStack
   ports:
   - name: test
     nodePort: 30056
     port: 80
     protocol: TCP
     targetPort: 80
   selector:
     app: dynamic
   sessionAffinity: None
   type: LoadBalancer
status:
   loadBalancer:
     ingress:
     - ip: 172.18.0.18
```

3.13.2 Testing LoadBalancerIP access

Refer to the following YAML to create a test Pod that serves as the Endpoints for the Service:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: dynamic
  name: dynamic
  namespace: default
spec:
  replicas: 2
  selector:
    matchLabels:
      app: dynamic
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: dynamic
    spec:
      containers:
      - image: docker.io/library/nginx:alpine
        imagePullPolicy: IfNotPresent
        name: nginx
      dnsPolicy: ClusterFirst
      restartPolicy: Always
```

Under normal circumstances, the provided subnet addresses should be accessible from outside the cluster. To verify, access the Service's `LoadBalancerIP:Port` from within the cluster and check if the access is successful.

```
# curl 172.18.0.11:80
<html>
<head>
        <title>Hello World!</title>
        <link href='//fonts.googleapis.com/css?family=Open+Sans:400,700' rel='stylesheet' type='text/css'>
        <style>
        body {
                background-color: white;
                text-align: center;
                padding: 50px;
                font-family: "Open Sans","Helvetica Neue",Helvetica,Arial,sans-serif;
        }
        #logo {
                margin-bottom: 40px;
        }
        </style>
</head>
<body>
                <h1>Hello World!</h1>
                                <h3>Links found</h3>
        <h3>I am on  dynamic-7d8d7874f5-hsgc4</h3>
        <h3>Cookie                  =</h3>
                                        <b>KUBERNETES</b> listening in 443 available at tcp://10.96.0.1:443<br />
                                                <h3>my name is hanhouchao!</h3>
                        <h3> RequestURI='/'</h3>
</body>
</html>
```

Enter the Pod created by the Service and check the network information:

```
# ip a
4: net1@if62: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether ba:85:f7:02:9f:42 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 172.18.0.18/16 scope global net1
       valid_lft forever preferred_lft forever
    inet6 fe80::b885:f7ff:fe02:9f42/64 scope link
       valid_lft forever preferred_lft forever
36: eth0@if37: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1400 qdisc noqueue state UP group default
    link/ether 00:00:00:45:f4:29 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.16.0.2/16 brd 10.16.255.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::200:ff:fe45:f429/64 scope link
       valid_lft forever preferred_lft forever

# ip rule
0: from all lookup local
32764: from all iif eth0 lookup 100
32765: from all iif net1 lookup 100
32766: from all lookup main
32767: from all lookup default

# ip route show table 100
default via 172.18.0.1 dev net1
10.109.201.193 via 10.16.0.1 dev eth0
172.18.0.0/16 dev net1 scope link

# iptables -t nat -L -n -v
Chain PREROUTING (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 DNAT       tcp  --  *      *       0.0.0.0/0            172.18.0.18          tcp dpt:80 to:10.109.201.193:80
```

```
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain POSTROUTING (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 MASQUERADE  all  --  *      *       0.0.0.0/0            10.109.201.193
```

Configure the `nodeSelector` for the LB Service Pod

You can specify the node where the LoadBalancer service gateway Pod is deployed by adjusting the `nodeSelector` in the `ovn-vpc-nat-config` ConfigMap.

```yaml
apiVersion: v1
data:
  image: docker.io/kubeovn/vpc-nat-gateway:v1.14.0
  nodeSelector: |
    kubernetes.io/hostname: kube-ovn-control-plane
    kubernetes.io/os: linux
kind: ConfigMap
metadata:
  name: ovn-vpc-nat-config
  namespace: kube-system
```

PDF Slack Support

July 30, 2025

August 18, 2023

GitHub

3.13.3 Comments

3.14 Monitor and Dashboard

Kube-OVN can export network control plane information and network data plane quality information metrics to the external in formats supported by Prometheus.

We use the CRD provided by kube-prometheus to define the corresponding Prometheus monitoring rules. For all monitoring metrics supported by Kube-OVN, please refer to Kube-OVN Monitoring Metrics.

If you are using native Prometheus, please refer to Configuring Native Prometheus for configuration.

3.14.1 Install Prometheus Monitor

Kube-OVN uses Prometheus Monitor CRD to manage the monitoring output.

```
# network quality related monitoring metrics
kubectl apply -f https://raw.githubusercontent.com/kubeovn/kube-ovn/master/dist/monitoring/pinger-monitor.yaml
# kube-ovn-controller metrics
kubectl apply -f https://raw.githubusercontent.com/kubeovn/kube-ovn/master/dist/monitoring/controller-monitor.yaml
# kube-ovn-cni metrics
kubectl apply -f https://raw.githubusercontent.com/kubeovn/kube-ovn/master/dist/monitoring/cni-monitor.yaml
# ovn metrics
kubectl apply -f https://raw.githubusercontent.com/kubeovn/kube-ovn/master/dist/monitoring/ovn-monitor.yaml
```

The default interval for Prometheus pull is 15s, if you need to adjust it, modify the `interval` value in yaml.

3.14.2 Import Grafana Dashboard

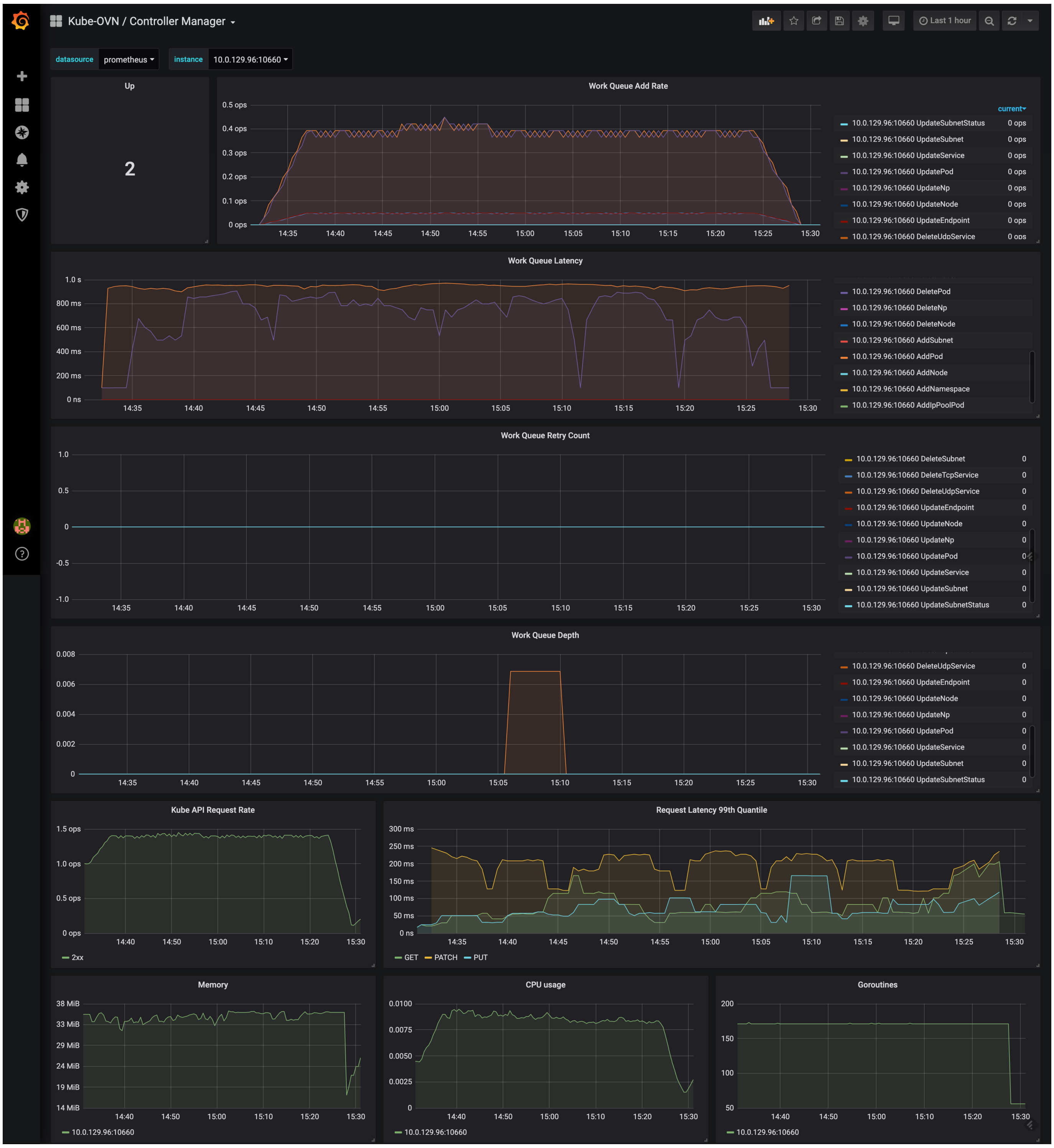
Kube-OVN provides a predefined Grafana Dashboard to display control plane and data plane related metrics.

Download the corresponding Dashboard template:

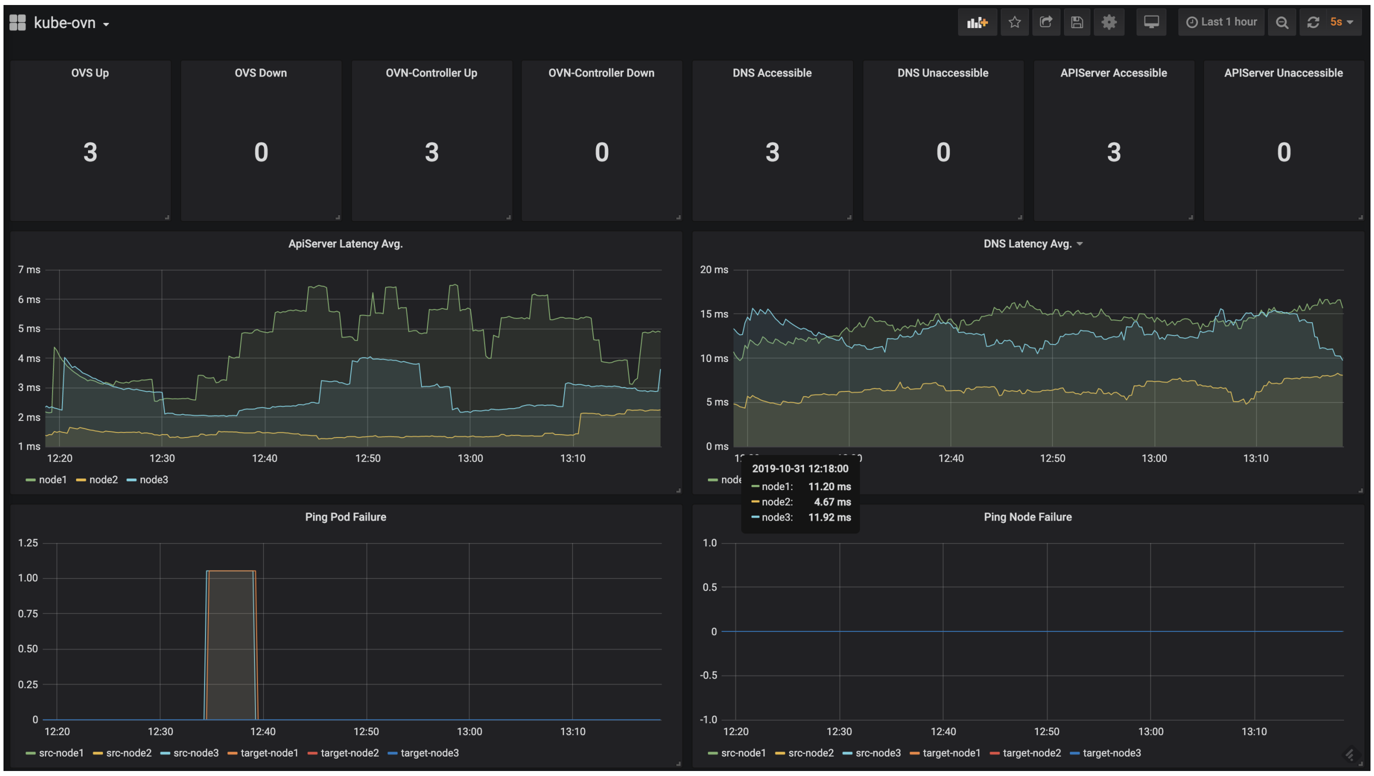
```
# network quality related monitoring dashboard
wget https://raw.githubusercontent.com/kubeovn/kube-ovn/master/dist/monitoring/pinger-grafana.json
# kube-ovn-controller dashboard
wget https://raw.githubusercontent.com/kubeovn/kube-ovn/master/dist/monitoring/controller-grafana.json
# kube-ovn-cni dashboard
wget https://raw.githubusercontent.com/kubeovn/kube-ovn/master/dist/monitoring/cni-grafana.json
# ovn dashboard
wget https://raw.githubusercontent.com/kubeovn/kube-ovn/master/dist/monitoring/ovn-grafana.json
# ovs dashboard
wget https://raw.githubusercontent.com/kubeovn/kube-ovn/master/dist/monitoring/ovs-grafana.json
```

Import these templates into Grafana and set the data source to the corresponding Prometheus to see the following Dashboards.

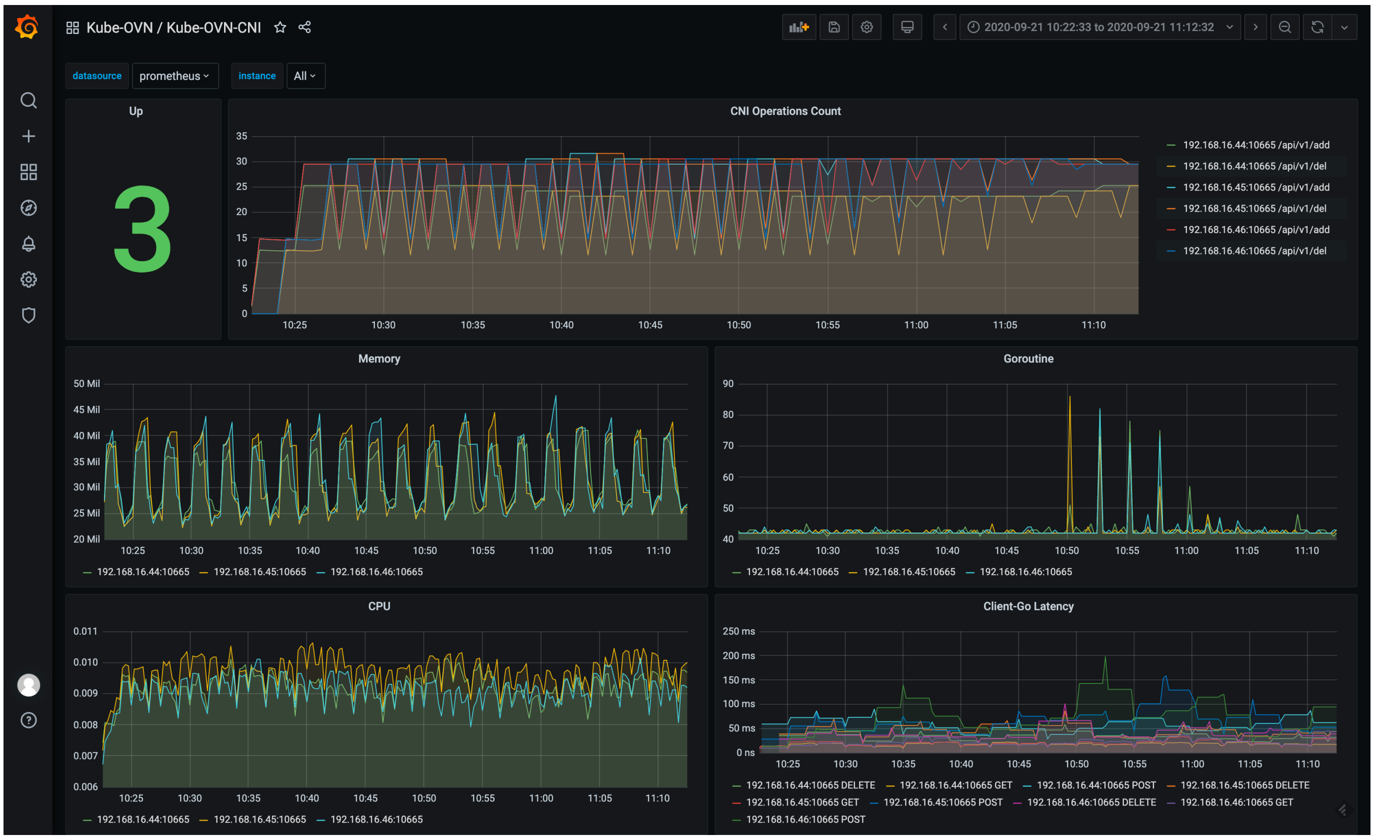
`kube-ovn-controller` dashboard:


Kube-OVN / Controller Manager
datasource prometheus
instance 10.0.129.96:10660
Last 1 hour
Up
2
Work Queue Add Rate
current
10.0.129.96:10660 UpdateSubnetStatus 0 ops
10.0.129.96:10660 UpdateSubnet 0 ops
10.0.129.96:10660 UpdateService 0 ops
10.0.129.96:10660 UpdatePod 0 ops
10.0.129.96:10660 UpdateNp 0 ops
10.0.129.96:10660 UpdateNode 0 ops
10.0.129.96:10660 UpdateEndpoint 0 ops
10.0.129.96:10660 DeleteUdpService 0 ops
Work Queue Latency
10.0.129.96:10660 DeletePod
10.0.129.96:10660 DeleteNp
10.0.129.96:10660 DeleteNode
10.0.129.96:10660 AddSubnet
10.0.129.96:10660 AddPod
10.0.129.96:10660 AddNode
10.0.129.96:10660 AddNamespace
10.0.129.96:10660 AddIpPoolPod
Work Queue Retry Count
10.0.129.96:10660 DeleteSubnet 0
10.0.129.96:10660 DeleteTcpService 0
10.0.129.96:10660 DeleteUdpService 0
10.0.129.96:10660 UpdateEndpoint 0
10.0.129.96:10660 UpdateNode 0
10.0.129.96:10660 UpdateNp 0
10.0.129.96:10660 UpdatePod 0
10.0.129.96:10660 UpdateService 0
10.0.129.96:10660 UpdateSubnet 0
10.0.129.96:10660 UpdateSubnetStatus 0
Work Queue Depth
10.0.129.96:10660 DeleteUdpService 0
10.0.129.96:10660 UpdateEndpoint 0
10.0.129.96:10660 UpdateNode 0
10.0.129.96:10660 UpdateNp 0
10.0.129.96:10660 UpdatePod 0
10.0.129.96:10660 UpdateService 0
10.0.129.96:10660 UpdateSubnet 0
10.0.129.96:10660 UpdateSubnetStatus 0
Kube API Request Rate
2xx
Request Latency 99th Quantile
GET
PATCH
PUT
Memory
10.0.129.96:10660
CPU usage
10.0.129.96:10660
Goroutines
10.0.129.96:10660

`kube-ovn-pinger` dashboard:



`kube-ovn-cni` dashboard:



September 6, 2022

May 23, 2022

GitHub

3.14.3 Comments

3.15 Config Native Prometheus

Kube-OVN provides rich monitoring data for OVN/OVS health status checks and connectivity checks of container and host networks, and Kube-OVN is configured with ServiceMonitor for Prometheus to dynamically obtain monitoring metrics.

In some cases, where only Prometheus Server is installed and no other components are installed, you can dynamically obtain monitoring data for the cluster environment by modifying the configuration of Prometheus.

3.15.1 Config Prometheus

The following configuration documentation, referenced from Prometheus Service Discovery.

Permission Configuration

Prometheus is deployed in the cluster and needs to access the k8s apiserver to query the monitoring data of the containers.

Refer to the following yaml to configure the permissions required by Prometheus:

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: prometheus
rules:
- apiGroups: [""]
  resources:
  - nodes
  - nodes/proxy
  - services
  - endpoints
  - pods
  verbs: ["get", "list", "watch"]
- apiGroups:
  - extensions
  resources:
  - ingresses
  verbs: ["get", "list", "watch"]
- nonResourceURLs: ["/metrics"]
  verbs: ["get"]
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: prometheus
  namespace: default
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: prometheus
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: prometheus
subjects:
- kind: ServiceAccount
  name: prometheus
  namespace: default
```

Prometheus ConfigMap

The startup of Prometheus relies on the configuration file prometheus.yml, the contents of which can be configured in ConfigMap and dynamically mounted to the Pod.

Create the ConfigMap file used by Prometheus by referring to the following yaml:

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-config
data:
  prometheus.yml: |-
    global:
      scrape_interval:     15s
      evaluation_interval: 15s
    scrape_configs:
```

```
    - job_name: 'prometheus'
      static_configs:
      - targets: ['localhost:9090']

    - job_name: 'kubernetes-nodes'
      tls_config:
        ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
      bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
      kubernetes_sd_configs:
      - role: node

    - job_name: 'kubernetes-service'
      tls_config:
        ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
      bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
      kubernetes_sd_configs:
      - role: service

    - job_name: 'kubernetes-endpoints'
      tls_config:
        ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
      bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
      kubernetes_sd_configs:
      - role: endpoints

    - job_name: 'kubernetes-ingress'
      tls_config:
        ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
      bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
      kubernetes_sd_configs:
      - role: ingress

    - job_name: 'kubernetes-pods'
      tls_config:
        ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
      bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
      kubernetes_sd_configs:
      - role: pod
```

Prometheus provides role-based querying of Kubernetes resource monitoring operations, which can be configured in the official documentation kubernetes_sd_config.

Deploy Prometheus

Deploy Prometheus Server by referring to the following yaml:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: prometheus
  name: prometheus
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: prometheus
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: prometheus
    spec:
      serviceAccountName: prometheus
      serviceAccount: prometheus
      containers:
      - image: docker.io/prom/prometheus:latest
        imagePullPolicy: IfNotPresent
        name: prometheus
        command:
        - "/bin/prometheus"
        args:
        - "--config.file=/etc/prometheus/prometheus.yml"
        ports:
        - containerPort: 9090
          protocol: TCP
        volumeMounts:
        - mountPath: "/etc/prometheus"
          name: prometheus-config
      volumes:
      - name: prometheus-config
        configMap:
          name: prometheus-config
```

Deploy Prometheus Service by referring to the following yaml:

```yaml
kind: Service
apiVersion: v1
metadata:
  name: prometheus
  namespace: default
  labels:
    name: prometheus
spec:
  ports:
    - name: test
      protocol: TCP
      port: 9090
      targetPort: 9090
  type: NodePort
  selector:
    app: prometheus
  sessionAffinity: None
```

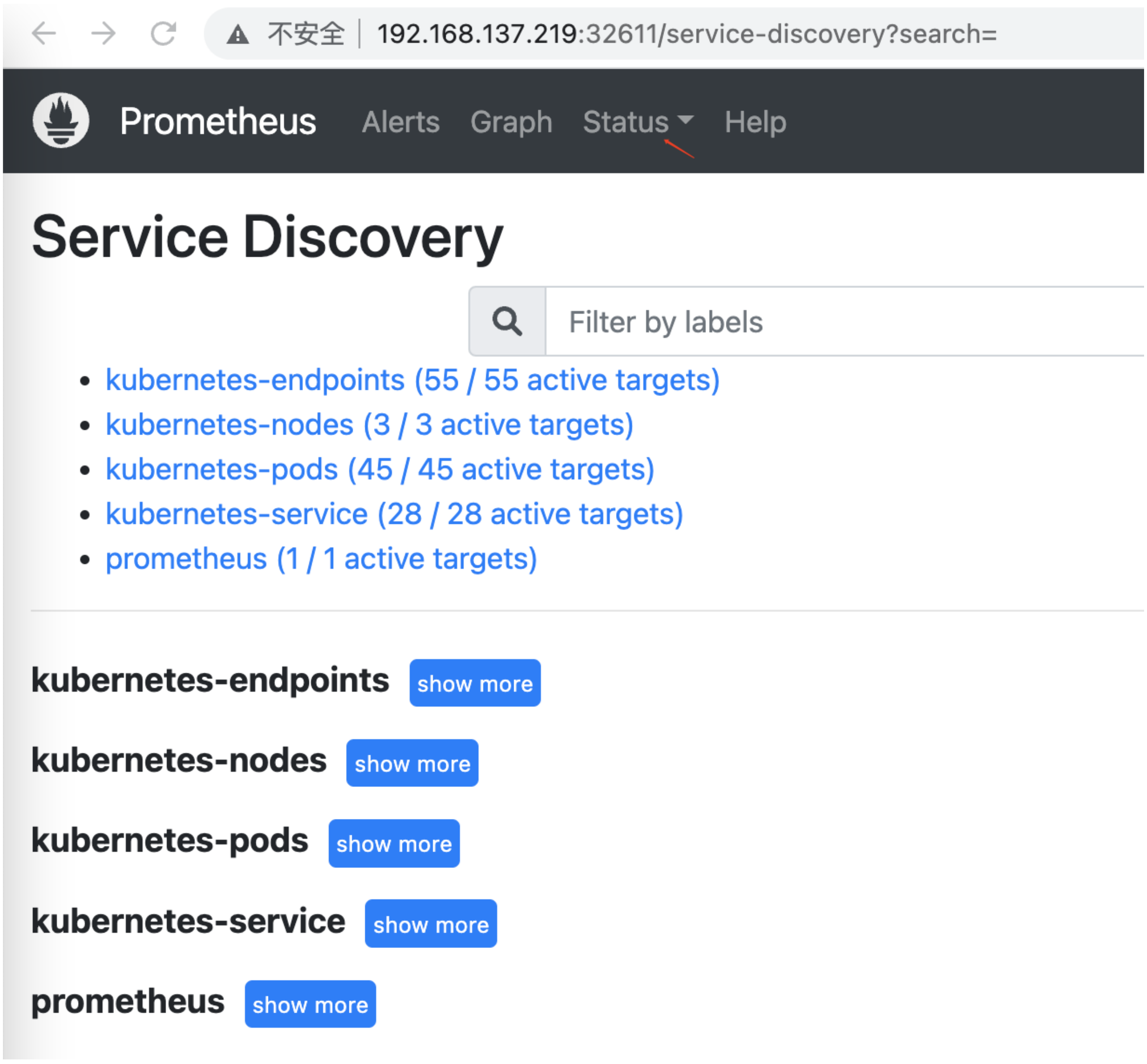
After exposing Prometheus through NodePort, Prometheus can be accessed through the node address.

3.15.2 Prometheus Metrics Config

View information about Prometheus on the environment:

```
# kubectl get svc
NAME         TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE
kubernetes   ClusterIP   10.4.0.1       <none>        443/TCP          8d
prometheus   NodePort    10.4.102.222   <none>        9090:32611/TCP   8d
# kubectl get pod -o wide
NAME                          READY   STATUS    RESTARTS   AGE    IP          NODE              NOMINATED NODE   READINESS GATES
prometheus-7544b6b84d-v9m8s   1/1     Running   0          3d5h   10.3.0.7    192.168.137.219   <none>           <none>
# kubectl get endpoints -o wide
NAME         ENDPOINTS                                                        AGE
kubernetes   192.168.136.228:6443,192.168.136.232:6443,192.168.137.219:6443   8d
prometheus   10.3.0.7:9090                                                    8d
```

Access Prometheus via NodePort to see the data dynamically queried by Status/Service Discovery:



Service Discovery

Filter by labels

- kubernetes-endpoints (55 / 55 active targets)
- kubernetes-nodes (3 / 3 active targets)
- kubernetes-pods (45 / 45 active targets)
- kubernetes-service (28 / 28 active targets)
- prometheus (1 / 1 active targets)

kubernetes-endpoints show more

kubernetes-nodes show more

kubernetes-pods show more

kubernetes-service show more

prometheus show more

You can see that you can currently query all the service data information on the cluster.

Configure to Query Specified Resource

The ConfigMap configuration above queries all resource data. If you only need resource data for a certain role, you can add filter conditions.

Take Service as an example, modify the ConfigMap content to query only the service monitoring data:

```
    - job_name: 'kubernetes-service'
      tls_config:
        ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
      bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
      kubernetes_sd_configs:
        - role: service
      relabel_configs:
        - source_labels: [__meta_kubernetes_service_annotation_prometheus_io_scrape]
          action: "keep"
          regex: "true"
        - action: labelmap
          regex: __meta_kubernetes_service_label_(.+)
        - source_labels: [__meta_kubernetes_namespace]
          target_label: kubernetes_namespace
        - source_labels: [__meta_kubernetes_service_name]
          target_label: kubernetes_service_name
        - source_labels: [__meta_kubernetes_service_annotation_prometheus_io_path]
          action: replace
          target_label: __metrics_path__
          regex: "(.+)"
```

Check the Kube-OVN Service in kube-system Namespace:

```
# kubectl get svc -n kube-system
NAME                  TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)                  AGE
kube-dns              ClusterIP   10.4.0.10      <none>        53/UDP,53/TCP,9153/TCP   13d
kube-ovn-cni          ClusterIP   10.4.228.60    <none>        10665/TCP                13d
kube-ovn-controller   ClusterIP   10.4.172.213   <none>        10660/TCP                13d
kube-ovn-monitor      ClusterIP   10.4.242.9     <none>        10661/TCP                13d
kube-ovn-pinger       ClusterIP   10.4.122.52    <none>        8080/TCP                 13d
ovn-nb                ClusterIP   10.4.80.213    <none>        6641/TCP                 13d
ovn-northd            ClusterIP   10.4.126.234   <none>        6643/TCP                 13d
ovn-sb                ClusterIP   10.4.216.249   <none>        6642/TCP                 13d
```

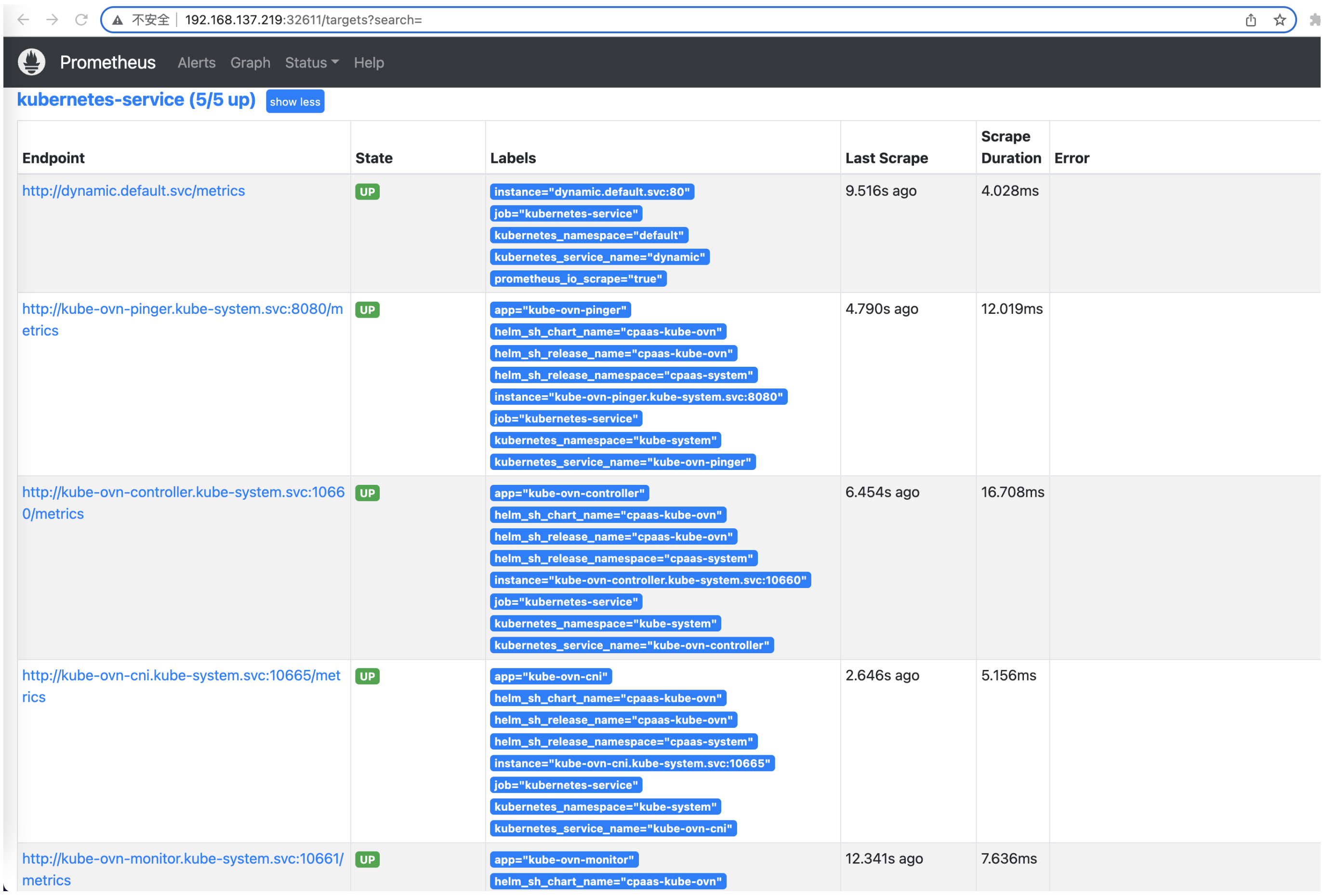
Add annotation `prometheus.io/scrape="true"` to Service:

```
# kubectl annotate svc -n kube-system kube-ovn-cni  prometheus.io/scrape=true
service/kube-ovn-cni annotated
# kubectl annotate svc -n kube-system kube-ovn-controller  prometheus.io/scrape=true
service/kube-ovn-controller annotated
# kubectl annotate svc -n kube-system kube-ovn-monitor  prometheus.io/scrape=true
service/kube-ovn-monitor annotated
# kubectl annotate svc -n kube-system kube-ovn-pinger  prometheus.io/scrape=true
service/kube-ovn-pinger annotated
```

Check the configured Service information:

```
# kubectl get svc -o yaml -n kube-system kube-ovn-controller
apiVersion: v1
kind: Service
metadata:
  annotations:
    helm.sh/chart-version: v3.10.0-alpha.55
    helm.sh/original-name: kube-ovn-controller
    ovn.kubernetes.io/vpc: ovn-cluster
    prometheus.io/scrape: "true"                        // added annotation
  labels:
    app: kube-ovn-controller
  name: kube-ovn-controller
  namespace: kube-system
spec:
  clusterIP: 10.4.172.213
  clusterIPs:
  - 10.4.172.213
  internalTrafficPolicy: Cluster
  ipFamilies:
  - IPv4
  ipFamilyPolicy: SingleStack
  ports:
  - name: metrics
    port: 10660
    protocol: TCP
    targetPort: 10660
  selector:
    app: kube-ovn-controller
  sessionAffinity: None
  type: ClusterIP
status:
  loadBalancer: {}
```

Looking at the Prometheus Status Targets information, you can only see the Services with annotation:



kubernetes-service (5/5 up)

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://dynamic.default.svc/metrics	UP	instance="dynamic.default.svc:80" job="kubernetes-service" kubernetes_namespace="default" kubernetes_service_name="dynamic" prometheus_io_scrape="true"	9.516s ago	4.028ms	
http://kube-ovn-pinger.kube-system.svc:8080/metrics	UP	app="kube-ovn-pinger" helm_sh_chart_name="cpaas-kube-ovn" helm_sh_release_name="cpaas-kube-ovn" helm_sh_release_namespace="cpaas-system" instance="kube-ovn-pinger.kube-system.svc:8080" job="kubernetes-service" kubernetes_namespace="kube-system" kubernetes_service_name="kube-ovn-pinger"	4.790s ago	12.019ms	
http://kube-ovn-controller.kube-system.svc:10660/metrics	UP	app="kube-ovn-controller" helm_sh_chart_name="cpaas-kube-ovn" helm_sh_release_name="cpaas-kube-ovn" helm_sh_release_namespace="cpaas-system" instance="kube-ovn-controller.kube-system.svc:10660" job="kubernetes-service" kubernetes_namespace="kube-system" kubernetes_service_name="kube-ovn-controller"	6.454s ago	16.708ms	
http://kube-ovn-cni.kube-system.svc:10665/metrics	UP	app="kube-ovn-cni" helm_sh_chart_name="cpaas-kube-ovn" helm_sh_release_name="cpaas-kube-ovn" helm_sh_release_namespace="cpaas-system" instance="kube-ovn-cni.kube-system.svc:10665" job="kubernetes-service" kubernetes_namespace="kube-system" kubernetes_service_name="kube-ovn-cni"	2.646s ago	5.156ms	
http://kube-ovn-monitor.kube-system.svc:10661/metrics	UP	app="kube-ovn-monitor" helm_sh_chart_name="cpaas-kube-ovn"	12.341s ago	7.636ms	

For more information about adding filter parameters to relabel, please check Prometheus-Relabel.

PDF Slack Support

July 30, 2025

August 23, 2022

GitHub

3.15.3 Comments

4. KubeVirt

4.1 Fixed VM IP

In container environments, container IP addresses are typically dynamically assigned and may change after container restarts. However, VM users prefer their VM's IP address to be fixed for subsequent management and operations.

However, most common CNIs have the following limitations:

- **Unable to bind IP addresses to the VM lifecycle**: The VM's IP address changes after VM restarts or shutdowns.
- **IP addresses are bound to Nodes**: When a VM migrates to a new node, the previous IP cannot be reused.
- **Unable to support IP address configuration**: Users cannot specify the VM's IP address.

Therefore, the `masquerade` network mode of KubeVirt is often used. It forwards the VM's traffic to the host's network interface via iptables to achieve a fixed VM IP. However, compared to the `bridge` mode, `masquerade` has the following issues:

- **Inconsistent Pod and VM IPs**: This increases management complexity. After restarts and live migrations, Pod IPs change, making the external address still not fixed.
- **Performance**: `masquerade` uses iptables for traffic forwarding, resulting in lower performance compared to the `bridge` mode.
- **Limited to Layer 3 Traffic Forwarding**: Some Layer 2 network functionalities cannot be achieved.
- **Potential Traffic Interruptions**: `masquerade` traffic is tracked by conntrack, which may cause traffic interruptions during live migrations.

Kube-OVN supports binding IP addresses to the VM lifecycle under KubeVirt's `bridge` and `managedTap` network modes. The IP address remains unchanged after operations such as VM restarts and live migrations. It also supports configuring fixed IP addresses for VMs by adding annotations.

4.1.1 Binding IP and VM Lifecycle

For scenarios where users only want the VM's IP address to remain fixed during the VM's lifecycle without specifying the IP address, users can create the VM as usual. Kube-OVN's internal IPAM automatically records the VM's lifecycle, ensuring the VM uses the same IP address after restarts and migrations.

Below is an example using the `bridge` network mode: creating a VM, performing restarts and live migrations, and observing the IP address changes.

1. **Create VM**

```
kubectl apply -f - <<EOF
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: testvm
spec:
  runStrategy: Always
  template:
    metadata:
      labels:
        kubevirt.io/size: small
        kubevirt.io/domain: testvm
      annotations:
        kubevirt.io/allow-pod-bridge-network-live-migration: "true"
    spec:
      domain:
        devices:
          disks:
            - name: containerdisk
              disk:
                bus: virtio
            - name: cloudinitdisk
              disk:
                bus: virtio
          interfaces:
          - name: default
            bridge: {}
        resources:
          requests:
            memory: 64M
      networks:
      - name: default
        pod: {}
      volumes:
        - name: containerdisk
          containerDisk:
            image: quay.io/kubevirt/cirros-container-disk-demo
        - name: cloudinitdisk
          cloudInitNoCloud:
            userDataBase64: SGkuXG4=
EOF
```

2. **View VM Status**

```
kubectl get vmi testvm
```

3. **Restart VM**

```
virtctl restart testvm
```

4. **Live Migrate VM**

```
virtctl migrate testvm
```

You can observe that in bridge mode, the VM's IP address remains unchanged after restarts and live migrations.

4.1.2 Specifying IP Address

For scenarios where users need to specify the VM's IP address, they can add an annotation to the VM when creating it to assign a specific IP address. Other usage methods are consistent with native KubeVirt.

```
kubectl apply -f - <<EOF
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: testvm
spec:
  runStrategy: Always
  template:
    metadata:
      labels:
        kubevirt.io/size: small
        kubevirt.io/domain: testvm
      annotations:
        ovn.kubernetes.io/ip_address: 10.16.0.15
        kubevirt.io/allow-pod-bridge-network-live-migration: "true"
    spec:
      domain:
        devices:
          disks:
            - name: containerdisk
              disk:
                bus: virtio
            - name: cloudinitdisk
              disk:
                bus: virtio
          interfaces:
          - name: default
            bridge: {}
        resources:
          requests:
            memory: 64M
      networks:
      - name: default
        pod: {}
      volumes:
        - name: containerdisk
          containerDisk:
            image: quay.io/kubevirt/cirros-container-disk-demo
        - name: cloudinitdisk
          cloudInitNoCloud:
            userDataBase64: SGkuXG4=
EOF
```

PDF Slack Support

March 3, 2025

March 3, 2025

GitHub

4.1.3 Comments

4.2 Dual-Stack Network

In KubeVirt's Bridge network mode, the DHCP service is provided by `virt-launcher`. However, KubeVirt currently only implements DHCP for IPv4 single-stack, which prevents KubeVirt VMs in Bridge mode from dynamically obtaining IPv6 addresses through the RA (Router Advertisement) protocol. Although Kube-OVN provides DHCP and RA capabilities, these features are ineffective because KubeVirt intercepts DHCP/RA requests in advance.

In versions of KubeVirt after 1.4.0, a new Network Binding Plugin introduces a Bridge-like network mode called `managedTap`. In this mode, KubeVirt does not intercept DHCP requests. Therefore, by combining the new `managedTap` mode with Kube-OVN's DHCP/RA capabilities, it is possible to automatically obtain dual-stack network addresses for VMs.

4.2.1 Usage

1. Enable DHCP and IPv6 RA features in the Subnet of Kube-OVN, as shown in the following YAML configuration:

```yaml
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
  name: dual-stack-subnet
spec:
  cidrBlock: "10.244.0.0/16,fd00:10:244::/64"
  enableDHCP: true
  enableIPv6RA: true
```

2. Register the `managedTap` Network Binding Plugin in KubeVirt:

```
# kubectl patch kubevirts -n kubevirt kubevirt --type=json -p=\
'[{"op": "add", "path": "/spec/configuration/network",   "value": {
    "binding": {
        "managedtap": {
            "domainAttachmentType": "managedTap"
        }
    }
}}]'
```

3. Create a virtual machine, specifying the use of the `managedTap` network type:

```yaml
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: dual-stack-vm
  namespace: default
spec:
  running: false
  template:
    spec:
      domain:
        devices:
          interfaces:
            - name: default
              binding:
                name: managedtap
        networks:
        - name: default
          pod: {}
```

By following these steps, VMs can obtain their corresponding IPv4/IPv6 addresses through the DHCP and IPv6 RA protocols.

PDF Slack Support

March 3, 2025

March 3, 2025

GitHub

4.2.2 Comments

4.3 Live Migration

In virtual machine usage scenarios, live migration allows a virtual machine to be moved from one node to another for operations such as node maintenance, upgrades, and failover.

KubeVirt faces the following challenges during live migration:

- KubeVirt does not support live migration of virtual machines using bridge network mode by default.
- KubeVirt only handles memory and disk migration without specific optimizations for network migration.
- If the virtual machine's IP changes during migration, it cannot achieve a seamless live migration.
- If the network is interrupted during migration, it cannot achieve a seamless live migration.

Kube-OVN specifically addresses the above issues during the virtual machine migration process, allowing users to perform network-transparent live migrations. Our tests show that network interruption time can be controlled within 0.5 seconds, and TCP connections remain uninterrupted.

4.3.1 Usage

Users only need to add the annotation `kubevirt.io/allow-pod-bridge-network-live-migration: "true"` in the VM Spec. Kube-OVN will automatically handle network migration during the process.

1. **Create VM**

```
kubectl apply -f - <<EOF
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: testvm
spec:
  runStrategy: Always
  template:
    metadata:
      labels:
        kubevirt.io/size: small
        kubevirt.io/domain: testvm
      annotations:
        kubevirt.io/allow-pod-bridge-network-live-migration: "true"
    spec:
      domain:
        devices:
          disks:
            - name: containerdisk
              disk:
                bus: virtio
            - name: cloudinitdisk
              disk:
                bus: virtio
          interfaces:
          - name: default
            bridge: {}
        resources:
          requests:
            memory: 64M
      networks:
      - name: default
        pod: {}
      volumes:
        - name: containerdisk
          containerDisk:
            image: quay.io/kubevirt/cirros-container-disk-demo
        - name: cloudinitdisk
          cloudInitNoCloud:
            userDataBase64: SGkuXG4=
EOF
```

2. **SSH into the Virtual Machine and Test Network Connectivity**

```
# password: gocubsgo
virtctl ssh cirros@testvm
ping 8.8.8.8
```

3. **Perform Migration in Another Terminal and Observe Virtual Machine Network Connectivity**

```
virtctl migrate testvm
```

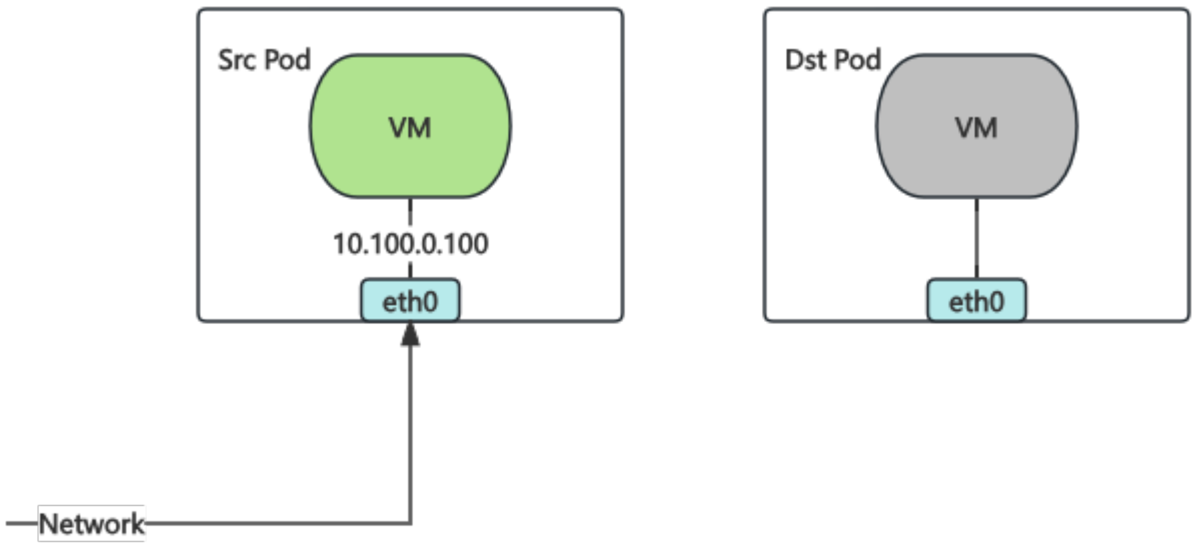
It can be observed that during the VM live migration process, the SSH connection remains uninterrupted, and ping only experiences packet loss in a few instances.

4.3.2 Live Migration Principles

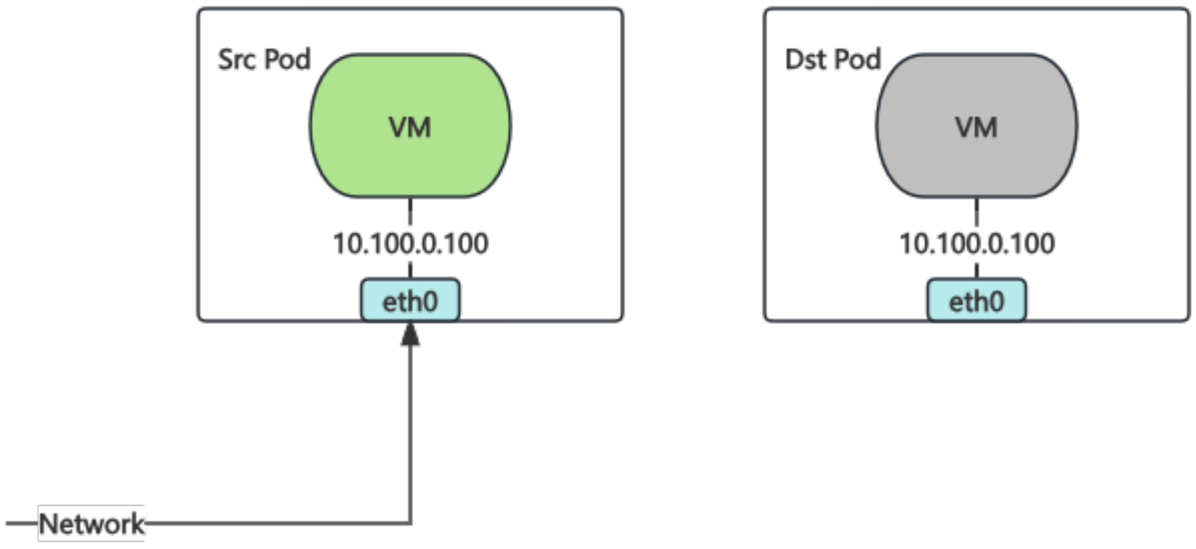
During the live migration process, Kube-OVN implements techniques inspired by the Red Hat team's Live migration - Reducing downtime with multi-chassis port bindings.

To ensure network consistency between the source and target virtual machines during migration, the same IP address exists on the network for both the source and target VMs. This requires handling network conflicts and traffic confusion. The specific steps are as follows: Here's the translation:

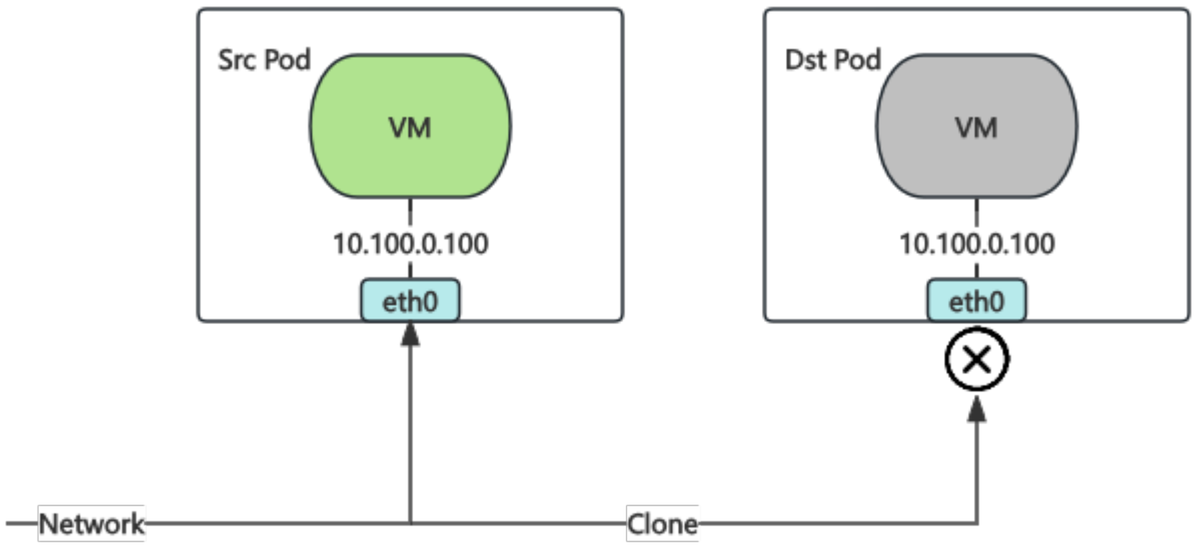
1. KubeVirt initiates the migration and creates the corresponding Pod on the target machine.



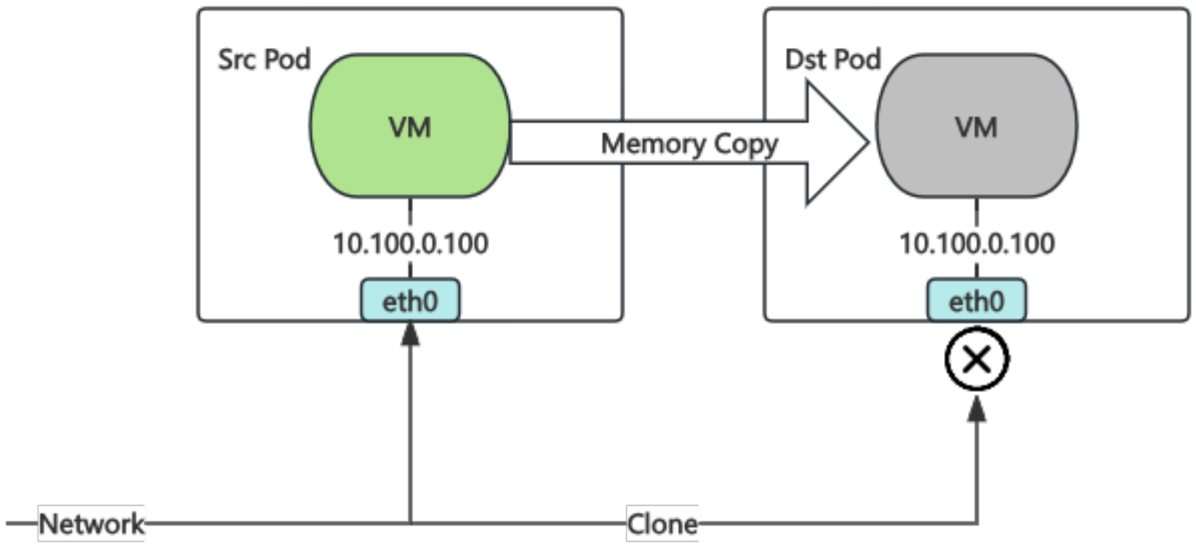
1. Kube-OVN detects that the Pod is the target Pod for a live migration and reuses the network port information from the source Pod.



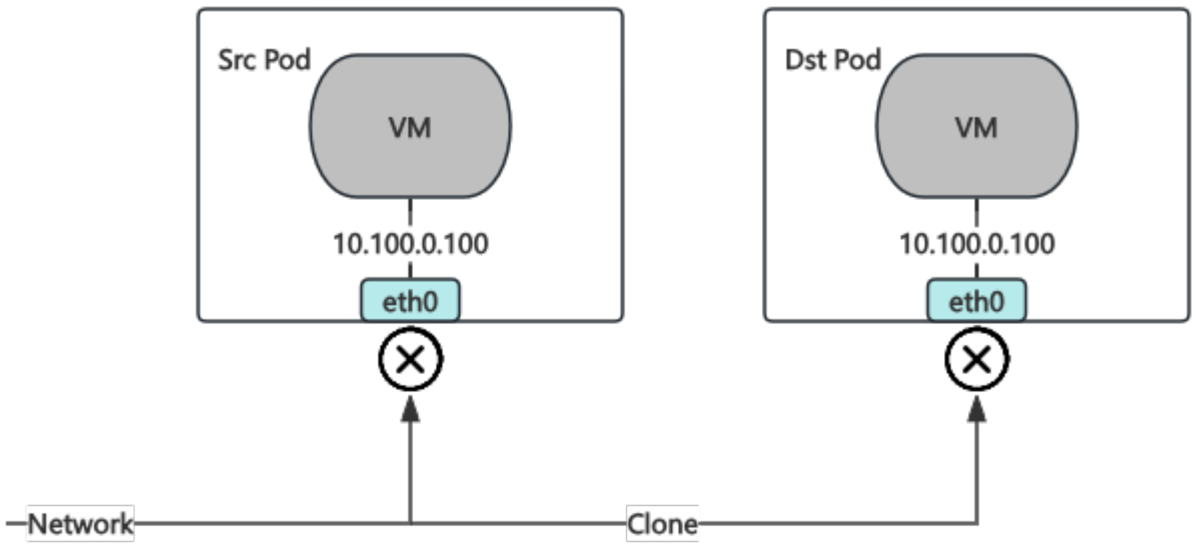
1. Kube-OVN sets up traffic replication, so network traffic will be copied to both the source Pod and the target Pod. This helps reduce the interruption time caused by control plane switch during network migration. The network port of the target Pod is temporarily disabled, so the target Pod will not actually receive the replicated traffic, avoiding traffic confusion.



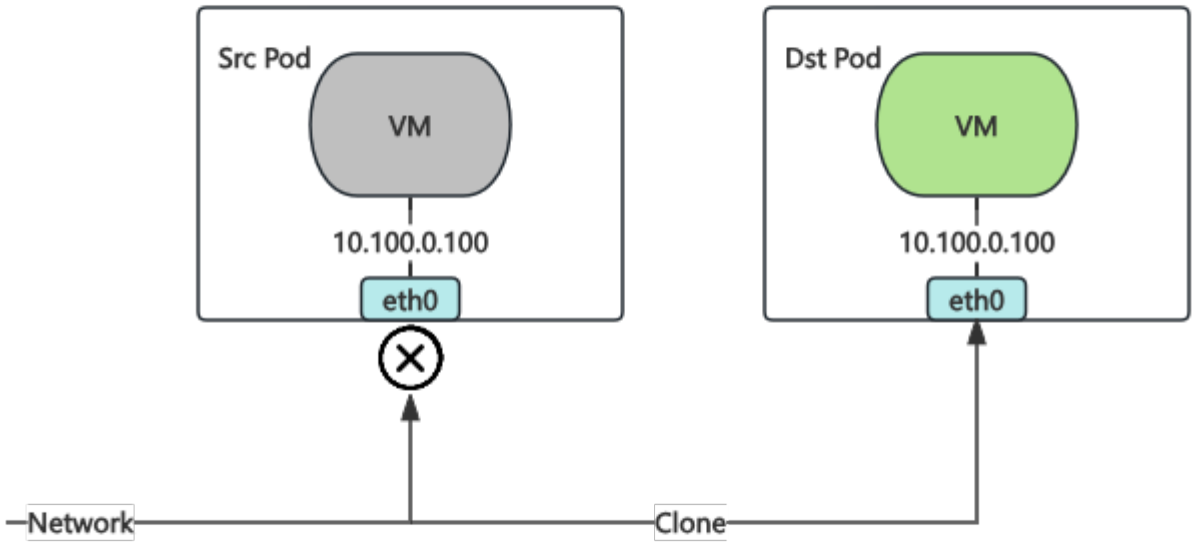
1. KubeVirt synchronizes the VM memory.



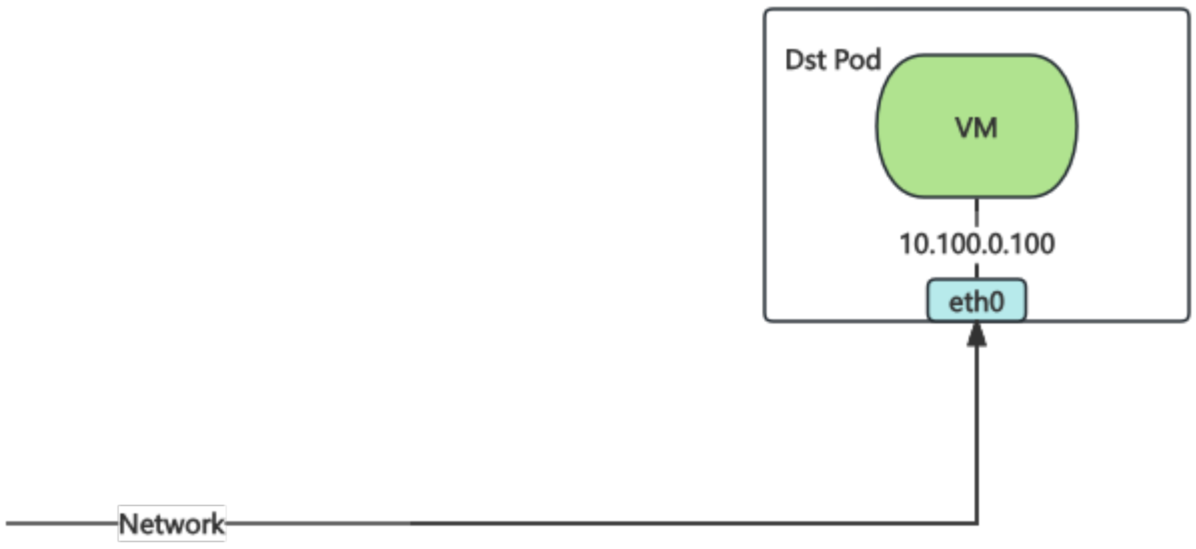
1. KubeVirt completes the memory synchronization and deactivates the source Pod. At this point, the source Pod will not handle network traffic.



1. KubeVirt activates the target Pod. At this point, libvirt sends a RARP to activate the network port of the target Pod, and the target Pod starts processing traffic.



1. KubeVirt deletes the source Pod, completing the live migration. Kube-OVN listens for the migration completion event through the Watch Migration CR and stops traffic replication after the migration is finished.



In this process, the network interruption mainly occurs between steps 5 and 6. The network interruption time primarily depends on the time it takes for libvirt to send the RARP. Tests show that the network interruption time can be controlled within 0.5 seconds, and TCP connections will not experience interruptions due to the retry mechanism.

PDF Slack Support

July 30, 2025

March 3, 2025

GitHub

4.3.3 Comments

4.4 DHCP

When using SR-IOV or DPDK type networks, KubeVirt's built-in DHCP does not work in this network mode. Kube-OVN can use the DHCP capabilities of OVN to set DHCP options at the subnet level to help KubeVirt VMs of these network types to properly use DHCP to obtain assigned IP addresses. Kube-OVN supports both DHCPv4 and DHCPv6.

The subnet DHCP is configured as follows:

```yaml
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
  name: sn-dual
spec:
  cidrBlock: "10.0.0.0/24,240e::a00/120"
  default: false
  disableGatewayCheck: true
  disableInterConnection: false
  excludeIps:
    - 10.0.0.1
    - 240e::a01
  gateway: 10.0.0.1,240e::a01
  gatewayNode: ''
  gatewayType: distributed
  natOutgoing: false
  private: false
  protocol: Dual
  provider: ovn
  vpc: vpc-test
  enableDHCP: true
  dhcpV4Options: "lease_time=3600,router=10.0.0.1,server_id=169.254.0.254,server_mac=00:00:00:2E:2F:B8"
  dhcpV6Options: "server_id=00:00:00:2E:2F:C5"
  enableIPv6RA: true
  ipv6RAConfigs: "address_mode=dhcpv6_stateful,max_interval=30,min_interval=5,send_periodic=true"
```

- `enableDHCP` : Whether to enable the DHCP function for the subnet.
- `dhcpV4Options` , `dhcpV6Options` : This field directly exposes DHCP-related options within ovn-nb, please reade DHCP Options for more detail. The default value is `"lease_time=3600, router=$ipv4_gateway, server_id=169.254.0.254, server_mac=$random_mac"` and `server_id=$random_mac` .
- `enableIPv6RA` : Whether to enable the route broadcast function of DHCPv6.
- `ipv6RAConfigs` : This field directly exposes DHCP-related options within ovn-nb Logical_Router_Port, please read Logical Router Port for more detail. The default value is `address_mode=dhcpv6_stateful, max_interval=30, min_interval=5, send_periodic=true` .

PDF Slack Support

July 30, 2025

May 24, 2022

GitHub

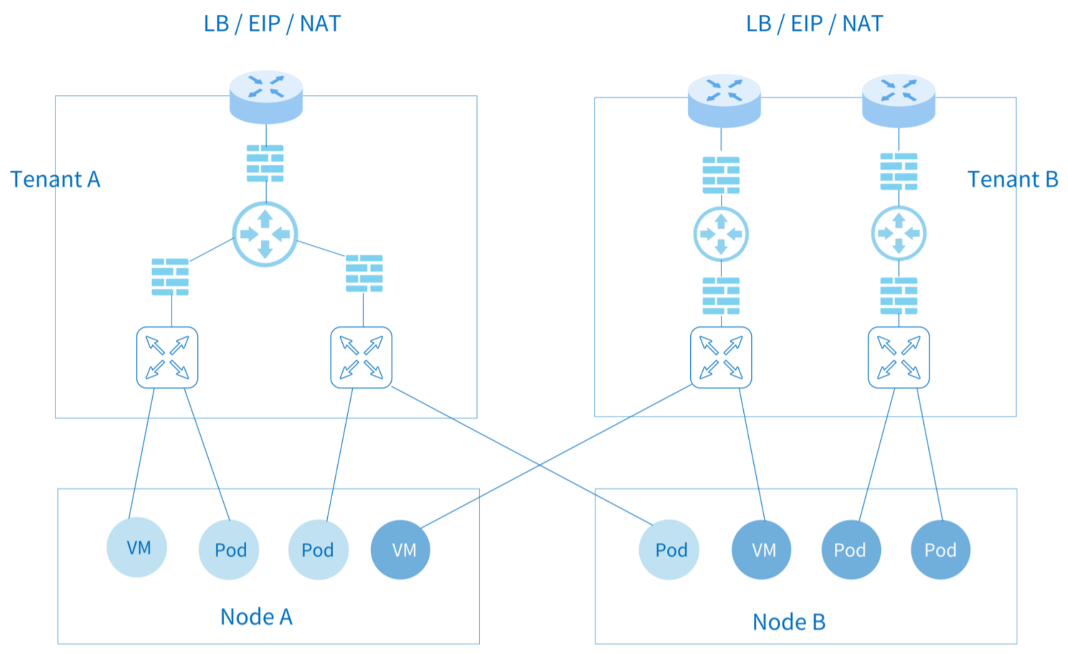
4.4.1 Comments

5. VPC Network

5.1 Config VPC

Kube-OVN supports multi-tenant isolation level VPC networks. Different VPC networks are independent of each other and can be configured separately with Subnet CIDRs, routing policies, security policies, outbound gateways, EIP, etc.

VPC is mainly used in scenarios where there requires strong isolation of multi-tenant networks and some Kubernetes networking features conflict under multi-tenant networks. For example, node and pod access, NodePort functionality, network access-based health checks, and DNS capabilities are not supported in multi-tenant network scenarios at this time. In order to facilitate common Kubernetes usage scenarios, Kube-OVN has a special design for the default VPC where the Subnet under the VPC can meet the Kubernetes specification. The custom VPC supports static routing, EIP and NAT gateways as described in this document. Common isolation requirements can be achieved through network policies and Subnet ACLs under the default VPC, so before using a custom VPC, please make sure whether you need VPC-level isolation and understand the limitations under the custom VPC. For Underlay subnets, physical switches are responsible for data-plane forwarding, so VPCs cannot isolate Underlay subnets.



5.1.1 Implementation Principle

In Kube-OVN, each VPC is mapped to a Logical Router in OVN. Multiple logical routers are independent network units, and each logical router can be associated with its own Logical Switches, thereby having independent network ports and IP address spaces. Traffic from different VPCs is distinguished and isolated by different Datapath IDs when traversing tunnels, and forwarding is performed based on Datapath IDs. This ensures IP address space isolation, allowing the same IP addresses to be used in different VPCs without conflicts. For tunnel encapsulation formats, refer to OVN Architecture Design Decisions.

5.1.2 Creating Custom VPCs

Create two VPCs:

```yaml
kind: Vpc
apiVersion: kubeovn.io/v1
metadata:
  name: test-vpc-1
spec:
  namespaces:
  - ns1
---
kind: Vpc
apiVersion: kubeovn.io/v1
metadata:
  name: test-vpc-2
spec:
  namespaces:
    - ns2
```

- `namespaces` : Limit which namespaces can use this VPC. If empty, all namespaces can use this VPC.

Create two Subnets, belonging to two different VPCs and having the same CIDR:

```yaml
kind: Subnet
apiVersion: kubeovn.io/v1
metadata:
  name: net1
spec:
  vpc: test-vpc-1
  cidrBlock: 10.0.1.0/24
  protocol: IPv4
  namespaces:
    - ns1
---
kind: Subnet
apiVersion: kubeovn.io/v1
metadata:
  name: net2
spec:
  vpc: test-vpc-2
  cidrBlock: 10.0.1.0/24
  protocol: IPv4
  namespaces:
    - ns2
```

Create Pods under two separate Namespaces:

```yaml
apiVersion: v1
kind: Pod
metadata:
  namespace: ns1
  name: vpc1-pod
spec:
  containers:
    - name: vpc1-pod
      image: docker.io/library/nginx:alpine
---
apiVersion: v1
kind: Pod
metadata:
  namespace: ns2
  name: vpc2-pod
spec:
  containers:
    - name: vpc2-pod
      image: docker.io/library/nginx:alpine
```

After running successfully, you can observe that the two Pod addresses belong to the same CIDR, but the two Pods cannot access each other because they are running on different tenant VPCs.

Custom VPC Pod supports livenessProbe and readinessProbe

Since the Pods under the custom VPC do not communicate with the network of the node, the probe packets sent by the kubelet cannot reach the Pods in the custom VPC. Kube-OVN uses TProxy to redirect the detection packets sent by kubelet to Pods in the custom VPC to achieve this function.

The configuration method is as follows, add the parameter `--enable-tproxy=true` in DaemonSet `kube-ovn-cni` :

```yaml
spec:
  template:
    spec:
      containers:
      - args:
        - --enable-tproxy=true
```

Restrictions for this feature:

1. When Pods under different VPCs have the same IP under the same node, the detection function fails.
2. Currently, only `tcpSocket` and `httpGet` are supported.

5.1.3 Create VPC NAT Gateway

Subnets under custom VPCs do not support distributed gateways and centralized gateways under default VPCs.

Pod access to the external network within the VPC requires a VPC gateway, which bridges the physical and tenant networks and provides floating IP, SNAT and DNAT capabilities.

The VPC gateway function relies on Multus-CNI function, please refer to multus-cni.

Configuring the External Network

```yaml
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
  name: ovn-vpc-external-network
spec:
  protocol: IPv4
  provider: ovn-vpc-external-network.kube-system
  cidrBlock: 192.168.0.0/24
  gateway: 192.168.0.1  # IP address of the physical gateway
  excludeIps:
  - 192.168.0.1..192.168.0.10
---
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: ovn-vpc-external-network
  namespace: kube-system
spec:
  config: '{
      "cniVersion": "0.3.0",
      "type": "macvlan",
      "master": "eth1",
      "mode": "bridge",
      "ipam": {
        "type": "kube-ovn",
        "server_socket": "/run/openvswitch/kube-ovn-daemon.sock",
        "provider": "ovn-vpc-external-network.kube-system"
      }
    }'
```

- This Subnet is used to manage the available external addresses and the address will be allocated to VPC NAT Gateway through Macvlan, so please communicate with your network administrator to give you the available physical segment IPs.
- The VPC gateway uses Macvlan for physical network configuration, and `master` of `NetworkAttachmentDefinition` should be the NIC name of the corresponding physical network NIC.
- `name` : External network name.

For macvlan mode, the nic will send packets directly through that node NIC, relying on the underlying network devices for L2/L3 level forwarding capabilities. You need to configure the corresponding gateway, Vlan and security policy in the underlying network device in advance.

1. For OpenStack VM environments, you need to turn off `PortSecurity` on the corresponding network port.
2. For VMware vSwitch networks, `MAC Address Changes` , `Forged Transmits` and `Promiscuous Mode Operation` should be set to `allow` .
3. For Hyper-V virtualization, `MAC Address Spoofing` should be enabled in VM nic advanced features.
4. Public clouds, such as AWS, GCE, AliCloud, etc., do not support user-defined Mac, so they cannot support Macvlan mode network.
5. Due to the limitations of Macvlan itself, a Macvlan sub-interface cannot access the address of the parent interface, which means that it is not possible to access the Pod through the network on the host machine where the VpcNATGateway Pod is located.
6. If the physical network card corresponds to a switch interface in Trunk mode, a sub-interface needs to be created on the network card and provided to Macvlan for use.

Enabling the VPC Gateway

The VPC gateway functionality can be enabled by configuring the `ovn-vpc-nat-gw-config` in the `kube-system` namespace. The `nodeSelector` can be used to specify the node where the gateway is deployed:

```yaml
---
kind: ConfigMap
apiVersion: v1
metadata:
  name: ovn-vpc-nat-config
  namespace: kube-system
data:
  image: docker.io/kubeovn/vpc-nat-gateway:v1.14.4
  nodeSelector: |
    kubernetes.io/hostname: kube-ovn-control-plane
---
kind: ConfigMap
apiVersion: v1
metadata:
  name: ovn-vpc-nat-gw-config
  namespace: kube-system
data:
  enable-vpc-nat-gw: 'true'
```

- `image` : The image used by the Gateway Pod.
- `enable-vpc-nat-gw` : Controls whether the VPC Gateway feature is enabled.

Create VPC Gateway

```yaml
kind: VpcNatGateway
apiVersion: kubeovn.io/v1
metadata:
  name: gw1
spec:
  vpc: test-vpc-1
  subnet: net1
  lanIp: 10.0.1.254
  selector:
    - "kubernetes.io/hostname: kube-ovn-worker"
    - "kubernetes.io/os: linux"
  externalSubnets:
    - ovn-vpc-external-network
```

- `vpc` : The VPC to which this VpcNatGateway belongs.
- `subnet` : A Subnet within the VPC, the VPC Gateway Pod will use `lanIp` to connect to the tenant network under that subnet.
- `lanIp` : An unused IP within the `subnet` that the VPC Gateway Pod will eventually use. When configuring routing for a VPC, the `nextHopIP` needs to be set to the `lanIp` of the current VpcNatGateway.
- `selector` : The node selector for VpcNatGateway Pod has the same format as NodeSelector in Kubernetes.
- `externalSubnets` : External network used by the VPC gateway, if not configured, `ovn-vpc-external-network` is used by default, and only one external network is supported in the current version.

Other configurable parameters:

- `tolerations` : Configure tolerance for the VPC gateway. For details, see Taints and Tolerations
- `affinity` : Configure affinity for the Pod or node of the VPC gateway. For details, see Assigning Pods to Nodes

Things to note when using VPC-NAT-GW:

1. After the nat gw pod is created, the nic net1 arp is disabled and the physical gateway cannot be pinged. After creating the eip, the arp will be automatically enabled and the ping to underlay gateway should be ok.

Create EIP

EIP allows for floating IP, SNAT, and DNAT operations after assigning an IP from an external network segment to a VPC gateway.

Randomly assign an address to the EIP:

```yaml
kind: IptablesEIP
apiVersion: kubeovn.io/v1
metadata:
  name: eip-random
spec:
  natGwDp: gw1
```

Fixed EIP address assignment:

```yaml
kind: IptablesEIP
apiVersion: kubeovn.io/v1
metadata:
  name: eip-static
spec:
  natGwDp: gw1
  v4ip: 192.168.0.100
```

Specify the external network on which the EIP is located:

```yaml
kind: IptablesEIP
apiVersion: kubeovn.io/v1
metadata:
  name: eip-random
spec:
  natGwDp: gw1
  externalSubnet: ovn-vpc-external-network
```

- `externalSubnet` : The name of the external network on which the EIP is located. If not specified, it defaults to `ovn-vpc-external-network`. If specified, it must be one of the `externalSubnets` of the VPC gateway.

Create DNAT Rules

Through the DNAT rules, external can access to an IP and port within a VPC through an EIP and port.

```yaml
kind: IptablesEIP
apiVersion: kubeovn.io/v1
metadata:
  name: eipd01
spec:
  natGwDp: gw1

---
kind: IptablesDnatRule
apiVersion: kubeovn.io/v1
metadata:
  name: dnat01
spec:
  eip: eipd01
  externalPort: '8888'
  internalIp: 10.0.1.10
  internalPort: '80'
  protocol: tcp
```

Create SNAT Rules

Through SNAT rules, when a Pod in the VPC accesses an external address, it will go through the corresponding EIP for SNAT.

```yaml
---
kind: IptablesEIP
apiVersion: kubeovn.io/v1
metadata:
  name: eips01
spec:
  natGwDp: gw1
---
kind: IptablesSnatRule
apiVersion: kubeovn.io/v1
metadata:
  name: snat01
spec:
  eip: eips01
  internalCIDR: 10.0.1.0/24
```

Create Floating IP

Through floating IP rules, one IP in the VPC will be completely mapped to the EIP, and the external can access the IP in the VPC through this EIP. When the IP in the VPC accesses the external address, it will be SNAT to this EIP

```yaml
---
kind: IptablesEIP
```

```yaml
apiVersion: kubeovn.io/v1
metadata:
  name: eipf01
spec:
  natGwDp: gw1

---
kind: IptablesFIPRule
apiVersion: kubeovn.io/v1
metadata:
  name: fip01
spec:
  eip: eipf01
  internalIp: 10.0.1.5
```

5.1.4 Custom Routing

Within the custom VPC, users can customize the routing rules within the VPC and combine it with the gateway for more flexible forwarding. Kube-OVN supports static routes and more flexible policy routes.

Static Routes

```yaml
kind: Vpc
apiVersion: kubeovn.io/v1
metadata:
  name: test-vpc-1
spec:
  staticRoutes:
    - cidr: 0.0.0.0/0
      nextHopIP: 10.0.1.254
      policy: policyDst
    - cidr: 172.31.0.0/24
      nextHopIP: 10.0.1.253
      policy: policySrc
      routeTable: "rtb1"
```

- `policy` : Supports destination routing `policyDst` and source routing `policySrc` .
- When there are overlapping routing rules, the rule with the longer CIDR mask has higher priority, and if the mask length is the same, the destination route has a higher priority over the source route.
- `routeTable` : You can store the route in specific table, default is main table. Associate with subnet please defer to Create Custom Subnets

Policy Routes

Traffic matched by static routes can be controlled at a finer granularity by policy routing. Policy routing provides more precise matching rules, priority control and more forwarding actions. This feature brings the OVN internal logical router policy function directly to the outside world, for more information on its use, please refer to Logical Router Policy.

An example of policy routes:

```yaml
kind: Vpc
apiVersion: kubeovn.io/v1
metadata:
  name: test-vpc-1
spec:
  policyRoutes:
    - action: drop
      match: ip4.src==10.0.1.0/24 && ip4.dst==10.0.1.250
      priority: 11
    - action: reroute
      match: ip4.src==10.0.1.0/24
      nextHopIP: 10.0.1.252
      priority: 10
```

5.1.5 Custom vpc-dns

Due to the isolation between custom VPCs and default VPC networks, Pods in VPCs cannot use the default coredns service for domain name resolution. If you want to use coredns to resolve Service domain names within the custom VPC, you can use the `vpc-dns` resource provided by Kube-OVN.

Create an Additional Network

```yaml
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: ovn-nad
  namespace: default
spec:
  config: '{
      "cniVersion": "0.3.0",
      "type": "kube-ovn",
      "server_socket": "/run/openvswitch/kube-ovn-daemon.sock",
      "provider": "ovn-nad.default.ovn"
    }'
```

Modify the Provider of the ovn-default Logical Switch

Modify the provider of ovn-default to the provider `ovn-nad.default.ovn` configured above in nad:

```yaml
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
  name: ovn-default
spec:
  cidrBlock: 10.16.0.0/16
  default: true
  disableGatewayCheck: false
  disableInterConnection: false
  enableDHCP: false
  enableIPv6RA: false
  excludeIps:
  - 10.16.0.1
  gateway: 10.16.0.1
  gatewayType: distributed
  logicalGateway: false
  natOutgoing: true
  private: false
  protocol: IPv4
  provider: ovn-nad.default.ovn
  vpc: ovn-cluster
```

Modify the vpc-dns ConfigMap

Create a ConfigMap in the kube-system namespace, configure the vpc-dns parameters to be used for the subsequent vpc-dns feature activation:

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: vpc-dns-config
  namespace: kube-system
data:
  coredns-vip: 10.96.0.3
  enable-vpc-dns: "true"
  nad-name: ovn-nad
  nad-provider: ovn-nad.default.ovn
```

- `enable-vpc-dns` : (optional) `true` to enable the feature, `false` to disable the feature. Default `true` .
- `coredns-image` : (optional): DNS deployment image. Default is the cluster coredns deployment version.
- `coredns-template` : (optional): URL of the DNS deployment template. Default: `yamls/coredns-template.yaml` in the current version repository.
- `coredns-vip` : VIP providing LB service for coredns.
- `nad-name` : Name of the configured `network-attachment-definitions` resource.
- `nad-provider` : Name of the used provider.
- `k8s-service-host` : (optional) IP used by coredns to access the k8s apiserver service.
- `k8s-service-port` : (optional) Port used by coredns to access the k8s apiserver service.

Deploying VPC-DNS Dependent Resources

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    kubernetes.io/bootstrapping: rbac-defaults
  name: system:vpc-dns
rules:
  - apiGroups:
    - ""
    resources:
    - endpoints
    - services
    - pods
    - namespaces
    verbs:
    - list
    - watch
  - apiGroups:
    - discovery.k8s.io
    resources:
    - endpointslices
    verbs:
    - list
    - watch
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  annotations:
    rbac.authorization.kubernetes.io/autoupdate: "true"
  labels:
    kubernetes.io/bootstrapping: rbac-defaults
  name: vpc-dns
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: system:vpc-dns
subjects:
- kind: ServiceAccount
  name: vpc-dns
  namespace: kube-system
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: vpc-dns
  namespace: kube-system
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: vpc-dns-corefile
  namespace: kube-system
data:
  Corefile: |
    .:53 {
        errors
        health {
          lameduck 5s
        }
        ready
        kubernetes cluster.local in-addr.arpa ip6.arpa {
          pods insecure
          fallthrough in-addr.arpa ip6.arpa
        }
        prometheus :9153
        forward . /etc/resolv.conf {
          prefer_udp
        }
        cache 30
        loop
        reload
        loadbalance
    }
```

Deploy vpc-dns

```yaml
kind: VpcDns
apiVersion: kubeovn.io/v1
metadata:
  name: test-cjh1
spec:
  vpc: cjh-vpc-1
  subnet: cjh-subnet-1
```

- `vpc` : The VPC name used to deploy the DNS component.
- `subnet` : The subnet name used to deploy the DNS component.

View resource information:

```
[root@hci-dev-mst-1 kubeovn]# kubectl get vpc-dns
NAME        ACTIVE   VPC         SUBNET
test-cjh1   false    cjh-vpc-1   cjh-subnet-1
test-cjh2   true     cjh-vpc-1   cjh-subnet-2
```

- `ACTIVE` : if the custom vpc-dns is ready.

Restrictions

- Only one custom DNS component will be deployed in one VPC;
- When multiple VPC-DNS resources (i.e. different subnets in the same VPC) are configured in one VPC, only one VPC-DNS resource with status `true` will be active, while the others will be `false` ;
- When the `true` VPC-DNS is deleted, another `false` VPC-DNS will be deployed.

5.1.6 Default subnet selection for custom VPC

If the custom VPC is running multiple Subnets, you can specify the default Subnet for that VPC.

```
kind: Vpc
apiVersion: kubeovn.io/v1
metadata:
  name: test-vpc-1
spec:
  namespaces:
  - ns1
  defaultSubnet: test
```

- `defaultSubnet` : Name of the subnet that should be used by custom VPC as the default one.

Please note that it will annotate the VPC namespaces with just one logical switch using `"ovn.kubernetes.io/logical_switch"` annotation. Any of the new Pods without `"ovn.kubernetes.io/logical_switch"` annotation will be added to the default Subnet.

PDF Slack Support

July 14, 2025

May 24, 2022

GitHub

5.1.7 Comments

5.2 VPC Egress Gateway

VPC Egress Gateway is used to control external network access for Pods within a VPC (including the default VPC) with a group of static addresses and has the following features:

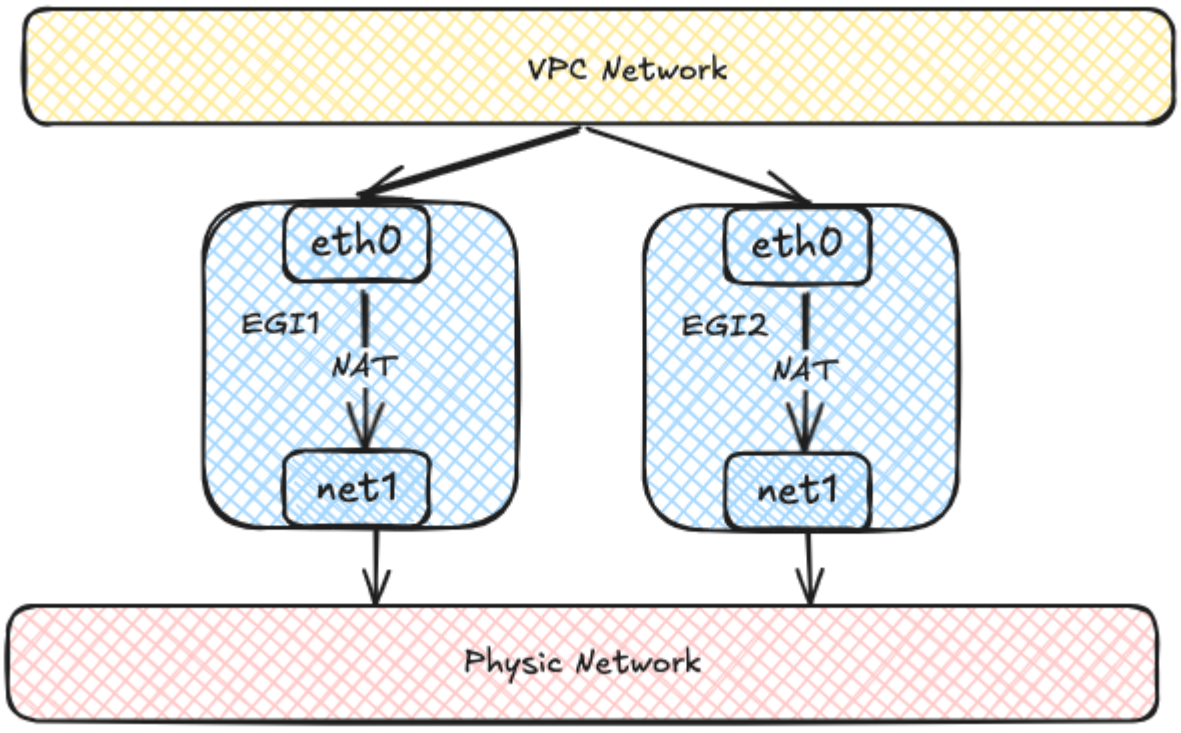
- Achieves Active-Active high availability through ECMP, enabling horizontal throughput scaling
- Implements fast failover (<1s) via BFD
- Supports IPv6 and dual-stack
- Enables granular routing control through `NamespaceSelector` and `PodSelector`
- Allows flexible scheduling of Egress Gateway through `NodeSelector`

At the same time, VPC Egress Gateway has the following limitations:

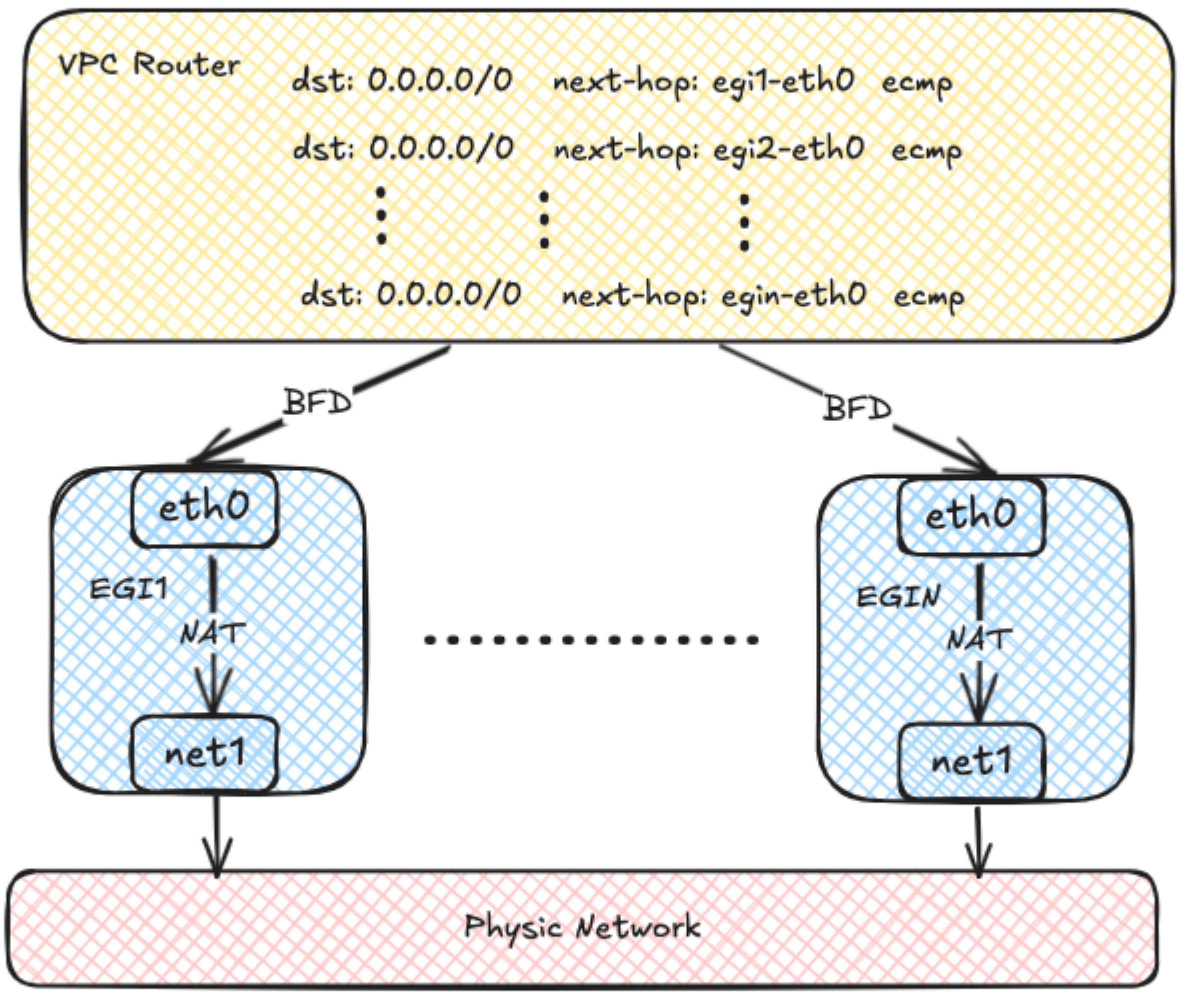
- Uses macvlan for underlying network connectivity, requiring Underlay support from the underlying network
- In multi-instance Gateway mode, multiple Egress IPs are required
- Currently, only supports SNAT; EIP and DNAT are not supported
- Currently, recording source address translation relationships is not supported

5.2.1 Implementation Details

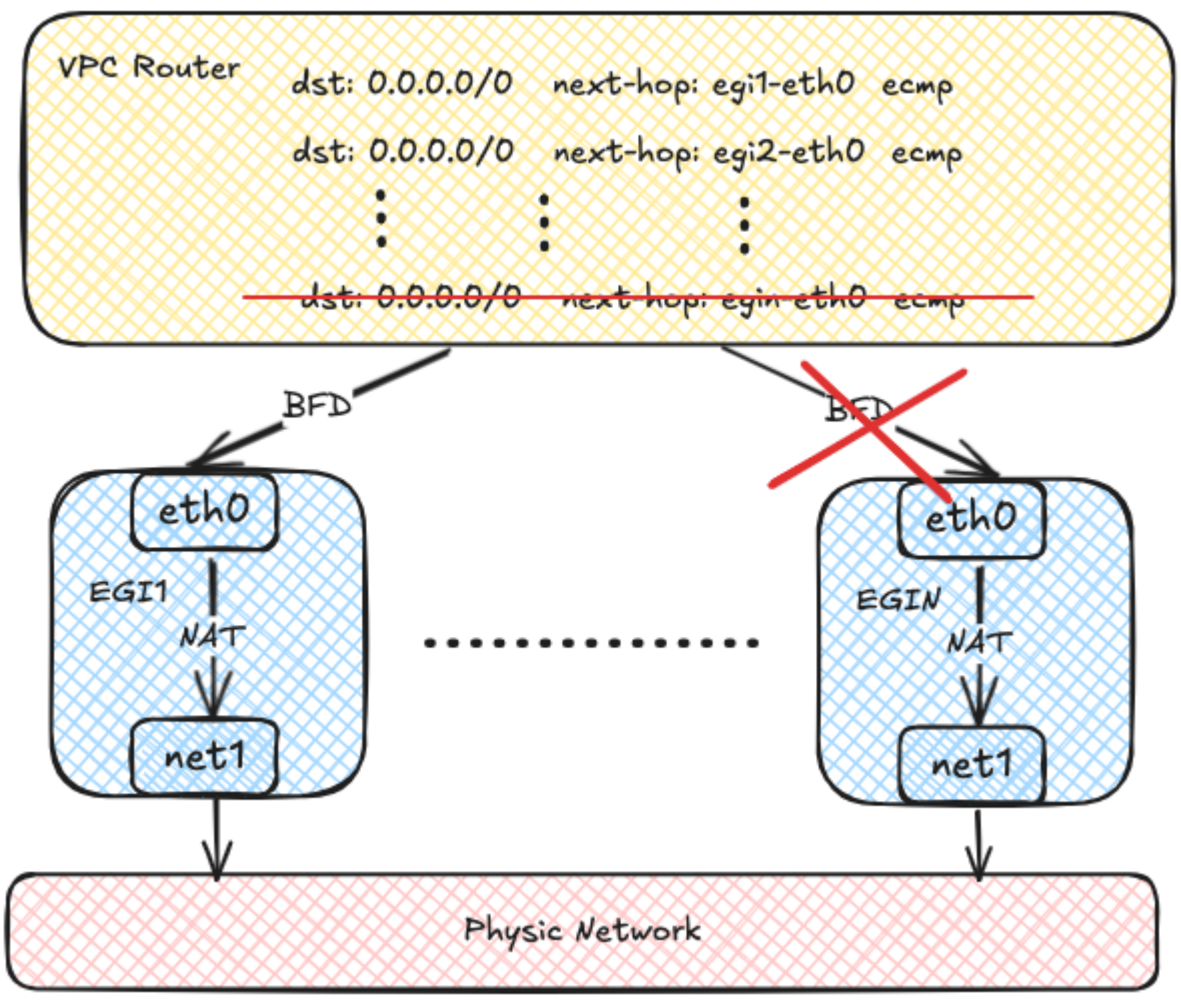
Each Egress Gateway consists of multiple Pods with multiple network interfaces. Each Pod has two network interfaces: one joins the virtual network for communication within the VPC, and the other connects to the underlying physical network via Macvlan for external network communication. Virtual network traffic ultimately accesses the external network through NAT within the Egress Gateway instances.



Each Egress Gateway instance registers its address in the OVN routing table. When a Pod within the VPC needs to access the external network, OVN uses source address hashing to forward traffic to multiple Egress Gateway instance addresses, achieving load balancing. As the number of Egress Gateway instances increases, throughput can also scale horizontally.


VPC Router
dst: 0.0.0.0/0 next-hop: egi1-eth0 ecmp
dst: 0.0.0.0/0 next-hop: egi2-eth0 ecmp
dst: 0.0.0.0/0 next-hop: egin-eth0 ecmp
BFD
BFD
eth0
EGI1
NAT
net1
eth0
EGIN
NAT
net1
Physic Network

OVN uses the BFD protocol to probe multiple Egress Gateway instances. When an Egress Gateway instance fails, OVN marks the corresponding route as unavailable, enabling rapid failure detection and recovery.


VPC Router
dst: 0.0.0.0/0 next-hop: egi1-eth0 ecmp
dst: 0.0.0.0/0 next-hop: egi2-eth0 ecmp
~~dst: 0.0.0.0/0 next-hop: egin-eth0 ecmp~~
BFD
BFD
eth0
EGI1
NAT
net1
eth0
EGIN
NAT
net1
Physic Network

5.2.2 Requirements

The VPC Egress Gateway is the same as the VPC NAT Gateway in that it requires Multus-CNI.

No ConfigMap needs to be configured to use VPC Egress Gateway.

5.2.3 Usage

Creating a Network Attachment Definition

The VPC Egress Gateway uses multiple NICs to access both the VPC and the external network, so you need to create a Network Attachment Definition to connect to the external network. An example of using the `macvlan` plugin with IPAM provided by Kube-OVN is shown below:

```yaml
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  name: eth1
  namespace: default
spec:
  config: '{
      "cniVersion": "0.3.0",
      "type": "macvlan",
      "master": "eth1",
      "mode": "bridge",
      "ipam": {
        "type": "kube-ovn",
        "server_socket": "/run/openvswitch/kube-ovn-daemon.sock",
        "provider": "eth1.default"
      }
    }'
---
```

```yaml
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
  name: macvlan1
spec:
  protocol: IPv4
  provider: eth1.default
  cidrBlock: 172.17.0.0/16
  gateway: 172.17.0.1
  excludeIps:
    - 172.17.0.0..172.17.0.10
```

You can create a Network Attachment Definition with any CNI plugin to access the corresponding network.

For details on how to use multi-nic, please refer to Manage Multiple Interface.

Creating a VPC Egress Gateway

Create a VPC Egress Gateway resource as shown in the example below:

```yaml
apiVersion: kubeovn.io/v1
kind: VpcEgressGateway
metadata:
  name: gateway1
  namespace: default
spec:
  vpc: ovn-cluster
  replicas: 1
  externalSubnet: macvlan1
  policies:
    - snat: true
      subnets:
        - ovn-default
```

The above resource creates a VPC Egress Gateway named gateway1 for VPC `ovn-cluster` under the default namespace, and all Pods under the `ovn-default` subnet (10.16.0.0/16) within `ovn-cluster` VPC will access the external network via the `macvlan1` subnet with SNAT applied.

After the creation is complete, check out the VPC Egress Gateway resource:

```
$ kubectl get veg gateway1
NAME       VPC           REPLICAS   BFD ENABLED   EXTERNAL SUBNET   PHASE       READY   AGE
gateway1   ovn-cluster   1          false         macvlan1          Completed   true    13s
```

To view more informations:

```
kubectl get veg gateway1 -o wide
NAME       VPC           REPLICAS   BFD ENABLED   EXTERNAL SUBNET   PHASE       READY   INTERNAL IPS     EXTERNAL IPS     WORKING NODES         AGE
gateway1   ovn-cluster   1          false         macvlan1          Completed   true    ["10.16.0.12"]   ["172.17.0.11"]   ["kube-ovn-worker"]   82s
```

To view the workload:

```
$ kubectl get deployment -l ovn.kubernetes.io/vpc-egress-gateway=gateway1
NAME       READY   UP-TO-DATE   AVAILABLE   AGE
gateway1   1/1     1            1           4m40s

$ kubectl get pod -l ovn.kubernetes.io/vpc-egress-gateway=gateway1 -o wide
NAME                        READY   STATUS    RESTARTS   AGE     IP           NODE              NOMINATED NODE   READINESS GATES
gateway1-b9f8b4448-76lhm   1/1     Running   0          4m48s   10.16.0.12   kube-ovn-worker   <none>           <none>
```

To view IP addresses, routes, and iptables rules in the Pod:

```
$ kubectl exec gateway1-b9f8b4448-76lhm -c gateway -- ip address show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: net1@if13: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether 62:d8:71:90:7b:86 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 172.17.0.11/16 brd 172.17.255.255 scope global net1
       valid_lft forever preferred_lft forever
    inet6 fe80::60d8:71ff:fe90:7b86/64 scope link
       valid_lft forever preferred_lft forever
17: eth0@if18: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1400 qdisc noqueue state UP group default
    link/ether 36:7c:6b:c7:82:6b brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.16.0.12/16 brd 10.16.255.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::347c:6bff:fec7:826b/64 scope link
       valid_lft forever preferred_lft forever

$ kubectl exec gateway1-b9f8b4448-76lhm -c gateway -- ip route show
default via 172.17.0.1 dev net1
10.16.0.0/16 dev eth0 proto kernel scope link src 10.16.0.12
172.17.0.0/16 dev net1 proto kernel scope link src 172.17.0.11

$ kubectl exec gateway1-b9f8b4448-76lhm -c gateway -- iptables -t nat -S
-P PREROUTING ACCEPT
-P INPUT ACCEPT
-P OUTPUT ACCEPT
-P POSTROUTING ACCEPT
-A POSTROUTING -s 10.16.0.0/16 -j MASQUERADE --random-fully
```

Capture packets in the Gateway Pod to verify network traffic:

```
$ kubectl exec -ti gateway1-b9f8b4448-76lhm -c gateway -- bash
nobody@gateway1-b9f8b4448-76lhm:/kube-ovn$ tcpdump -i any -nnve icmp and host 172.17.0.1
tcpdump: data link type LINUX_SLL2
tcpdump: listening on any, link-type LINUX_SLL2 (Linux cooked v2), snapshot length 262144 bytes
06:50:58.936528 eth0  In  ifindex 17 92:26:b8:9e:f2:1c ethertype IPv4 (0x0800), length 104: (tos 0x0, ttl 63, id 30481, offset 0, flags [DF], proto ICMP (1), length 84)
    10.16.0.9 > 172.17.0.1: ICMP echo request, id 37989, seq 0, length 64
06:50:58.936574 net1  Out ifindex 2 62:d8:71:90:7b:86 ethertype IPv4 (0x0800), length 104: (tos 0x0, ttl 62, id 30481, offset 0, flags [DF], proto ICMP (1), length 84)
    172.17.0.11 > 172.17.0.1: ICMP echo request, id 39449, seq 0, length 64
06:50:58.936613 net1  In  ifindex 2 02:42:39:79:7f:08 ethertype IPv4 (0x0800), length 104: (tos 0x0, ttl 64, id 26701, offset 0, flags [none], proto ICMP (1), length 84)
    172.17.0.1 > 172.17.0.11: ICMP echo reply, id 39449, seq 0, length 64
06:50:58.936621 eth0  Out ifindex 17 36:7c:6b:c7:82:6b ethertype IPv4 (0x0800), length 104: (tos 0x0, ttl 63, id 26701, offset 0, flags [none], proto ICMP (1), length 84)
    172.17.0.1 > 10.16.0.9: ICMP echo reply, id 37989, seq 0, length 64
```

Routing policies (static routes for custom VPCs) are automatically created on the OVN Logical Router:

```
$ kubectl ko nbctl lr-policy-list ovn-cluster
Routing Policies
     31000                            ip4.dst == 10.16.0.0/16           allow
     31000                           ip4.dst == 100.64.0.0/16           allow
     30000                              ip4.dst == 172.18.0.2         reroute                100.64.0.3
     30000                              ip4.dst == 172.18.0.3         reroute                100.64.0.2
     30000                              ip4.dst == 172.18.0.4         reroute                100.64.0.4
     29100                            ip4.src == 10.16.0.0/16         reroute                10.16.0.12
     29000 ip4.src == $ovn.default.kube.ovn.control.plane_ip4         reroute                100.64.0.2
     29000       ip4.src == $ovn.default.kube.ovn.worker2_ip4         reroute                100.64.0.4
     29000        ip4.src == $ovn.default.kube.ovn.worker_ip4         reroute                100.64.0.3
```

If you need to enable load balancing, modify `.spec.replicas` as shown in the following example:

```
$ kubectl scale veg gateway1 --replicas=2
vpcegressgateway.kubeovn.io/gateway1 scaled

$ kubectl get veg gateway1
NAME       VPC           REPLICAS   BFD ENABLED   EXTERNAL SUBNET   PHASE       READY   AGE
gateway1   ovn-cluster   2          false         macvlan           Completed   true    39m

$ kubectl get pod -l ovn.kubernetes.io/vpc-egress-gateway=gateway1 -o wide
NAME                       READY   STATUS    RESTARTS   AGE   IP           NODE               NOMINATED NODE   READINESS GATES
gateway1-b9f8b4448-76lhm   1/1     Running   0          40m   10.16.0.12   kube-ovn-worker    <none>           <none>
gateway1-b9f8b4448-zd4dl   1/1     Running   0          64s   10.16.0.13   kube-ovn-worker2   <none>           <none>

$ kubectl ko nbctl lr-policy-list ovn-cluster
Routing Policies
     31000                            ip4.dst == 10.16.0.0/16           allow
     31000                           ip4.dst == 100.64.0.0/16           allow
     30000                              ip4.dst == 172.18.0.2         reroute                100.64.0.3
     30000                              ip4.dst == 172.18.0.3         reroute                100.64.0.2
     30000                              ip4.dst == 172.18.0.4         reroute                100.64.0.4
     29100                            ip4.src == 10.16.0.0/16         reroute                10.16.0.12, 10.16.0.13
     29000 ip4.src == $ovn.default.kube.ovn.control.plane_ip4         reroute                100.64.0.2
     29000       ip4.src == $ovn.default.kube.ovn.worker2_ip4         reroute                100.64.0.4
     29000        ip4.src == $ovn.default.kube.ovn.worker_ip4         reroute                100.64.0.3
```

Enabling BFD-based High Availability

BFD-based high availability relies on the VPC BFD LRP function, so you need to modify the VPC resource to enable BFD Port. Here is an example:

```yaml
apiVersion: kubeovn.io/v1
kind: Vpc
metadata:
  name: vpc1
spec:
  bfdPort:
    enabled: true
    ip: 10.255.255.255
---
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
  name: subnet1
spec:
  vpc: vpc1
  protocol: IPv4
  cidrBlock: 192.168.0.0/24
```

After the BFD Port is enabled, an LRP dedicated to BFD is automatically created on the corresponding OVN LR:

```
$ kubectl ko nbctl show vpc1
router 0c1d1e8f-4c86-4d96-88b2-c4171c7ff824 (vpc1)
    port bfd@vpc1
        mac: "8e:51:4b:16:3c:90"
        networks: ["10.255.255.255"]
    port vpc1-subnet1
        mac: "de:c9:5c:38:7a:61"
        networks: ["192.168.0.1/24"]
```

After that, set `.spec.bfd.enabled` to `true` in VPC Egress Gateway. An example is shown below:

```
apiVersion: kubeovn.io/v1
kind: VpcEgressGateway
metadata:
  name: gateway2
  namespace: default
spec:
  vpc: vpc1
  replicas: 2
  internalSubnet: subnet1
  externalSubnet: macvlan
  bfd:
    enabled: true
  policies:
    - snat: true
      ipBlocks:
        - 192.168.0.0/24
```

To view VPC Egress Gateway information:

```
$ kubectl get veg gateway2 -o wide
NAME       VPC    REPLICAS   BFD ENABLED   EXTERNAL SUBNET   PHASE       READY   INTERNAL IPS                    EXTERNAL IPS                    WORKING
NODES                               AGE
gateway2   vpc1   2          true          macvlan           Completed   true    ["192.168.0.2","192.168.0.3"]   ["172.17.0.13","172.17.0.14"]   ["kube-ovn-
worker","kube-ovn-worker2"]   58s

$ kubectl get pod -l ovn.kubernetes.io/vpc-egress-gateway=gateway2 -o wide
NAME                        READY   STATUS    RESTARTS   AGE     IP            NODE               NOMINATED NODE   READINESS GATES
gateway2-fcc6b8b87-8lgvx   1/1     Running   0          2m18s   192.168.0.3   kube-ovn-worker2   <none>           <none>
gateway2-fcc6b8b87-wmww6   1/1     Running   0          2m18s   192.168.0.2   kube-ovn-worker    <none>           <none>

$ kubectl ko nbctl lr-route-list vpc1
IPv4 Routes
Route Table <main>:
           192.168.0.0/24               192.168.0.2 src-ip ecmp ecmp-symmetric-reply bfd
           192.168.0.0/24               192.168.0.3 src-ip ecmp ecmp-symmetric-reply bfd

$ kubectl ko nbctl list bfd
_uuid               : 223ede10-9169-4c7d-9524-a546e24bfab5
detect_mult         : 3
dst_ip              : "192.168.0.2"
external_ids        : {af="4", vendor=kube-ovn, vpc-egress-gateway="default/gateway2"}
logical_port        : "bfd@vpc1"
min_rx              : 1000
min_tx              : 1000
options             : {}
status              : up

_uuid               : b050c75e-2462-470b-b89c-7bd38889b758
detect_mult         : 3
dst_ip              : "192.168.0.3"
external_ids        : {af="4", vendor=kube-ovn, vpc-egress-gateway="default/gateway2"}
logical_port        : "bfd@vpc1"
min_rx              : 1000
min_tx              : 1000
options             : {}
status              : up
```

To view BFD connections:

```
$ kubectl exec gateway2-fcc6b8b87-8lgvx -c bfdd -- bfdd-control status
There are 1 sessions:
```

```
Session 1
 id=1 local=192.168.0.3 (p) remote=10.255.255.255 state=Up

$ kubectl exec gateway2-fcc6b8b87-wmww6 -c bfdd -- bfdd-control status
There are 1 sessions:
Session 1
 id=1 local=192.168.0.2 (p) remote=10.255.255.255 state=Up
```

Configuration Parameters

VPC BFD PORT

Fields	Type	Optional	Default Value	Description	Example
`enabled`	`boolean`	Yes	`false`	Whether to enable the BFD Port.	`true`
`ip`	`string`	No	-	The IP address used by the BFD Port. Must NOT conflict with other addresses. IPv4, IPv6 and dual-stack are supported.	`169.255.255.` `fdff::1` / `169.255.255.` `1`
`nodeSelector`	`object`	Yes	-	Label selector used to select nodes that carries the BFD Port work. the BFD Port binds an OVN HA Chassis Group of selected nodes and works in Active/Backup mode. If this field is not specified, Kube-OVN automatically selects up to three nodes. You can view all OVN HA Chassis Group resources by executing `kubectl ko nbctl list ha_chassis_group`.	-
`nodeSelector.matchLabels`	`dict/map`	Yes	-	A map of {key,value} pairs.	-
`nodeSelector.matchExpressions`	`object array`	Yes	-	A list of label selector requirements. The requirements are ANDed.	-

VPC EGRESS GATEWAY

Spec:

Fields	Type	Optional	Default Value	Description	Example
`vpc`	`string`	Yes	Name of the default VPC (ovn-cluster)	VPC name.	`vpc1`
`replicas`	`integer/int32`	Yes	`1`	Replicas.	`2`
`prefix`	`string`	Yes	-	Prefix of the workload deployment name. This field is immutable.	`veg-`
`image`	`string`	Yes	-	The image used by the workload deployment.	`docker.io/kubeovn/kube-ovn:v1.14.0-debug`
`internalSubnet`	`string`	Yes	Name of the default subnet within the VPC.	Name of the subnet used to access the VPC network.	`subnet1`
`externalSubnet`	`string`	No	-	Name of the subnet used to access the external network.	`ext1`
`internalIPs`	`string array`	Yes	-	IP addresses used for accessing the VPC network. IPv4, IPv6 and dual-stack are supported. The number of IPs specified must NOT be less than `replicas`. It is recommended to set the number to `<replicas> + 1` to avoid extreme cases where the Pod is not created properly.	`10.16.0.101` / `fd00::11` / `10.16.0.101,fd00::11`
`externalIPs`	`string array`	Yes	-	IP addresses used for accessing the external network. IPv4, IPv6 and dual-stack are supported. The number of IPs	`10.16.0.101` / `fd00::11` / `10.16.0.101,fd00::11`

Fields	Type	Optional	Default Value	Description	Example
				specified must NOT be less than `replicas`. It is recommended to set the number to `<replicas> + 1` to avoid extreme cases where the Pod is not created properly.	
`bfd`	`object`	Yes	-	BFD Configuration.	-
`policies`	`object array`	Yes	-	Egress policies. Configurable when `selectors` is configured.	-
`selectors`	`object array`	Yes	-	Configure Egress policies by namespace selectors and Pod selectors. SNAT/MASQUERADE will be applied to the matched Pods. Configurable when `policies` is configured.	-
`nodeSelector`	`object array`	Yes	-	Node selector applied to the workload. The workload (Deployment/Pod) will run on the selected nodes.	-
`trafficPolicy`	`string`	Yes	`Cluster`	Available values: `Cluster`/`Local`. **Effective only when BFD is enabled**. When set to `Local`, Egress traffic will be redirected to the VPC Egress	`Local`

Fields	Type	Optional	Default Value	Description	Example
				Gateway instance running on the same node if available. If the instance is down, Egress traffic will be redirected to other instances.	

BFD Configuration:

Fields	Type	Optional	Default Value	Description	Example
`enabled`	`boolean`	Yes	`false`	Whether to enable BFD.	`true`
`minRX`	`integer/int32`	Yes	`1000`	BFD minRX in milliseconds.	`500`
`minTX`	`integer/int32`	Yes	`1000`	BFD minTX in milliseconds.	`500`
`multiplier`	`integer/int32`	Yes	`3`	BFD multiplier.	`1`

Egress Policies:

Fields	Type	Optional	Default Value	Description	Example
`snat`	`boolean`	Yes	`false`	Whether to enable SNAT/ MASQUERADE.	`true`
`ipBlocks`	`string array`	Yes	-	IP range segments applied to this Gateway. Both IPv4 and IPv6 are supported.	`192.168.0.1` / `192.168.0.0/24`
`subnets`	`string array`	Yes	-	The VPC subnet name applied to this Gateway. IPv4, IPv6 and dual-stack subnets are supported.	`subnet1`

Selectors:

Fields	Type	Optional	Default Value	Description	Example
`namespaceSelector`	`object`	Yes	-	Namespace selector. An empty label selector matches all namespaces.	-
`namespaceSelector.matchLabels`	`dict/map`	Yes	-	A map of {key,value} pairs.	-
`namespaceSelector.matchExpressions`	`object array`	Yes	-	A list of label selector requirements. The requirements are ANDed.	-
`podSelector`	`object`	Yes	-	Pod selector. An empty label selector matches all Pods.	-
`podSelector.matchLabels`	`dict/map`	Yes	-	A map of {key,value} pairs.	-
`podSelector.matchExpressions`	`object array`	Yes	-	A list of label selector requirements. The requirements are ANDed.	-

Node selector:

Fields	Type	Optional	Default Value	Description	Example
`matchLabels`	`dict/map`	Yes	-	A map of {key,value} pairs.	-
`matchExpressions`	`object array`	Yes	-	A list of label selector requirements. The requirements are ANDed.	-
`matchFields`	`object array`	Yes	-	A list of field selector requirements. The requirements are ANDed.	-

Status:

Fields	Type	Description	Example
`ready`	`boolean`	Whether the gateway is ready.	`true`
`phase`	`string`	The gateway processing phase.	`Pending` / `Processing` / `Completed`
`internalIPs`	`string array`	IP addresses used to access the VPC network.	-
`externalIPs`	`string array`	IP addresses used to access the external network.	-
`workload`	`object`	Workload information.	-
`workload.apiVersion`	`string`	Workload API version.	`apps/v1`
`workload.kind`	`string`	Workload kind.	`Deployment`
`workload.name`	`string`	Workload name.	`gateway1`
`workload.nodes`	`string array`	Names of the nodes where the workload resides.	-
`conditions`	`object array`	-	-

PDF **Slack** **Support**

July 29, 2025

December 12, 2024

GitHub

5.2.4 Comments

5.3 VPC QoS

Kube-OVN supports using QoSPolicy CRD to limit the traffic rate of custom VPC.

5.3.1 EIP QoS

Limit the speed of EIP to 1Mbps and the priority to 1, and `shared=false` here means that this QoSPolicy can only be used for this EIP and support dynamically modifying QoSPolicy to change QoS rules.

The QoSPolicy configuration is as follows:

```yaml
apiVersion: kubeovn.io/v1
kind: QoSPolicy
metadata:
  name: qos-eip-example
spec:
  shared: false
  bindingType: EIP
  bandwidthLimitRules:
  - name: eip-ingress
    rateMax: "1" # Mbps
    burstMax: "1" # Mbps
    priority: 1
    direction: ingress
  - name: eip-egress
    rateMax: "1" # Mbps
    burstMax: "1" # Mbps
    priority: 1
    direction: egress
```

The IptablesEIP configuration is as follows:

```yaml
kind: IptablesEIP
apiVersion: kubeovn.io/v1
metadata:
  name: eip-1
spec:
  natGwDp: gw1
  qosPolicy: qos-eip-example
```

The value of `.spec.qosPolicy` supports being specified during creation and also supports modification after creation.

5.3.2 View EIPs with QoS enabled

View the corresponding EIPs that have been set up using `label`:

```
# kubectl get eip  -l ovn.kubernetes.io/qos=qos-eip-example
NAME    IP             MAC                 NAT   NATGWDP   READY
eip-1   172.18.11.24   00:00:00:34:41:0B   fip   gw1       true
```

5.3.3 QoS for VPC NATGW net1 NIC

Limit the speed of the net1 NIC on VPC NATGW to 10Mbps and set the priority to 3. Here `shared=true`, which means that this QoSPolicy can be used by multiple resources at the same time, and does not allow the modification of the contents of the QoSPolicy in this scenario.

The QoSPolicy configuration is as follows:

```yaml
apiVersion: kubeovn.io/v1
kind: QoSPolicy
metadata:
  name: qos-natgw-example
spec:
  shared: true
  bindingType: NATGW
  bandwidthLimitRules:
  - name: net1-ingress
    interface: net1
    rateMax: "10" # Mbps
    burstMax: "10" # Mbps
    priority: 3
    direction: ingress
  - name: net1-egress
    interface: net1
    rateMax: "10" # Mbps
    burstMax: "10" # Mbps
    priority: 3
    direction: egress
```

The VpcNatGateway configuration is as follows:

```yaml
kind: VpcNatGateway
apiVersion: kubeovn.io/v1
metadata:
  name: gw1
spec:
  vpc: test-vpc-1
  subnet: net1
  lanIp: 10.0.1.254
  qosPolicy: qos-natgw-example
  selector:
    - "kubernetes.io/hostname: kube-ovn-worker"
    - "kubernetes.io/os: linux"
```

The value of `.spec.qosPolicy` supports both creation and subsequent modification.

5.3.4 QoS for specific traffic on net1 NIC

Limit the specific traffic on net1 NIC to 5Mbps and set the priority to 2. Here `shared=true`, which means that this QoSPolicy can be used by multiple resources at the same time, and does not allow the modification of the contents of the QoSPolicy in this scenario.

The QoSPolicy configuration is as follows:

```yaml
apiVersion: kubeovn.io/v1
kind: QoSPolicy
metadata:
  name: qos-natgw-example
spec:
  shared: true
  bindingType: NATGW
  bandwidthLimitRules:
  - name: net1-extip-ingress
    interface: net1
    rateMax: "5" # Mbps
    burstMax: "5" # Mbps
    priority: 2
    direction: ingress
    matchType: ip
    matchValue: src 172.18.11.22/32
  - name: net1-extip-egress
    interface: net1
    rateMax: "5" # Mbps
    burstMax: "5" # Mbps
    priority: 2
    direction: egress
    matchType: ip
    matchValue: dst 172.18.11.23/32
```

The VpcNatGateway configuration is as follows:

```yaml
kind: VpcNatGateway
apiVersion: kubeovn.io/v1
metadata:
  name: gw1
spec:
  vpc: test-vpc-1
  subnet: net1
  lanIp: 10.0.1.254
  qosPolicy: qos-natgw-example
  selector:
    - "kubernetes.io/hostname: kube-ovn-worker"
    - "kubernetes.io/os: linux"
```

5.3.5 View NATGWs with QoS enabled

View the corresponding NATGWs that have been set up using `label`:

```
# kubectl get vpc-nat-gw  -l ovn.kubernetes.io/qos=qos-natgw-example
NAME   VPC          SUBNET   LANIP
gw1    test-vpc-1   net1     10.0.1.254
```

5.3.6 View QoS rules

```
# kubectl get qos -A
NAME                 SHARED   BINDINGTYPE
qos-eip-example      false    EIP
qos-natgw-example    true     NATGW
```

5.3.7 Limitations

- QoSPolicy can only be deleted when it is not in use. Therefore, before deleting the QoSPolicy, please check the EIP and NATGW that have enabled QoS, and remove their `spec.qosPolicy` configuration.

5.3.8 Comments

5.4 Customize VPC Internal Load Balancing

The Service provided by Kubernetes can be used for load balancing within the cluster. However, there are several issues with using Service as internal load balancing in customize VPC mode:

1. The Service IP range is a cluster resource, shared by all customize VPCs, and cannot overlap.
2. Users cannot set internal load balancing IP addresses according to their own preferences.

To address the above issues, Kube OVN introduced the `SwitchLBRule` CRD in 1.11, allowing users to set internal load balancing rules within customize VPCs.

`SwitchLBRule` support the following two ways to set internal load balancing rules within a customize VPC.

5.4.1 Automatically Generate Load Balancing Rules by `Selector`

Load balancing rules can be generated by `selector` automatic association with `pod` configuration through `label`.

example of `SwitchLBRule` is as follows:

```
apiVersion: kubeovn.io/v1
kind: SwitchLBRule
metadata:
  name:  cjh-slr-nginx
spec:
  vip: 1.1.1.1
  sessionAffinity: ClientIP
  namespace: default
  selector:
    - app:nginx
  ports:
  - name: dns
    port: 8888
    targetPort: 80
    protocol: TCP
```

- usage of `selector`, `sessionAffinity`, and `port` is the same as Kubernetes Service.
- `vip`: customize load balancing IP address.
- `namespace`: namespace of the `pod` selected by `selector`.

Kube OVN will determine the VPC of the selected `pod` based on the `SwitchLBRule` definition and set the corresponding L2 LB.

5.4.2 Manually Defined Load Balancing Rules by `Endpoints`

Load balancing rules can be customized configured by `endpoints`, to support scenarios where load balancing rules cannot be automatically generated through `selector`. For example, the load balancing backend is `vm` created by `kubevirt`.

example of `SwitchLBRule` is as follows:

```
apiVersion: kubeovn.io/v1
kind: SwitchLBRule
metadata:
  name:  cjh-slr-nginx
spec:
  vip: 1.1.1.1
  sessionAffinity: ClientIP
  namespace: default
  endpoints:
    - 192.168.0.101
    - 192.168.0.102
    - 192.168.0.103
  ports:
  - name: dns
    port: 8888
    targetPort: 80
    protocol: TCP
```

usage of `sessionAffinity`, and `port` is the same as Kubernetes Service.

- `vip`:customize load balancing IP address.
- `namespace`:namespace of the `pod` selected by `selector`.
- `endpoints`:load balancing backend IP list.

If both `selector` and `endpoints` are configured, the `selector` configuration will be automatically ignored.

5.4.3 Health Check

`OVN` supports health checks for load balancer endpoints, for IPv4 load balancers only. When health checks are enabled, the load balancer uses only healthy endpoints.

[Health Checks](https://www.ovn.org/support/dist-docs/ovn-nb.5.html)

Add a health check to `SwitchLBRule` based on the health check of the `ovn` load balancer.While creating the `SwitchLBRule`, obtain a reusable `vip` from the corresponding `VPC` and `subnet` as the detection endpoint and associate the corresponding `IP_Port_Mappings` and `Load_Balancer_Health_Check` to the corresponding load balancer.

- The detection endpoint `vip` will be automatically determined whether it exists in the corresponding `subnet` with the same name of the `subnet`. If it does not exist, it will be automatically created and deleted after all associated `SwitchLBRule` are deleted.
- Currently, only `SwitchLBRule` automatically generated through `Selector` are supported.

Create `SwitchLBRule`

```
root@server:~# kubectl get po -o wide -n vulpecula
NAME                     READY   STATUS    RESTARTS   AGE     IP          NODE     NOMINATED NODE   READINESS GATES
nginx-78d9578975-f4qn4   1/1     Running   3          4d16h   10.16.0.4   worker   <none>           <none>
nginx-78d9578975-t8tm5   1/1     Running   3          4d16h   10.16.0.6   worker   <none>           <none>

# create slr

root@server:~# cat << END > slr.yaml
apiVersion: kubeovn.io/v1
kind: SwitchLBRule
metadata:
  name:  nginx
  namespace:  vulpecula
spec:
  vip: 1.1.1.1
  sessionAffinity: ClientIP
  namespace: default
  selector:
    - app:nginx
  ports:
  - name: dns
    port: 8888
    targetPort: 80
    protocol: TCP
END

root@server:~# kubectl apply -f slr.yaml
root@server:~# kubectl get slr
NAME              VIP       PORT(S)    SERVICE                       AGE
vulpecula-nginx   1.1.1.1   8888/TCP   default/slr-vulpecula-nginx   3d21h
```

The `vip` with the same name of the `subnet` has been created.

```
# vip for check

root@server:~# kubectl get vip
NAME             NS    V4IP        MAC                 V6IP    PMAC   SUBNET           READY   TYPE
vulpecula-subnet     10.16.0.2   00:00:00:39:95:C1   <nil>          vulpecula-subnet   true
```

Query the `Load_Balancer_Health_Check` and `Service_Monitor` by commands.

```
root@server:~# kubectl ko nbctl list Load_Balancer
_uuid               : 3cbb6d43-44aa-4028-962f-30d2dba9f0b8
external_ids        : {}
health_check        : [5bee3f12-6b54-411c-9cc8-c9def8f67356]
ip_port_mappings    : {"10.16.0.4"="nginx-78d9578975-f4qn4.default:10.16.0.2", "10.16.0.6"="nginx-78d9578975-t8tm5.default:10.16.0.2"}
name                : cluster-tcp-session-loadbalancer
options             : {affinity_timeout="10800"}
protocol            : tcp
```

```
selection_fields    : [ip_src]
vips                : {"1.1.1.1:8888"="10.16.0.4:80,10.16.0.6:80"}

root@server:~# kubectl ko nbctl list Load_Balancer_Health_Check
_uuid               : 5bee3f12-6b54-411c-9cc8-c9def8f67356
external_ids        : {switch_lb_subnet=vulpecula-subnet}
options             : {failure_count="3", interval="5", success_count="3", timeout="20"}
vip                 : "1.1.1.1:8888"

root@server:~# kubectl ko sbctl list Service_Monitor
_uuid               : 1bddc541-cc49-44ea-9935-a4208f627a91
external_ids        : {}
ip                  : "10.16.0.4"
logical_port        : nginx-78d9578975-f4qn4.default
options             : {failure_count="3", interval="5", success_count="3", timeout="20"}
port                : 80
protocol            : tcp
src_ip              : "10.16.0.2"
src_mac             : "c6:d4:b8:08:54:e7"
status              : online

_uuid               : 84dd24c5-e1b4-4e97-9daa-13687ed59785
external_ids        : {}
ip                  : "10.16.0.6"
logical_port        : nginx-78d9578975-t8tm5.default
options             : {failure_count="3", interval="5", success_count="3", timeout="20"}
port                : 80
protocol            : tcp
src_ip              : "10.16.0.2"
src_mac             : "c6:d4:b8:08:54:e7"
status              : online
```

At this point, the service response can be successfully obtained through load balancer `vip`.

```
root@server:~# kubectl exec -it -n vulpecula nginx-78d9578975-t8tm5 -- curl 1.1.1.1:8888
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

Update load balance service endpoints

Update the service endpoints of the load balancer by deleting the `pod`.

```
kubectl delete po nginx-78d9578975-f4qn4
kubectl get po -o wide -n vulpecula
NAME                     READY   STATUS    RESTARTS   AGE     IP          NODE     NOMINATED NODE   READINESS GATES
nginx-78d9578975-lxmvh   1/1     Running   0          31s     10.16.0.8   worker   <none>           <none>
nginx-78d9578975-t8tm5   1/1     Running   3          4d16h   10.16.0.6   worker   <none>           <none>
```

Query the `Load_Balancer_Health_Check` and `Service_Monitor` by commands, the results have undergone corresponding changes.

```
root@server:~# kubectl ko nbctl list Load_Balancer
_uuid               : 3cbb6d43-44aa-4028-962f-30d2dba9f0b8
external_ids        : {}
health_check        : [5bee3f12-6b54-411c-9cc8-c9def8f67356]
ip_port_mappings    : {"10.16.0.4"="nginx-78d9578975-f4qn4.default:10.16.0.2", "10.16.0.6"="nginx-78d9578975-t8tm5.default:10.16.0.2",
"10.16.0.8"="nginx-78d9578975-lxmvh.default:10.16.0.2"}
name                : cluster-tcp-session-loadbalancer
options             : {affinity_timeout="10800"}
protocol            : tcp
selection_fields    : [ip_src]
vips                : {"1.1.1.1:8888"="10.16.0.6:80,10.16.0.8:80"}

root@server:~# kubectl ko nbctl list Load_Balancer_Health_Check
_uuid               : 5bee3f12-6b54-411c-9cc8-c9def8f67356
external_ids        : {switch_lb_subnet=vulpecula-subnet}
options             : {failure_count="3", interval="5", success_count="3", timeout="20"}
vip                 : "1.1.1.1:8888"

root@server:~# kubectl ko sbctl list Service_Monitor
_uuid               : 84dd24c5-e1b4-4e97-9daa-13687ed59785
external_ids        : {}
ip                  : "10.16.0.6"
logical_port        : nginx-78d9578975-t8tm5.default
options             : {failure_count="3", interval="5", success_count="3", timeout="20"}
port                : 80
protocol            : tcp
src_ip              : "10.16.0.2"
src_mac             : "c6:d4:b8:08:54:e7"
status              : online

_uuid               : 5917b7b7-a999-49f2-a42d-da81f1eeb28f
external_ids        : {}
```

```
ip                 : "10.16.0.8"
logical_port       : nginx-78d9578975-lxmvh.default
options            : {failure_count="3", interval="5", success_count="3", timeout="20"}
port               : 80
protocol           : tcp
src_ip             : "10.16.0.2"
src_mac            : "c6:d4:b8:08:54:e7"
status             : online
```

Delete `SwitchLBRule` and confirm the resource status, `Load_Balancer_Health_Check` adn `Service_Monitor` has been deleted, and the corresponding `vip` has also been deleted.

```
root@server:~# kubectl delete -f slr.yaml
switchlbrule.kubeovn.io "vulpecula-nginx" deleted
root@server:~# kubectl get vip
No resources found
root@server:~# kubectl ko sbctl list Service_Monitor
root@server:~#
root@server:~# kubectl ko nbctl list Load_Balancer_Health_Check
root@server:~#
```

PDF Slack Support

July 30, 2025

May 18, 2023

GitHub

5.4.4 Comments

5.5 Custom VPC Internal DNS

Due to the isolation of the user-defined VPC and the default VPC network, the coredns deployed in the default VPC cannot be accessed from within the custom VPC. If you wish to use the intra-cluster domain name resolution capability provided by Kubernetes within your custom VPC, you can refer to this document and utilize the vpc-dns CRD to do so.

This CRD eventually deploys a coredns that has two NICs, one in the user-defined VPC and the other in the default VPC to enable network interoperability and provide an internal load balancing within the custom VPC through the custom VPC internal load balancing.

Note

This DNS address **will not** be automatically injected into Pods or VMs. Users need to modify the content of `/etc/resolv.conf` through Webhook or VM image templates.

5.5.1 Deployment of vpc-dns dependent resources

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    kubernetes.io/bootstrapping: rbac-defaults
  name: system:vpc-dns
rules:
  - apiGroups:
    - ""
    resources:
    - endpoints
    - services
    - pods
    - namespaces
    verbs:
    - list
    - watch
  - apiGroups:
    - discovery.k8s.io
    resources:
    - endpointslices
    verbs:
    - list
    - watch
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  annotations:
    rbac.authorization.kubernetes.io/autoupdate: "true"
  labels:
    kubernetes.io/bootstrapping: rbac-defaults
  name: vpc-dns
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: system:vpc-dns
subjects:
- kind: ServiceAccount
  name: vpc-dns
  namespace: kube-system
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: vpc-dns
  namespace: kube-system
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: vpc-dns-corefile
  namespace: kube-system
data:
  Corefile: |
    .:53 {
        errors
        health {
          lameduck 5s
        }
        ready
        kubernetes cluster.local in-addr.arpa ip6.arpa {
          pods insecure
          fallthrough in-addr.arpa ip6.arpa
        }
        prometheus :9153
        forward . /etc/resolv.conf {
          prefer_udp
        }
        cache 30
        loop
        reload
        loadbalance
    }
```

In addition to the above resources, the feature relies on the nat-gw-pod image for routing configuration.

5.5.2 Configuring Additional Network

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: ovn-nad
  namespace: default
spec:
  config: '{
      "cniVersion": "0.3.0",
      "type": "kube-ovn",
      "server_socket": "/run/openvswitch/kube-ovn-daemon.sock",
      "provider": "ovn-nad.default.ovn"
    }'
```

5.5.3 Configuring Configmap for vpc-dns

Create a configmap under the kube-system namespace to configure the vpc-dns usage parameters that will be used later to start the vpc-dns function:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: vpc-dns-config
  namespace: kube-system
data:
  coredns-vip: 10.96.0.3
  enable-vpc-dns: "true"
  nad-name: ovn-nad
  nad-provider: ovn-nad.default.ovn
```

- `enable-vpc-dns` : enable vpc dns feature, true as default
- `coredns-image` : dns deployment image. Defaults to the clustered coredns deployment version
- `coredns-vip` : the vip that provides lb services for coredns.
- `coredns-template` : the URL where the coredns deployment template is located. defaults to the current version of the ovn directory. `coredns-template.yaml` default is `https://raw.githubusercontent.com/kubeovn/kube-ovn/<kube-ovn version>/yamls/coredns-template.yaml` .
- `nad-name` : configured network-attachment-definitions Resource name.
- `nad-provider` : the name of the provider to use.
- `k8s-service-host` : the ip used for coredns to access the k8s apiserver service, defaults to the apiserver address within the cluster.
- `k8s-service-port` : the port used for coredns to access the k8s apiserver service, defaults to the apiserver port within the cluster.

5.5.4 Deploying vpc-dns

configure vpc-dns yaml:

```
kind: VpcDns
apiVersion: kubeovn.io/v1
metadata:
  name: test-cjh1
spec:
  vpc: cjh-vpc-1
  subnet: cjh-subnet-1
  replicas: 2
```

- `vpc` : the name of the vpc used to deploy the dns component.
- `subnet` : subnet name for deploying dns components.
- `replicas` : vpc dns deployment replicas

View information about deployed resources:

```
# kubectl get vpc-dns
NAME        ACTIVE   VPC         SUBNET
test-cjh1   false    cjh-vpc-1   cjh-subnet-1
test-cjh2   true     cjh-vpc-1   cjh-subnet-2
```

ACTIVE : true Customized dns component deployed, false No deployment.

Restrictions: only one custom dns component will be deployed under a VPC

- When multiple vpc-dns resources are configured under a VPC (i.e., different subnets for the same VPC), only one vpc-dns resource is in the state `true``, and the others are`fasle`.
- When the `true` vpc-dns is removed, the other `false` vpc-dns will be obtained for deployment.

5.5.5 Validate deployment results

To view vpc-dns Pod status, use label `app=vpc-dns` to view all vpc-dns Pod status:

```
# kubectl -n kube-system get pods -l app=vpc-dns
NAME                                 READY   STATUS    RESTARTS   AGE
vpc-dns-test-cjh1-7b878d96b4-g5979   1/1     Running   0          28s
vpc-dns-test-cjh1-7b878d96b4-ltmf9   1/1     Running   0          28s
```

View switch lb rule status information:

```
# kubectl -n kube-system get slr
NAME                VIP         PORT(S)                  SERVICE                             AGE
vpc-dns-test-cjh1   10.96.0.3   53/UDP,53/TCP,9153/TCP   kube-system/slr-vpc-dns-test-cjh1   113s
```

Go to the Pod under this VPC and test the dns resolution:

```
nslookup kubernetes.default.svc.cluster.local 10.96.0.3
```

The subnet where the switch lb rule under this VPC is located and the Pods under other subnets under the same VPC can be resolved.

PDF **Slack** **Support**

July 31, 2025

October 8, 2023

GitHub

5.5.6 Comments

5.6 SecurityGroup Usage

Kube-OVN has support for the configuration of security-groups through the SecurityGroup CRD.

Kube-OVN also supports **port security** to prevent MAC and IP spoofing by allowing only L2/L3 source addresses matching the ones allocated by the IPAM.

5.6.1 SecurityGroup Example

```yaml
apiVersion: kubeovn.io/v1
kind: SecurityGroup
metadata:
  name: sg-example
spec:
  allowSameGroupTraffic: true
  egressRules:
  - ipVersion: ipv4
    policy: allow
    priority: 1
    protocol: all
    remoteAddress: 10.16.0.13 # 10.16.0.0/16 Configure network segment
    remoteType: address
  ingressRules:
  - ipVersion: ipv4
    policy: deny
    priority: 1
    protocol: icmp
    remoteAddress: 10.16.0.14
    remoteType: address
```

The specific meaning of each field of the SecurityGroup can be found in the Kube-OVN API Reference.

Pods bind security-groups by adding annotations, two annotations are used:

- `port_security`: Source address verification. If this function is enabled, only packets with L2/L3 addresses assigned by Kube-OVN's IPAM can be exported from the pod network adapter. After this function is disabled, any L2/L3 address can be exported.
- security_groups: indicates a security group that contains a series of ACL rules
- When configuring a security group, the `priority` value ranges from 1 to 200, with smaller values indicating higher priority. When implementing a security group through ACLs, the security group's priority is mapped to the ACL priority. The specific mapping relationship is as follows: ACL priority=2300−Security group priority, therefore, it is essential to distinguish between the priorities of security groups and subnet ACLs.

These two annotations are responsible for functions that are independent of each other.

```yaml
    ovn.kubernetes.io/port_security: "true"
    ovn.kubernetes.io/security_groups: sg-example
```

5.6.2 Caution

- Security-groups are finally restricted by setting ACL rules, and as mentioned in the OVN documentation, if two ACL rules match with the same priority, it is uncertain which ACL will actually work. Therefore, when setting up security-group rules, you need to be careful to differentiate the priority.
- When adding a security-group, it is important to know what restrictions are being added. As a CNI, Kube-OVN will perform a Pod-to-Gateway connectivity test after creating a Pod.

5.6.3 Actual test

Create a Pod using the following YAML, and specify the security-group in the annotation for the pod.

```yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    app: static
  annotations:
    ovn.kubernetes.io/port_security: 'true'
    ovn.kubernetes.io/security_groups: 'sg-example'
  name: sg-test-pod
  namespace: default
spec:
  nodeName: kube-ovn-worker
  containers:
  - image: docker.io/library/nginx:alpine
    imagePullPolicy: IfNotPresent
    name: qatest
```

The actual test results show as follows:

```
# kubectl get pod -o wide
NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES
sg-test-pod 0/1 ContainerCreating 0 5h32m <none> kube-ovn-worker <none> <none>
test-99fff7f86-52h9r 1/1 Running 0 5h41m 10.16.0.14 kube-ovn-control-plane <none> <none>
test-99fff7f86-qcgjw 1/1 Running 0 5h43m 10.16.0.13 kube-ovn-worker <none> <none>
```

Execute `kubectl describe pod` to see information about the pod, and you can see the error message:

```
# kubectl describe pod sg-test-pod
Name: sg-test-pod
Namespace: default
Priority: 0
Node: kube-ovn-worker/172.18.0.2
Start Time: Tue, 28 Feb 2023 10:29:36 +0800
Labels: app=static
Annotations: ovn.kubernetes.io/allocated: true
              ovn.kubernetes.io/cidr: 10.16.0.0/16
              ovn.kubernetes.io/gateway: 10.16.0.1
              ovn.kubernetes.io/ip_address: 10.16.0.15
              ovn.kubernetes.io/logical_router: ovn-cluster
              ovn.kubernetes.io/logical_switch: ovn-default
              ovn.kubernetes.io/mac_address: 00:00:00:FA:17:97
              ovn.kubernetes.io/pod_nic_type: veth-pair
              ovn.kubernetes.io/port_security: true
              ovn.kubernetes.io/routed: true
              ovn.kubernetes.io/security_groups: sg-allow-reject
Status: Pending
IP:
IPs: <none>
-
- -
- -
Events:
  Type Reason Age From Message
  ---- ------ ---- ---- -------
  Warning FailedCreatePodSandBox 5m3s (x70 over 4h59m) kubelet (combined from similar events): Failed to create pod sandbox: rpc error: code = Unknown desc =
failed to setup network for sandbox "40636e0c7f1ade5500fa958486163d74f2e2300051a71522a9afd7ba0538afb6": plugin type="kube-ovn" failed ( add): RPC failed;
request ip return 500 configure nic failed 10.16.0.15 network not ready after 200 ping 10.16.0.1
```

Modify the rules for the security group to add access rules to the gateway, refer to the following:

```
apiVersion: kubeovn.io/v1
kind: SecurityGroup
metadata:
  name: sg-gw-both
spec:
  allowSameGroupTraffic: true
  egressRules:
  - ipVersion: ipv4
    policy: allow
    priority: 2
    protocol: all
    remoteAddress: 10.16.0.13
    remoteType: address
  - ipVersion: ipv4
    policy: allow
    priority: 1
    protocol: all
    remoteAddress: 10.16.0.1
    remoteType: address
  ingressRules:
  - ipVersion: ipv4
    policy: deny
    priority: 2
    protocol: icmp
    remoteAddress: 10.16.0.14
    remoteType: address
  - ipVersion: ipv4
    policy: allow
    priority: 1
    protocol: icmp
    remoteAddress: 10.16.0.1
    remoteType: address
```

In the inbound and outbound rules respectively, add a rule to allow access to the gateway, and set the rule to have the highest priority.

Deploying with the following yaml to bind security group, confirm that the Pod is operational:

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    app: static
  annotations:
    ovn.kubernetes.io/port_security: 'true'
    ovn.kubernetes.io/security_groups: 'sg-gw-both'
  name: sg-gw-both
  namespace: default
spec:
  nodeName: kube-ovn-worker
  containers:
  - image: docker.io/library/nginx:alpine
    imagePullPolicy: IfNotPresent
    name: qatest
```

To view Pod information after deployment:

```
# kubectl get pod -o wide
NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES
sg-test-pod 0/1 ContainerCreating 0 5h41m <none> kube-ovn-worker <none> <none>
sg-gw-both 1/1 Running 0 5h37m 10.16.0.19 kube-ovn-worker <none> <none>
```

So for the use of security groups, be particularly clear about the effect of the added restriction rules. If it is simply to restrict traffic access, consider using a network policy instead.

PDF Slack Support

July 30, 2025

February 28, 2023

GitHub

5.6.4 Comments

5.7 Support OVN EIP,FIP and SNAT

Support the use of any number of `provider-network vlan (external) subnet` resources by any VPC OVN NAT function, which is independent of the default VPC EIP/SNAT function.

5.7.1 Two independent ways of use

- `default external network`: If only one external network is needed, the startup parameters need to be specified in `kube-ovn-controller` and `kube-ovn-cni`. Then use this default external subnet through the `ovn-external-gw-config` or `VPC spec enableExternal` attribute.
- `CRD`: Create the `provider-network` `vlan` `subnet` resources, and then use any external subnets by `VPC spec extraExternalSubnets`, and then use `ovn-eip, ovn-dnat, ovn-fip, ovn-snat`.

```
graph LR

pod-->subnet-->vpc-->lrp--bind-->gw-chassis-->snat-->lsp-->external-subnet
lrp-.-peer-.-lsp
```

The pod access the public network based on the snat

Pod uses a centralized gateway based on Fip, and the path is similar.

```
graph LR


pod-->subnet-->vpc-->lrp--bind-->local-chassis-->snat-->lsp-->external-subnet


lrp-.-peer-.-lsp
```

Pod is based on the general flow of distributed gateway FIP (dnat_and_snat) to exit the public network. Finally, POD can exit the public network based on the public network NIC of the local node.

The CRD supported by this function is basically the same as the iptables nat gw public network solution.

- ovn eip: occupies a public ip address and is allocated from the underlay provider network vlan subnet
- ovn fip: one-to-one dnat snat, which provides direct public network access for ip addresses and vip in a VPC
- ovn snat: a subnet cidr or a single VPC ip or vip can access public networks based on snat
- ovn dnat: based router lb, which enables direct access to a group of endpoints in a VPC based on a public endpoint

5.7.2 1. Deployment

If the user selects the `default external network` mode for use:

During the deployment phase, you may need to specify a default public network logical switch based on actual conditions. If no vlan is in use (vlan 0), the following startup vlan id do not need to be configured.

```
# When deploying you need to refer to the above scenario and specify the following parameters as needed according to the actual situation
# 1. kube-ovn-controller Startup parameters to be configured
          - --external-gateway-vlanid=204
          - --external-gateway-switch=external204

# 2. kube-ovn-cni Startup parameters to be configured:
          - --external-gateway-switch=external204

# The above configuration is consistent with the following public network configuration vlan id and resource name,
# currently only support to specify one underlay public network as the default external public network.
```

The design and use of this configuration item takes into account the following factors:

- Based on this configuration item can be docked to the provider network, vlan, subnet resources.
- Based on this configuration item, the default VPC enable_eip_snat function can be docked to the existing vlan, subnet resources, while supporting the ipam
- If only the default VPC's enable_eip_snat mode is used with the old pod annotation based eip fip snat, then the following configuration is not required.
- Based on this configuration you can not use the default VPC enable_eip_snat process, only by corresponding to vlan, subnet process, can be compatible with only custom VPC use eip snat usage scenarios.

The neutron ovn mode also has a certain static file configuration designation that is, for now, generally consistent.

1.1 Create the underlay public network

```
# provider-network,  vlan,  subnet
# cat 01-provider-network.yaml

apiVersion: kubeovn.io/v1
kind: ProviderNetwork
metadata:
  name: external204
spec:
  defaultInterface: vlan

# cat 02-vlan.yaml

apiVersion: kubeovn.io/v1
kind: Vlan
metadata:
  name: vlan204
spec:
  id: 204
  provider: external204

# cat 03-vlan-subnet.yaml

apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
  name: external204
spec:
  protocol: IPv4
  cidrBlock: 10.5.204.0/24
  gateway: 10.5.204.254
  vlan: vlan204
  excludeIps:
  - 10.5.204.1..10.5.204.100
```

1.2 Default VPC enable eip_snat

```
# Enable the default VPC and the above underlay public provider subnet interconnection

cat 00-centralized-external-gw-no-ip.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: ovn-external-gw-config
  namespace: kube-system
data:
  enable-external-gw: "true"
  external-gw-nodes: "pc-node-1,pc-node-2,pc-node-3"
  type: "centralized"
  external-gw-nic: "vlan"
  external-gw-addr: "10.5.204.254/24"
```

This feature currently supports the ability to create lrp type ovn eip resources without specifying the lrp ip and mac, which is already supported for automatic acquisition. If specified, it is equivalent to specifying the ip to create an ovn-eip of type lrp. Of course, you can also manually create the lrp type ovn eip in advance.

1.3 Custom VPC enable eip snat fip function

Clusters generally require multiple gateway nodes to achieve high availability. The configuration is as follows:

```
# First specify external-gw-nodes by adding label
kubectl label nodes pc-node-1 pc-node-2 pc-node-3 ovn.kubernetes.io/external-gw=true
```

```
# cat 00-ns.yml

apiVersion: v1
kind: Namespace
metadata:
  name: vpc1

# cat 01-vpc-ecmp-enable-external-bfd.yml

kind: Vpc
apiVersion: kubeovn.io/v1
metadata:
  name: vpc1
spec:
  namespaces:
  - vpc1
  enableExternal: true
  staticRoutes:
  - cidr: 0.0.0.0/0
    nextHopIP: 10.5.204.254
    policy: policyDst

# VPC enableExternal will automatically create an lrp association to the default public network specified above

# cat 02-subnet.yml
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
  name: vpc1-subnet1
spec:
  cidrBlock: 192.168.0.0/24
  default: false
  disableGatewayCheck: false
  disableInterConnection: true
  enableEcmp: true
  gatewayNode: ""
  gatewayType: distributed
  #gatewayType: centralized
  natOutgoing: false
  private: false
  protocol: IPv4
  provider: ovn
  vpc: vpc1
  namespaces:
  - vpc1
```

After the above template is applied, you should see the following resources exist

```
# kubectl ko nbctl show vpc1

router 87ad06fd-71d5-4ff8-a1f0-54fa3bba1a7f (vpc1)
    port vpc1-vpc1-subnet1
        mac: "00:00:00:ED:8E:C7"
        networks: ["192.168.0.1/24"]
    port vpc1-external204
        mac: "00:00:00:EF:05:C7"
        networks: ["10.5.204.105/24"]
        gateway chassis: [7cedd14f-265b-42e5-ac17-e03e7a1f2342 276baccb-fe9c-4476-b41d-05872a94976d fd9f140c-c45d-43db-a6c0-0d4f8ea298dd]
    nat 21d853b0-f7b4-40bd-9a53-31d2e2745739
        external ip: "10.5.204.115"
        logical ip: "192.168.0.0/24"
        type: "snat"
```

```
# kubectl ko nbctl lr-route-list vpc1

IPv4 Routes
Route Table <main>:
                0.0.0.0/0              10.5.204.254 dst-ip

# Please configure this default route in the VPC CRD definition
```

Note: Considering that enableExternal supports multiple external networks and it is impossible to determine which external network uses which route, automatic maintenance of public network routes is currently not supported. Users can specify policy routes or static routes through the VPC CRD definition

1.4 Use additional public network

1.4.1 CREATE ADDITIONAL UNDERLAY PUBLIC NETWORK

Additional public network functions will be enabled after the default eip snat fip function is enabled. If there is only 1 public network card, please use the default eip snat fip function.

```
# provider-network, vlan, subnet
# cat 01-extra-provider-network.yaml
apiVersion: kubeovn.io/v1
kind: ProviderNetwork
metadata:
  name: extra
spec:
  defaultInterface: vlan
# cat 02-extra-vlan.yaml
apiVersion: kubeovn.io/v1
kind: Vlan
metadata:
  name: vlan0
spec:
  id: 0
  provider: extra
# cat 03-extra-vlan-subnet.yaml
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
  name: extra
spec:
  protocol: IPv4
  cidrBlock: 10.10.204.0/24
  gateway: 10.10.204.254
  vlan: vlan0
  excludeIps:
  - 10.10.204.1..10.10.204.100
```

1.4.2 CUSTOM VPC CONFIGURATION

```
apiVersion: kubeovn.io/v1
kind: Vpc
metadata:
  name: vpc1
spec:
  namespaces:
  - vpc1
  enableExternal: true  # VPC enableExternal will automatically create an lrp association to the default external network specified above
  extraExternalSubnets: # configure extraExternalSubnets to support connecting any multiple public networks
  - extra
```

After the above template is applied, you should see the following resources exist

```
# kubectl ko nbctl show vpc1
router 87ad06fd-71d5-4ff8-a1f0-54fa3bba1a7f (vpc1)
    port vpc1-vpc1-subnet1
        mac: "00:00:00:ED:8E:C7"
        networks: ["192.168.0.1/24"]
    port vpc1-external204
        mac: "00:00:00:EF:05:C7"
        networks: ["10.5.204.105/24"]
        gateway chassis: [7cedd14f-265b-42e5-ac17-e03e7a1f2342 276baccb-fe9c-4476-b41d-05872a94976d fd9f140c-c45d-43db-a6c0-0d4f8ea298dd]
    port vpc1-extra
        mac: "00:00:00:EF:6A:C7"
        networks: ["10.10.204.105/24"]
        gateway chassis: [7cedd14f-265b-42e5-ac17-e03e7a1f2342 276baccb-fe9c-4476-b41d-05872a94976d fd9f140c-c45d-43db-a6c0-0d4f8ea298dd]
```

5.7.3 2. ovn-eip

This function is designed and used in the same way as iptables-eip, ovn-eip currently has three types

- nat: indicates ovn dnat, fip, and snat.
- lrp: indicates the resource used to connect a VPC to the public network
- lsp: In the ovn BFD-based ecmp static route scenario, an ovs internal port is provided on the gateway node as the next hop of the ecmp route

```
---
kind: OvnEip
apiVersion: kubeovn.io/v1
metadata:
  name: eip-static
spec:
  externalSubnet: external204
  type: nat

# Dynamically allocate an eip resource that is reserved for fip dnat_and_snat scenarios
```

The externalSubnet field does not need to be configured. If not configured, the default public network will be used. In the above configuration, the default public network is external204.

If you want to use an additional public network, you need to explicitly specify the public network to be extended through externalSubnet. In the above configuration, the extended public network is extra.

2.1 Create an fip for pod

```
# kubectl get po -o wide -n vpc1 vpc-1-busybox01
NAME              READY   STATUS    RESTARTS   AGE     IP            NODE
vpc-1-busybox01   1/1     Running   0          3d15h   192.168.0.2   pc-node-2

# kubectl get ip vpc-1-busybox01.vpc1
NAME                   V4IP          V6IP   MAC                 NODE        SUBNET
vpc-1-busybox01.vpc1   192.168.0.2          00:00:00:0A:DD:27   pc-node-2   vpc1-subnet1

---

kind: OvnEip
apiVersion: kubeovn.io/v1
metadata:
  name: eip-static
spec:
  externalSubnet: external204
  type: nat

---
kind: OvnFip
apiVersion: kubeovn.io/v1
metadata:
  name: eip-static
spec:
  ovnEip: eip-static
  ipName: vpc-1-busybox01.vpc1  # the name of the ip crd, which is unique
  type: "centralized"           # centralized or distributed

--
# Alternatively, you can specify a VPC or Intranet ip address

kind: OvnFip
apiVersion: kubeovn.io/v1
metadata:
  name: eip-static
spec:
  ovnEip: eip-static
  vpc: vpc1
  v4Ip: 192.168.0.2
  type: "centralized"           # centralized or distributed
```

```
# kubectl get ofip
NAME          VPC    V4EIP          V4IP          READY   IPTYPE   IPNAME
eip-for-vip   vpc1   10.5.204.106   192.168.0.3   true    vip      test-fip-vip
eip-static    vpc1   10.5.204.101   192.168.0.2   true             vpc-1-busybox01.vpc1
# kubectl get ofip eip-static
NAME         VPC    V4EIP          V4IP          READY   IPTYPE   IPNAME
eip-static   vpc1   10.5.204.101   192.168.0.2   true             vpc-1-busybox01.vpc1

[root@pc-node-1 03-cust-vpc]# ping 10.5.204.101
PING 10.5.204.101 (10.5.204.101) 56(84) bytes of data.
64 bytes from 10.5.204.101: icmp_seq=2 ttl=62 time=1.21 ms
64 bytes from 10.5.204.101: icmp_seq=3 ttl=62 time=0.624 ms
64 bytes from 10.5.204.101: icmp_seq=4 ttl=62 time=0.368 ms
^C
--- 10.5.204.101 ping statistics ---
4 packets transmitted, 3 received, 25% packet loss, time 3049ms
rtt min/avg/max/mdev = 0.368/0.734/1.210/0.352 ms
[root@pc-node-1 03-cust-vpc]#

# pod <--> node ping is working
```

```
# The key resources that this public ip can pass include the following ovn nb resources

# kubectl ko nbctl show vpc1
router 87ad06fd-71d5-4ff8-a1f0-54fa3bba1a7f (vpc1)
    port vpc1-vpc1-subnet1
        mac: "00:00:00:ED:8E:C7"
        networks: ["192.168.0.1/24"]
    port vpc1-external204
        mac: "00:00:00:EF:05:C7"
        networks: ["10.5.204.105/24"]
        gateway chassis: [7cedd14f-265b-42e5-ac17-e03e7a1f2342 276baccb-fe9c-4476-b41d-05872a94976d fd9f140c-c45d-43db-a6c0-0d4f8ea298dd]
    nat 813523e7-c68c-408f-bd8c-cba30cb2e4f4
```

```
        external ip: "10.5.204.101"
        logical ip: "192.168.0.2"
        type: "dnat_and_snat"
```

2.2 Create an fip for vip

In order to facilitate the use of some vip scenarios, such as inside kubevirt VM, keepalived use vip, kube-vip use vip, etc. the vip need public network access.

```
# First create vip, eip, then bind eip to vip
# cat vip.yaml

apiVersion: kubeovn.io/v1
kind: Vip
metadata:
  name: test-fip-vip
spec:
  subnet: vpc1-subnet1

# cat 04-fip.yaml

---
kind: OvnEip
apiVersion: kubeovn.io/v1
metadata:
  name: eip-for-vip
spec:
  externalSubnet: external204
  type: nat

---
kind: OvnFip
apiVersion: kubeovn.io/v1
metadata:
  name: eip-for-vip
spec:
  ovnEip: eip-for-vip
  ipType: vip         # By default fip is for pod ip, here you need to specify the docking to vip resources
  ipName: test-fip-vip

---
# Alternatively, you can specify a VPC or Intranet ip address

kind: OvnFip
apiVersion: kubeovn.io/v1
metadata:
  name: eip-for-vip
spec:
  ovnEip: eip-for-vip
  ipType: vip         # By default fip is for pod ip, here you need to specify the docking to vip resources
  vpc: vpc1
  v4Ip: 192.168.0.3
```

```
# kubectl get ofip
NAME          VPC    V4EIP          V4IP          READY   IPTYPE   IPNAME
eip-for-vip   vpc1   10.5.204.106   192.168.0.3   true    vip      test-fip-vip


[root@pc-node-1 fip-vip]# ping  10.5.204.106
PING 10.5.204.106 (10.5.204.106) 56(84) bytes of data.
64 bytes from 10.5.204.106: icmp_seq=1 ttl=62 time=0.694 ms
64 bytes from 10.5.204.106: icmp_seq=2 ttl=62 time=0.436 ms

# node <--> pod fip is working

# The way ip is used inside the pod is roughly as follows

[root@pc-node-1 fip-vip]# kubectl -n vpc1 exec -it vpc-1-busybox03 -- bash
[root@vpc-1-busybox03 /]#
[root@vpc-1-busybox03 /]#
[root@vpc-1-busybox03 /]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
1568: eth0@if1569: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 00:00:00:56:40:e5 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 192.168.0.5/24 brd 192.168.0.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet 192.168.0.3/24 scope global secondary eth0  # vip here
       valid_lft forever preferred_lft forever
    inet6 fe80::200:ff:fe56:40e5/64 scope link
       valid_lft forever preferred_lft forever

[root@vpc-1-busybox03 /]# tcpdump -i eth0 host  192.168.0.3 -netvv
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
00:00:00:ed:8e:c7 > 00:00:00:56:40:e5, ethertype IPv4 (0x0800), length 98: (tos 0x0, ttl 62, id 44830, offset 0, flags [DF], proto ICMP (1), length 84)
```

```
    10.5.32.51 > 192.168.0.3: ICMP echo request, id 177, seq 1, length 64
00:00:00:56:40:e5 > 00:00:00:ed:8e:c7, ethertype IPv4 (0x0800), length 98: (tos 0x0, ttl 64, id 43962, offset 0, flags [none], proto ICMP (1), length 84)
    192.168.0.3 > 10.5.32.51: ICMP echo reply, id 177, seq 1, length 64

# pod internal can catch fip related icmp packets
```

5.7.4 3. ovn-snat

3.1 ovn-snat corresponds to the CIDR of a subnet

This feature is designed and used in much the same way as iptables-snat

```
# cat 03-subnet-snat.yaml

---
kind: OvnEip
apiVersion: kubeovn.io/v1
metadata:
  name: snat-for-subnet-in-vpc
spec:
  externalSubnet: external204
  type: nat

---
kind: OvnSnatRule
apiVersion: kubeovn.io/v1
metadata:
  name: snat-for-subnet-in-vpc
spec:
  ovnEip: snat-for-subnet-in-vpc
  vpcSubnet: vpc1-subnet1 # eip corresponds to the entire network segment

---
# Alternatively, you can specify a VPC and subnet cidr on an Intranet

kind: OvnSnatRule
apiVersion: kubeovn.io/v1
metadata:
  name: snat-for-subnet-in-vpc
spec:
  ovnEip: snat-for-subnet-in-vpc
  vpc: vpc1
  v4IpCidr: 192.168.0.0/24 # VPC subnet cidr or ip address
```

If you want to use an additional public network, you need to explicitly specify the public network to be extended through externalSubnet. In the above configuration, the extended public network is extra.

3.2 ovn-snat corresponds to a pod IP

This feature is designed and used in much the same way as iptables-snat

```
# cat 03-pod-snat.yaml
---
kind: OvnEip
apiVersion: kubeovn.io/v1
metadata:
  name: snat-for-pod-vpc-ip
spec:
  externalSubnet: external204
  type: nat

---
kind: OvnSnatRule
apiVersion: kubeovn.io/v1
metadata:
  name: snat01
spec:
  ovnEip: snat-for-pod-vpc-ip
  ipName: vpc-1-busybox02.vpc1 # eip corresponds to a single pod ip

---
# Alternatively, you can specify a VPC or Intranet ip address

kind: OvnSnatRule
apiVersion: kubeovn.io/v1
metadata:
  name: snat-for-subnet-in-vpc
spec:
  ovnEip: snat-for-subnet-in-vpc
  vpc: vpc1
  v4IpCidr: 192.168.0.4
```

If you want to use an additional public network, you need to explicitly specify the public network to be extended through externalSubnet. In the above configuration, the extended public network is extra.

After the above resources are created, you can see the following resources that the snat public network feature depends on.

```
# kubectl ko nbctl show vpc1
router 87ad06fd-71d5-4ff8-a1f0-54fa3bba1a7f (vpc1)
    port vpc1-vpc1-subnet1
        mac: "00:00:00:ED:8E:C7"
        networks: ["192.168.0.1/24"]
    port vpc1-external204
        mac: "00:00:00:EF:05:C7"
        networks: ["10.5.204.105/24"]
        gateway chassis: [7cedd14f-265b-42e5-ac17-e03e7a1f2342 276baccb-fe9c-4476-b41d-05872a94976d fd9f140c-c45d-43db-a6c0-0d4f8ea298dd]
    nat 21d853b0-f7b4-40bd-9a53-31d2e2745739
        external ip: "10.5.204.115"
        logical ip: "192.168.0.0/24"
        type: "snat"
    nat da77a11f-c523-439c-b1d1-72c664196a0f
        external ip: "10.5.204.116"
        logical ip: "192.168.0.4"
        type: "snat"
```

```
[root@pc-node-1 03-cust-vpc]# kubectl get po -A -o wide  | grep busy
vpc1            vpc-1-busybox01                                 1/1     Running   0                3d15h   192.168.0.2   pc-node-2   <none>           <none>
vpc1            vpc-1-busybox02                                 1/1     Running   0                17h     192.168.0.4   pc-node-1   <none>           <none>
vpc1            vpc-1-busybox03                                 1/1     Running   0                17h     192.168.0.5   pc-node-1   <none>           <none>
vpc1            vpc-1-busybox04                                 1/1     Running   0                17h     192.168.0.6   pc-node-3   <none>           <none>
vpc1            vpc-1-busybox05                                 1/1     Running   0                17h     192.168.0.7   pc-node-1   <none>           <none>

# kubectl exec -it -n vpc1            vpc-1-busybox04   bash
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] instead.
[root@vpc-1-busybox04 /]#
[root@vpc-1-busybox04 /]#
[root@vpc-1-busybox04 /]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
17095: eth0@if17096: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 00:00:00:76:94:55 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 192.168.0.6/24 brd 192.168.0.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::200:ff:fe76:9455/64 scope link
       valid_lft forever preferred_lft forever
[root@vpc-1-busybox04 /]# ping 223.5.5.5
PING 223.5.5.5 (223.5.5.5) 56(84) bytes of data.
64 bytes from 223.5.5.5: icmp_seq=1 ttl=114 time=22.2 ms
64 bytes from 223.5.5.5: icmp_seq=2 ttl=114 time=21.8 ms

[root@pc-node-1 03-cust-vpc]# kubectl exec -it -n vpc1            vpc-1-busybox02   bash
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] instead.
[root@vpc-1-busybox02 /]#
[root@vpc-1-busybox02 /]#
[root@vpc-1-busybox02 /]#
[root@vpc-1-busybox02 /]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
1566: eth0@if1567: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 00:00:00:0b:e9:d0 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 192.168.0.4/24 brd 192.168.0.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::200:ff:fe0b:e9d0/64 scope link
       valid_lft forever preferred_lft forever
[root@vpc-1-busybox02 /]# ping 223.5.5.5
PING 223.5.5.5 (223.5.5.5) 56(84) bytes of data.
64 bytes from 223.5.5.5: icmp_seq=2 ttl=114 time=22.7 ms
64 bytes from 223.5.5.5: icmp_seq=3 ttl=114 time=22.6 ms
64 bytes from 223.5.5.5: icmp_seq=4 ttl=114 time=22.1 ms
^C
--- 223.5.5.5 ping statistics ---
4 packets transmitted, 3 received, 25% packet loss, time 3064ms
rtt min/avg/max/mdev = 22.126/22.518/22.741/0.278 ms

# the two pods can access the external network based on these two type snat resources respectively
```

5.7.5 4. ovn-dnat

4.1 ovn-dnat binds a DNAT to a pod

```yaml
kind: OvnEip
apiVersion: kubeovn.io/v1
metadata:
  name: eip-dnat
spec:
  externalSubnet: underlay
  type: nat
---
kind: OvnDnatRule
apiVersion: kubeovn.io/v1
metadata:
  name: eip-dnat
spec:
  ovnEip: eip-dnat
  ipName: vpc-1-busybox01.vpc1 # Note that this is the name of the pod IP CRD and it is unique
  protocol: tcp
  internalPort: "22"
  externalPort: "22"

---
# Alternatively, you can specify a VPC or Intranet ip address

kind: OvnDnatRule
apiVersion: kubeovn.io/v1
metadata:
  name: eip-dnat
spec:
  ovnEip: eip-dnat
  protocol: tcp
  internalPort: "22"
  externalPort: "22"
  vpc: vpc1
  v4Ip: 192.168.0.3
```

If you want to use an additional public network, you need to explicitly specify the public network to be extended through externalSubnet. In the above configuration, the extended public network is extra.

The configuration of OvnDnatRule is similar to that of IptablesDnatRule.

```
# kubectl get oeip eip-dnat
NAME       V4IP        V6IP   MAC                 TYPE   READY
eip-dnat   10.5.49.4          00:00:00:4D:CE:49   dnat   true

# kubectl get odnat
NAME                    EIP                     PROTOCOL   V4EIP        V4IP           INTERNALPORT   EXTERNALPORT   IPNAME                                READY
eip-dnat                eip-dnat                tcp        10.5.49.4    192.168.0.3    22             22             vpc-1-busybox01.vpc1                  true
```

4.2 ovn-dnat binds a DNAT to a VIP

```yaml
kind: OvnDnatRule
apiVersion: kubeovn.io/v1
metadata:
  name: eip-dnat
spec:
  ipType: vip  # By default, Dnat is oriented towards pod IPs. Here, it is necessary to specify that it is connected to VIP resources
  ovnEip: eip-dnat
  ipName: test-dnat-vip
  protocol: tcp
  internalPort: "22"
  externalPort: "22"


---
# Alternatively, you can specify a VPC or Intranet ip address

kind: OvnDnatRule
apiVersion: kubeovn.io/v1
metadata:
  name: eip-dnat
spec:
  ipType: vip  # By default, Dnat is oriented towards pod IPs. Here, it is necessary to specify that it is connected to VIP resources
  ovnEip: eip-dnat
  ipName: test-dnat-vip
  protocol: tcp
  internalPort: "22"
  externalPort: "22"
  vpc: vpc1
  v4Ip: 192.168.0.4
```

The configuration of OvnDnatRule is similar to that of IptablesDnatRule.

```
# kubectl get vip test-dnat-vip
NAME            V4IP          PV4IP   MAC                 PMAC   V6IP   PV6IP   SUBNET         READY
test-dnat-vip   192.168.0.4           00:00:00:D0:C0:B5                         vpc1-subnet1   true
```

```
# kubectl get oeip eip-dnat
NAME       V4IP        V6IP   MAC                 TYPE   READY
eip-dnat   10.5.49.4          00:00:00:4D:CE:49   dnat   true

# kubectl get odnat eip-dnat
NAME       EIP        PROTOCOL   V4EIP       V4IP          INTERNALPORT   EXTERNALPORT   IPNAME          READY
eip-dnat   eip-dnat   tcp        10.5.49.4   192.168.0.4   22             22             test-dnat-vip   true
```

PDF Slack Support

July 31, 2025

March 3, 2023

GitHub

5.7.6 Comments

5.8 Support OVN SNAT L3 HA Based ECMP and BFD Static Route

Custom vpc based on ovn snat after ecmp based static route hash to multiple gw node ovnext0 NICs out of the public network

- Supports bfd-based high availability
- Only supports hash load balancing

```
graph LR

pod-->vpc-subnet-->vpc-->snat-->ecmp-->external-subnet-->gw-node1-ovnext0--> node1-external-switch
external-subnet-->gw-node2-ovnext0--> node2-external-switch
external-subnet-->gw-node3-ovnext0--> node3-external-switch
```

This functions basically the same as ovn-eip-fip-snat.md .

As for the different parts, which will be specified in the following sections, mainly including the creation of ovn-eip of lsp type and the automatic maintenance of bfd as well as ecmp static routes based on vpc enable_bfd.

5.8.1 1. Deployment

1.1 Create the underlay public network

1.2 Default vpc enable eip_snat

1.3 Custom vpc enable eip snat fip function

The above section is exactly the same with ovn-eip-fip-snat.md.

After these functions are verified, the vpc can be switched directly to the ecmp-based bfd static route based on the following way, or of course, switched directly back.

Before customizing vpc to use this feature, you need to provide some gateway nodes, at least 2. Note that the name of the current implementation of ovn-eip must be consistent with the gateway node name, no automated maintenance is currently done for this resource.

```yaml
# cat gw-node-eip.yaml
---
kind: OvnEip
apiVersion: kubeovn.io/v1
metadata:
  name: pc-node-1
spec:
  externalSubnet: external204
  type: lsp

---
kind: OvnEip
apiVersion: kubeovn.io/v1
metadata:
  name: pc-node-2
spec:
  externalSubnet: external204
  type: lsp

---
kind: OvnEip
apiVersion: kubeovn.io/v1
metadata:
  name: pc-node-3
spec:
  externalSubnet: external204
  type: lsp
```

Since this scenario is currently designed for vpc ecmp out of the public network, the gateway node above will not trigger the creation of a gateway NIC when there is no vpc enabled bfd, i.e. when there is no ovn eip (lrp) with enable bfd labeled, and will not be able to successfully start listening to the bfd session on the other side.

5.8.2 2. Custom vpc enable ecmp bfd L3 HA public network function

```
# cat 01-vpc-ecmp-enable-external-bfd.yml
kind: Vpc
apiVersion: kubeovn.io/v1
metadata:
  name: vpc1
spec:
  namespaces:
  - vpc1
  enableExternal: true
  enableBfd: true # bfd switch can be switched at will
  #enableBfd: false


# cat 02-subnet.yml
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
  name: vpc1-subnet1
spec:
  cidrBlock: 192.168.0.0/24
  default: false
  disableGatewayCheck: false
  disableInterConnection: true
  enableEcmp: true  # enable ecmp
  gatewayNode: ""
  gatewayType: distributed
  #gatewayType: centralized
  natOutgoing: false
  private: false
  protocol: IPv4
  provider: ovn
  vpc: vpc1
  namespaces:
  - vpc1
```

note:

1. Customize ecmp under vpc to use only static ecmp bfd routes. vpc enableBfd and subnet enableEcmp will only take effect if they are enabled at the same time, before static ecmp bfd routes are automatically managed.
2. If the above configuration is turned off, it will automatically switch back to the regular default static route.
3. This feature is not available for the default vpc, only custom vpc is supported, the default vpc has more complex policy routing.
4. The enableEcmp of the subnet of the custom vpc uses only static routes, the gateway type gatewayType has no effect.
5. When EnableExternal is turned off in vpc, the external network cannot be passed inside vpc.
6. When EnableExternal is enabled on vpc, when EnableBfd is turned off, it will be based on the normal default route to the external network and will not have high availability.

```
# After the above template is applied the ovn logic layer should see the following resources
# k get vpc
NAME          ENABLEEXTERNAL   ENABLEBFD   STANDBY   SUBNETS                                NAMESPACES
ovn-cluster   true                         true      ["external204","join","ovn-default"]
vpc1          true             true        true      ["vpc1-subnet1"]                       ["vpc1"]

# Default vpc does not support ENABLEBFD
# Custom vpc is supported and enabled

# 1. bfd table created
# k ko nbctl list bfd
_uuid               : be7df545-2c4c-4751-878f-b3507987f050
detect_mult         : 3
dst_ip              : "10.5.204.121"
external_ids        : {}
logical_port        : vpc1-external204
min_rx              : 100
min_tx              : 100
options             : {}
status              : up

_uuid               : 684c4489-5b59-4693-8d8c-3beab93f8093
detect_mult         : 3
dst_ip              : "10.5.204.109"
external_ids        : {}
logical_port        : vpc1-external204
min_rx              : 100
min_tx              : 100
options             : {}
status              : up

_uuid               : f0f62077-2ae9-4e79-b4f8-a446ec6e784c
detect_mult         : 3
dst_ip              : "10.5.204.108"
external_ids        : {}
```

```
logical_port        : vpc1-external204
min_rx              : 100
min_tx              : 100
options             : {}
status              : up

### Note that all statuses should normally be up

# 2. bfd ecmp static routes table created
# k ko nbctl lr-route-list vpc1
IPv4 Routes
Route Table <main>:
           192.168.0.0/24              10.5.204.108 src-ip ecmp ecmp-symmetric-reply bfd
           192.168.0.0/24              10.5.204.109 src-ip ecmp ecmp-symmetric-reply bfd
           192.168.0.0/24              10.5.204.121 src-ip ecmp ecmp-symmetric-reply bfd

# 3. Static Route Details
# k ko nbctl find Logical_Router_Static_Route  policy=src-ip options=ecmp_symmetric_reply="true"
_uuid               : 3aacb384-d5ee-4b14-aebf-59e8c11717ba
bfd                 : 684c4489-5b59-4693-8d8c-3beab93f8093
external_ids        : {}
ip_prefix           : "192.168.0.0/24"
nexthop             : "10.5.204.109"
options             : {ecmp_symmetric_reply="true"}
output_port         : []
policy              : src-ip
route_table         : ""

_uuid               : 18bcc585-bc05-430b-925b-ef673c8e1aef
bfd                 : f0f62077-2ae9-4e79-b4f8-a446ec6e784c
external_ids        : {}
ip_prefix           : "192.168.0.0/24"
nexthop             : "10.5.204.108"
options             : {ecmp_symmetric_reply="true"}
output_port         : []
policy              : src-ip
route_table         : ""

_uuid               : 7d0a4e6b-cde0-4110-8176-fbaf19738498
bfd                 : be7df545-2c4c-4751-878f-b3507987f050
external_ids        : {}
ip_prefix           : "192.168.0.0/24"
nexthop             : "10.5.204.121"
options             : {ecmp_symmetric_reply="true"}
output_port         : []
policy              : src-ip
route_table         : ""
```

```
# Also, the following resources should be available at all gateway nodes

[root@pc-node-1 ~]# ip netns exec ovnext bash ip a
/usr/sbin/ip: /usr/sbin/ip: cannot execute binary file
[root@pc-node-1 ~]#
[root@pc-node-1 ~]# ip netns exec ovnext ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
1541: ovnext0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1400 qdisc noqueue state UNKNOWN group default qlen 1000
    link/ether 00:00:00:ab:bd:87 brd ff:ff:ff:ff:ff:ff
    inet 10.5.204.108/24 brd 10.5.204.255 scope global ovnext0
       valid_lft forever preferred_lft forever
    inet6 fe80::200:ff:feab:bd87/64 scope link
       valid_lft forever preferred_lft forever
[root@pc-node-1 ~]#
[root@pc-node-1 ~]# ip netns exec ovnext route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         10.5.204.254    0.0.0.0         UG    0      0        0 ovnext0
10.5.204.0      0.0.0.0         255.255.255.0   U     0      0        0 ovnext0


[root@pc-node-1 ~]# ip netns exec ovnext bfdd-control status
There are 1 sessions:
Session 1
 id=1 local=10.5.204.108 (p) remote=10.5.204.122 state=Up

## This is the other end of the lrp bfd session and one of the next hops of the lrp ecmp


[root@pc-node-1 ~]# ip netns exec ovnext ping -c1 223.5.5.5
PING 223.5.5.5 (223.5.5.5) 56(84) bytes of data.
64 bytes from 223.5.5.5: icmp_seq=1 ttl=115 time=21.6 ms

# No problem to the public network
```

catch outgoing packets within the ovnext ns of a gateway node

```
# tcpdump -i ovnext0 host 223.5.5.5 -netvv
dropped privs to tcpdump
```

```
tcpdump: listening on ovnext0, link-type EN10MB (Ethernet), capture size 262144 bytes
^C
0 packets captured
0 packets received by filter
0 packets dropped by kernel
[root@pc-node-1 ~]# exit
[root@pc-node-1 ~]# ssh pc-node-2
Last login: Thu Feb 23 09:21:08 2023 from 10.5.32.51
[root@pc-node-2 ~]# ip netns exec ovnext bash
[root@pc-node-2 ~]# tcpdump -i ovnext0 host 223.5.5.5 -netvv
dropped privs to tcpdump
tcpdump: listening on ovnext0, link-type EN10MB (Ethernet), capture size 262144 bytes
^C
0 packets captured
0 packets received by filter
0 packets dropped by kernel
[root@pc-node-2 ~]# exit
[root@pc-node-2 ~]# logout
Connection to pc-node-2 closed.
[root@pc-node-1 ~]# ssh pc-node-3
Last login: Thu Feb 23 08:32:41 2023 from 10.5.32.51
[root@pc-node-3 ~]#  ip netns exec ovnext bash
[root@pc-node-3 ~]# tcpdump -i ovnext0 host 223.5.5.5 -netvv
dropped privs to tcpdump
tcpdump: listening on ovnext0, link-type EN10MB (Ethernet), capture size 262144 bytes
00:00:00:2d:f8:ce > 00:00:00:fd:b2:a4, ethertype IPv4 (0x0800), length 98: (tos 0x0, ttl 63, id 57978, offset 0, flags [DF], proto ICMP (1), length 84)
    10.5.204.102 > 223.5.5.5: ICMP echo request, id 22, seq 71, length 64
00:00:00:fd:b2:a4 > dc:ef:80:5a:44:1a, ethertype IPv4 (0x0800), length 98: (tos 0x0, ttl 62, id 57978, offset 0, flags [DF], proto ICMP (1), length 84)
    10.5.204.102 > 223.5.5.5: ICMP echo request, id 22, seq 71, length 64
^C
2 packets captured
2 packets received by filter
0 packets dropped by kernel
[root@pc-node-3 ~]#
```

5.8.3 3. Turn off bfd mode

In some scenarios, you may want to use a (centralized) single gateway directly out of the public network, which is the same as the default vpc enable_eip_snat usage pattern

```
# cat 01-vpc-ecmp-enable-external-bfd.yml
kind: Vpc
apiVersion: kubeovn.io/v1
metadata:
  name: vpc2
spec:
  namespaces:
  - vpc2
  enableExternal: true
  #enableBfd: true
  enableBfd: false

## set it false add apply

# k ko nbctl lr-route-list vpc2
IPv4 Routes
Route Table <main>:
                0.0.0.0/0               10.5.204.254 dst-ip

# After application the route will switch back to the normal default static route
# nbctl list bfd, the bfd session associated with lrp has been removed
# And the opposite side of the bfd session in ovnext ns is automatically removed
```

PDF   Slack   Support

March 3, 2025

March 3, 2023

GitHub

5.8.4 Comments

5.9 VPC Peering

VPC peering provides a mechanism for bridging two VPC networks through logical routes so that workloads within two VPCs can access each other through private addresses as if they were on the same private network, without the need for NAT forwarding through a gateway.

5.9.1 Prerequisites

1. This feature is only available for customized VPCs.
2. To avoid route overlap the subnet CIDRs within the two VPCs cannot overlap.
3. Currently, only interconnection of two VPCs is supported.

5.9.2 Usage

First create two non-interconnected VPCs with one Subnet under each VPC, and the CIDRs of the Subnets do not overlap with each other.

```yaml
kind: Vpc
apiVersion: kubeovn.io/v1
metadata:
  name: vpc-1
spec: {}
---
kind: Subnet
apiVersion: kubeovn.io/v1
metadata:
  name: net1
spec:
  vpc: vpc-1
  cidrBlock: 10.0.0.0/16
---
kind: Vpc
apiVersion: kubeovn.io/v1
metadata:
  name: vpc-2
spec: {}
---
kind: Subnet
apiVersion: kubeovn.io/v1
metadata:
  name: net2
spec:
  vpc: vpc-2
  cidrBlock: 172.31.0.0/16
```

Add `vpcPeerings` and the corresponding static routes within each VPC:

```yaml
kind: Vpc
apiVersion: kubeovn.io/v1
metadata:
  name: vpc-1
spec:
  vpcPeerings:
    - remoteVpc: vpc-2
      localConnectIP: 169.254.0.1/30
  staticRoutes:
    - cidr: 172.31.0.0/16
      nextHopIP: 169.254.0.2
      policy: policyDst
---
kind: Vpc
apiVersion: kubeovn.io/v1
metadata:
  name: vpc-2
spec:
  vpcPeerings:
    - remoteVpc: vpc-1
      localConnectIP: 169.254.0.2/30
  staticRoutes:
    - cidr: 10.0.0.0/16
```

```
      nextHopIP: 169.254.0.1
      policy: policyDst
```

- `remoteVpc` : the name of another peering VPC.
- `localConnectIP` : as the IP address and CIDR of the interconnection endpoint. Note that both IPs should belong to the same CIDR and should not conflict with existing subnets.
- `cidr` : CIDR of the peering Subnet.
- `nextHopIP` : the `localConnectIP` on the other end of the peering VPC.

Create Pods under the two Subnets

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    ovn.kubernetes.io/logical_switch: net1
  name: vpc-1-pod
spec:
  containers:
    - name: vpc-1-pod
      image: docker.io/library/nginx:alpine
---
apiVersion: v1
kind: Pod
metadata:
  annotations:
    ovn.kubernetes.io/logical_switch: net2
  name: vpc-2-pod
spec:
  containers:
    - name: vpc-2-pod
      image: docker.io/library/nginx:alpine
```

Test the network connectivity

```
# kubectl exec -it vpc-1-pod -- ping $(kubectl get pod vpc-2-pod -o jsonpath='{.status.podIP}')
PING 172.31.0.2 (172.31.0.2): 56 data bytes
64 bytes from 172.31.0.2: seq=0 ttl=62 time=0.655 ms
64 bytes from 172.31.0.2: seq=1 ttl=62 time=0.086 ms
64 bytes from 172.31.0.2: seq=2 ttl=62 time=0.098 ms
^C
--- 172.31.0.2 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.086/0.279/0.655 ms
# kubectl exec -it vpc-2-pod -- ping $(kubectl get pod vpc-1-pod -o jsonpath='{.status.podIP}')
PING 10.0.0.2 (10.0.0.2): 56 data bytes
64 bytes from 10.0.0.2: seq=0 ttl=62 time=0.594 ms
64 bytes from 10.0.0.2: seq=1 ttl=62 time=0.093 ms
64 bytes from 10.0.0.2: seq=2 ttl=62 time=0.088 ms
^C
--- 10.0.0.2 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.088/0.258/0.594 ms
```

PDF **Slack** **Support**

July 30, 2025

May 24, 2022

GitHub

5.9.3 Comments

6. Operations

6.1 Kubectl Plugin

To facilitate daily operations and maintenance, Kube-OVN provides the kubectl plug-in tool, which allows administrators to perform daily operations through this command. For examples: Check OVN database information and status, OVN database backup and restore, OVS related information, tcpdump specific containers, specific link logical topology, network problem diagnosis and performance optimization.

6.1.1 Plugin Installation

Kube-OVN installation will deploy the plugin to each node by default. If the machine that runs kubectl is not in the cluster, or if you need to reinstall the plugin, please refer to the following steps:

Download `kubectl-ko` file:

```
wget https://raw.githubusercontent.com/kubeovn/kube-ovn/release-1.14/dist/images/kubectl-ko
```

Move file to `$PATH` :

```
mv kubectl-ko /usr/local/bin/kubectl-ko
```

Add executable permissions:

```
chmod +x /usr/local/bin/kubectl-ko
```

Check if the plugin works properly:

```
# kubectl plugin list
The following compatible plugins are available:

/usr/local/bin/kubectl-ko
```

6.1.2 Plugin Usage

Running `kubectl ko` will show all the available commands and usage descriptions, as follows:

```
# kubectl ko
kubectl ko {subcommand} [option...]
Available Subcommands:
  [nb|sb] [status|kick|backup|dbstatus|restore]     ovn-db operations show cluster status, kick stale server, backup database, get db consistency status or
restore ovn nb db when met 'inconsistent data' error
  nbctl [ovn-nbctl options ...]    invoke ovn-nbctl
  sbctl [ovn-sbctl options ...]    invoke ovn-sbctl
  vsctl {nodeName} [ovs-vsctl options ...]   invoke ovs-vsctl on the specified node
  ofctl {nodeName} [ovs-ofctl options ...]   invoke ovs-ofctl on the specified node
  dpctl {nodeName} [ovs-dpctl options ...]   invoke ovs-dpctl on the specified node
  appctl {nodeName} [ovs-appctl options ...]   invoke ovs-appctl on the specified node
  tcpdump {namespace/podname} [tcpdump options ...]     capture pod traffic
  {trace|ovn-trace} ...    trace ovn microflow of specific packet"
    {trace|ovn-trace} {namespace/podname} {target ip address} [target mac address] {icmp|tcp|udp} [target tcp/udp port]    trace ICMP/TCP/UDP
    {trace|ovn-trace} {namespace/podname} {target ip address} [target mac address] arp {request|reply}                     trace ARP request/reply
    {trace|ovn-trace} {node//nodename} {target ip address} [target mac address] {icmp|tcp|udp} [target tcp/udp port]       trace ICMP/TCP/UDP
    {trace|ovn-trace} {node//nodename} {target ip address} [target mac address] arp {request|reply}                       trace ARP request/reply
  echo "  diagnose {all|node|subnet|IPPorts} [nodename|subnetName|{proto1}-{IP1}-{Port1},{proto2}-{IP2}-{Port2}]    diagnose connectivity of all nodes or a
specific node or specify subnet's ds pod or IPPorts like 'tcp-172.18.0.2-53,udp-172.18.0.3-53'"
  tuning {install-fastpath|local-install-fastpath|remove-fastpath|install-stt|local-install-stt|remove-stt} {centos7|centos8}} [kernel-devel-version]  deploy
kernel optimisation components to the system
  reload    restart all kube-ovn components
  log {kube-ovn|ovn|ovs|linux|all}    save log to ./kubectl-ko-log/
  perf [image] performance test default image is kubeovn/test:v1.12.0
```

The specific functions and usage of each command are described below.

[nb | sb] [status | kick | backup | dbstatus | restore]

This subcommand mainly operates on OVN northbound or southbound databases, including database cluster status check, database node offline, database backup, database storage status check and database repair.

DB CLUSTER STATUS CHECK

This command executes `ovs-appctl cluster/status` on the leader node of the corresponding OVN database to show the cluster status:

```
# kubectl ko nb status
306b
Name: OVN_Northbound
Cluster ID: 9a87 (9a872522-3e7d-47ca-83a3-d74333e1a7ca)
Server ID: 306b (306b256b-b5e1-4eb0-be91-4ca96adf6bad)
Address: tcp:[172.18.0.2]:6643
Status: cluster member
Role: leader
Term: 1
Leader: self
Vote: self

Last Election started 280309 ms ago, reason: timeout
Last Election won: 280309 ms ago
Election timer: 5000
Log: [139, 139]
Entries not yet committed: 0
Entries not yet applied: 0
Connections: <-8723 ->8723 <-85d6 ->85d6
Disconnections: 0
Servers:
    85d6 (85d6 at tcp:[172.18.0.4]:6643) next_index=139 match_index=138 last msg 763 ms ago
    8723 (8723 at tcp:[172.18.0.3]:6643) next_index=139 match_index=138 last msg 763 ms ago
    306b (306b at tcp:[172.18.0.2]:6643) (self) next_index=2 match_index=138
status: ok
```

If the `match_index` under `Server` has a large difference and the `last msg` time is long, the corresponding Server may not respond for a long time and needs to be checked further.

DB NODES OFFLINE

This command removes a node from the OVN database and is required when a node is taken offline or replaced. The following is an example of the cluster status from the previous command, to offline the `172.18.0.3` node:

```
# kubectl ko nb kick 8723
started removal
```

Check the database cluster status again to confirm that the node has been removed:

```
# kubectl ko nb status
306b
Name: OVN_Northbound
Cluster ID: 9a87 (9a872522-3e7d-47ca-83a3-d74333e1a7ca)
Server ID: 306b (306b256b-b5e1-4eb0-be91-4ca96adf6bad)
Address: tcp:[172.18.0.2]:6643
Status: cluster member
Role: leader
Term: 1
Leader: self
Vote: self

Last Election started 324356 ms ago, reason: timeout
Last Election won: 324356 ms ago
Election timer: 5000
Log: [140, 140]
Entries not yet committed: 0
Entries not yet applied: 0
Connections: <-85d6 ->85d6
Disconnections: 2
Servers:
    85d6 (85d6 at tcp:[172.18.0.4]:6643) next_index=140 match_index=139 last msg 848 ms ago
    306b (306b at tcp:[172.18.0.2]:6643) (self) next_index=2 match_index=139
status: ok
```

DB BACKUP

This subcommand backs up the current OVN database locally and can be used for disaster recovery:

```
# kubectl ko nb backup
tar: Removing leading `/' from member names
backup ovn-nb db to /root/ovnnb_db.060223191654183154.backup
```

DATABASE STORAGE STATUS CHECK

This command is used to check if the database file is corrupt:

```
# kubectl ko nb dbstatus
status: ok
```

If error happens, `inconsistent data` is displayed and needs to be fixed with the following command.

DATABASE REPAIR

If the database status goes to `inconsistent data`, this command can be used to repair:

```
# kubectl ko nb restore
deployment.apps/ovn-central scaled
ovn-central original replicas is 3
first nodeIP is 172.18.0.5
ovs-ovn pod on node 172.18.0.5 is ovs-ovn-8jxv9
ovs-ovn pod on node 172.18.0.3 is ovs-ovn-sjzb6
ovs-ovn pod on node 172.18.0.4 is ovs-ovn-t87zk
backup nb db file
restore nb db file, operate in pod ovs-ovn-8jxv9
deployment.apps/ovn-central scaled
finish restore nb db file and ovn-central replicas
recreate ovs-ovn pods
pod "ovs-ovn-8jxv9" deleted
pod "ovs-ovn-sjzb6" deleted
pod "ovs-ovn-t87zk" deleted
```

[nbctl | sbctl] [options ...]

This subcommand executes the `ovn-nbctl` and `ovn-sbctl` commands directly into the leader node of the OVN northbound or southbound database. For more detailed usage of this command, please refer to the official documentation of the upstream OVN ovn-nbctl(8) ovn-sbctl(8).

```
# kubectl ko nbctl show
switch c7cd17e8-ceee-4a91-9bb3-e5a313fe1ece (snat)
    port snat-ovn-cluster
        type: router
        router-port: ovn-cluster-snat
switch 20e0c6d0-023a-4756-aec5-200e0c60f95d (join)
    port node-liumengxin-ovn3-192.168.137.178
        addresses: ["00:00:00:64:FF:A8 100.64.0.4"]
    port node-liumengxin-ovn1-192.168.137.176
        addresses: ["00:00:00:AF:98:62 100.64.0.2"]
    port node-liumengxin-ovn2-192.168.137.177
        addresses: ["00:00:00:D9:58:B8 100.64.0.3"]
    port join-ovn-cluster
        type: router
        router-port: ovn-cluster-join
switch 0191705c-f827-427b-9de3-3c3b7d971ba5 (central)
    port central-ovn-cluster
        type: router
        router-port: ovn-cluster-central
switch 2a45ff05-388d-4f85-9daf-e6fccd5833dc (ovn-default)
    port alertmanager-main-0.monitoring
        addresses: ["00:00:00:6C:DF:A3 10.16.0.19"]
    port kube-state-metrics-5d6885d89-4nf8h.monitoring
        addresses: ["00:00:00:6F:02:1C 10.16.0.15"]
    port fake-kubelet-67c55dfd89-pv86k.kube-system
        addresses: ["00:00:00:5C:12:E8 10.16.19.177"]
    port ovn-default-ovn-cluster
        type: router
        router-port: ovn-cluster-ovn-default
router 212f73dd-d63d-4d72-864b-a537e9afbee1 (ovn-cluster)
    port ovn-cluster-snat
        mac: "00:00:00:7A:82:8F"
        networks: ["172.22.0.1/16"]
    port ovn-cluster-join
        mac: "00:00:00:F8:18:5A"
        networks: ["100.64.0.1/16"]
    port ovn-cluster-central
        mac: "00:00:00:4D:8C:F5"
        networks: ["192.101.0.1/16"]
    port ovn-cluster-ovn-default
        mac: "00:00:00:A3:F8:18"
        networks: ["10.16.0.1/16"]
```

vsctl {nodeName} [options ...]

This command will go to the `ovs-ovn` container on the corresponding `nodeName` and execute the corresponding `ovs-vsctl` command to query and configure `vswitchd`. For more detailed usage of this command, please refer to the official documentation of the upstream OVS ovs-vsctl(8).

```
# kubectl ko vsctl kube-ovn-01 show
0d4c4675-c9cc-440a-8c1a-878e17f81b88
    Bridge br-int
        fail_mode: secure
        datapath_type: system
        Port a2c1a8a8b83a_h
            Interface a2c1a8a8b83a_h
        Port "4fa5c4cbb1a5_h"
            Interface "4fa5c4cbb1a5_h"
        Port ovn-eef07d-0
            Interface ovn-eef07d-0
                type: stt
                options: {csum="true", key=flow, remote_ip="192.168.137.178"}
        Port ovn0
            Interface ovn0
                type: internal
        Port mirror0
            Interface mirror0
                type: internal
        Port ovn-efa253-0
            Interface ovn-efa253-0
                type: stt
                options: {csum="true", key=flow, remote_ip="192.168.137.177"}
        Port br-int
            Interface br-int
                type: internal
    ovs_version: "2.17.2"
```

ofctl {nodeName} [options ...]

This command will go to the `ovs-ovn` container on the corresponding `nodeName` and execute the corresponding `ovs-ofctl` command to query or manage OpenFlow. For more detailed usage of this command, please refer to the official documentation of the upstream OVS ovs-ofctl(8).

```
# kubectl ko ofctl kube-ovn-01 dump-flows br-int
NXST_FLOW reply (xid=0x4): flags=[more]
 cookie=0xcf3429e6, duration=671791.432s, table=0, n_packets=0, n_bytes=0, idle_age=65534, hard_age=65534, priority=100,in_port=2 actions=load:0x4-
>NXM_NX_REG13[],load:0x9->NXM_NX_REG11[],load:0xb->NXM_NX_REG12[],load:0x4->OXM_OF_METADATA[],load:0x1->NXM_NX_REG14[],resubmit(,8)
 cookie=0xc91413c6, duration=671791.431s, table=0, n_packets=907489, n_bytes=99978275, idle_age=0, hard_age=65534, priority=100,in_port=7 actions=load:0x1-
>NXM_NX_REG13[],load:0x9->NXM_NX_REG11[],load:0xb->NXM_NX_REG12[],load:0x4->OXM_OF_METADATA[],load:0x4->NXM_NX_REG14[],resubmit(,8)
 cookie=0xf180459, duration=671791.431s, table=0, n_packets=17348582, n_bytes=2667811214, idle_age=0, hard_age=65534, priority=100,in_port=6317 actions=load:
0xa->NXM_NX_REG13[],load:0x9->NXM_NX_REG11[],load:0xb->NXM_NX_REG12[],load:0x4->OXM_OF_METADATA[],load:0x9->NXM_NX_REG14[],resubmit(,8)
 cookie=0x7806dd90, duration=671791.431s, table=0, n_packets=3235428, n_bytes=833821312, idle_age=0, hard_age=65534, priority=100,in_port=1 actions=load:0xd-
>NXM_NX_REG13[],load:0x9->NXM_NX_REG11[],load:0xb->NXM_NX_REG12[],load:0x4->OXM_OF_METADATA[],load:0x3->NXM_NX_REG14[],resubmit(,8)
...
```

dpctl {nodeName} [options ...]

This command will go to the `ovs-ovn` container on the corresponding `nodeName` and execute the corresponding `ovs-dpctl` command to query or manage the OVS datapath. For more detailed usage of this command, please refer to the official documentation of the upstream OVS ovs-dpctl(8).

```
# kubectl ko dpctl kube-ovn-01 show
system@ovs-system:
  lookups: hit:350805055 missed:21983648 lost:73
  flows: 105
  masks: hit:1970748791 total:22 hit/pkt:5.29
  port 0: ovs-system (internal)
  port 1: ovn0 (internal)
  port 2: mirror0 (internal)
  port 3: br-int (internal)
  port 4: stt_sys_7471 (stt: packet_type=ptap)
  port 5: eeb4d9e51b5d_h
  port 6: a2c1a8a8b83a_h
  port 7: 4fa5c4cbb1a5_h
```

appctl {nodeName} [options ...]

This command will enter the `ovs-ovn` container on the corresponding `nodeName` and execute the corresponding `ovs-appctl` command to operate the associated daemon process. For more detailed usage of this command, please refer to the official documentation of the upstream OVS ovs-appctl(8).

```
# kubectl ko appctl kube-ovn-01 vlog/list
                 console    syslog    file
                 -------    ------    ------
backtrace          OFF        ERR       INFO
bfd                OFF        ERR       INFO
bond               OFF        ERR       INFO
bridge             OFF        ERR       INFO
bundle             OFF        ERR       INFO
bundles            OFF        ERR       INFO
...
```

tcpdump {namespace/podname} [tcpdump options ...]

This command will enter the `kube-ovn-cni` container on the machine where `namespace/podname` is located, and run `tcpdump` to capture the traffic on the veth NIC of the corresponding container, which can be used to troubleshoot network-related problems.

```
# kubectl ko tcpdump default/ds1-l6n7p icmp
+ kubectl exec -it kube-ovn-cni-wlg4s -n kube-ovn -- tcpdump -nn -i d7176fe7b4e0_h icmp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on d7176fe7b4e0_h, link-type EN10MB (Ethernet), capture size 262144 bytes
06:52:36.619688 IP 100.64.0.3 > 10.16.0.4: ICMP echo request, id 2, seq 1, length 64
06:52:36.619746 IP 10.16.0.4 > 100.64.0.3: ICMP echo reply, id 2, seq 1, length 64
06:52:37.619588 IP 100.64.0.3 > 10.16.0.4: ICMP echo request, id 2, seq 2, length 64
06:52:37.619630 IP 10.16.0.4 > 100.64.0.3: ICMP echo reply, id 2, seq 2, length 64
06:52:38.619933 IP 100.64.0.3 > 10.16.0.4: ICMP echo request, id 2, seq 3, length 64
06:52:38.619973 IP 10.16.0.4 > 100.64.0.3: ICMP echo reply, id 2, seq 3, length 64
```

trace [arguments ...]

This command will print the OVN logical flow table and the final Openflow flow table when the Pod/node accesses an address through a specific protocol, so that it make locate flow table related problems during development or troubleshooting much easy.

Supported commands:

```
kubectl ko trace {namespace/podname} {target ip address} [target mac address] {icmp|tcp|udp} [target tcp/udp port]
kubectl ko trace {namespace/podname} {target ip address} [target mac address] arp {request|reply}
kubectl ko trace {node//nodename} {target ip address} [target mac address] {icmp|tcp|udp} [target tcp/udp port]
kubectl ko trace {node//nodename} {target ip address} [target mac address] arp {request|reply}
```

Example:

```
# kubectl ko trace default/ds1-l6n7p 8.8.8.8 icmp
+ kubectl exec ovn-central-5bc494cb5-np9hm -n kube-ovn -- ovn-trace --ct=new ovn-default 'inport == "ds1-l6n7p.default" && ip.ttl == 64 && icmp && eth.src ==
0a:00:00:10:00:05 && ip4.src == 10.16.0.4 && eth.dst == 00:00:00:B8:CA:43 && ip4.dst == 8.8.8.8'
# icmp,reg14=0xf,vlan_tci=0x0000,dl_src=0a:00:00:10:00:05,dl_dst=00:00:00:b8:ca:
43,nw_src=10.16.0.4,nw_dst=8.8.8.8,nw_tos=0,nw_ecn=0,nw_ttl=64,icmp_type=0,icmp_code=0

ingress(dp="ovn-default", inport="ds1-l6n7p.default")
-----------------------------------------------------
 0. ls_in_port_sec_l2 (ovn-northd.c:4143): inport == "ds1-l6n7p.default" && eth.src == {0a:00:00:10:00:05}, priority 50, uuid 39453393
    next;
 1. ls_in_port_sec_ip (ovn-northd.c:2898): inport == "ds1-l6n7p.default" && eth.src == 0a:00:00:10:00:05 && ip4.src == {10.16.0.4}, priority 90, uuid 81bcd485
    next;
 3. ls_in_pre_acl (ovn-northd.c:3269): ip, priority 100, uuid 7b4f4971
    reg0[0] = 1;
    next;
 5. ls_in_pre_stateful (ovn-northd.c:3396): reg0[0] == 1, priority 100, uuid 36cdd577
    ct_next;

ct_next(ct_state=new|trk)
-------------------------
 6. ls_in_acl (ovn-northd.c:3759): ip && (!ct.est || (ct.est && ct_label.blocked == 1)), priority 1, uuid 7608af5b
    reg0[1] = 1;
    next;
10. ls_in_stateful (ovn-northd.c:3995): reg0[1] == 1, priority 100, uuid 2aba1b90
    ct_commit(ct_label=0/0x1);
    next;
16. ls_in_l2_lkup (ovn-northd.c:4470): eth.dst == 00:00:00:b8:ca:43, priority 50, uuid 5c9c3c9f
    outport = "ovn-default-ovn-cluster";
    output;

...
```

If the trace object is a virtual machine running in Underlay network, additional parameters is needed to specify the destination Mac address.

```
kubectl ko trace default/virt-handler-7lvml 8.8.8.8 82:7c:9f:83:8c:01 icmp
```

diagnose {all|node|subnet|IPPorts} [nodename|subnetName|{proto1}-{IP1}-{Port1},{proto2}-{IP2}-{Port2}]

Diagnose the status of cluster network components and go to the corresponding node's `kube-ovn-pinger` to detect connectivity and network latency from the current node to other nodes and critical services.

```
# kubectl ko diagnose all
switch c7cd17e8-ceee-4a91-9bb3-e5a313fe1ece (snat)
    port snat-ovn-cluster
        type: router
        router-port: ovn-cluster-snat
switch 20e0c6d0-023a-4756-aec5-200e0c60f95d (join)
    port node-liumengxin-ovn3-192.168.137.178
        addresses: ["00:00:00:64:FF:A8 100.64.0.4"]
    port node-liumengxin-ovn1-192.168.137.176
        addresses: ["00:00:00:AF:98:62 100.64.0.2"]
    port join-ovn-cluster
        type: router
        router-port: ovn-cluster-join
switch 0191705c-f827-427b-9de3-3c3b7d971ba5 (central)
    port central-ovn-cluster
        type: router
        router-port: ovn-cluster-central
switch 2a45ff05-388d-4f85-9daf-e6fccd5833dc (ovn-default)
    port ovn-default-ovn-cluster
        type: router
        router-port: ovn-cluster-ovn-default
    port prometheus-k8s-1.monitoring
        addresses: ["00:00:00:AA:37:DF 10.16.0.23"]
router 212f73dd-d63d-4d72-864b-a537e9afbee1 (ovn-cluster)
    port ovn-cluster-snat
        mac: "00:00:00:7A:82:8F"
        networks: ["172.22.0.1/16"]
    port ovn-cluster-join
        mac: "00:00:00:F8:18:5A"
        networks: ["100.64.0.1/16"]
    port ovn-cluster-central
        mac: "00:00:00:4D:8C:F5"
        networks: ["192.101.0.1/16"]
    port ovn-cluster-ovn-default
        mac: "00:00:00:A3:F8:18"
        networks: ["10.16.0.1/16"]
Routing Policies
     31000                            ip4.dst == 10.16.0.0/16           allow
     31000                           ip4.dst == 100.64.0.0/16           allow
     30000                         ip4.dst == 192.168.137.177         reroute                100.64.0.3
     30000                         ip4.dst == 192.168.137.178         reroute                100.64.0.4
     29000                 ip4.src == $ovn.default.fake.6_ip4         reroute               100.64.0.22
     29000                 ip4.src == $ovn.default.fake.7_ip4         reroute               100.64.0.21
     29000                 ip4.src == $ovn.default.fake.8_ip4         reroute               100.64.0.23
     29000 ip4.src == $ovn.default.liumengxin.ovn3.192.168.137.178_ip4         reroute                100.64.0.4
     20000 ip4.src == $ovn.default.liumengxin.ovn1.192.168.137.176_ip4 && ip4.dst != $ovn.cluster.overlay.subnets.IPv4         reroute                100.
64.0.2
     20000 ip4.src == $ovn.default.liumengxin.ovn2.192.168.137.177_ip4 && ip4.dst != $ovn.cluster.overlay.subnets.IPv4         reroute                100.
64.0.3
     20000 ip4.src == $ovn.default.liumengxin.ovn3.192.168.137.178_ip4 && ip4.dst != $ovn.cluster.overlay.subnets.IPv4         reroute                100.
64.0.4
IPv4 Routes
Route Table <main>:
                0.0.0.0/0                100.64.0.1 dst-ip
UUID                                    LB                  PROTO      VIP                     IPs
e9bcfd9d-793e-4431-9073-6dec96b75d71    cluster-tcp-load    tcp        10.100.209.132:10660    192.168.137.176:10660
                                                            tcp        10.101.239.192:6641     192.168.137.177:6641
                                                            tcp        10.101.240.101:3000     10.16.0.7:3000
                                                            tcp        10.103.184.186:6642     192.168.137.177:6642
35d2b7a5-e3a7-485a-a4b7-b4970eb0e63b    cluster-tcp-sess    tcp        10.100.158.128:8080     10.16.0.10:8080,10.16.0.5:8080,10.16.63.30:8080
                                                            tcp        10.107.26.215:8080      10.16.0.19:8080,10.16.0.20:8080,10.16.0.21:8080
                                                            tcp        10.107.26.215:9093      10.16.0.19:9093,10.16.0.20:9093,10.16.0.21:9093
                                                            tcp        10.98.187.99:8080       10.16.0.22:8080,10.16.0.23:8080
                                                            tcp        10.98.187.99:9090       10.16.0.22:9090,10.16.0.23:9090
f43303e4-89aa-4d3e-a3dc-278a552fe27b    cluster-udp-load    udp        10.96.0.10:53           10.16.0.4:53,10.16.0.9:53
_uuid               : 06776304-5a96-43ed-90c4-c4854c251699
addresses           : []
external_ids        : {vendor=kube-ovn}
name                : node_liumengxin_ovn2_192.168.137.177_underlay_v6

_uuid               : 62690625-87d5-491c-8675-9fd83b1f433c
addresses           : []
external_ids        : {vendor=kube-ovn}
name                : node_liumengxin_ovn1_192.168.137.176_underlay_v6

_uuid               : b03a9bae-94d5-4562-b34c-b5f6198e180b
addresses           : ["10.16.0.0/16", "100.64.0.0/16", "172.22.0.0/16", "192.101.0.0/16"]
external_ids        : {vendor=kube-ovn}
```

```
name                : ovn.cluster.overlay.subnets.IPv4

_uuid               : e1056f3a-24cc-4666-8a91-75ee6c3c2426
addresses           : []
external_ids        : {vendor=kube-ovn}
name                : ovn.cluster.overlay.subnets.IPv6

_uuid               : 3e5d5fff-e670-47b2-a2f5-a39f4698a8c5
addresses           : []
external_ids        : {vendor=kube-ovn}
name                : node_liumengxin_ovn3_192.168.137.178_underlay_v6
_uuid               : 2d85dbdc-d0db-4abe-b19e-cc806d32b492
action              : drop
direction           : from-lport
external_ids        : {}
label               : 0
log                 : false
match               : "inport==@ovn.sg.kubeovn_deny_all && ip"
meter               : []
name                : []
options             : {}
priority            : 2003
severity            : []

_uuid               : de790cc8-f155-405f-bb32-5a51f30c545f
action              : drop
direction           : to-lport
external_ids        : {}
label               : 0
log                 : false
match               : "outport==@ovn.sg.kubeovn_deny_all && ip"
meter               : []
name                : []
options             : {}
priority            : 2003
severity            : []
Chassis "e15ed4d4-1780-4d50-b09e-ea8372ed48b8"
    hostname: liumengxin-ovn1-192.168.137.176
    Encap stt
        ip: "192.168.137.176"
        options: {csum="true"}
    Port_Binding node-liumengxin-ovn1-192.168.137.176
    Port_Binding perf-6vxkn.default
    Port_Binding kube-state-metrics-5d6885d89-4nf8h.monitoring
    Port_Binding alertmanager-main-0.monitoring
    Port_Binding kube-ovn-pinger-6ftdf.kube-system
    Port_Binding fake-kubelet-67c55dfd89-pv86k.kube-system
    Port_Binding prometheus-k8s-0.monitoring
Chassis "eef07da1-f8ad-4775-b14d-bd6a3b4eb0d5"
    hostname: liumengxin-ovn3-192.168.137.178
    Encap stt
        ip: "192.168.137.178"
        options: {csum="true"}
    Port_Binding kube-ovn-pinger-7twb4.kube-system
    Port_Binding prometheus-adapter-86df476d87-rl88g.monitoring
    Port_Binding prometheus-k8s-1.monitoring
    Port_Binding node-liumengxin-ovn3-192.168.137.178
    Port_Binding perf-ff475.default
    Port_Binding alertmanager-main-1.monitoring
    Port_Binding blackbox-exporter-676d976865-tvsjd.monitoring
Chassis "efa253c9-494d-4719-83ae-b48ab0f11c03"
    hostname: liumengxin-ovn2-192.168.137.177
    Encap stt
        ip: "192.168.137.177"
        options: {csum="true"}
    Port_Binding grafana-6c4c6b8fb7-pzd2c.monitoring
    Port_Binding node-liumengxin-ovn2-192.168.137.177
    Port_Binding alertmanager-main-2.monitoring
    Port_Binding coredns-6789c94dd8-9jqsz.kube-system
    Port_Binding coredns-6789c94dd8-25d4r.kube-system
    Port_Binding prometheus-operator-7bbc99fc8b-wgjm4.monitoring
    Port_Binding prometheus-adapter-86df476d87-gdxmc.monitoring
    Port_Binding perf-fjnws.default
    Port_Binding kube-ovn-pinger-vh2xg.kube-system
ds kube-proxy ready
kube-proxy ready
deployment ovn-central ready
deployment kube-ovn-controller ready
ds kube-ovn-cni ready
ds ovs-ovn ready
deployment coredns ready
ovn-nb leader check ok
ovn-sb leader check ok
ovn-northd leader check ok
### kube-ovn-controller recent log

### start to diagnose node liumengxin-ovn1-192.168.137.176
#### ovn-controller log:
2022-06-03T00:56:44.897Z|16722|inc_proc_eng|INFO|User triggered force recompute.
2022-06-03T01:06:44.912Z|16723|inc_proc_eng|INFO|User triggered force recompute.
2022-06-03T01:16:44.925Z|16724|inc_proc_eng|INFO|User triggered force recompute.
2022-06-03T01:26:44.936Z|16725|inc_proc_eng|INFO|User triggered force recompute.
2022-06-03T01:36:44.959Z|16726|inc_proc_eng|INFO|User triggered force recompute.
2022-06-03T01:46:44.974Z|16727|inc_proc_eng|INFO|User triggered force recompute.
```

```
2022-06-03T01:56:44.988Z|16728|inc_proc_eng|INFO|User triggered force recompute.
2022-06-03T02:06:45.001Z|16729|inc_proc_eng|INFO|User triggered force recompute.
2022-06-03T02:16:45.025Z|16730|inc_proc_eng|INFO|User triggered force recompute.
2022-06-03T02:26:45.040Z|16731|inc_proc_eng|INFO|User triggered force recompute.

#### ovs-vswitchd log:
2022-06-02T23:03:00.137Z|00079|dpif(handler1)|WARN|system@ovs-system: execute ct(commit,zone=14,label=0/0x1,nat(src)),8 failed (Invalid argument) on packet
icmp,vlan_tci=0x0000,dl_src=00:00:00:f8:07:c8,dl_dst=00:00:00:fa:1e:50,nw_src=10.16.0.5,nw_dst=10.16.0.10,nw_tos=0,nw_ecn=0,nw_ttl=64,icmp_type=8,icmp_code=0
icmp_csum:f9d1
 with metadata skb_priority(0),tunnel(tun_id=0x160017000004,src=192.168.137.177,dst=192.168.137.176,ttl=64,tp_src=38881,tp_dst=7471,flags(csum|
key)),skb_mark(0),ct_state(0x21),ct_zone(0xe),ct_tuple4(src=10.16.0.5,dst=10.16.0.10,proto=1,tp_src=8,tp_dst=0),in_port(4) mtu 0
2022-06-02T23:23:31.840Z|00080|dpif(handler1)|WARN|system@ovs-system: execute ct(commit,zone=14,label=0/0x1,nat(src)),8 failed (Invalid argument) on packet
icmp,vlan_tci=0x0000,dl_src=00:00:00:f8:07:c8,dl_dst=00:00:00:fa:1e:50,nw_src=10.16.0.5,nw_dst=10.16.0.10,nw_tos=0,nw_ecn=0,nw_ttl=64,icmp_type=8,icmp_code=0
icmp_csum:15b2
 with metadata skb_priority(0),tunnel(tun_id=0x160017000004,src=192.168.137.177,dst=192.168.137.176,ttl=64,tp_src=38881,tp_dst=7471,flags(csum|
key)),skb_mark(0),ct_state(0x21),ct_zone(0xe),ct_tuple4(src=10.16.0.5,dst=10.16.0.10,proto=1,tp_src=8,tp_dst=0),in_port(4) mtu 0
2022-06-03T00:09:15.659Z|00081|dpif(handler1)|WARN|system@ovs-system: execute ct(commit,zone=14,label=0/0x1,nat(src)),8 failed (Invalid argument) on packet
icmp,vlan_tci=0x0000,dl_src=00:00:00:dc:e3:63,dl_dst=00:00:00:fa:1e:50,nw_src=10.16.63.30,nw_dst=10.
16.0.10,nw_tos=0,nw_ecn=0,nw_ttl=64,icmp_type=8,icmp_code=0 icmp_csum:e5a5
 with metadata skb_priority(0),tunnel(tun_id=0x150017000004,src=192.168.137.178,dst=192.168.137.176,ttl=64,tp_src=9239,tp_dst=7471,flags(csum|
key)),skb_mark(0),ct_state(0x21),ct_zone(0xe),ct_tuple4(src=10.16.63.30,dst=10.16.0.10,proto=1,tp_src=8,tp_dst=0),in_port(4) mtu 0
2022-06-03T00:30:13.409Z|00064|dpif(handler2)|WARN|system@ovs-system: execute ct(commit,zone=14,label=0/0x1,nat(src)),8 failed (Invalid argument) on packet
icmp,vlan_tci=0x0000,dl_src=00:00:00:f8:07:c8,dl_dst=00:00:00:fa:1e:50,nw_src=10.16.0.5,nw_dst=10.16.0.10,nw_tos=0,nw_ecn=0,nw_ttl=64,icmp_type=8,icmp_code=0
icmp_csum:6b4a
 with metadata skb_priority(0),tunnel(tun_id=0x160017000004,src=192.168.137.177,dst=192.168.137.176,ttl=64,tp_src=38881,tp_dst=7471,flags(csum|
key)),skb_mark(0),ct_state(0x21),ct_zone(0xe),ct_tuple4(src=10.16.0.5,dst=10.16.0.10,proto=1,tp_src=8,tp_dst=0),in_port(4) mtu 0
2022-06-03T02:02:33.832Z|00082|dpif(handler1)|WARN|system@ovs-system: execute ct(commit,zone=14,label=0/0x1,nat(src)),8 failed (Invalid argument) on packet
icmp,vlan_tci=0x0000,dl_src=00:00:00:f8:07:c8,dl_dst=00:00:00:fa:1e:50,nw_src=10.16.0.5,nw_dst=10.16.0.10,nw_tos=0,nw_ecn=0,nw_ttl=64,icmp_type=8,icmp_code=0
icmp_csum:a819
 with metadata skb_priority(0),tunnel(tun_id=0x160017000004,src=192.168.137.177,dst=192.168.137.176,ttl=64,tp_src=38881,tp_dst=7471,flags(csum|
key)),skb_mark(0),ct_state(0x21),ct_zone(0xe),ct_tuple4(src=10.16.0.5,dst=10.16.0.10,proto=1,tp_src=8,tp_dst=0),in_port(4) mtu 0

#### ovs-vsctl show results:
0d4c4675-c9cc-440a-8c1a-878e17f81b88
    Bridge br-int
        fail_mode: secure
        datapath_type: system
        Port a2c1a8a8b83a_h
            Interface a2c1a8a8b83a_h
        Port "4fa5c4cbb1a5_h"
            Interface "4fa5c4cbb1a5_h"
        Port ovn-eef07d-0
            Interface ovn-eef07d-0
                type: stt
                options: {csum="true", key=flow, remote_ip="192.168.137.178"}
        Port ovn0
            Interface ovn0
                type: internal
        Port "04d03360e9a0_h"
            Interface "04d03360e9a0_h"
        Port eeb4d9e51b5d_h
            Interface eeb4d9e51b5d_h
        Port mirror0
            Interface mirror0
                type: internal
        Port "8e5d887ccd80_h"
            Interface "8e5d887ccd80_h"
        Port ovn-efa253-0
            Interface ovn-efa253-0
                type: stt
                options: {csum="true", key=flow, remote_ip="192.168.137.177"}
        Port "17512d5be1f1_h"
            Interface "17512d5be1f1_h"
        Port br-int
            Interface br-int
                type: internal
    ovs_version: "2.17.2"

#### pinger diagnose results:
I0603 10:35:04.349404   17619 pinger.go:19]
-------------------------------------------------------------------------------
Kube-OVN:
  Version:       v1.14.4
  Build:         2022-04-24_08:02:50
  Commit:        git-73f9d15
  Go Version:    go1.17.8
  Arch:          amd64
-------------------------------------------------------------------------------
I0603 10:35:04.376797   17619 config.go:166] pinger config is &{KubeConfigFile: KubeClient:0xc000493380 Port:8080 DaemonSetNamespace:kube-system
DaemonSetName:kube-ovn-pinger Interval:5 Mode:job ExitCode:0 InternalDNS:kubernetes.default ExternalDNS: NodeName:liumengxin-ovn1-192.168.137.176 HostIP:
192.168.137.176 PodName:kube-ovn-pinger-6ftdf PodIP:10.16.0.10 PodProtocols:[IPv4] ExternalAddress: NetworkMode:kube-ovn PollTimeout:2 PollInterval:15
SystemRunDir:/var/run/openvswitch DatabaseVswitchName:Open_vSwitch DatabaseVswitchSocketRemote:unix:/var/run/openvswitch/db.sock DatabaseVswitchFileDataPath:/
etc/openvswitch/conf.db DatabaseVswitchFileLogPath:/var/log/openvswitch/ovsdb-server.log DatabaseVswitchFilePidPath:/var/run/openvswitch/ovsdb-server.pid
DatabaseVswitchFileSystemIDPath:/etc/openvswitch/system-id.conf ServiceVswitchdFileLogPath:/var/log/openvswitch/ovs-vswitchd.log ServiceVswitchdFilePidPath:/
var/run/openvswitch/ovs-vswitchd.pid ServiceOvnControllerFileLogPath:/var/log/ovn/ovn-controller.log ServiceOvnControllerFilePidPath:/var/run/ovn/ovn-
controller.pid}
I0603 10:35:04.449166   17619 exporter.go:75] liumengxin-ovn1-192.168.137.176: exporter connect successfully
I0603 10:35:04.554011   17619 ovn.go:21] ovs-vswitchd and ovsdb are up
I0603 10:35:04.651293   17619 ovn.go:33] ovn_controller is up
I0603 10:35:04.651342   17619 ovn.go:39] start to check port binding
I0603 10:35:04.749613   17619 ovn.go:135] chassis id is 1d7f3d6c-eec5-4b3c-adca-2969d9cdfd80
I0603 10:35:04.763487   17619 ovn.go:49] port in sb is [node-liumengxin-ovn1-192.168.137.176 perf-6vxkn.default kube-state-metrics-5d6885d89-4nf8h.monitoring
alertmanager-main-0.monitoring kube-ovn-pinger-6ftdf.kube-system fake-kubelet-67c55dfd89-pv86k.kube-system prometheus-k8s-0.monitoring]
I0603 10:35:04.763583   17619 ovn.go:61] ovs and ovn-sb binding check passed
```

```
I0603 10:35:05.049309   17619 ping.go:259] start to check apiserver connectivity
I0603 10:35:05.053666   17619 ping.go:268] connect to apiserver success in 4.27ms
I0603 10:35:05.053786   17619 ping.go:129] start to check pod connectivity
I0603 10:35:05.249590   17619 ping.go:159] ping pod: kube-ovn-pinger-6ftdf 10.16.0.10, count: 3, loss count 0, average rtt 16.30ms
I0603 10:35:05.354135   17619 ping.go:159] ping pod: kube-ovn-pinger-7twb4 10.16.63.30, count: 3, loss count 0, average rtt 1.81ms
I0603 10:35:05.458460   17619 ping.go:159] ping pod: kube-ovn-pinger-vh2xg 10.16.0.5, count: 3, loss count 0, average rtt 1.92ms
I0603 10:35:05.458523   17619 ping.go:83] start to check node connectivity
```

If the target of diagnose is specified as subnet, the script will create a daemonset on the subnet, and `kube-ovn-pinger` will detect the connectivity and network delay of all pods in this daemonset, and automatically destroy the daemonset after the test.

If the target of diagnose is specified as IPPorts, the script will let each `kube-ovn-pinger` pod detect whether the target protocol, IP, and Port are reachable.

tuning {install-fastpath|local-install-fastpath|remove-fastpath|install-stt|local-install-stt|remove-stt} {centos7|centos8}} [kernel-devel-version]

This command performs performance tuning related operations, please refer to Performance Tuning.

reload

This command restarts all Kube-OVN related components:

```
# kubectl ko reload
pod "ovn-central-8684dd94bd-vzgcr" deleted
Waiting for deployment "ovn-central" rollout to finish: 0 of 1 updated replicas are available...
deployment "ovn-central" successfully rolled out
pod "ovs-ovn-bsnvz" deleted
pod "ovs-ovn-m9b98" deleted
pod "kube-ovn-controller-8459db5ff4-64c62" deleted
Waiting for deployment "kube-ovn-controller" rollout to finish: 0 of 1 updated replicas are available...
deployment "kube-ovn-controller" successfully rolled out
pod "kube-ovn-cni-2klnh" deleted
pod "kube-ovn-cni-t2jz4" deleted
Waiting for daemon set "kube-ovn-cni" rollout to finish: 0 of 2 updated pods are available...
Waiting for daemon set "kube-ovn-cni" rollout to finish: 1 of 2 updated pods are available...
daemon set "kube-ovn-cni" successfully rolled out
pod "kube-ovn-pinger-ln72z" deleted
pod "kube-ovn-pinger-w8lrk" deleted
Waiting for daemon set "kube-ovn-pinger" rollout to finish: 0 of 2 updated pods are available...
Waiting for daemon set "kube-ovn-pinger" rollout to finish: 1 of 2 updated pods are available...
daemon set "kube-ovn-pinger" successfully rolled out
pod "kube-ovn-monitor-7fb67d5488-7q6zb" deleted
Waiting for deployment "kube-ovn-monitor" rollout to finish: 0 of 1 updated replicas are available...
deployment "kube-ovn-monitor" successfully rolled out
```

log

Using this command will capture the logs of Kube-OVN, OVN, Open vSwitch on all nodes of kube-ovn and some debug information commonly used in linux.

```
# kubectl ko log all
Collecting kube-ovn logging files
Collecting ovn logging files
Collecting openvswitch logging files
Collecting linux dmesg files
Collecting linux iptables-legacy files
Collecting linux iptables-nft files
Collecting linux route files
Collecting linux link files
Collecting linux neigh files
Collecting linux memory files
Collecting linux top files
Collecting linux sysctl files
Collecting linux netstat files
Collecting linux addr files
Collecting linux ipset files
Collecting linux tcp files
Collected files have been saved in the directory /root/kubectl-ko-log
```

The directory is as follows:

```
# tree kubectl-ko-log/
kubectl-ko-log/
|-- kube-ovn-control-plane
|   |-- kube-ovn
|   |   |-- kube-ovn-cni.log
|   |   |-- kube-ovn-monitor.log
|   |   `-- kube-ovn-pinger.log
|   |-- linux
|   |   |-- addr.log
|   |   |-- dmesg.log
|   |   |-- ipset.log
|   |   |-- iptables-legacy.log
|   |   |-- iptables-nft.log
|   |   |-- link.log
|   |   |-- memory.log
|   |   |-- neigh.log
|   |   |-- netstat.log
|   |   |-- route.log
|   |   |-- sysctl.log
|   |   |-- tcp.log
|   |   `-- top.log
|   |-- openvswitch
|   |   |-- ovs-vswitchd.log
|   |   `-- ovsdb-server.log
|   `-- ovn
|       |-- ovn-controller.log
|       |-- ovn-northd.log
|       |-- ovsdb-server-nb.log
|       `-- ovsdb-server-sb.log
```

perf [image]

This command will test some performance indicators of Kube-OVN as follows:

1. The performance indicators of the container network;
2. Hostnetwork network performance indicators;
3. Container network multicast packet performance indicators;
4. Time required for OVN-NB, OVN-SB, and OVN-Northd leader deletion recovery. The parameter image is used to specify the image used by the performance test pod. By default, it is `kubeovn/test:v1.12.0`. This parameter is mainly set for offline scenarios, and the image name may change when the image is pulled to the intranet environment.

```
# kubectl ko perf
============================== Prepareing Performance Test Resources ===============================
pod/test-client created
pod/test-host-client created
pod/test-server created
pod/test-host-server created
service/test-server created
pod/test-client condition met
pod/test-host-client condition met
pod/test-host-server condition met
pod/test-server condition met
====================================================================================================
============================ Start Pod Network Unicast Performance Test ============================
Size            TCP Latency     TCP Bandwidth   UDP Latency     UDP Lost Rate   UDP Bandwidth
64              82.8 us         97.7 Mbits/sec  67.6 us         (0%)            8.42 Mbits/sec
128             85.4 us         167 Mbits/sec   67.2 us         (0%)            17.2 Mbits/sec
512             85.8 us         440 Mbits/sec   68.7 us         (0%)            68.4 Mbits/sec
1k              85.1 us         567 Mbits/sec   68.7 us         (0%)            134 Mbits/sec
4k              138 us          826 Mbits/sec   78.1 us         (1.4%)          503 Mbits/sec
====================================================================================================
=============================== Start Host Network Performance Test ===============================
Size            TCP Latency     TCP Bandwidth   UDP Latency     UDP Lost Rate   UDP Bandwidth
64              49.7 us         120 Mbits/sec   37.9 us         (0%)            18.6 Mbits/sec
128             49.7 us         200 Mbits/sec   38.1 us         (0%)            35.5 Mbits/sec
512             51.9 us         588 Mbits/sec   38.9 us         (0%)            142 Mbits/sec
1k              51.7 us         944 Mbits/sec   37.2 us         (0%)            279 Mbits/sec
4k              74.9 us         1.66 Gbits/sec  39.9 us         (0%)            1.20 Gbits/sec
====================================================================================================
============================== Start Service Network Performance Test ==============================
Size            TCP Latency     TCP Bandwidth   UDP Latency     UDP Lost Rate   UDP Bandwidth
64              111 us          96.3 Mbits/sec  88.4 us         (0%)            7.59 Mbits/sec
128             83.7 us         150 Mbits/sec   69.2 us         (0%)            16.9 Mbits/sec
512             87.4 us         374 Mbits/sec   75.8 us         (0%)            60.9 Mbits/sec
1k              88.2 us         521 Mbits/sec   73.1 us         (0%)            123 Mbits/sec
4k              148 us          813 Mbits/sec   77.6 us         (0.0044%)       451 Mbits/sec
====================================================================================================
=========================== Start Pod Multicast Network Performance Test ===========================
Size            UDP Latency     UDP Lost Rate   UDP Bandwidth
64              0.014 ms        (0.17%)         5.80 Mbits/sec
128             0.012 ms        (0%)            11.4 Mbits/sec
512             0.016 ms        (0%)            46.1 Mbits/sec
1k              0.023 ms        (0.073%)        89.8 Mbits/sec
4k              0.035 ms        (1.3%)          126 Mbits/sec
====================================================================================================
============================= Start Host Multicast Network Performance =============================
Size            UDP Latency     UDP Lost Rate   UDP Bandwidth
64              0.007 ms        (0%)            9.95 Mbits/sec
128             0.005 ms        (0%)            21.8 Mbits/sec
512             0.008 ms        (0%)            86.8 Mbits/sec
1k              0.013 ms        (0.045%)        168 Mbits/sec
4k              0.010 ms        (0.31%)         242 Mbits/sec
```

```
====================================================================================================
================================== Start Leader Recover Time Test ==================================
Delete ovn central nb pod
pod "ovn-central-5cb9c67d75-tlz9w" deleted
Waiting for ovn central nb pod running
=============================== OVN nb Recovery takes 3.305236803 s ================================
Delete ovn central sb pod
pod "ovn-central-5cb9c67d75-szx4c" deleted
Waiting for ovn central sb pod running
=============================== OVN sb Recovery takes 3.462698535 s ================================
Delete ovn central northd pod
pod "ovn-central-5cb9c67d75-zqmqv" deleted
Waiting for ovn central northd pod running
============================= OVN northd Recovery takes 2.691291403 s ==============================
====================================================================================================
================================= Remove Performance Test Resource =================================
rm -f unicast-test-client.log
rm -f unicast-test-host-client.log
rm -f unicast-test-client.log
kubectl ko nbctl lb-del test-server
rm -f multicast-test-server.log
kubectl exec ovs-ovn-gxdrf -n kube-system -- ip maddr del 01:00:5e:00:00:64 dev eth0
kubectl exec ovs-ovn-h57bf -n kube-system -- ip maddr del 01:00:5e:00:00:64 dev eth0
rm -f multicast-test-host-server.log
pod "test-client" deleted
pod "test-host-client" deleted
pod "test-host-server" deleted
pod "test-server" deleted
service "test-server" deleted
====================================================================================================
```

PDF **Slack** **Support**

July 30, 2025

May 24, 2022

GitHub

6.1.3 Comments

6.2 Delete Work Node

If the node is simply removed from Kubernetes, the `ovn-controller` process running in `ovs-ovn` on the node will periodically connect to `ovn-central` to register relevant network information. This leads to additional resource waste and potential rule conflict risk. Therefore, when removing nodes from within Kubernetes, follow the steps below to ensure that related resources are cleaned up properly.

This document describes the steps to delete a worker node, if you want to change the node where `ovn-central` is located, please refer to Replace ovn-central Node.

6.2.1 Evict Pods on the Node

```
 # kubectl drain kube-ovn-worker --ignore-daemonsets --force
 node/kube-ovn-worker cordoned
 WARNING: ignoring DaemonSet-managed Pods: kube-system/kube-ovn-cni-zt74b, kube-system/kube-ovn-pinger-5rxfs, kube-system/kube-proxy-jpmnm, kube-system/ovs-
ovn-v2kll
 evicting pod kube-system/coredns-64897985d-qsgpt
 evicting pod local-path-storage/local-path-provisioner-5ddd94ff66-llss6
 evicting pod kube-system/kube-ovn-controller-8459db5ff4-94lxb
 pod/kube-ovn-controller-8459db5ff4-94lxb evicted
 pod/coredns-64897985d-qsgpt evicted
 pod/local-path-provisioner-5ddd94ff66-llss6 evicted
 node/kube-ovn-worker drained
```

6.2.2 Stop kubelet and docker

This step stops the `ovs-ovn` container to avoid registering information to `ovn-central`. Log into to the corresponding node and ruu the following commands:

```
systemctl stop kubelet
systemctl stop docker
```

If using containerd as the CRI, the following command needs to be executed to stop the `ovs-ovn` container:

```
crictl rm -f $(crictl ps | grep openvswitch | awk '{print $1}')
```

6.2.3 Cleanup Files on Node

```
rm -rf /var/run/openvswitch
rm -rf /var/run/ovn
rm -rf /etc/origin/openvswitch/
rm -rf /etc/origin/ovn/
rm -rf /etc/cni/net.d/00-kube-ovn.conflist
rm -rf /etc/cni/net.d/01-kube-ovn.conflist
rm -rf /var/log/openvswitch
rm -rf /var/log/ovn
```

6.2.4 Delete the Node

```
kubectl delete no kube-ovn-01
```

6.2.5 Check If Node Removed from OVN-SB

In the example below, the node `kube-ovn-worker` is not removed:

```
# kubectl ko sbctl show
Chassis "b0564934-5a0d-4804-a4c0-476c93596a17"
  hostname: kube-ovn-worker
  Encap geneve
      ip: "172.18.0.2"
      options: {csum="true"}
  Port_Binding kube-ovn-pinger-5rxfs.kube-system
Chassis "6a29de7e-d731-4eaf-bacd-2f239ee52b28"
  hostname: kube-ovn-control-plane
  Encap geneve
      ip: "172.18.0.3"
```

```
        options: {csum="true"}
    Port_Binding coredns-64897985d-nbfln.kube-system
    Port_Binding node-kube-ovn-control-plane
    Port_Binding local-path-provisioner-5ddd94ff66-h4tn9.local-path-storage
    Port_Binding kube-ovn-pinger-hf2p6.kube-system
    Port_Binding coredns-64897985d-fhwlw.kube-system
```

6.2.6 Delete the Chassis Manually

Use the uuid find above to delete the chassis:

```
# kubectl ko sbctl chassis-del b0564934-5a0d-4804-a4c0-476c93596a17
# kubectl ko sbctl show
Chassis "6a29de7e-d731-4eaf-bacd-2f239ee52b28"
  hostname: kube-ovn-control-plane
  Encap geneve
      ip: "172.18.0.3"
      options: {csum="true"}
  Port_Binding coredns-64897985d-nbfln.kube-system
  Port_Binding node-kube-ovn-control-plane
  Port_Binding local-path-provisioner-5ddd94ff66-h4tn9.local-path-storage
  Port_Binding kube-ovn-pinger-hf2p6.kube-system
  Port_Binding coredns-64897985d-fhwlw.kube-system
```

PDF Slack Support

July 30, 2025

June 3, 2022

GitHub

6.2.7 Comments

6.3 Replace ovn-central Node

Since `ovn-nb` and `ovn-sb` within `ovn-central` create separate etcd-like raft clusters, replacing the `ovn-central` node requires additional operations to ensure correct cluster state and consistent data. It is recommended that only one node be up and down at a time to avoid the cluster going into an unavailable state and affecting the overall cluster network.

6.3.1 ovn-central Nodes Offline

This document use the cluster below to describes how to remove the `kube-ovn-control-plane2` node from the `ovn-central` as an example.

```
# kubectl -n kube-system get pod -o wide | grep central
ovn-central-6bf58cbc97-2cdhg                      1/1     Running   0             21m   172.18.0.3   kube-ovn-control-plane    <none>           <none>
ovn-central-6bf58cbc97-crmfp                      1/1     Running   0             21m   172.18.0.5   kube-ovn-control-plane2   <none>           <none>
ovn-central-6bf58cbc97-lxmpl                      1/1     Running   0             21m   172.18.0.4   kube-ovn-control-plane3   <none>           <none>
```

Kick Node in ovn-nb

First check the ID of the node within the cluster for subsequent operations.

```
# kubectl ko nb status
1b9a
Name: OVN_Northbound
Cluster ID: 32ca (32ca07fb-739b-4257-b510-12fa18e7cce8)
Server ID: 1b9a (1b9a5d76-e69b-410c-8085-39943d0cd38c)
Address: tcp:[172.18.0.3]:6643
Status: cluster member
Role: leader
Term: 1
Leader: self
Vote: self

Last Election started 2135194 ms ago, reason: timeout
Last Election won: 2135188 ms ago
Election timer: 5000
Log: [135, 135]
Entries not yet committed: 0
Entries not yet applied: 0
Connections: <-d64b ->d64b <-4984 ->4984
Disconnections: 0
Servers:
    4984 (4984 at tcp:[172.18.0.4]:6643) next_index=135 match_index=134 last msg 1084 ms ago
    1b9a (1b9a at tcp:[172.18.0.3]:6643) (self) next_index=2 match_index=134
    d64b (d64b at tcp:[172.18.0.5]:6643) next_index=135 match_index=134 last msg 1084 ms ago
status: ok
```

`kube-ovn-control-plane2` corresponds to a node IP of `172.18.0.5` and the corresponding ID within the cluster is `d64b`. Next, kick the node out of the ovn-nb cluster.

```
# kubectl ko nb kick d64b
started removal
```

Check if the node has been kicked:

```
# kubectl ko nb status
1b9a
Name: OVN_Northbound
Cluster ID: 32ca (32ca07fb-739b-4257-b510-12fa18e7cce8)
Server ID: 1b9a (1b9a5d76-e69b-410c-8085-39943d0cd38c)
Address: tcp:[172.18.0.3]:6643
Status: cluster member
Role: leader
Term: 1
Leader: self
Vote: self

Last Election started 2297649 ms ago, reason: timeout
Last Election won: 2297643 ms ago
Election timer: 5000
Log: [136, 136]
Entries not yet committed: 0
Entries not yet applied: 0
Connections: <-4984 ->4984
Disconnections: 2
Servers:
    4984 (4984 at tcp:[172.18.0.4]:6643) next_index=136 match_index=135 last msg 1270 ms ago
    1b9a (1b9a at tcp:[172.18.0.3]:6643) (self) next_index=2 match_index=135
status: ok
```

Kick Node in ovn-sb

Next, for the ovn-sb cluster, you need to first check the ID of the node within the cluster for subsequent operations.

```
kubectl ko sb status
3722
Name: OVN_Southbound
Cluster ID: d4bd (d4bd37a4-0400-499f-b4df-b4fd389780f0)
Server ID: 3722 (3722d5ae-2ced-4820-a6b2-8b744d11fb3e)
Address: tcp:[172.18.0.3]:6644
Status: cluster member
Role: leader
Term: 1
Leader: self
Vote: self

Last Election started 2395317 ms ago, reason: timeout
Last Election won: 2395316 ms ago
Election timer: 5000
Log: [130, 130]
Entries not yet committed: 0
Entries not yet applied: 0
Connections: <-e9f7 ->e9f7 <-6e84 ->6e84
Disconnections: 0
Servers:
    e9f7 (e9f7 at tcp:[172.18.0.5]:6644) next_index=130 match_index=129 last msg 1006 ms ago
    6e84 (6e84 at tcp:[172.18.0.4]:6644) next_index=130 match_index=129 last msg 1004 ms ago
    3722 (3722 at tcp:[172.18.0.3]:6644) (self) next_index=2 match_index=129
status: ok
```

`kube-ovn-control-plane2` corresponds to node IP `172.18.0.5` and the corresponding ID within the cluster is `e9f7`. Next, kick the node out of the ovn-sb cluster.

```
# kubectl ko sb kick e9f7
started removal
```

Check if the node has been kicked:

```
# kubectl ko sb status
3722
Name: OVN_Southbound
Cluster ID: d4bd (d4bd37a4-0400-499f-b4df-b4fd389780f0)
Server ID: 3722 (3722d5ae-2ced-4820-a6b2-8b744d11fb3e)
Address: tcp:[172.18.0.3]:6644
Status: cluster member
Role: leader
Term: 1
Leader: self
Vote: self

Last Election started 2481636 ms ago, reason: timeout
Last Election won: 2481635 ms ago
Election timer: 5000
Log: [131, 131]
Entries not yet committed: 0
Entries not yet applied: 0
Connections: <-6e84 ->6e84
Disconnections: 2
Servers:
    6e84 (6e84 at tcp:[172.18.0.4]:6644) next_index=131 match_index=130 last msg 642 ms ago
    3722 (3722 at tcp:[172.18.0.3]:6644) (self) next_index=2 match_index=130
status: ok
```

Delete Node Label and Downscale ovn-central

Note that you need to remove the offline node from the node address of the ovn-central environment variable `NODE_IPS`.

```
kubectl label node kube-ovn-control-plane2 kube-ovn/role-
kubectl scale deployment -n kube-system ovn-central --replicas=2
kubectl set env deployment/ovn-central -n kube-system NODE_IPS="172.18.0.3,172.18.0.4"
kubectl rollout status deployment/ovn-central -n kube-system
```

Modify Components Address to ovn-central

Modify `ovs-ovn` to remove the offline Node address:

```
# kubectl set env daemonset/ovs-ovn -n kube-system OVN_DB_IPS="172.18.0.3,172.18.0.4"
daemonset.apps/ovs-ovn env updated
# kubectl delete pod -n kube-system -lapp=ovs
pod "ovs-ovn-4f6jc" deleted
pod "ovs-ovn-csn2w" deleted
pod "ovs-ovn-mpbmb" deleted
```

Modify `kube-ovn-controller` to remove the offline Node address:

```
# kubectl set env deployment/kube-ovn-controller -n kube-system OVN_DB_IPS="172.18.0.3,172.18.0.4"
deployment.apps/kube-ovn-controller env updated

# kubectl rollout status deployment/kube-ovn-controller -n kube-system
Waiting for deployment "kube-ovn-controller" rollout to finish: 1 of 3 updated replicas are available...
Waiting for deployment "kube-ovn-controller" rollout to finish: 2 of 3 updated replicas are available...
deployment "kube-ovn-controller" successfully rolled out
```

Clean Node

Delete the database files in the `kube-ovn-control-plane2` node to avoid errors when adding the node again:

```
rm -rf /etc/origin/ovn
```

To take a node offline from a Kubernetes cluster entirely, please continue with Delete Work Node.

6.3.2 ovn-central Online

The following steps will add a new Kubernetes node to the `ovn-central` cluster.

Directory Check

Check if the `ovnnb_db.db` or `ovnsb_db.db` file exists in the `/etc/origin/ovn` directory of the new node, and if so, delete it:

```
rm -rf /etc/origin/ovn
```

Check Current ovn-central Status

If the current `ovn-central` cluster state is already abnormal, adding new nodes may cause the voting election to fail to pass the majority, affecting subsequent operations.

```
# kubectl ko nb status
1b9a
Name: OVN_Northbound
Cluster ID: 32ca (32ca07fb-739b-4257-b510-12fa18e7cce8)
Server ID: 1b9a (1b9a5d76-e69b-410c-8085-39943d0cd38c)
Address: tcp:[172.18.0.3]:6643
Status: cluster member
Role: leader
Term: 44
Leader: self
Vote: self

Last Election started 1855739 ms ago, reason: timeout
Last Election won: 1855729 ms ago
Election timer: 5000
Log: [147, 147]
Entries not yet committed: 0
Entries not yet applied: 0
Connections: ->4984 <-4984
Disconnections: 0
Servers:
    4984 (4984 at tcp:[172.18.0.4]:6643) next_index=147 match_index=146 last msg 367 ms ago
    1b9a (1b9a at tcp:[172.18.0.3]:6643) (self) next_index=140 match_index=146
status: ok

# kubectl ko sb status
3722
Name: OVN_Southbound
Cluster ID: d4bd (d4bd37a4-0400-499f-b4df-b4fd389780f0)
Server ID: 3722 (3722d5ae-2ced-4820-a6b2-8b744d11fb3e)
Address: tcp:[172.18.0.3]:6644
Status: cluster member
Role: leader
Term: 33
Leader: self
Vote: self
```

```
Last Election started 1868589 ms ago, reason: timeout
Last Election won: 1868579 ms ago
Election timer: 5000
Log: [142, 142]
Entries not yet committed: 0
Entries not yet applied: 0
Connections: ->6e84 <-6e84
Disconnections: 0
Servers:
    6e84 (6e84 at tcp:[172.18.0.4]:6644) next_index=142 match_index=141 last msg 728 ms ago
    3722 (3722 at tcp:[172.18.0.3]:6644) (self) next_index=134 match_index=141
status: ok
```

Label Node and Scale ovn-central

Note that you need to add the online node address to the node address of the ovn-central environment variable `NODE_IPS`.

```
kubectl label node kube-ovn-control-plane2 kube-ovn/role=master
kubectl scale deployment -n kube-system ovn-central --replicas=3
kubectl set env deployment/ovn-central -n kube-system NODE_IPS="172.18.0.3,172.18.0.4,172.18.0.5"
kubectl rollout status deployment/ovn-central -n kube-system
```

Modify Components Address to ovn-central

Modify `ovs-ovn` to add the online Node address:

```
# kubectl set env daemonset/ovs-ovn -n kube-system OVN_DB_IPS="172.18.0.3,172.18.0.4,172.18.0.5"
daemonset.apps/ovs-ovn env updated
# kubectl delete pod -n kube-system -lapp=ovs
pod "ovs-ovn-4f6jc" deleted
pod "ovs-ovn-csn2w" deleted
pod "ovs-ovn-mpbmb" deleted
```

Modify `kube-ovn-controller` to add the online Node address:

```
# kubectl set env deployment/kube-ovn-controller -n kube-system OVN_DB_IPS="172.18.0.3,172.18.0.4,172.18.0.5"
deployment.apps/kube-ovn-controller env updated

# kubectl rollout status deployment/kube-ovn-controller -n kube-system
Waiting for deployment "kube-ovn-controller" rollout to finish: 1 of 3 updated replicas are available...
Waiting for deployment "kube-ovn-controller" rollout to finish: 2 of 3 updated replicas are available...
deployment "kube-ovn-controller" successfully rolled out
```

PDF Slack Support

February 15, 2023

May 24, 2022

GitHub

6.3.3 Comments

6.4 OVN DB Backup and Recovery

This document describes how to perform database backups and how to perform cluster recovery from existing database files in different situations.

6.4.1 Database Backup

The database files can be backed up for recovery in case of failure. Use the backup command of the kubectl plugin:

```
# kubectl ko nb backup
tar: Removing leading `/' from member names
backup ovn-nb db to /root/ovnnb_db.060223191654183154.backup

# kubectl ko sb backup
tar: Removing leading `/' from member names
backup ovn-nb db to /root/ovnsb_db.060223191654183154.backup
```

6.4.2 Cluster Partial Nodes Failure Recovery

If some nodes in the cluster are working abnormally due to power failure, file system failure or lack of disk space, but the cluster is still working normally, you can recover it by following the steps below.

Check the Logs to Confirm Status

Check the log in `/var/log/ovn/ovn-northd.log`, if it shows similar error as follows, you can make sure that there is an exception in the database:

```
 * ovn-northd is not running
ovsdb-server: ovsdb error: error reading record 2739 from OVN_Northbound log: record 2739 advances commit index to 6308 but last log index is 6307
 * Starting ovsdb-nb
```

Kick Node from Cluster

Select the corresponding database for the operation based on whether the log prompt is `OVN_Northbound` or `OVN_Southbound`. The above log prompt is `OVN_Northbound` then for ovn-nb do the following:

```
# kubectl ko nb status
9182
Name: OVN_Northbound
Cluster ID: e75f (e75fa340-49ed-45ab-990e-26cb865ebc85)
Server ID: 9182 (9182e8dd-b5b0-4dd8-8518-598cc1e374f3)
Address: tcp:[10.0.128.61]:6643
Status: cluster member
Role: leader
Term: 1454
Leader: self
Vote: self

Last Election started 1732603 ms ago, reason: timeout
Last Election won: 1732587 ms ago
Election timer: 1000
Log: [7332, 12512]
Entries not yet committed: 1
Entries not yet applied: 1
Connections: ->f080 <-f080 <-e631 ->e631
Disconnections: 1
Servers:
    f080 (f080 at tcp:[10.0.129.139]:6643) next_index=12512 match_index=12510 last msg 63 ms ago
    9182 (9182 at tcp:[10.0.128.61]:6643) (self) next_index=10394 match_index=12510
    e631 (e631 at tcp:[10.0.131.173]:6643) next_index=12512 match_index=0
```

Kick abnormal nodes from the cluster:

```
kubectl ko nb kick e631
```

Log in to the abnormal node and delete the database file:

```
mv /etc/origin/ovn/ovnnb_db.db /tmp
```

Delete the `ovn-central` pod of the corresponding node and wait for the cluster to recover:

```
kubectl delete pod -n kube-system ovn-central-xxxx
```

6.4.3 Recover when Total Cluster Failed

If the majority of the cluster nodes are broken and the leader cannot be elected, please refer to the following steps to recover.

Stop ovn-central

Record the current replicas of `ovn-central` and stop `ovn-central` to avoid new database changes that affect recovery:

```
kubectl scale deployment -n kube-system ovn-central --replicas=0
```

Select a Backup

As most of the nodes are damaged, the cluster needs to be rebuilt by recovering from one of the database files. If you have previously backed up the database you can use the previous backup file to restore it. If not you can use the following steps to generate a backup from an existing file.

Since the database file in the default folder is a cluster format database file containing information about the current cluster, you can't rebuild the database directly with this file, you need to use `ovsdb-tool cluster-to-standalone` to convert the format.

Select the first node in the `ovn-central` environment variable `NODE_IPS` to restore the database files. If the database file of the first node is corrupted, copy the file from the other machine `/etc/origin/ovn` to the first machine. Run the following command to generate a database file backup.

```
docker run -it -v /etc/origin/ovn:/etc/ovn kubeovn/kube-ovn:v1.14.4 bash
cd /etc/ovn/
ovsdb-tool cluster-to-standalone ovnnb_db_standalone.db ovnnb_db.db
ovsdb-tool cluster-to-standalone ovnsb_db_standalone.db ovnsb_db.db
```

Delete the Database Files on All ovn-central Nodes

In order to avoid rebuilding the cluster with the wrong data, the existing database files need to be cleaned up:

```
mv /etc/origin/ovn/ovnnb_db.db /tmp
mv /etc/origin/ovn/ovnsb_db.db /tmp
```

Recovering Database Cluster

Rename the backup databases to `ovnnb_db.db` and `ovnsb_db.db` respectively, and copy them to the `/etc/origin/ovn/` directory of the first machine in the `ovn-central` environment variable `NODE_IPS`:

```
mv /etc/origin/ovn/ovnnb_db_standalone.db /etc/origin/ovn/ovnnb_db.db
mv /etc/origin/ovn/ovnsb_db_standalone.db /etc/origin/ovn/ovnsb_db.db
```

Restore the number of replicas of `ovn-central`:

```
kubectl scale deployment -n kube-system ovn-central --replicas=3
kubectl rollout status deployment/ovn-central -n kube-system
```

PDF Slack Support

July 30, 2025

May 24, 2022

GitHub

6.4.4 Comments

6.5 Change Subnet CIDR

If a subnet CIDR is created that conflicts or does not meet expectations, it can be modified by following the steps in this document.

After modifying the subnet CIDR, the previously created Pods will not be able to access the network properly and need to be rebuilt. Careful consideration is recommended before operating. This document is only for business subnet CIDR changes, if you need to Change the Join subnet CIDR, please refer to Change Join CIDR.

6.5.1 Edit Subnet

Use `kubectl edit` to modify `cidrBlock`, `gateway` and `excludeIps`.

```
kubectl edit subnet test-subnet
```

6.5.2 Rebuild all Pods under this Subnet

Take the subnet binding `test` Namespace as example:

```
for pod in $(kubectl get pod --no-headers -n "$ns" --field-selector spec.restartPolicy=Always -o custom-columns=NAME:.metadata.name,HOST:spec.hostNetwork |
awk '{if ($2!="true") print $1}'); do
  kubectl delete pod "$pod" -n test --ignore-not-found
done
```

If only the default subnet is used, you can delete all Pods that are not in host network mode using the following command:

```
for ns in $(kubectl get ns --no-headers -o custom-columns=NAME:.metadata.name); do
  for pod in $(kubectl get pod --no-headers -n "$ns" --field-selector spec.restartPolicy=Always -o custom-columns=NAME:.metadata.name,HOST:spec.hostNetwork |
awk '{if ($2!="true") print $1}'); do
    kubectl delete pod "$pod" -n "$ns" --ignore-not-found
  done
done
```

6.5.3 Change Default Subnet Settings

If you are modifying the CIDR for the default Subnet, you also need to change the args of the `kube-ovn-controller` Deployment:

```
args:
- --default-cidr=10.17.0.0/16
- --default-gateway=10.17.0.1
- --default-exclude-ips=10.17.0.1
```

PDF **Slack** **Support**

July 30, 2025

May 24, 2022

GitHub

6.5.4 Comments

6.6 Change Join Subnet CIDR

If the Join subnet CIDR created conflicts or does not meet expectations, you can use this document to modify.

After modifying the Join Subnet CIDR, the previously created Pods will not be able to access the external network normally and need to wait for the rebuild completed.

6.6.1 Delete Join Subnet

```
kubectl patch subnet join --type='json' -p '[{"op": "replace", "path": "/metadata/finalizers", "value": []}]'
kubectl delete subnet join
```

6.6.2 Cleanup Allocated Config

```
kubectl annotate node ovn.kubernetes.io/allocated=false --all --overwrite
```

6.6.3 Modify Join Subnet

Change Join Subnet args in `kube-ovn-controller`:

```
kubectl edit deployment -n kube-system kube-ovn-controller
```

Change the CIDR below:

```
args:
- --node-switch-cidr=100.51.0.0/16
```

Reboot the `kube-ovn-controller` and rebuild `join` Subnet:

```
kubectl delete pod -n kube-system -lapp=kube-ovn-controller
```

Check the new Join Subnet information:

```
# kubectl get subnet
NAME         PROVIDER    VPC           PROTOCOL    CIDR            PRIVATE    NAT      DEFAULT    GATEWAYTYPE    V4USED    V4AVAILABLE    V6USED    V6AVAILABLE
EXCLUDEIPS
join         ovn         ovn-cluster   IPv4        100.51.0.0/16   false      false    false      distributed    2         65531          0         0
["100.51.0.1"]
ovn-default  ovn         ovn-cluster   IPv4        10.17.0.0/16    false      true     true       distributed    5         65528          0         0
["10.17.0.1"]
```

6.6.4 Reconfigure ovn0 NIC Address

The `ovn0` NIC information for each node needs to be re-updated, which can be done by restarting `kube-ovn-cni`:

```
kubectl delete pod -n kube-system -l app=kube-ovn-cni
```

PDF **Slack** **Support**

February 15, 2023

May 24, 2022

GitHub

6.6.5 Comments

6.7 Change Log Level

Open `kube-ovn.yaml` and set the log level in the parameter list of the service startup script, such as:

```
vi kube-ovn.yaml
# ...
        - name: kube-ovn-controller
          image: "docker.io/kubeovn/kube-ovn:v1.14.4"
          imagePullPolicy: IfNotPresent
          args:
          - /kube-ovn/start-controller.sh
          - --v=3
# ...
# The higher the log level, the more detailed the log
```

PDF Slack Support

May 9, 2023

April 4, 2023

GitHub

6.7.1 Comments

6.8 FAQ

6.8.1 Kylin ARM system cross-host container access intermittently fails

Behavior

There is a problem with Kylin ARM system and some NIC offload, which can cause intermittent container network failure.

Use `netstat` to identify the problem:

```
# netstat -us
IcmpMsg:
    InType0: 22
    InType3: 24
    InType8: 117852
    OutType0: 117852
    OutType3: 29
    OutType8: 22
Udp:
    3040636 packets received
    0 packets to unknown port received.
    4 packet receive errors
    602 packets sent
    0 receive buffer errors
    0 send buffer errors
    InCsumErrors: 4
UdpLite:
IpExt:
    InBcastPkts: 10244
    InOctets: 4446320361
    OutOctets: 1496815600
    InBcastOctets: 3095950
    InNoECTPkts: 7683903
```

If `InCsumErrors` is present and increases with network failures, you can confirm that this is the problem.

Solution

The fundamental solution requires communication with Kylin and the corresponding network card manufacturer to update the system and drivers. A temporary solution would be to turn off `tx offload` on the physical NIC, but this would cause a significant degradation in tcp performance.

```
ethtool -K eth0 tx off
```

From the community feedback, the problem can be solved by the `4.19.90-25.16.v2101` kernel.

6.8.2 Pod can not Access Service

Behavior

Pod can not access Service, and `dmesg` show errors:

```
netlink Unknown conntrack attr (type=6, max=5)
openvswitch: netlink: Flow actions may not be safe on all matching packets.
```

This log indicates that the in-kernel OVS version is too low to support the corresponding NAT operation.

Solution

1. Upgrade the kernel module or compile the OVS kernel module manually.
2. If you are using an Overlay network you can change the `kube-ovn-controller` args, setting `--enable-lb=false` to disable the OVN LB to use kube-proxy for service forwarding.

6.8.3 Frequent leader selection occurs in ovn-central

Behavior

Starting from the v1.11.x version, in a cluster with 1w Pod or more, if OVN NB or SB frequently elects the master, the possible reason is that Kube-OVN periodically performs the ovsdb-server/compact action, which affects the master selection logic.

Solution

You can configure the environment variables for ovn-central as follows and turn off compact:

```
- name: ENABLE_COMPACT
  value: "false"
```

PDF **Slack** **Support**

July 30, 2025

May 24, 2022

GitHub

6.8.4 Comments

7. Advanced Features

7.1 Manage Multiple Interface

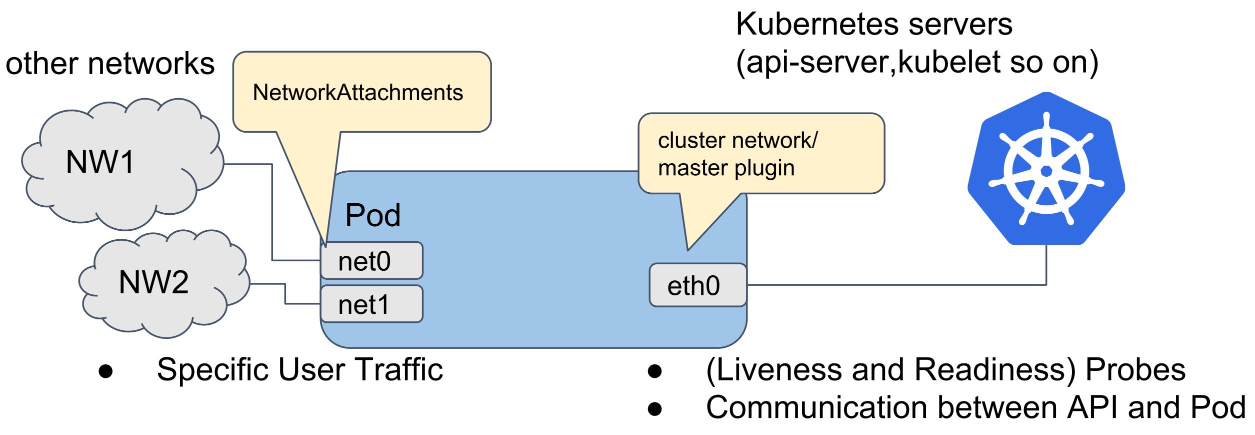
Kube-OVN can provide cluster-level IPAM capabilities for other CNI network plugins such as macvlan, vlan, host-device, etc. Other network plugins can then use the subnet and fixed IP capabilities in Kube-OVN.

Kube-OVN also supports address management when multiple NICs are all of Kube-OVN type.

7.1.1 Working Principle

Multi-nic management:

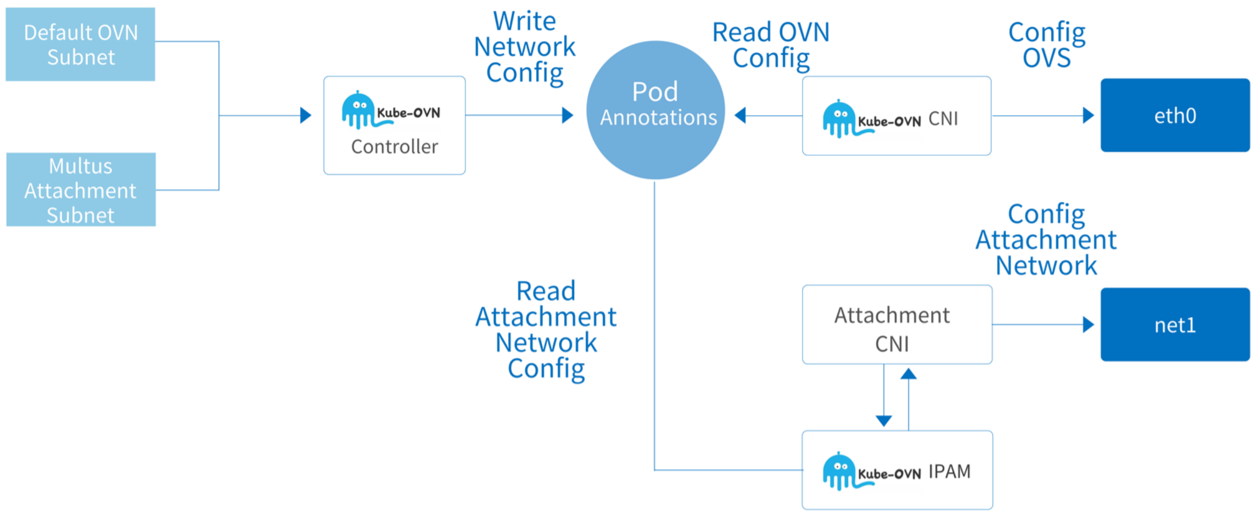
Here's an illustration of the network interfaces attached to a pod, as provisioned by Multus CNI. The diagram shows the pod with three interfaces: eth0, net0 and net1. eth0 connects kubernetes cluster network to connect with kubernetes server/services (e.g. kubernetes api-server, kubelet and so on). net0 and net1 are additional network attachments and connect to other networks by using other CNI plugins (e.g. vlan/vxlan/ptp).



IPAM:

By using Multus CNI, we can add multiple NICs of different networks to a Pod. However, we still lack the ability to manage the IP addresses of different networks within a cluster. In Kube-OVN, we have been able to perform advanced IP management such as subnet management, IP reservation, random assignment, fixed assignment, etc. through CRD of Subnet and IP. Now Kube-OVN extend the subnet to integrate with other different network plugins, so that other network plugins can also use the IPAM functionality of Kube-OVN.

Workflow



The above diagram shows how to manage the IP addresses of other network plugins via Kube-OVN. The eth0 NIC of the container is connected to the OVN network and the net1 NIC is connected to other CNI networks. The network definition for the net1 network is taken from the NetworkAttachmentDefinition resource definition in multus-cni.

When a Pod is created, `kube-ovn-controller` will get the Pod add event, find the corresponding Subnet according to the annotation in the Pod, then manage the address from it, and write the address information assigned to the Pod back to the Pod annotation.

The CNI on the container machine can configure `kube-ovn-cni` as the ipam plugin. `kube-ovn-cni` will read the Pod annotation and return the address information to the corresponding CNI plugin using the standard format of the CNI protocol.

7.1.2 Usage

Install Kube-OVN and Multus

Please refer One-Click Installation and Multus how to use to install Kube-OVN and Multus-CNI.

Provide IPAM for other types of CNI

CREATE NETWORKATTACHMENTDEFINITION

Here we use macvlan as the second network of the container network and set its ipam to `kube-ovn`:

```
# load macvlan module
sudo modprobe macvlan
```

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: macvlan
  namespace: default
spec:
  config: '{
      "cniVersion": "0.3.0",
      "type": "macvlan",
      "master": "eth0",
      "mode": "bridge",
      "ipam": {
        "type": "kube-ovn",
        "server_socket": "/run/openvswitch/kube-ovn-daemon.sock",
        "provider": "macvlan.default"
      }
    }'
```

- `spec.config.ipam.type` : Need to be set to `kube-ovn` to call the kube-ovn plugin to get the address information.
- `server_socket` : The socket file used for communication to Kube-OVN. The default location is `/run/openvswitch/kube-ovn-daemon.sock` .
- `provider` : The current NetworkAttachmentDefinition's `<name>. <namespace>` , Kube-OVN will use this information to find the corresponding Subnet resource.
- `master` : the host's physical network card

CREATE A KUBE-OVN SUBNET

Create a Kube-OVN Subnet, set the corresponding `cidrBlock` and `exclude_ips` , the `provider` should be set to the `<name>. <namespace>` of corresponding NetworkAttachmentDefinition. For example, to provide additional NICs with macvlan, create a Subnet as follows:

```
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
  name: macvlan
spec:
  protocol: IPv4
  provider: macvlan.default
  cidrBlock: 172.17.0.0/16
  gateway: 172.17.0.1
  excludeIps:
  - 172.17.0.0..172.17.0.10
```

`gateway` , `private` , `nat` are only valid for networks with `provider` type ovn, not for attachment networks.

Create a Pod with Multiple NIC

For Pods with randomly assigned addresses, simply add the following annotation `k8s.v1.cni.cncf.io/networks` , taking the value `<namespace>/<name>` of the corresponding NetworkAttachmentDefinition:

```
apiVersion: v1
kind: Pod
metadata:
  name: samplepod
  namespace: default
  annotations:
    k8s.v1.cni.cncf.io/networks: default/macvlan
spec:
  containers:
  - name: samplepod
    command: ["/bin/ash", "-c", "trap : TERM INT; sleep infinity & wait"]
    image: docker.io/library/alpine:edge
```

Create Pod with a Fixed IP

For Pods with fixed IPs, add `<networkAttachmentName>.<networkAttachmentNamespace>.kubernetes.io/ip_address` annotation:

```
apiVersion: v1
kind: Pod
metadata:
  name: static-ip
  namespace: default
  annotations:
    k8s.v1.cni.cncf.io/networks: default/macvlan
    ovn.kubernetes.io/ip_address: 10.16.0.15
    ovn.kubernetes.io/mac_address: 00:00:00:53:6B:B6
    macvlan.default.kubernetes.io/ip_address: 172.17.0.100
    macvlan.default.kubernetes.io/mac_address: 00:00:00:53:6B:BB
spec:
  containers:
  - name: static-ip
    image: docker.io/library/nginx:alpine
```

Create Workloads with Fixed IPs

For workloads that use ippool, add `<networkAttachmentName>.<networkAttachmentNamespace>.kubernetes.io/ip_pool` annotations:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  namespace: default
  name: static-workload
  labels:
    app: static-workload
spec:
  replicas: 2
  selector:
    matchLabels:
      app: static-workload
  template:
    metadata:
      labels:
        app: static-workload
      annotations:
        k8s.v1.cni.cncf.io/networks: default/macvlan
        ovn.kubernetes.io/ip_pool: 10.16.0.15,10.16.0.16,10.16.0.17
        macvlan.default.kubernetes.io/ip_pool: 172.17.0.200,172.17.0.201,172.17.0.202
    spec:
      containers:
      - name: static-workload
        image: docker.io/library/nginx:alpine
```

Create a Pod using macvlan as default route

For Pods that use macvlan as an accessory network card, if you want to use the accessory network card as the default route of the Pod, you only need to add the following annotation, `default-route` is the gateway address:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: samplepod-route
  namespace: default
  annotations:
    k8s.v1.cni.cncf.io/networks: '[{
      "name": "macvlan",
      "namespace": "default",
      "default-route": ["172.17.0.1"]
    }]'
spec:
  containers:
  - name: samplepod-route
    command: ["/bin/ash", "-c", "trap : TERM INT; sleep infinity & wait"]
    image: docker.io/library/alpine:edge
```

Create a Pod using macvlan as the main nic

For Pods that use macvlan as the main network card, you only need to add the following annotation `v1.multus-cni.io/default-network`, whose value is `<namespace>/<name>` of the corresponding NetworkAttachmentDefinition:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: samplepod-macvlan
  namespace: default
  annotations:
    v1.multus-cni.io/default-network: default/macvlan
spec:
  containers:
  - name: samplepod-macvlan
    command: ["/bin/ash", "-c", "trap : TERM INT; sleep infinity & wait"]
    image: docker.io/library/alpine:edge
```

CREATE A KUBE-OVN SUBNET (PROVIDER OVN)

Create a Kube-OVN Subnet, set the corresponding `cidrBlock` and `exclude_ips`, `provider` is ovn, and create the Subnet as follows:

```yaml
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
  name: macvlan
spec:
  protocol: IPv4
  provider: ovn
  cidrBlock: 172.17.0.0/16
  gateway: 172.17.0.1
  excludeIps:
  - 172.17.0.0..172.17.0.10
```

Create a Pod with Multiple NIC

For Pods that need to obtain IP from the subnet with `provider` type ovn, you need to combine the annotation `k8s.v1.cni.cncf.io/networks` and `<networkAttachmentName>.<networkAttachmentNamespace>.kubernetes.io/logical_switch` use:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: samplepod
  namespace: default
  annotations:
    k8s.v1.cni.cncf.io/networks: default/macvlan
    macvlan.default.kubernetes.io/logical_switch: macvlan
spec:
  containers:
  - name: samplepod
    command: ["/bin/ash", "-c", "trap : TERM INT; sleep infinity & wait"]
    image: docker.io/library/alpine:edge
```

- `k8s.v1.cni.cncf.io/networks` : The value is `<namespace>/<name>` of the corresponding NetworkAttachmentDefinition
- `macvlan.default.kubernetes.io/logical_switch` : The value is the subnet name

Note: Specifying a subnet through `<networkAttachmentName>.<networkAttachmentNamespace>.kubernetes.io/logical_switch` has a higher priority than specifying a subnet through provider. Subnets based on ovn type provide ipam and also support the creation of fixed IP Pods and the creation of fixed IP pods. IP workload, create a Pod with the default route as macvlan, but creating a Pod with the main network card as macvlan is not supported.

The attached NIC is a Kube-OVN type NIC

At this point, the multiple NICs are all Kube-OVN type NICs.

CREATE NETWORKATTACHMENTDEFINITION

Set the `provider` suffix to `ovn` :

```yaml
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: attachnet
  namespace: default
spec:
  config: '{
      "cniVersion": "0.3.0",
      "type": "kube-ovn",
      "server_socket": "/run/openvswitch/kube-ovn-daemon.sock",
      "provider": "attachnet.default.ovn"
    }'
```

- `spec.config.ipam.type` : Need to be set to `kube-ovn` to call the kube-ovn plugin to get the address information.
- `server_socket` : The socket file used for communication to Kube-OVN. The default location is `/run/openvswitch/kube-ovn-daemon.sock` .
- `provider` : The current NetworkAttachmentDefinition's `<name>. <namespace>` , Kube-OVN will use this information to find the corresponding Subnet resource. It should have the suffix `ovn` here.

CREATE A KUBE-OVN SUBNET

If you are using Kube-OVN as an attached NIC, `provider` should be set to the `<name>. <namespace>.ovn` of the corresponding NetworkAttachmentDefinition, and should end with `ovn` as a suffix.

An example of creating a Subnet with an additional NIC provided by Kube-OVN is as follows:

```yaml
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
  name: attachnet
spec:
  protocol: IPv4
  provider: attachnet.default.ovn
  cidrBlock: 172.17.0.0/16
  gateway: 172.17.0.1
  excludeIps:
  - 172.17.0.0..172.17.0.10
```

Create a Pod with Multiple NIC

For Pods with randomly assigned addresses, simply add the following annotation `k8s.v1.cni.cncf.io/networks`, taking the value `<namespace>/<name>` of the corresponding NetworkAttachmentDefinition.:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: samplepod
  namespace: default
  annotations:
    k8s.v1.cni.cncf.io/networks: default/attachnet
spec:
  containers:
  - name: samplepod
    command: ["/bin/ash", "-c", "trap : TERM INT; sleep infinity & wait"]
    image: docker.io/library/alpine:edge
```

CREATE A KUBE-OVN SUBNET (PROVIDER OVN)

Create a Kube-OVN Subnet, set the corresponding `cidrBlock` and `exclude_ips`, `provider` is ovn, and create the Subnet as follows:

```yaml
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
  name: attachnet
spec:
  protocol: IPv4
  provider: ovn
  cidrBlock: 172.17.0.0/16
  gateway: 172.17.0.1
  excludeIps:
  - 172.17.0.0..172.17.0.10
```

Create a Pod with Multiple NIC

For Pods that need to obtain IP from the subnet whose `provider` type is ovn, the annotation `k8s.v1.cni.cncf.io/networks` and `<networkAttachmentName>.<networkAttachmentNamespace>.ovn.kubernetes.io/logical_switch` need to be used in conjunction with:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: samplepod
  namespace: default
  annotations:
    k8s.v1.cni.cncf.io/networks: default/attachnet
    attachnet.default.ovn.kubernetes.io/logical_switch: attachnet
spec:
  containers:
  - name: samplepod
    command: ["/bin/ash", "-c", "trap : TERM INT; sleep infinity & wait"]
    image: docker.io/library/alpine:edge
```

- `k8s.v1.cni.cncf.io/networks`: The value is `<namespace>/<name>` of the corresponding NetworkAttachmentDefinition
- `attachnet.default.ovn.kubernetes.io/logical_switch`: The value is the subnet name

Note: Specifying a subnet through `<networkAttachmentName>.<networkAttachmentNamespace>.ovn.kubernetes.io/logical_switch` has a higher priority than specifying a subnet through provider. For Pods with Kube-OVN attached network cards, the creation of fixed IPs is supported. Pod, create a workload using a fixed IP, create a Pod with the default route as macvlan, and also support the creation of a Pod with the main network card as Kube-OVN type. For the configuration method, please refer to the previous section.

PDF Slack Support

July 30, 2025

May 20, 2022

GitHub

7.1.3 Comments

7.2 Performance Tuning

To keep the installation simple and feature-complete, the default installation script for Kube-OVN does not have performance-specific optimizations. If the applications are sensitive to latency and throughput, administrators can use this document to make specific performance optimizations.

The community will continue to iterate on the performance. Some general performance optimizations have been integrated into the latest version, so it is recommended to use the latest version to get better default performance.

For more on the process and methodology of performance optimization, please watch the video Kube-OVN .

7.2.1 Benchmarking

Because the hardware and software environments vary greatly, the performance test data provided here can only be used as a reference, and the actual test results may differ significantly from the results in this document. It is recommended to compare the performance test results before and after optimization, and the performance comparison between the host network and the container network.

Overlay Performance Comparison before and after Optimization

Environment:

- Kubernetes: 1.22.0
- OS: CentOS 7
- Kube-OVN: 1.8.0 *Overlay* Mode
- CPU: Intel(R) Xeon(R) E-2278G
- Network: 2*10Gbps, xmit_hash_policy=layer3+4

We use `qperf -t 60 <server ip> -ub -oo msg_size:1 -vu tcp_lat tcp_bw udp_lat udp_bw` to test bandwidth and latency of tcp/udp in 1-byte packets and the host network, respectively.

Type	tcp_lat (us)	udp_lat (us)	tcp_bw (Mb/s)	udp_bw(Mb/s)
Kube-OVN Default	25.7	22.9	27.1	1.59
Kube-OVN Optimized	13.9	12.9	27.6	5.57
HOST Network	13.1	12.4	28.2	6.02

Overlay and Underlay Comparison

Next, we compare the overlay and underlay performance of the optimized Kube-OVN at different packet sizes with the host network.

Environment:

- Kubernetes: 1.22.0
- OS: CentOS 7
- Kube-OVN: 1.8.0
- CPU: AMD EPYC 7402P 24-Core Processor
- Network: Intel Corporation Ethernet Controller XXV710 for 25GbE SFP28

`qperf -t 60 <server ip> -ub -oo msg_size:1 -vu tcp_lat tcp_bw udp_lat udp_bw`

Type	tcp_lat (us)	udp_lat (us)	tcp_bw (Mb/s)	udp_bw(Mb/s)
Kube-OVN Overlay	15.2	14.6	23.6	2.65
Kube-OVN Underlay	14.3	13.8	24.2	3.46
HOST Network	16.6	15.4	24.8	2.64

`qperf -t 60 <server ip> -ub -oo msg_size:1K -vu tcp_lat tcp_bw udp_lat udp_bw`

Type	tcp_lat (us)	udp_lat (us)	tcp_bw (Gb/s)	udp_bw(Gb/s)
Kube-OVN Overlay	16.5	15.8	10.2	2.77
Kube-OVN Underlay	15.9	14.5	9.6	3.22
HOST Network	18.1	16.6	9.32	2.66

`qperf -t 60 <server ip> -ub -oo msg_size:4K -vu tcp_lat tcp_bw udp_lat udp_bw`

Type	tcp_lat (us)	udp_lat (us)	tcp_bw (Gb/s)	udp_bw(Gb/s)
Kube-OVN Overlay	34.7	41.6	16.0	9.23
Kube-OVN Underlay	32.6	44	15.1	6.71
HOST Network	35.9	45.9	14.6	5.59

In some cases the container network outperforms the host network, this is because the container network path is optimized to completely bypass netfilter. Due to the existence of `kube-proxy`, all packets in host network have to go through netfilter, which will lead to more CPU consumption, so that container network in some environments has better performance.

7.2.2 Dataplane performance optimization methods

The optimization methods described here are related to the hardware and software environment and the desired functionality, so please carefully understand the prerequisites for optimization before attempting it.

CPU Performance Mode Tuning

In some environments the CPU is running in power saving mode, performance in this mode will be unstable and latency will increase significantly, it is recommended to use the CPU's performance mode for more stable performance.

```
cpupower frequency-set -g performance
```

NIC Hardware Queue Adjustment

In the case of increased traffic, a small buffer queue may lead to significant performance degradation due to a high packet loss rate and needs to be tuned.

Check the current NIC queue length:

```
# ethtool -g eno1
 Ring parameters for eno1:
 Pre-set maximums:
 RX:             4096
 RX Mini:        0
 RX Jumbo:       0
 TX:             4096
 Current hardware settings:
 RX:             255
 RX Mini:        0
 RX Jumbo:       0
 TX:             255
```

Increase the queue length to the maximum:

```
ethtool -G eno1 rx 4096
ethtool -G eno1 tx 4096
```

Optimize with tuned

tuned can use a series of preconfigured profile files to perform system optimizations for a specific scenario.

For latency-first scenarios:

```
tuned-adm profile network-latency
```

For throughput-first scenarios:

```
tuned-adm profile network-throughput
```

Interrupt Binding

We recommend disabling `irqbalance` and binding NIC interrupts to specific CPUs to avoid performance fluctuations caused by switching between multiple CPUs.

Disable OVN LB

The L2 LB implementation of OVN requires calling the kernel's `conntrack` module and recirculate, resulting in a significant CPU overhead, which is tested to be around 20%. For Overlay networks you can use `kube-proxy` to complete the service forwarding function for better Pod-to-Pod performance. This can be turned off in `kube-ovn-controller` args:

```
command:
- /kube-ovn/start-controller.sh
args:
...
- --enable-lb=false
...
```

In Underlay mode `kube-proxy` cannot use iptables or ipvs to control container network traffic, if you want to disable the LB function, you need to confirm whether you do not need the Service function.

FastPath Kernel Module

Since the container network and the host network are on different network ns, the packets will pass through the netfilter module several times when they are transmitted across the host, which results in a CPU overhead of nearly 20%. The `FastPath` module can reduce CPU overhead by bypassing netfilter, since in most cases applications within a container network do not need to use the functionality of the netfilter module.

If you need to use the functions provided by netfilter such as iptables, ipvs, nftables, etc. in the container network, this module will disable the related functions.

Since kernel modules are kernel version dependent, it is not possible to provide a single kernel module artifact that adapts to all kernels. We pre-compiled the `FastPath` module for part of the kernels, which can be accessed by tuning-package.

You can also compile it manually, see Compiling FastPath Module

After obtaining the kernel module, you can load the `FastPath` module on each node using `insmod kube_ovn_fastpath.ko` and verify that the module was loaded successfully using `dmesg`:

```
# dmesg
...
[619631.323788] init_module,kube_ovn_fastpath_local_out
[619631.323798] init_module,kube_ovn_fastpath_post_routing
[619631.323800] init_module,kube_ovn_fastpath_pre_routing
```

```
[619631.323801] init_module,kube_ovn_fastpath_local_in
...
```

OVS Kernel Module Optimization

OVS flow processing including hashing, matching, etc. consumes about 10% of the CPU resources. Some instruction sets on modern x86 CPUs such as `popcnt` and `sse4.2` can speed up the computation process, but the kernel is not compiled with these options enabled. It has been tested that the CPU consumption of flow-related operations is reduced to about 5% when the corresponding instruction set optimizations are enabled.

Similar to the compilation of the `FastPath` module, it is not possible to provide a single kernel module artifact for all kernels. Users need to compile manually or go to tuning-package to see if a compiled package is available for download.

Before using this kernel module, please check if the CPU supports the following instruction set:

```
cat /proc/cpuinfo  | grep popcnt
cat /proc/cpuinfo  | grep sse4_2
```

COMPILE AND INSTALL IN CENTOS

Install the relevant compilation dependencies and kernel headers:

```
yum install -y gcc kernel-devel-$(uname -r) python3 autoconf automake libtool rpm-build openssl-devel
```

Compile the OVS kernel module and generate the corresponding RPM:

```
git clone -b branch-2.17 --depth=1 https://github.com/openvswitch/ovs.git
cd ovs
curl -s  https://github.com/kubeovn/ovs/commit/2d2c83c26d4217446918f39d5cd5838e9ac27b32.patch |  git apply
./boot.sh
./configure --with-linux=/lib/modules/$(uname -r)/build CFLAGS="-g -O2 -mpopcnt -msse4.2"
make rpm-fedora-kmod
cd rpm/rpmbuild/RPMS/x86_64/
```

Copy the RPM to each node and install:

```
rpm -i openvswitch-kmod-2.15.2-1.el7.x86_64.rpm
```

If you have previously started Kube-OVN and the older version of the OVS module has been loaded into the kernel. It is recommended to reboot the machine to reload the new version of the kernel module.

COMPILE AND INSTALL IN UBUNTU

Install the relevant compilation dependencies and kernel headers:

```
apt install -y autoconf automake libtool gcc build-essential libssl-dev
```

Compile the OVS kernel module and install:

```
git clone -b branch-2.17 --depth=1 https://github.com/openvswitch/ovs.git
cd ovs
curl -s  https://github.com/kubeovn/ovs/commit/2d2c83c26d4217446918f39d5cd5838e9ac27b32.patch |  git apply
./boot.sh
./configure --prefix=/usr/ --localstatedir=/var --enable-ssl --with-linux=/lib/modules/$(uname -r)/build
make -j `nproc`
make install
make modules_install

cat > /etc/depmod.d/openvswitch.conf << EOF
override openvswitch * extra
override vport-* * extra
EOF

depmod -a
cp debian/openvswitch-switch.init /etc/init.d/openvswitch-switch
/etc/init.d/openvswitch-switch force-reload-kmod
```

If you have previously started Kube-OVN and the older version of the OVS module has been loaded into the kernel. It is recommended to reboot the machine to reload the new version of the kernel module.

Using STT Type Tunnel

Common tunnel encapsulation protocols such as Geneve and Vxlan use the UDP protocol to encapsulate packets and are well supported in the kernel. However, when TCP packets are encapsulated using UDP, the optimization and offload features of modern operating systems and network cards for the TCP protocol do not work well, resulting in a significant drop in TCP throughput. In some virtualization scenarios, due to CPU limitations, TCP packet throughput may even be a tenth of that of the host network.

STT provides an innovative tunneling protocol that uses TCP formatted header for encapsulation. This encapsulation only emulates the TCP protocol header format without actually establishing a TCP connection, but can take full advantage of the TCP optimization capabilities of modern operating systems and network cards. In our tests TCP packet throughput can be improved several times, reaching performance levels close to those of the host network.

The STT tunnel is not pre-installed in the kernel and needs to be installed by compiling the OVS kernel module, which can be found in the previous section.

Enable STT tunnel:

```
kubectl set env daemonset/ovs-ovn -n kube-system TUNNEL_TYPE=stt

kubectl delete pod -n kube-system -lapp=ovs
```

PDF Slack Support

July 30, 2025

May 24, 2022

GitHub

7.2.3 Comments

7.3 Compile FastPath Module

After a data plane performance profile, `netfilter` consumes about 20% of CPU resources for related processing within the container and on the host. The FastPath module can bypass `netfilter` to reduce CPU consumption and latency, and increase throughput. This document will describe how to compile the FastPath module manually.

7.3.1 Download Related Code

```
git clone --depth=1 https://github.com/kubeovn/kube-ovn.git
```

7.3.2 Install Dependencies

Here is an example of CentOS dependencies to download:

```
yum install -y kernel-devel-$(uname -r) gcc elfutils-libelf-devel
```

7.3.3 Compile the Module

For the 3.x kernel:

```
cd kube-ovn/fastpath
make all
```

For the 4.x kernel:

```
cd kube-ovn/fastpath/4.18
cp ../Makefile .
make all
```

7.3.4 Instal the Kernel Module

Copy `kube_ovn_fastpath.ko` to each node that needs performance optimization, and run the following command:

```
insmod kube_ovn_fastpath.ko
```

Use `dmesg` to confirm successful installation:

```
# dmesg
[619631.323788] init_module,kube_ovn_fastpath_local_out
[619631.323798] init_module,kube_ovn_fastpath_post_routing
[619631.323800] init_module,kube_ovn_fastpath_pre_routing
[619631.323801] init_module,kube_ovn_fastpath_local_in
```

To uninstall a module, use the following command.

```
rmmod kube_ovn_fastpath.ko
```

This module will not be loaded automatically after machine reboot. If you want to load it automatically, please write the corresponding autostart script according to the system configuration.

PDF Slack Support

February 15, 2023

May 24, 2022

GitHub

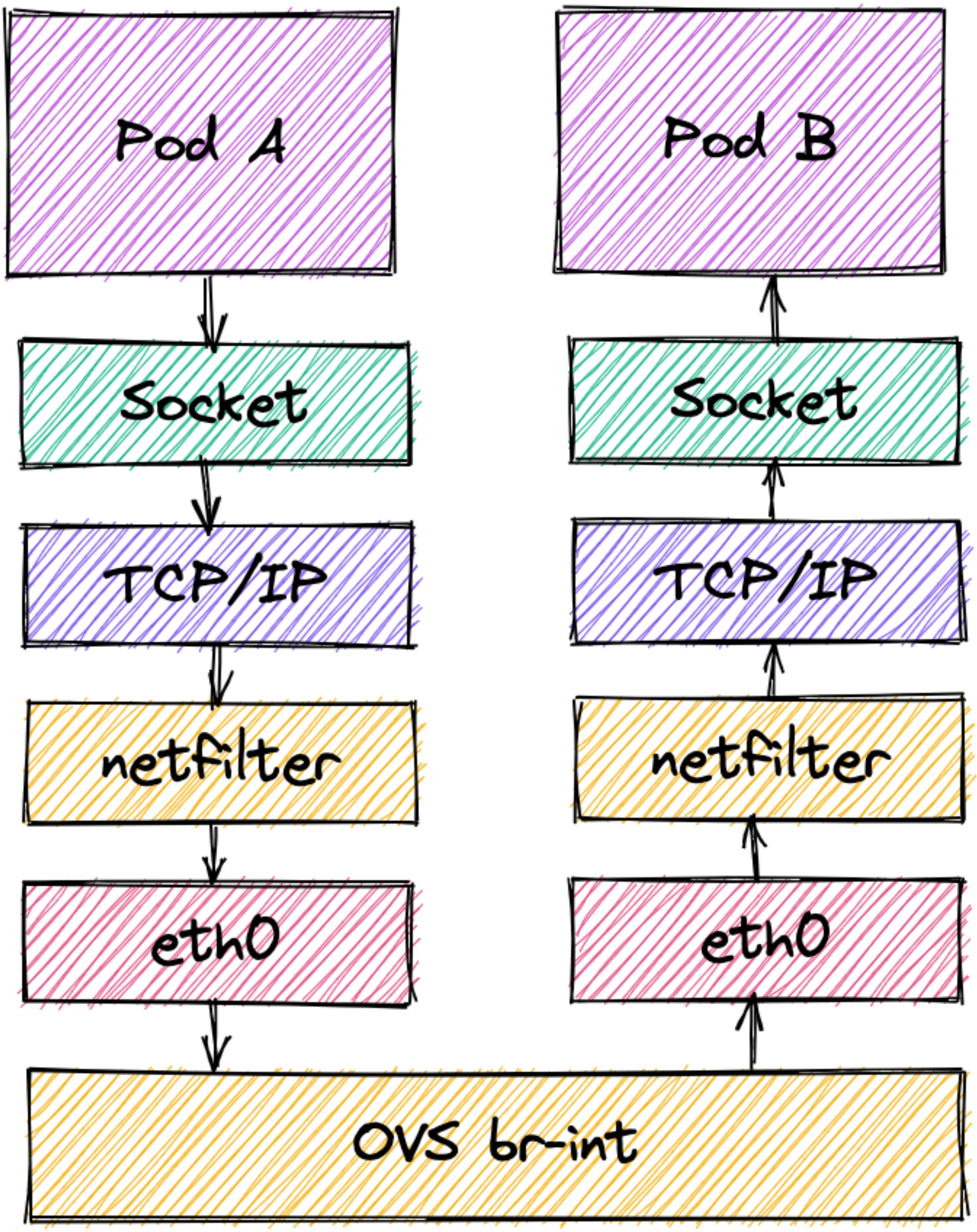
7.3.5 Comments

7.4 Accelerate TCP Communication in Node with eBPF

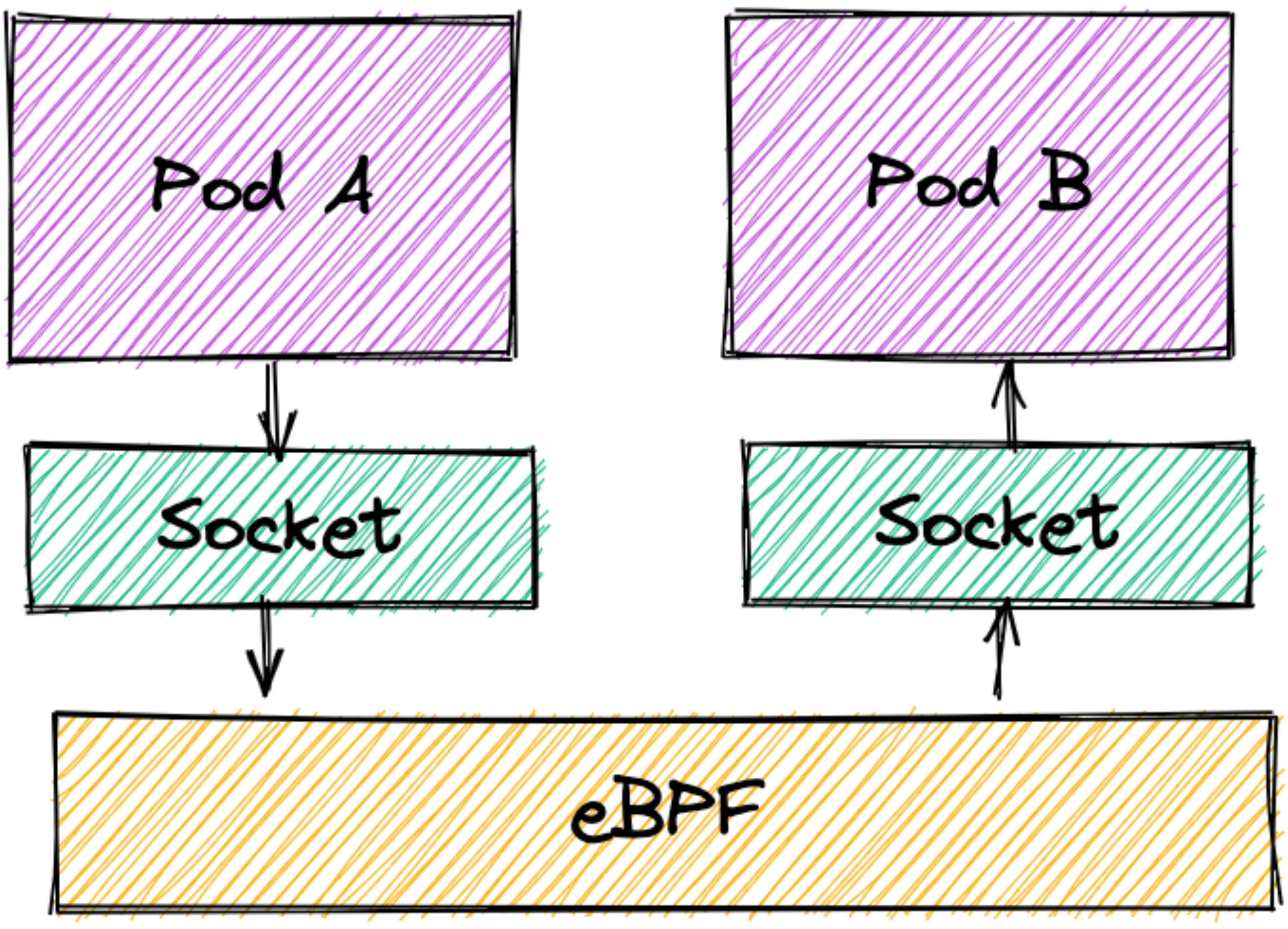
At some edge and 5G scenarios, there will be a lot of TCP communication between Pods on the same node. By using the open source istio-tcpip-bypass project from Intel, Pods can use the ability of eBPF to bypass the host's TCP/IP protocol stack and communicate directly through sockets, thereby greatly reducing latency and improving throughput.

7.4.1 Basic Principle

At present, two Pods on the same host need to go through a lot of network stacks, including TCP/IP, netfilter, OVS, etc., as shown in the following figure:



istio-tcpip-bypass plugin can automatically analyze and identify TCP communication within the same host, and bypass the complex kernel stack so that socket data transmission can be performed directly to reduce network stack processing overhead, as shown in the following figure:



Due to the fact that this component can automatically identify TCP communication within the same host and optimize it. In the Service Mesh environment based on the proxy mode, this component can also enhance the performance of Service Mesh.

For more technical implementation details, please refer to Tanzu Service Mesh Acceleration using eBPF.

7.4.2 Prerequisites

eBPF requires a kernel version of at least 5.4.0-74-generic. It is recommended to use Ubuntu 20.04 and Linux 5.4.0-74-generic kernel version for testing.

7.4.3 Experimental Steps

Deploy two performance test Pods on the same node. If there are multiple machines in the cluster, you need to specify `nodeSelector`:

```
# kubectl create deployment perf --image=kubeovn/perf:dev --replicas=2
deployment.apps/perf created
# kubectl get pod -o wide
NAME                    READY   STATUS    RESTARTS   AGE   IP            NODE     NOMINATED NODE   READINESS GATES
perf-7697bc6ddf-b2cpv   1/1     Running   0          28s   100.64.0.3    sealos   <none>           <none>
perf-7697bc6ddf-p2xpt   1/1     Running   0          28s   100.64.0.2    sealos   <none>           <none>
```

Enter one of the Pods to start the qperf server, and start the qperf client in another Pod for performance testing:

```
# kubectl exec -it perf-7697bc6ddf-b2cpv sh
/ # qperf

# kubectl exec -it perf-7697bc6ddf-p2xpt sh
/ # qperf -t 60 100.64.0.3 -ub -oo msg_size:1:16K:*4 -vu tcp_lat tcp_bw
```

Deploy the istio-tcpip-bypass plugin:

```
kubectl apply -f https://raw.githubusercontent.com/intel/istio-tcpip-bypass/main/bypass-tcpip-daemonset.yaml
```

Enter the perf client container again for performance testing:

```
# kubectl exec -it perf-7697bc6ddf-p2xpt sh
/ # qperf -t 60 100.64.0.3 -ub -oo msg_size:1:16K:*4 -vu tcp_lat tcp_bw
```

7.4.4 Test Results

According to the test results, the TCP latency will decrease by 40% ~ 60% under different packet sizes, and the throughput will increase by 40% ~ 80% when the packet size is greater than 1024 bytes.

Packet Size (byte)	eBPF tcp_lat (us)	Default tcp_lat (us)	eBPF tcp_bw (Mb/s)	Default tcp_bw(Mb/s)
1	20.2	44.5	1.36	4.27
4	20.2	48.7	5.48	16.7
16	19.6	41.6	21.7	63.5
64	18.8	41.3	96.8	201
256	19.2	36	395	539
1024	18.3	42.4	1360	846
4096	16.5	62.6	4460	2430
16384	20.2	58.8	9600	6900

In the hardware environment under test, when the packet size is less than 512 bytes, the throughput indicator optimized by eBPF is lower than the throughput under the default configuration. This situation may be related to the TCP aggregation optimization of the network card under the default configuration. If the application scenario is sensitive to small packet throughput, you need to test in the corresponding environment Determine whether to enable eBPF optimization. We will also optimize the throughput of eBPF TCP small packet scenarios in the future.

7.4.5 References

1. istio-tcpip-bypass
2. Deep Dive TCP/IP Bypass with eBPF in Service Mesh
3. Tanzu Service Mesh Acceleration using eBPF

PDF Slack Support

July 2, 2025

June 20, 2023

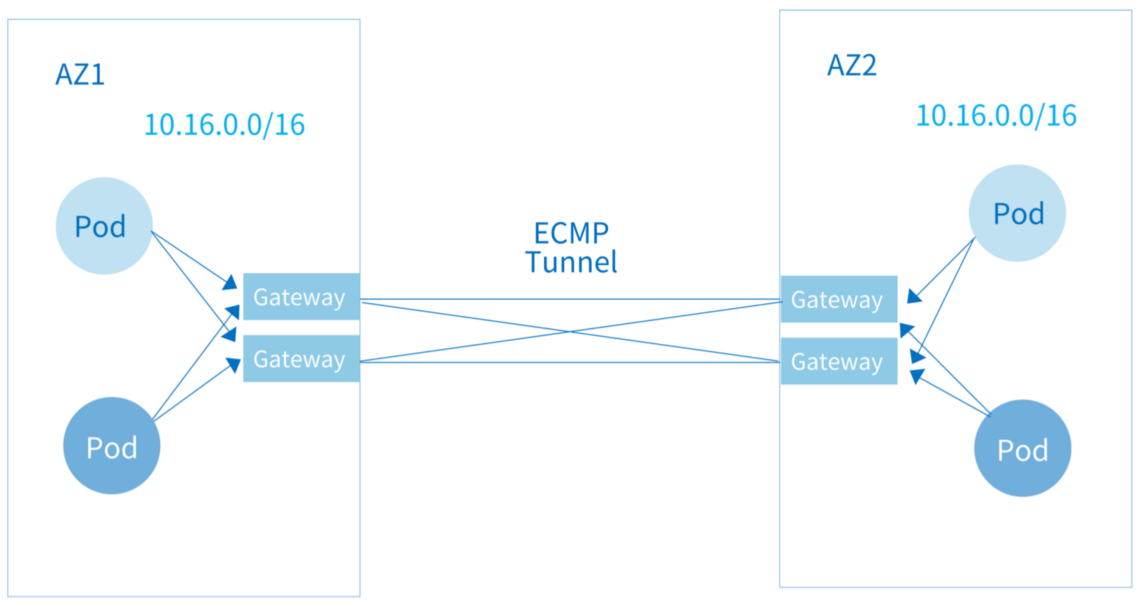
GitHub

7.4.6 Comments

7.5 Cluster Inter-Connection with OVN-IC

Kube-OVN supports interconnecting two Kubernetes cluster Pod networks via OVN-IC, and the Pods in the two clusters can communicate directly via Pod IPs. Kube-OVN uses tunnels to encapsulate cross-cluster traffic, allowing container networks to interconnect between two clusters as long as there is a set of IP reachable machines.

This mode of multi-cluster interconnection is for Overlay network. For Underlay network, it needs the underlying infrastructure to do the inter-connection work.



Limitation

The OVN-IC method can only achieve cross-cluster connectivity for Pod IPs and cannot complete cross-cluster connectivity for Services, DNS, and NetworkPolicies. If cross-cluster service discovery capabilities are needed, please consider using Istio or other cross-cluster service governance projects.

7.5.1 Prerequisites

1. Clusters configured in versions after 1.11.16 have the cluster interconnection switch turned off by default. You need to mark the following in the configuration script `install.sh`:

```
ENABLE_IC=true
```

   After opening the switch and deploying the cluster, the component deployment ovn-ic-controller will appear.

2. The subnet CIDRs within OpenStack and Kubernetes cannot overlap with each other in auto-interconnect mode. If there is overlap, you need to refer to the subsequent manual interconnection process, which can only connect non-overlapping Subnets.
3. A set of machines needs to exist that can be accessed by each cluster over the network and used to deploy controllers that interconnect across clusters.
4. Each cluster needs to have a set of machines that can access each other across clusters via IP as the gateway nodes.
5. This solution only connects to the Kubernetes default VPCs.

7.5.2 Deploy a single-node OVN-IC DB

Single node deployment solution 1

Solution 1 is recommended first, supported after Kube-OVN v1.11.16.

This method does not distinguish between "single node" or "multi-node high availability" deployment. The controller will be deployed on the master node in the form of Deployment. The cluster master node is 1, which is a single node deployment, and the number of master nodes is multiple, that is, multi-node. Highly available deployment.

First get the script `install-ovn-ic.sh` and use the following command:

```
wget https://raw.githubusercontent.com/kubeovn/kube-ovn/release-1.14/dist/images/install-ic-server.sh
```

Execute the command installation, where `TS_NUM` represents the number of ECMP Paths connected to the cluster:

```
sed 's/VERSION=.*/VERSION=v1.14.4/' dist/images/install-ic-server.sh | TS_NUM=3 bash
```

The output of successful execution is as follows:

```
deployment.apps/ovn-ic-server created
Waiting for deployment spec update to be observed...
Waiting for deployment "ovn-ic-server" rollout to finish: 0 out of 3 new replicas have been updated...
Waiting for deployment "ovn-ic-server" rollout to finish: 0 of 3 updated replicas are available...
Waiting for deployment "ovn-ic-server" rollout to finish: 1 of 3 updated replicas are available...
Waiting for deployment "ovn-ic-server" rollout to finish: 2 of 3 updated replicas are available...
deployment "ovn-ic-server" successfully rolled out
OVN IC Server installed Successfully
```

You can view the status of the current interconnected controller through the `kubectl ko icsbctl show` command. The command is as follows:

```
kubectl ko icsbctl show
availability-zone az0
    gateway 059b5c54-c540-4d77-b009-02d65f181a02
        hostname: kube-ovn-worker
        type: geneve
            ip: 172.18.0.3
        port ts-az0
            transit switch: ts
            address: ["00:00:00:B4:8E:BE 169.254.100.97/24"]
    gateway 74ee4b9a-ba48-4a07-861e-1a8e4b9f905f
        hostname: kube-ovn-worker2
        type: geneve
            ip: 172.18.0.2
        port ts1-az0
            transit switch: ts1
            address: ["00:00:00:19:2E:F7 169.254.101.90/24"]
    gateway 7e2428b6-344c-4dd5-a0d5-972c1ccec581
        hostname: kube-ovn-control-plane
        type: geneve
            ip: 172.18.0.4
        port ts2-az0
            transit switch: ts2
            address: ["00:00:00:EA:32:BA 169.254.102.103/24"]
availability-zone az1
    gateway 034da7cb-3826-4318-81ce-6a877a9bf285
        hostname: kube-ovn1-worker
        type: geneve
            ip: 172.18.0.6
        port ts-az1
            transit switch: ts
            address: ["00:00:00:25:3A:B9 169.254.100.51/24"]
    gateway 2531a683-283e-4fb8-a619-bdbcb33539b8
        hostname: kube-ovn1-worker2
        type: geneve
            ip: 172.18.0.5
        port ts1-az1
            transit switch: ts1
            address: ["00:00:00:52:87:F4 169.254.101.118/24"]
    gateway b0efb0be-e5a7-4323-ad4b-317637a757c4
        hostname: kube-ovn1-control-plane
        type: geneve
            ip: 172.18.0.8
        port ts2-az1
            transit switch: ts2
            address: ["00:00:00:F6:93:1A 169.254.102.17/24"]
```

Single node deployment solution 2

Deploy the `OVN-IC` DB on a machine accessible by `kube-ovn-controller`, This DB will hold the network configuration information synchronized up from each cluster.

An environment deploying `docker` can start the `OVN-IC` DB with the following command.

```
docker run --name=ovn-ic-db -d --env "ENABLE_OVN_LEADER_CHECK="false"" --network=host --privileged  -v /etc/ovn/:/etc/ovn -v /var/run/ovn:/var/run/ovn -v /
var/log/ovn:/var/log/ovn kubeovn/kube-ovn:v1.14.4 bash start-ic-db.sh
```

For deploying a `containerd` environment instead of `docker` you can use the following command:

```
ctr -n k8s.io run -d --env "ENABLE_OVN_LEADER_CHECK="false"" --net-host --privileged --mount="type=bind,src=/etc/ovn/,dst=/etc/ovn,options=rbind:rw" --
mount="type=bind,src=/var/run/ovn,dst=/var/run/ovn,options=rbind:rw" --mount="type=bind,src=/var/log/ovn,dst=/var/log/ovn,options=rbind:rw" docker.io/kubeovn/
kube-ovn:v1.14.4 ovn-ic-db bash start-ic-db.sh
```

7.5.3 Automatic Routing Mode

In auto-routing mode, each cluster synchronizes the CIDR information of the Subnet under its own default VPC to `OVN-IC`, so make sure there is no overlap between the Subnet CIDRs of the two clusters.

Create `ovn-ic-config` ConfigMap in `kube-system` Namespace:

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: ovn-ic-config
  namespace: kube-system
data:
  enable-ic: "true"
  az-name: "az1"
  ic-db-host: "192.168.65.3"
  ic-nb-port: "6645"
  ic-sb-port: "6646"
  gw-nodes: "az1-gw"
  auto-route: "true"
```

- `enable-ic`: Whether to enable cluster interconnection.
- `az-name`: Distinguish the cluster names of different clusters, each interconnected cluster needs to be different.
- `ic-db-host`: Address of the node where the `OVN-IC` DB is deployed.
- `ic-nb-port`: `OVN-IC` Northbound Database port, default 6645.
- `ic-sb-port`: `OVN-IC` Southbound Database port, default 6645.
- `gw-nodes`: The name of the nodes in the cluster interconnection that takes on the work of the gateways, separated by commas.
- `auto-route`: Whether to automatically publish and learn routes.

Note: To ensure the correct operation, the ConfigMap `ovn-ic-config` is not allowed to be modified. If any parameter needs to be changed, please delete this ConfigMap, modify it and then apply it again.

Check if the interconnected logical switch `ts` has been established in the `ovn-ic` container with the following command:

```
# ovn-ic-sbctl show
availability-zone az1
    gateway deee03e0-af16-4f45-91e9-b50c3960f809
        hostname: az1-gw
        type: geneve
            ip: 192.168.42.145
        port ts-az1
            transit switch: ts
            address: ["00:00:00:50:AC:8C 169.254.100.45/24"]
availability-zone az2
    gateway e94cc831-8143-40e3-a478-90352773327b
        hostname: az2-gw
        type: geneve
            ip: 192.168.42.149
        port ts-az2
            transit switch: ts
            address: ["00:00:00:07:4A:59 169.254.100.63/24"]
```

At each cluster observe if logical routes have learned peer routes:

```
# kubectl ko nbctl lr-route-list ovn-cluster
IPv4 Routes
                10.42.1.1            169.254.100.45 dst-ip (learned)
                10.42.1.3                100.64.0.2 dst-ip
                10.16.0.2                100.64.0.2 src-ip
                10.16.0.3                100.64.0.2 src-ip
                10.16.0.4                100.64.0.2 src-ip
                10.16.0.6                100.64.0.2 src-ip
             10.17.0.0/16            169.254.100.45 dst-ip (learned)
            100.65.0.0/16            169.254.100.45 dst-ip (learned)
```

Next, you can try `ping` a Pod IP in Cluster 1 directly from a Pod in Cluster 2 to see if you can work.

For a subnet that does not want to automatically publish routes to the other end, you can disable route broadcasting by modifying `disableInterConnection` in the Subnet spec.

```
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
  name: no-advertise
spec:
  cidrBlock: 10.199.0.0/16
  disableInterConnection: true
```

7.5.4 Manual Routing Mode

For cases where there are overlapping CIDRs between clusters, and you only want to do partial subnet interconnection, you can manually publish subnet routing by following the steps below.

Create `ovn-ic-config` ConfigMap in `kube-system` Namespace, and set `auto-route` to `false`:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: ovn-ic-config
  namespace: kube-system
data:
  enable-ic: "true"
  az-name: "az1"
  ic-db-host: "192.168.65.3"
  ic-nb-port: "6645"
  ic-sb-port: "6646"
  gw-nodes: "az1-gw"
  auto-route: "false"
```

Find the address of the remote logical ports in each cluster separately, for later manual configuration of the route:

```
[root@az1 ~]# kubectl ko nbctl show
switch a391d3a1-14a0-4841-9836-4bd930c447fb (ts)
    port ts-az1
        type: router
        router-port: az1-ts
    port ts-az2
        type: remote
        addresses: ["00:00:00:4B:E2:9F 169.254.100.31/24"]

[root@az2 ~]# kubectl ko nbctl show
switch da6138b8-de81-4908-abf9-b2224ec4edf3 (ts)
    port ts-az2
        type: router
        router-port: az2-ts
    port ts-az1
        type: remote
        addresses: ["00:00:00:FB:2A:F7 169.254.100.79/24"]
```

The output above shows that the remote address from cluster `az1` to cluster `az2` is `169.254.100.31` and the remote address from `az2` to `az1` is `169.254.100.79`.

In this example, the subnet CIDR within cluster `az1` is `10.16.0.0/24` and the subnet CIDR within cluster `az2` is `10.17.0.0/24`.

Set up a route from cluster `az1` to cluster `az2` in cluster `az1`:

```
kubectl ko nbctl lr-route-add ovn-cluster 10.17.0.0/24 169.254.100.31
```

Set up a route to cluster `az1` in cluster `az2`:

```
kubectl ko nbctl lr-route-add ovn-cluster 10.16.0.0/24 169.254.100.79
```

7.5.5 Highly Available OVN-IC DB Installation

High availability deployment solution 1

Solution 1 is recommended first, supported after Kube-OVN v1.11.16.

The method is the same as Single node deployment solution 1

High availability deployment solution 2

A highly available cluster can be formed between `OVN-IC` DB via the Raft protocol, which requires a minimum of 3 nodes for this deployment model.

First start the leader of the `OVN-IC` DB on the first node.

Users deploying a `docker` environment can use the following command:

```
docker run --name=ovn-ic-db -d --env "ENABLE_OVN_LEADER_CHECK="false"" --network=host --privileged -v /etc/ovn/:/etc/ovn -v /var/run/ovn:/var/run/ovn -v /var/
log/ovn:/var/log/ovn -e LOCAL_IP="192.168.65.3"  -e NODE_IPS="192.168.65.3,192.168.65.2,192.168.65.1"   kubeovn/kube-ovn:v1.14.4 bash start-ic-db.sh
```

If you are using `containerd` you can use the following command:

```
ctr -n k8s.io run -d --env "ENABLE_OVN_LEADER_CHECK="false"" --net-host --privileged --mount="type=bind,src=/etc/ovn/,dst=/etc/ovn,options=rbind:rw" --
mount="type=bind,src=/var/run/ovn,dst=/var/run/ovn,options=rbind:rw" --mount="type=bind,src=/var/log/ovn,dst=/var/log/ovn,options=rbind:rw"  --
env="NODE_IPS="192.168.65.3,192.168.65.2,192.168.65.1"" --env="LOCAL_IP="192.168.65.3"" docker.io/kubeovn/kube-ovn:v1.14.4 ovn-ic-db bash start-ic-db.sh
```

- `LOCAL_IP` : The IP address of the node where the current container is located.
- `NODE_IPS` : The IP addresses of the three nodes running the `OVN-IC` database, separated by commas.

Next, deploy the follower of the `OVN-IC` DB on the other two nodes.

`docker` environment can use the following command.

```
docker run --name=ovn-ic-db -d --network=host --privileged -v /etc/ovn/:/etc/ovn -v /var/run/ovn:/var/run/ovn -v /var/log/ovn:/var/log/ovn -e
LOCAL_IP="192.168.65.2"  -e NODE_IPS="192.168.65.3,192.168.65.2,192.168.65.1" -e LEADER_IP="192.168.65.3"  kubeovn/kube-ovn:v1.14.4 bash start-ic-db.sh
```

If using `containerd` you can use the following command:

```
ctr -n k8s.io run -d --net-host --privileged --mount="type=bind,src=/etc/ovn/,dst=/etc/ovn,options=rbind:rw" --mount="type=bind,src=/var/run/ovn,dst=/var/run/
ovn,options=rbind:rw" --mount="type=bind,src=/var/log/ovn,dst=/var/log/ovn,options=rbind:rw"  --env="NODE_IPS="192.168.65.3,192.168.65.2,192.168.65.1"" --
env="LOCAL_IP="192.168.65.2"" --env="LEADER_IP="192.168.65.3"" docker.io/kubeovn/kube-ovn:v1.14.4 ovn-ic-db bash start-ic-db.sh
```

- `LOCAL_IP` : The IP address of the node where the current container is located.
- `NODE_IPS` : The IP addresses of the three nodes running the `OVN-IC` database, separated by commas.
- `LEADER_IP` : The IP address of the `OVN-IC` DB leader node.

Specify multiple `OVN-IC` database node addresses when creating `ovn-ic-config` for each cluster:

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: ovn-ic-config
  namespace: kube-system
data:
  enable-ic: "true"
  az-name: "az1"
  ic-db-host: "192.168.65.3,192.168.65.2,192.168.65.1"
  ic-nb-port: "6645"
  ic-sb-port: "6646"
  gw-nodes: "az1-gw"
  auto-route: "true"
```

7.5.6 Support cluster interconnection ECMP

The premise controller is deployed according to Single Node Deployment Solution 1

This solution supports cluster interconnection ECMP by default. The default ECMP path is 3. It also supports modifying the number of ECMP paths. Use the command:

```
kubectl edit deployment ovn-ic-server -n kube-system
```

Just modify the value of the environment variable 'TS_NUM'. `TS_NUM` represents the number of ECMP Paths accessed between the two clusters.

7.5.7 Manual Reset

In some cases, the entire interconnection configuration needs to be cleaned up due to configuration errors, you can refer to the following steps to clean up your environment.

Delete the current `ovn-ic-config` Configmap:

```
kubectl -n kube-system delete cm ovn-ic-config
```

Delete `ts` logical switch:

```
kubectl ko nbctl ls-del ts
```

Repeat the same steps at the peer cluster.

7.5.8 Clean OVN-IC

Delete the `ovn-ic-config` Configmap for all clusters:

```
kubectl -n kube-system delete cm ovn-ic-config
```

Delete all clusters' `ts` logical switches:

```
kubectl ko nbctl ls-del ts
```

Delete the cluster interconnect controller. If it is a high-availability OVN-IC database deployment, all need to be cleaned up.

If the controller is `docker` deploy execute command:

```
docker stop ovn-ic-db
docker rm ovn-ic-db
```

If the controller is `containerd` deploy the command:

```
ctr -n k8s.io task kill ovn-ic-db
ctr -n k8s.io containers rm ovn-ic-db
```

If the controller is deployed using deployment `ovn-ic-server`:

```
kubectl delete deployment ovn-ic-server -n kube-system
```

Then clean up the interconnection-related DB on each master node. The command is as follows:

```
rm -f /etc/origin/ovn/ovn_ic_nb_db.db
rm -f /etc/origin/ovn/ovn_ic_sb_db.db
```

PDF **Slack** **Support**

July 30, 2025

May 24, 2022

GitHub

7.5.9 Comments

7.6 Cluster Inter-Connection with Submariner

Submariner is an open source networking component that connects multiple Kubernetes cluster Pod and Service networks which can help Kube-OVN interconnect multiple clusters.

Compared to OVN-IC, Submariner can connect Kube-OVN and non-Kube-OVN cluster networks, and Submariner can provide cross-cluster capability for services. However, Submariner currently only enables the default subnets to be connected, and cannot selectively connect multiple subnets.

7.6.1 Prerequisites

- The Service CIDRs of the two clusters and the CIDR of the default Subnet cannot overlap.

7.6.2 Install Submariner

Download the `subctl` binary and deploy it to the appropriate path:

```
curl -Ls https://get.submariner.io | bash
export PATH=$PATH:~/.local/bin
echo export PATH=\$PATH:~/.local/bin >> ~/.profile
```

Change `kubeconfig` context to the cluster that need to deploy `submariner-broker`:

```
subctl deploy-broker
```

In this document the default subnet CIDR for `cluster0` is `10.16.0.0/16` and the join subnet CIDR for `cluster0` is `100.64.0.0/16`, the default subnet CIDR for `cluster1` is `11.16.0.0/16` and the join subnet CIDR for `cluster1` is `100.68.0.0/16`.

Switch `kubeconfig` to `cluster0` to register the cluster to the broker, and register the gateway node:

```
subctl  join broker-info.subm --clusterid  cluster0 --clustercidr 100.64.0.0/16,10.16.0.0/16  --natt=false --cable-driver vxlan --health-check=false
kubectl label nodes cluster0 submariner.io/gateway=true
```

Switch `kubeconfig` to `cluster1` to register the cluster to the broker, and register the gateway node:

```
subctl  join broker-info.subm --clusterid  cluster1 --clustercidr 100.68.0.0/16,11.16.0.0/16  --natt=false --cable-driver vxlan --health-check=false
kubectl label nodes cluster1 submariner.io/gateway=true
```

Next, you can start Pods in each of the two clusters and try to access each other using IPs.

Network communication problems can be diagnosed by using the `subctl` command:

```
subctl show all
subctl diagnose all
```

For more Submariner operations please read Submariner Usage.

PDF Slack Support

March 23, 2023

May 24, 2022

GitHub

7.6.3 Comments

7.7 Interconnection with Routes in Overlay Mode

In some scenarios, the network environment does not support Underlay mode, but still need Pods and external devices directly access through IP, then you can use the routing method to connect the container network and the external.

Only Overlay Subnets in default VPC support this method. In this case, the Pod IP goes directly to the underlying network, which needs to disable IP checks for source and destination addresses.

7.7.1 Prerequisites

- In this mode, the host needs to open the `ip_forward`.
- Check if there is a `Drop` rule in the forward chain in the host iptables that should be modified for container-related traffic.
- Due to the possibility of asymmetric routing, the host needs to allow packets with a ct status of `INVALID`.

7.7.2 Steps

For subnets that require direct external routing, you need to set `natOutgoing` of the subnet to `false` to turn off nat mapping and make the Pod IP directly accessible to the external network.

```yaml
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
  name: routed
spec:
  protocol: IPv4
  cidrBlock: 10.166.0.0/16
  default: false
  excludeIps:
  - 10.166.0.1
  gateway: 10.166.0.1
  gatewayType: distributed
  natOutgoing: false
```

At this point, the Pod's packets can reach the peer node via the host route, but the peer node does not yet know where the return packets should be sent to and needs to add a return route.

If the peer host and the container host are on the same Layer 2 network, we can add a static route directly to the peer host to point the next hop of the container network to any machine in the Kubernetes cluster.

```
ip route add 10.166.0.0/16 via 192.168.2.10 dev eth0
```

`10.166.0.0/16` is the container subnet CIDR, and `192.168.2.10` is one node in the Kubernetes cluster.

If the peer host and the container host are not in the same layer 2 network, you need to configure the corresponding rules on the router.

Note: Specifying an IP for a single node may lead to single point of failure. To achieve fast failover, Keepalived can be used to set up a VIP for multiple nodes, and the next hop of the route can be directed to the VIP.

In some virtualized environments, the virtual network identifies asymmetric traffic as illegal traffic and drops it. In this case, you need to adjust the `gatewayType` of the Subnet to `centralized` and set the next hop to the IP of the `gatewayNode` node during route setup.

```yaml
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
  name: routed
spec:
  protocol: IPv4
  cidrBlock: 10.166.0.0/16
  default: false
  excludeIps:
  - 10.166.0.1
  gateway: 10.166.0.1
  gatewayType: centralized
  gatewayNode: "node1"
  natOutgoing: false
```

If you still want to perform NAT processing for some traffic, such as traffic accessing the Internet, please refer to the Default VPC NAT Policy Rule.

PDF **Slack** **Support**

September 1, 2023

July 6, 2022

GitHub

7.7.3 Comments

7.8 BGP Support

Kube-OVN supports broadcasting the IP address of Pods/Subnets/Services/EIPs to the outside world via the BGP protocol.

To use this feature on Pods/Subnets/Services, you need to install `kube-ovn-speaker` on specific (or all) nodes and add the corresponding annotation to the Pod or Subnet that needs to be exposed to the outside world.
Kube-OVN also supports broadcasting the IP address of services of type `ClusterIP` via the same annotation.

To use this feature on EIPs, you need to create your NAT Gateway with special parameters to enable the BGP speaker sidecar. See Publishing EIPs for more information.

7.8.1 Installing `kube-ovn-speaker`

`kube-ovn-speaker` uses GoBGP to publish routing information to the outside world and to set the `next-hop` route to itself.

Since the nodes where `kube-ovn-speaker` is deployed need to carry return traffic, specific labeled nodes need to be selected for deployment:

```
kubectl label nodes speaker-node-1 ovn.kubernetes.io/bgp=true
kubectl label nodes speaker-node-2 ovn.kubernetes.io/bgp=true
```

When there are multiple instances of kube-ovn-speaker, each of them will publish routes to the outside world, the upstream router needs to support multi-path ECMP.

Download the corresponding yaml:

```
wget https://raw.githubusercontent.com/kubeovn/kube-ovn/release-1.14/yamls/speaker.yaml
```

Modify the corresponding configuration in yaml:

If you only have one switch:

```
- --neighbor-address=10.32.32.254
- --neighbor-ipv6-address=2409:AB00:AB00:2000::AFB:8AFE
- --neighbor-as=65030
- --cluster-as=65000
```

If you have a pair of switches:

```
- --neighbor-address=10.32.32.252,10.32.32.253
- --neighbor-ipv6-address=2409:AB00:AB00:2000::AFB:8AFC,2409:AB00:AB00:2000::AFB:8AFD
- --neighbor-as=65030
- --cluster-as=65000
```

- `neighbor-address` : The address of the BGP Peer, usually the router gateway address.
- `neighbor-as` : The AS number of the BGP Peer.
- `cluster-as` : The AS number of the container network.

Apply the YAML:

```
kubectl apply -f speaker.yaml
```

7.8.2 Publishing Pod/Subnet Routes

To use BGP for external routing on subnets, first set `natOutgoing` to `false` for the corresponding Subnet to allow the Pod IP to enter the underlying network directly.

Add annotation to publish routes:

```
kubectl annotate pod sample ovn.kubernetes.io/bgp=true
kubectl annotate subnet ovn-default ovn.kubernetes.io/bgp=true
```

Delete annotation to disable the publishing:

```
kubectl annotate pod sample ovn.kubernetes.io/bgp-
kubectl annotate subnet ovn-default ovn.kubernetes.io/bgp-
```

See Announcement Policies for the announcement behavior depending on the policy set in the annotation.

7.8.3 Publishing Services of type `ClusterIP`

To announce the ClusterIP of services to the outside world, the `kube-ovn-speaker` option `announce-cluster-ip` needs to be set to `true`. See the advanced options for more details.

Set the annotation to enable publishing:

```
kubectl annotate service sample ovn.kubernetes.io/bgp=true
```

Delete annotation to disable the publishing:

```
kubectl annotate service sample ovn.kubernetes.io/bgp-
```

The speakers will all start announcing the `ClusterIP` of that service to the outside world.

7.8.4 Publishing EIPs

EIPs can be announced by the NAT gateways to which they are attached.
There are 2 announcement modes:

- **ARP**: the NAT gateway uses ARP to advertise the EIPs attached to itself, this mode is always enabled
- **BGP**: the NAT gateway provisions a sidecar to publish the EIPs to another BGP speaker

When BGP is enabled on a `VpcNatGateway` a new BGP speaker sidecar gets injected to it. When the gateway is in BGP mode, the behaviour becomes cumulative with the **ARP** mode. This means that EIPs will be announced by **BGP** but also keep being advertised using traditional **ARP**.

To add BGP capabilities to NAT gateways, we first need to create a new `NetworkAttachmentDefinition` that can be attached to our BGP speaker sidecars. This NAD will reference a provider shared by a `Subnet` in the default VPC (in which the Kubernetes API is running).
This will enable the sidecar to reach the K8S API, automatically detecting new EIPs added to the gateway. This operation only needs to be done once. All the NAT gateways will use this provider from now on. This is the same principle used for the CoreDNS in a custom VPC, which means you can reuse that NAD if you've already done that setup before.

Create a `NetworkAttachmentDefinition` and a `Subnet` with the same `provider`. The name of the provider needs to be of the form `nadName.nadNamespace.ovn`:

```yaml
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: api-ovn-nad
  namespace: default
spec:
  config: '{
      "cniVersion": "0.3.0",
      "type": "kube-ovn",
      "server_socket": "/run/openvswitch/kube-ovn-daemon.sock",
      "provider": "api-ovn-nad.default.ovn"
    }'
---
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
  name: vpc-apiserver-subnet
spec:
  protocol: IPv4
  cidrBlock: 100.100.100.0/24
  provider: api-ovn-nad.default.ovn
```

The `ovn-vpc-nat-config` needs to be modified to reference our new provider and the image used by the BGP speaker:

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: ovn-vpc-nat-config
  namespace: kube-system
data:
  apiNadProvider: api-ovn-nad.default.ovn              # What NetworkAttachmentDefinition provider to use so that the sidecar
                                                       # can access the K8S API, as it can't by default due to VPC segmentation
  bgpSpeakerImage: docker.io/kubeovn/kube-ovn:v1.13.0  # Sets the BGP speaker image used
  image: docker.io/kubeovn/vpc-nat-gateway:v1.13.0
```

Some RBAC needs to be added so that the NAT gateways can poll the Kubernetes API, apply the following configuration:

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    kubernetes.io/bootstrapping: rbac-defaults
  name: system:vpc-nat-gw
rules:
  - apiGroups:
      - ""
    resources:
      - services
      - pods
    verbs:
      - list
      - watch
  - apiGroups:
      - kubeovn.io
    resources:
      - iptables-eips
      - subnets
      - vpc-nat-gateways
    verbs:
      - list
      - watch
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  annotations:
    rbac.authorization.kubernetes.io/autoupdate: "true"
  labels:
    kubernetes.io/bootstrapping: rbac-defaults
  name: vpc-nat-gw
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: system:vpc-nat-gw
subjects:
  - kind: ServiceAccount
    name: vpc-nat-gw
    namespace: kube-system
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: vpc-nat-gw
  namespace: kube-system
```

The NAT gateway(s) now needs to be created with BGP enabled so that the speaker sidecar gets created along it:

```yaml
kind: VpcNatGateway
apiVersion: kubeovn.io/v1
metadata:
  name: vpc-natgw
spec:
  vpc: vpc1
  subnet: net1
  lanIp: 10.0.1.10
  bgpSpeaker:
    enabled: true
    asn: 65500
    remoteAsn: 65000
    neighbors:
      - 100.127.4.161
      - fd:01::1
    enableGracefulRestart: true # Optional
    routerId: 1.1.1.1           # Optional
    holdTime: 1m                # Optional
    password: "password123"     # Optional
    extraArgs:                  # Optional, passed directly to the BGP speaker
      - -v5                     # Enables verbose debugging of the BGP speaker sidecar
  selector:
    - "kubernetes.io/os: linux"
  externalSubnets:
  - ovn-vpc-external-network # Network on which we'll speak BGP and receive/send traffic to the outside world
                             # BGP neighbors need to be on that network
```

This gateway is now capable of announcing any EIP that gets attached to it as long as it has the BGP annotation:

```
kubectl annotate eip sample ovn.kubernetes.io/bgp=true
```

7.8.5 Announcement policies

There are 2 policies used by `kube-ovn-speaker` to announce the routes:

- **Cluster**: this policy makes the Pod IPs/Subnet CIDRs be announced from every speaker, whether there's Pods that have that specific IP or that are part of the Subnet CIDR on that node. In other words, traffic may enter from any node hosting a speaker, and then be internally routed in the cluster to the actual Pod. In this configuration extra hops might be used. This is the default policy for Pods and Subnets.
- **Local**: this policy makes the Pod IPs be announced only from speakers on nodes that are actively hosting them. In other words, traffic will only enter from the node hosting the Pod marked as needing BGP advertisement, or from the node hosting a Pod with an IP belonging to a Subnet marked as needing BGP advertisement. This makes the network path shorter as external traffic arrives directly to the physical host of the Pod.

NOTE: You'll probably need to run `kube-ovn-speaker` on every node for the `Local` policy to work. If a Pod you're trying to announce lands on a node with no speaker on it, its IP will simply not be announced.

The default policy used is `Cluster`. Policies can be overridden for each Pod/Subnet using the `ovn.kubernetes.io/bgp` annotation:

- `ovn.kubernetes.io/bgp=cluster` or the default `ovn.kubernetes.io/bgp=yes` will use policy `Cluster`
- `ovn.kubernetes.io/bgp=local` will use policy `Local`

NOTE: Announcement of Services of type `ClusterIP` doesn't support any policy other than `Cluster` as routing to the actual pod is handled by a daemon such as `kube-proxy`. The annotation for Services only supports value `yes` and not `cluster`.

7.8.6 BGP Advanced Options

`kube-ovn-speaker` supports more BGP parameters for advanced configuration, which can be adjusted by users according to their network environment:

- `announce-cluster-ip`: Whether to publish routes for Services of type `ClusterIP` to the public, default is `false`.
- `auth-password`: The access password for the BGP peer.
- `holdtime`: The heartbeat detection time between BGP neighbors. Neighbors with no messages after the change time will be removed, the default is 90 seconds.
- `graceful-restart`: Whether to enable BGP Graceful Restart.
- `graceful-restart-time`: BGP Graceful restart time refer to RFC4724 3.
- `graceful-restart-deferral-time`: BGP Graceful restart deferral time refer to RFC4724 4.1.
- `passivemode`: The Speaker runs in Passive mode and does not actively connect to the peer.
- `ebgp-multihop`: The TTL value of EBGP Peer, default is 1.

7.8.7 BGP routes debug

```
# show peer neighbor
gobgp neighbor

# show announced routes to one peer
gobgp neighbor 10.32.32.254 adj-out
```

PDF Slack Support

October 15, 2024

June 14, 2022

GitHub

7.8.8 Comments

7.9 Integrating MetalLB with Kube-OVN Underlay

MetalLB is an open-source project that provides a network load balancer implementation for Kubernetes clusters, enabling network load balancing functionality for Service objects in Kubernetes clusters.

This document describes how to integrate MetalLB with Kube-OVN's Underlay subnet mode.

7.9.1 Feature Introduction

Starting from version 1.14.0, Kube-OVN supports the integration of MetalLB with Underlay subnets, primarily used in the following scenarios:

- Directly using MetalLB-assigned IPs as external access addresses in physical networks
- Service backend Pods and MetalLB VIPs are in the same Underlay network
- Preserve client source IP and support local forwarding without SNAT

7.9.2 Working Principle

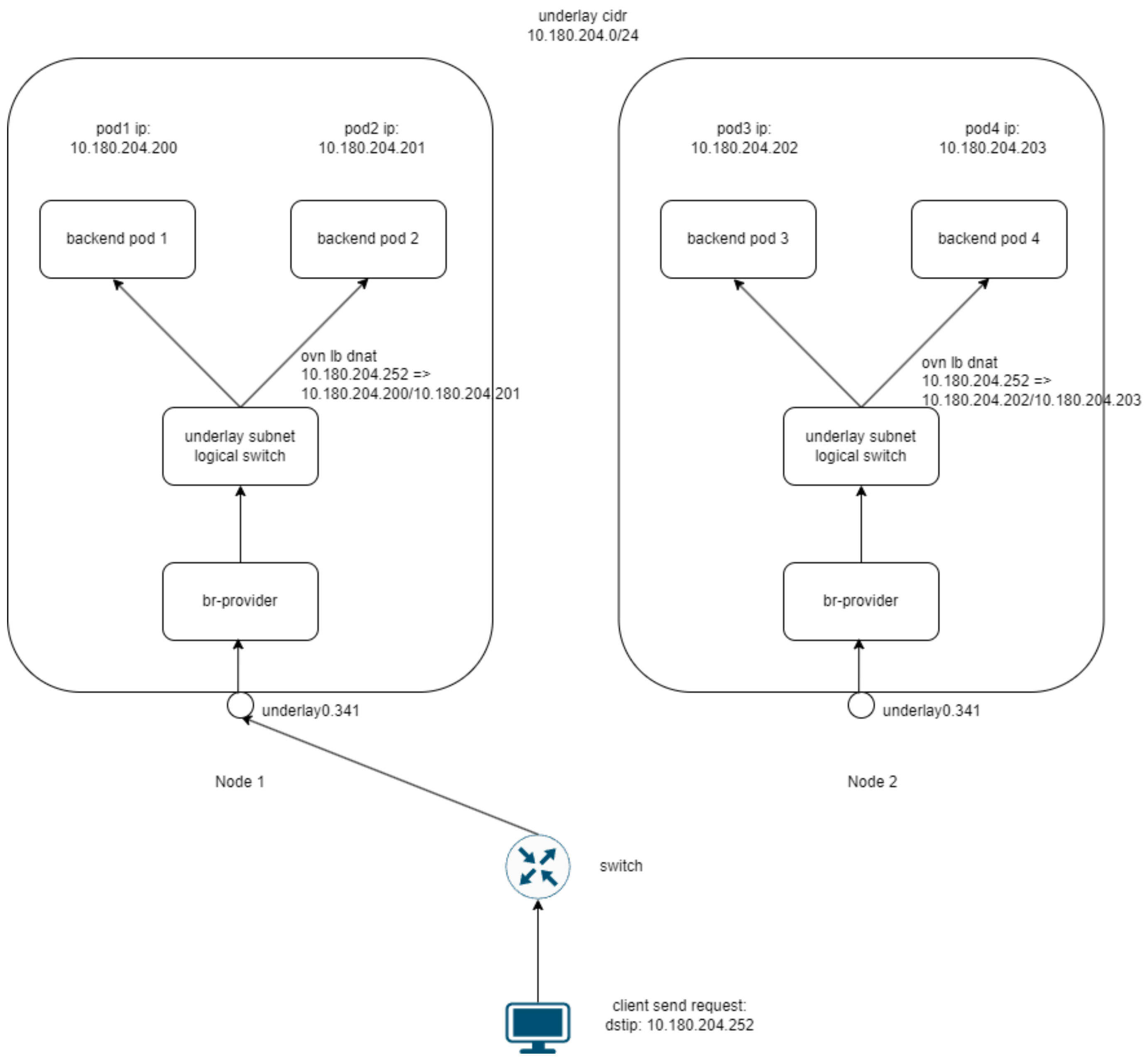


Figure 1: Network Traffic Path for MetalLB VIP Integration with Kube-OVN Underlay

The traffic flow for MetalLB integration with Kube-OVN Underlay is as follows:

1. External client sends requests to the target VIP (e.g., 10.180.204.252), which is announced by MetalLB in L2 mode. In this traffic diagram, Node1 announces the MetalLB VIP
2. Requests reach the node announcing the VIP through the physical network, entering the `underlay0.341` network interface on the node
3. Traffic reaches the `br-provider` bridge on the node, serving as the entry point for the Underlay network
4. `br-provider` forwards traffic to the OVN logical network through OpenFlow flow table rules
5. Traffic enters the `underlay subnet` logical switch, processed by the OVN load balancer (`ovn lb dnat`)
6. OVN load balancer forwards to any Pod on the local node

The entire subnet segment is 10.180.204.0/24, including both the VIP and backend Pod IP addresses within this range.

7.9.3 Prerequisites

- Kube-OVN controller enabled with `--enable-ovn-lb-prefer-local=true` option
- Underlay subnet configured with `enableExternalLBAddress=true`
- MetalLB address pool IP range added to Underlay subnet's `excludeIps`

7.9.4 Deployment Steps

1. Deploy Kube-OVN

Deploy Kube-OVN following the standard procedure, ensuring the Kube-OVN controller is enabled with `--enable-ovn-lb-prefer-local=true` and `--ls-ct-skip-dst-lport-ips=false` options:

```
# Add parameters to the kube-ovn-controller Deployment configuration
kubectl edit deployment -n kube-system kube-ovn-controller
```

Add the following parameters to the command line:

```
--enable-ovn-lb-prefer-local=true
--ls-ct-skip-dst-lport-ips=false
```

2. Configure Underlay Subnet

Create or modify the Underlay subnet to enable external LoadBalancer address support and exclude the IP range that MetalLB will use in excludeIps:

```yaml
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
  name: underlay-subnet
spec:
  protocol: IPv4
  provider: ovn
  cidrBlock: 10.180.204.0/24  # Matches the subnet segment in the diagram
  gateway: 10.180.204.1
  excludeIps:
  - 10.180.204.250
  - 10.180.204.251
  - 10.180.204.252  # MetalLB address pool range, includes the VIP 10.180.204.252
  natOutgoing: false
  enableExternalLBAddress: true   # When enabled, IPs in the subnet's CIDR can be used as MetalLB VIPs
```

3. Deploy MetalLB

Deploy MetalLB following the MetalLB official documentation:

```
kubectl apply -f https://raw.githubusercontent.com/metallb/metallb/v0.13.7/config/manifests/metallb-native.yaml
```

Configure MetalLB's address pool and L2 advertisement mode:

```yaml
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: underlay-pool
  namespace: metallb-system
spec:
  addresses:
  - 10.180.204.250-10.180.204.254  # Includes the VIP 10.180.204.252
---
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: l2-advert
  namespace: metallb-system
spec:
  ipAddressPools:
  - underlay-pool
```

4. Create LoadBalancer Service

Create a LoadBalancer type Service to direct traffic to Pods in the Underlay subnet:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: nginx
  name: deploy-169402624
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      annotations:
        ovn.kubernetes.io/logical_switch: underlay-subnet
      labels:
        app: nginx
    spec:
      containers:
      - args:
        - netexec
        - --http-port
        - "80"
        image: kubeovn/agnhost:2.47
        imagePullPolicy: IfNotPresent
        name: nginx
---
apiVersion: v1
kind: Service
metadata:
  name: nginx-lb
spec:
  externalTrafficPolicy: Local
  ipFamilies:
  - IPv4
  ipFamilyPolicy: PreferDualStack
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx
  type: LoadBalancer
```

7.9.5 Testing and Verification

1. Verify that the Service has obtained the MetalLB-assigned IP address:

   ```
   kubectl get svc nginx-lb
   ```

   You should see the assigned IP address (e.g., 10.180.204.252) in the EXTERNAL-IP column.

1. Access the Service IP address from external:

   ```
   curl http://10.180.204.252
   ```

1. Verify traffic is preferentially forwarded to local node Pods:

   You can verify that the local preference feature is working by checking the Service's endpoints and Pod distribution:

   ```bash
   # Check Service endpoints
   kubectl get endpoints nginx-lb

   # Check which nodes the Pods are distributed on
   kubectl get pods -l app=nginx -o wide
   ```

1. Verify client IP preservation:

   Check the access logs in the nginx Pod to confirm that the recorded client IP is the original client's real IP, not the SNAT-translated IP:

   ```bash
   kubectl exec -it $(kubectl get pods -l app=nginx -o name | head -n1) -- cat /var/log/nginx/access.log
   ```

7.9.6 Notes

IP Address Pool Configuration

The MetalLB address pool range must be a subset of the Underlay subnet CIDR and must be explicitly excluded in the Underlay subnet's `excludeIps` field to avoid IP allocation conflicts.

Network Interface Requirements

MetalLB must use the same network interface as the Kube-OVN Underlay subnet (e.g., `underlay0.341` in the example). This interface should be configured as a VLAN sub-interface to ensure proper broadcasting of ARP messages with VLAN tags for correct MetalLB VIP announcement.

Local Traffic Policy

To enable local preference, two conditions must be met: - Kube-OVN controller enabled with `--enable-ovn-lb-prefer-local=true` parameter - Service configured with `externalTrafficPolicy: Local`

PDF Slack Support

July 30, 2025

April 2, 2025

GitHub

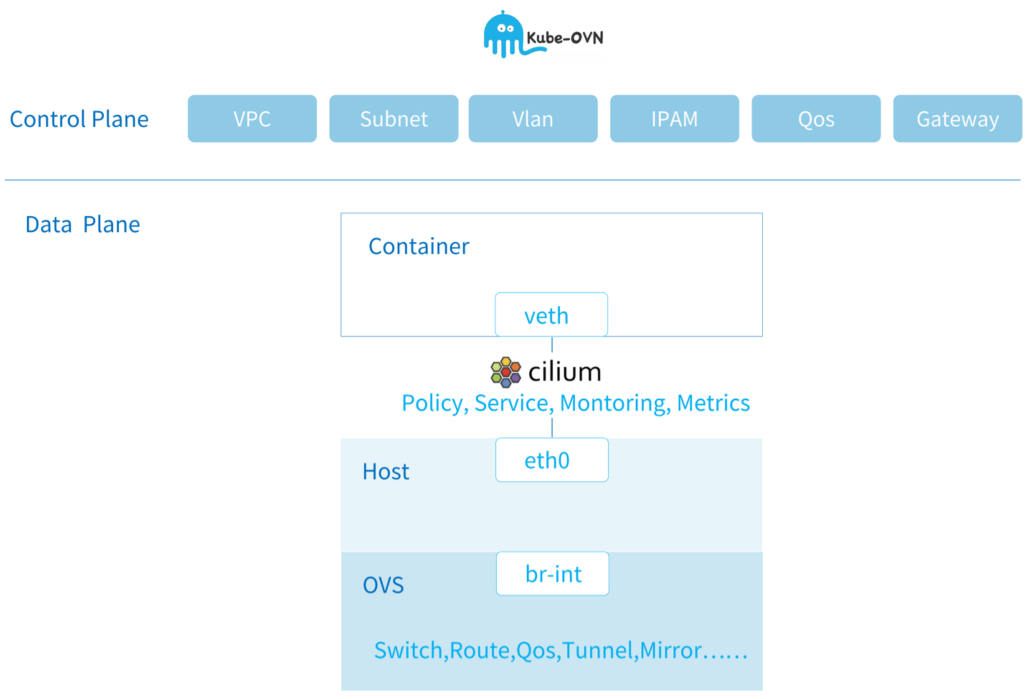
7.9.7 Comments

7.10 Integration with Cilium

Cilium is an eBPF-based networking and security component. Kube-OVN uses the CNI Chaining mode to enhance existing features. Users can use both the rich network abstraction capabilities of Kube-OVN and the monitoring and security capabilities that come with eBPF.

By integrating Cilium, Kube-OVN users can have the following gains:

- Richer and more efficient security policies.
- Hubble-based monitoring and UI.



7.10.1 Prerequisites

1. Linux kernel version above 4.19 or other compatible kernel for full eBPF capability support.
2. Install Helm in advance to prepare for the installation of Cilium, please refer to Installing Helm to deploy Helm.

7.10.2 Configure Kube-OVN

In order to fully utilize the security capabilities of Cilium, you need to disable the `networkpolicy` feature within Kube-OVN and adjust the CNI configuration priority.

Change the following variables in the `install.sh` script:

```
ENABLE_NP=false
CNI_CONFIG_PRIORITY=10
```

If the deployment is complete, you can adjust the args of `kube-ovn-controller`:

```
args:
- --enable-np=false
```

Modify the `kube-ovn-cni` args to adjust the CNI configuration priority:

```
args:
- --cni-conf-name=10-kube-ovn.conflist
```

Adjust the Kube-OVN cni configuration name on each node:

```
mv /etc/cni/net.d/01-kube-ovn.conflist /etc/cni/net.d/10-kube-ovn.conflist
```

7.10.3 Deploy Cilium

Create the `chaining.yaml` configuration file to use Cilium's `generic-veth` mode:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cni-configuration
  namespace: kube-system
data:
  cni-config: |-
    {
      "name": "generic-veth",
      "cniVersion": "0.3.1",
      "plugins": [
        {
          "type": "kube-ovn",
          "server_socket": "/run/openvswitch/kube-ovn-daemon.sock",
          "ipam": {
              "type": "kube-ovn",
              "server_socket": "/run/openvswitch/kube-ovn-daemon.sock"
          }
        },
        {
          "type": "portmap",
          "snat": true,
          "capabilities": {"portMappings": true}
        },
        {
          "type": "cilium-cni"
        }
      ]
    }
```

Installation of the chaining config:

```
kubectl apply -f chaining.yaml
```

Deploying Cilium with Helm:

```
helm repo add cilium https://helm.cilium.io/
helm install cilium cilium/cilium --version 1.11.6 \
    --namespace kube-system \
    --set cni.chainingMode=generic-veth \
    --set cni.customConf=true \
    --set cni.configMap=cni-configuration \
    --set routingMode=native \
    --set enableIPv4Masquerade=false \
    --set devices="eth+ ovn0 genev_sys_6081 vxlan_sys_4789" \
    --set enableIdentityMark=false
```

Confirm that the Cilium installation was successful:

```
# cilium  status
    /¯¯\
 /¯¯\__/¯¯\    Cilium:         OK
 \__/¯¯\__/    Operator:       OK
 /¯¯\__/¯¯\    Hubble:         disabled
 \__/¯¯\__/    ClusterMesh:    disabled
    \__/

DaemonSet         cilium             Desired: 2, Ready: 2/2, Available: 2/2
Deployment        cilium-operator    Desired: 2, Ready: 2/2, Available: 2/2
Containers:       cilium             Running: 2
                  cilium-operator    Running: 2
Cluster Pods:     8/11 managed by Cilium
Image versions    cilium             quay.io/cilium/cilium:v1.10.5@sha256:0612218e28288db360c63677c09fafa2d17edda4f13867bcabf87056046b33bb: 2
                  cilium-operator    quay.io/cilium/operator-generic:v1.10.5@sha256:2d2f730f219d489ff0702923bf24c0002cd93eb4b47ba344375566202f56d972: 2
```

7.10.4 Using Kube-OVN's NAT gateways with Cilium

Cilium natively runs security checks against the traffic going out of interfaces it manages, including the **default interface** of NAT gateways pods. Because the NAT gateway is doing SNAT and DNAT and forwarding the packets to pods within its VPC, the source address of most traffic coming out of the NAT gateway isn't using its Pod IP(s).

Cilium runs **SIP** (Source IP) address validation to prevent that. If the source address of a packet doesn't match one of the IP address of the Pod, the traffic is visibly dropped (can be seen on Hubble as **DROPPED** traffic).

This feature of Cilium is useful to enhance the network security of the cluster and prevent IP spoofing but it breaks NAT gateways. To fix this problem, SIP address validation **needs to be disabled on Cilium** in its Helm Chart by setting the `enableSourceIPVerification` value to `false`:

```
enableSourceIPVerification: false
```

PDF Slack Support

April 14, 2025

May 24, 2022

GitHub

7.10.5 Comments

7.11 Cilium NetworkPolicy Support

Kube-OVN currently supports integration with Cilium, and the specific operation can refer to Cilium integration.

After integrating Cilium, you can use Cilium's excellent network policy capabilities to control the access of Pods in the cluster.The following documents provide integration verification of Cilium L3 and L4 network policy capabilities.

7.11.1 Verification Steps

Create test Pod

Create namespace `test`. Refer to the following yaml, create Pod with label `app=test` in namespace `test` as the destination Pod for testing access.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: test
  name: test
  namespace: test
spec:
  replicas: 1
  selector:
    matchLabels:
      app: test
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: test
    spec:
      containers:
      - image: docker.io/library/nginx:alpine
        imagePullPolicy: IfNotPresent
        name: nginx
```

Similarly, refer to the following yaml, create Pod with label `app=dynamic` in namespace `default` as the Pod for testing access.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: dynamic
  name: dynamic
  namespace: default
spec:
  replicas: 2
  selector:
    matchLabels:
      app: dynamic
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: dynamic
    spec:
      containers:
      - image: docker.io/library/nginx:alpine
        imagePullPolicy: IfNotPresent
        name: nginx
```

View the test Pod and Label information:

```
# kubectl get pod -o wide --show-labels
NAME                       READY   STATUS    RESTARTS   AGE   IP            NODE                     NOMINATED NODE   READINESS GATES   LABELS
dynamic-7d8d7874f5-9v5c4   1/1     Running   0          28h   10.16.0.35    kube-ovn-worker          <none>           <none>            app=dynamic,pod-
template-hash=7d8d7874f5
dynamic-7d8d7874f5-s8z2n   1/1     Running   0          28h   10.16.0.36    kube-ovn-control-plane   <none>           <none>            app=dynamic,pod-
template-hash=7d8d7874f5
# kubectl get pod -o wide -n test --show-labels
```

```
NAME                           READY   STATUS    RESTARTS   AGE     IP             NODE                     NOMINATED NODE   READINESS GATES   LABELS
dynamic-7d8d7874f5-6dsg6       1/1     Running   0          7h20m   10.16.0.2      kube-ovn-control-plane   <none>           <none>            app=dynamic,pod-
template-hash=7d8d7874f5
dynamic-7d8d7874f5-tjgtp       1/1     Running   0          7h46m   10.16.0.42     kube-ovn-worker          <none>           <none>            app=dynamic,pod-
template-hash=7d8d7874f5
label-test1-77b6764857-swq4k   1/1     Running   0          3h43m   10.16.0.12     kube-ovn-worker          <none>           <none>            app=test1,pod-
template-hash=77b6764857

// As the destination Pod for testing access.
test-54c98bc466-mft5s          1/1     Running   0          8h      10.16.0.41     kube-ovn-worker          <none>           <none>            app=test,pod-
template-hash=54c98bc466
```

L3 Network Policy Test

Refer to the following yaml, create `CiliumNetworkPolicy` resource:

```yaml
apiVersion: "cilium.io/v2"
kind: CiliumNetworkPolicy
metadata:
  name: "l3-rule"
  namespace: test
spec:
  endpointSelector:
    matchLabels:
      app: test
  ingress:
  - fromEndpoints:
    - matchLabels:
        app: dynamic
```

At this point, the test Pod in the default namespace cannot access the destination Pod, but the test Pod to the destination Pod in the test namespace is accessible.

Test results in the default namespace:

```
# kubectl exec -it dynamic-7d8d7874f5-9v5c4 -- bash
bash-5.0# ping -c 3 10.16.0.41
PING 10.16.0.41 (10.16.0.41): 56 data bytes

--- 10.16.0.41 ping statistics ---
3 packets transmitted, 0 packets received, 100% packet loss
```

Test results in the test namespace:

```
# kubectl exec -it -n test dynamic-7d8d7874f5-6dsg6 -- bash
bash-5.0# ping -c 3 10.16.0.41
PING 10.16.0.41 (10.16.0.41): 56 data bytes
64 bytes from 10.16.0.41: seq=0 ttl=64 time=2.558 ms
64 bytes from 10.16.0.41: seq=1 ttl=64 time=0.223 ms
64 bytes from 10.16.0.41: seq=2 ttl=64 time=0.304 ms

--- 10.16.0.41 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.223/1.028/2.558 ms
```

Look at the Cilium official document explanation, the `CiliumNetworkPolicy` resource limits the control at the `namespace` level. For more information, please refer to Cilium Limitations.

If there is a network policy rule match, only the Pod in the **same namespace** can access according to the rule, and the Pod in the **other namespace** is denied access by default.

If you want to implement cross-namespace access, you need to specify the namespace information in the rule.

Refer to the document, modify the `CiliumNetworkPolicy` resource, and add namespace information:

```yaml
  ingress:
  - fromEndpoints:
    - matchLabels:
        app: dynamic
        k8s:io.kubernetes.pod.namespace: default    // control the Pod access in other namespace
```

Look at the modified `CiliumNetworkPolicy` resource information:

```
# kubectl get cnp -n test  -o yaml l3-rule
apiVersion: cilium.io/v2
kind: CiliumNetworkPolicy
metadata:
  name: l3-rule
  namespace: test
spec:
  endpointSelector:
    matchLabels:
      app: test
  ingress:
  - fromEndpoints:
    - matchLabels:
        app: dynamic
    - matchLabels:
        app: dynamic
        k8s:io.kubernetes.pod.namespace: default
```

Test the Pod access in the default namespace again, and the destination Pod access is normal:

```
# kubectl exec -it dynamic-7d8d7874f5-9v5c4 -n test -- bash
bash-5.0# ping -c 3 10.16.0.41
PING 10.16.0.41 (10.16.0.41): 56 data bytes
64 bytes from 10.16.0.41: seq=0 ttl=64 time=2.383 ms
64 bytes from 10.16.0.41: seq=1 ttl=64 time=0.115 ms
64 bytes from 10.16.0.41: seq=2 ttl=64 time=0.142 ms

--- 10.16.0.41 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.115/0.880/2.383 ms
```

Using the standard Kubernetes network policy networkpolicy, the test results show that Cilium also restricts access within the same namespace, and cross-namespace access is prohibited.

It is different from Kube-OVN implementation. Kube-OVN supports standard k8s network policy, which restricts the **destination Pod** in a specific namespace, but there is no namespace restriction on the source Pod. Any Pod that meets the restriction rules in any namespace can access the destination Pod.

L4 Network Policy Test

Refer to the following yaml, create `CiliumNetworkPolicy` resource:

```
apiVersion: "cilium.io/v2"
kind: CiliumNetworkPolicy
metadata:
  name: "l4-rule"
  namespace: test
spec:
  endpointSelector:
    matchLabels:
      app: test
  ingress:
  - fromEndpoints:
    - matchLabels:
        app: dynamic
    toPorts:
    - ports:
      - port: "80"
        protocol: TCP
```

Test the access of the Pod that meets the network policy rules in the same namespace

```
# kubectl exec -it -n test dynamic-7d8d7874f5-6dsg6 -- bash
bash-5.0# ping -c 3 10.16.0.41
PING 10.16.0.41 (10.16.0.41): 56 data bytes

--- 10.16.0.41 ping statistics ---
3 packets transmitted, 0 packets received, 100% packet loss
bash-5.0#
bash-5.0# curl 10.16.0.41:80
<html>
<head>
        <title>Hello World!</title>
        <link href='//fonts.googleapis.com/css?family=Open+Sans:400,700' rel='stylesheet' type='text/css'>
        <style>
        body {
                background-color: white;
                text-align: center;
                padding: 50px;
                font-family: "Open Sans","Helvetica Neue",Helvetica,Arial,sans-serif;
        }
        #logo {
                margin-bottom: 40px;
        }
        </style>
</head>
<body>
                <h1>Hello World!</h1>
```

```
                                  <h3>Links found</h3>
          <h3>I am on  test-54c98bc466-mft5s</h3>
          <h3>Cookie                  =</h3>
                                          <b>KUBERNETES</b> listening in 443 available at tcp://10.96.0.1:443<br />
                                                  <h3>my name is hanhouchao!</h3>
                          <h3> RequestURI='/'</h3>
</body>
</html>
```

The Pod that does not meet the network policy rules in the same namespace cannot access

```
# kubectl exec -it -n test label-test1-77b6764857-swq4k -- bash
bash-5.0# ping -c 3 10.16.0.41
PING 10.16.0.41 (10.16.0.41): 56 data bytes

--- 10.16.0.41 ping statistics ---
3 packets transmitted, 0 packets received, 100% packet loss
bash-5.0#
bash-5.0# curl -v 10.16.0.41:80 --connect-timeout 10
*   Trying 10.16.0.41:80...
* After 10000ms connect time, move on!
* connect to 10.16.0.41 port 80 failed: Operation timed out
* Connection timeout after 10001 ms
* Closing connection 0
curl: (28) Connection timeout after 10001 ms
```

After the network policy takes effect, cross-namespace access is still prohibited, which is consistent with the L3 network policy test results.

After the L4 network policy takes effect, ping cannot be used, but TCP access that meets the policy rules can be executed normally.

About the restriction of ICMP, please refer to the official description L4 Limitation Description.

L7 Network Policy Test

chaining mode, L7 network policy currently has problems. In the Cilium official document, there is an explanation for this situation, please refer to Generic Veth Chaining.

This problem is tracked using issue 12454, and it has not been resolved yet.

PDF Slack Support

June 20, 2023

August 2, 2022

GitHub

7.11.2 Comments

7.12 Cilium Network Traffic Observation

Kube-OVN supports Cilium integration, please refer to Cilium integration for details.

Cilium provides rich network traffic observation capabilities, and the flow observability is provided by Hubble. Hubble can observe the traffic across nodes, clusters, and even multi-cluster scenarios.

7.12.1 Install Hubble

In the default Cilium integration installation, the Hubble related components are not installed, so to support traffic observation, you need to supplement the installation of Hubble on the environment.

Execute the following command to install Hubble using helm:

```
helm upgrade cilium cilium/cilium --version 1.11.6 \
   --namespace kube-system \
   --reuse-values \
   --set hubble.relay.enabled=true \
   --set hubble.ui.enabled=true
```

After installing Hubble, execute `cilium status` to check the status of the component and confirm that the installation is successful.

```
# cilium status
    /¯¯\
 /¯¯\__/¯¯\    Cilium:         OK
 \__/¯¯\__/    Operator:       OK
 /¯¯\__/¯¯\    Hubble:         OK
 \__/¯¯\__/    ClusterMesh:    disabled
    \__/

Deployment        hubble-relay       Desired: 1, Ready: 1/1, Available: 1/1
Deployment        cilium-operator    Desired: 2, Ready: 2/2, Available: 2/2
DaemonSet         cilium             Desired: 2, Ready: 2/2, Available: 2/2
Deployment        hubble-ui          Desired: 1, Ready: 1/1, Available: 1/1
Containers:       cilium             Running: 2
                  hubble-ui          Running: 1
                  hubble-relay       Running: 1
                  cilium-operator    Running: 2
Cluster Pods:     16/17 managed by Cilium
Image versions    hubble-relay       quay.io/cilium/hubble-relay:v1.11.6@sha256:fd9034a2d04d5b973f1e8ed44f230ea195b89c37955ff32e34e5aa68f3ed675a: 1
                  cilium-operator    quay.io/cilium/operator-generic:v1.11.6@sha256:9f6063c7bcaede801a39315ec7c166309f6a6783e98665f6693939cf1701bc17: 2
                  cilium             quay.io/cilium/cilium:v1.11.6@sha256:f7f93c26739b6641a3fa3d76b1e1605b15989f25d06625260099e01c8243f54c: 2
                  hubble-ui          quay.io/cilium/hubble-ui:v0.9.0@sha256:0ef04e9a29212925da6bdfd0ba5b581765e41a01f1cc30563cef9b30b457fea0: 1
                  hubble-ui          quay.io/cilium/hubble-ui-backend:v0.9.0@sha256:000df6b76719f607a9edefb9af94dfd1811a6f1b6a8a9c537cba90bf12df474b: 1
apple@bogon cilium %
```

After installing the Hubble component, you need to install the command line to view the traffic information in the environment. Execute the following command to install Hubble CLI:

```
curl -L --fail --remote-name-all https://github.com/cilium/hubble/releases/download/v0.10.0/hubble-linux-amd64.tar.gz
sudo tar xzvfC hubble-linux-amd64.tar.gz /usr/local/bin
```

7.12.2 Deploy and test

Cilium offers a traffic test deployment solution, you can directly use the official deployment solution to deploy the test.

Execute the command `cilium connectivity test`, Cilium will automatically create the `cilium-test` namespace, and deploy the test under cilium-test.

After the normal deployment, you can view the resource information under the `cilium-test` namespace, as follows:

```
# kubectl get all -n cilium-test
NAME                                  READY   STATUS    RESTARTS   AGE
pod/client-7df6cfbf7b-z5t2j           1/1     Running   0          21s
pod/client2-547996d7d8-nvgxg          1/1     Running   0          21s
pod/echo-other-node-d79544ccf-hl4gg   2/2     Running   0          21s
pod/echo-same-node-5d466d5444-ml7tc   2/2     Running   0          21s

NAME                      TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
service/echo-other-node   NodePort   10.109.58.126   <none>        8080:32269/TCP   21s
service/echo-same-node    NodePort   10.108.70.32    <none>        8080:32490/TCP   21s
```

```
NAME                              READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/client            1/1     1            1           21s
deployment.apps/client2           1/1     1            1           21s
deployment.apps/echo-other-node   1/1     1            1           21s
deployment.apps/echo-same-node    1/1     1            1           21s

NAME                                        DESIRED   CURRENT   READY   AGE
replicaset.apps/client-7df6cfbf7b           1         1         1       21s
replicaset.apps/client2-547996d7d8          1         1         1       21s
replicaset.apps/echo-other-node-d79544ccf   1         1         1       21s
replicaset.apps/echo-same-node-5d466d5444   1         1         1       21s
```

7.12.3 Use the command line to observe traffic

By default, the network traffic observation only provides the traffic observed by the Cilium agent on each node.

Execute the `hubble observe` command in the Cilium agent pod under the `kube-system namespace` to view the traffic information on the node.

```
# kubectl get pod -n kube-system -o wide
NAME                                              READY   STATUS    RESTARTS   AGE     IP           NODE                     NOMINATED NODE   READINESS GATES
cilium-d6h56                                      1/1     Running   0          2d20h   172.18.0.2   kube-ovn-worker          <none>           <none>
cilium-operator-5887f78bbb-c7sb2                  1/1     Running   0          2d20h   172.18.0.2   kube-ovn-worker          <none>           <none>
cilium-operator-5887f78bbb-wj8gt                  1/1     Running   0          2d20h   172.18.0.3   kube-ovn-control-plane   <none>           <none>
cilium-tq5xb                                      1/1     Running   0          2d20h   172.18.0.3   kube-ovn-control-plane   <none>           <none>
kube-ovn-pinger-7lgk8                             1/1     Running   0          21h     10.16.0.19   kube-ovn-control-plane   <none>           <none>
kube-ovn-pinger-msvcn                             1/1     Running   0          21h     10.16.0.18   kube-ovn-worker          <none>           <none>

# kubectl exec -it -n kube-system cilium-d6h56 -- bash
root@kube-ovn-worker:/home/cilium# hubble observe --from-namespace kube-system
Jul 29 03:24:25.551: kube-system/kube-ovn-pinger-msvcn:35576 -> 172.18.0.3:6642 to-stack FORWARDED (TCP Flags: ACK, PSH)
Jul 29 03:24:25.561: kube-system/kube-ovn-pinger-msvcn:35576 -> 172.18.0.3:6642 to-stack FORWARDED (TCP Flags: RST)
Jul 29 03:24:25.561: kube-system/kube-ovn-pinger-msvcn:35576 -> 172.18.0.3:6642 to-stack FORWARDED (TCP Flags: ACK, RST)
Jul 29 03:24:25.572: kube-system/kube-ovn-pinger-msvcn:35578 -> 172.18.0.3:6642 to-stack FORWARDED (TCP Flags: SYN)
Jul 29 03:24:25.572: kube-system/kube-ovn-pinger-msvcn:35578 -> 172.18.0.3:6642 to-stack FORWARDED (TCP Flags: ACK)
Jul 29 03:24:25.651: kube-system/kube-ovn-pinger-msvcn:35578 -> 172.18.0.3:6642 to-stack FORWARDED (TCP Flags: ACK, PSH)
Jul 29 03:24:25.661: kube-system/kube-ovn-pinger-msvcn:35578 -> 172.18.0.3:6642 to-stack FORWARDED (TCP Flags: RST)
Jul 29 03:24:25.661: kube-system/kube-ovn-pinger-msvcn:35578 -> 172.18.0.3:6642 to-stack FORWARDED (TCP Flags: ACK, RST)
Jul 29 03:24:25.761: kube-system/kube-ovn-pinger-msvcn:52004 -> 172.18.0.3:6443 to-stack FORWARDED (TCP Flags: ACK, PSH)
Jul 29 03:24:25.779: kube-system/kube-ovn-pinger-msvcn -> kube-system/kube-ovn-pinger-7lgk8 to-stack FORWARDED (ICMPv4 EchoRequest)
Jul 29 03:24:25.779: kube-system/kube-ovn-pinger-msvcn <- kube-system/kube-ovn-pinger-7lgk8 to-endpoint FORWARDED (ICMPv4 EchoReply)
Jul 29 03:24:25.866: kube-system/hubble-ui-7596f7ff6f-7j6f2:55836 <- kube-system/hubble-relay-959988db5-zc5vv:4245 to-stack FORWARDED (TCP Flags: ACK)
Jul 29 03:24:25.866: kube-system/hubble-ui-7596f7ff6f-7j6f2:55836 <- kube-system/hubble-relay-959988db5-zc5vv:80 to-endpoint FORWARDED (TCP Flags: ACK)
Jul 29 03:24:25.866: kube-system/hubble-ui-7596f7ff6f-7j6f2:55836 -> kube-system/hubble-relay-959988db5-zc5vv:4245 to-stack FORWARDED (TCP Flags: ACK)
Jul 29 03:24:25.866: kube-system/hubble-ui-7596f7ff6f-7j6f2:55836 -> kube-system/hubble-relay-959988db5-zc5vv:4245 to-endpoint FORWARDED (TCP Flags: ACK)
Jul 29 03:24:25.975: kube-system/kube-ovn-pinger-7lgk8 -> kube-system/kube-ovn-pinger-msvcn to-endpoint FORWARDED (ICMPv4 EchoRequest)
Jul 29 03:24:25.975: kube-system/kube-ovn-pinger-7lgk8 <- kube-system/kube-ovn-pinger-msvcn to-stack FORWARDED (ICMPv4 EchoReply)
Jul 29 03:24:25.979: kube-system/kube-ovn-pinger-msvcn -> 172.18.0.3 to-stack FORWARDED (ICMPv4 EchoRequest)
Jul 29 03:24:26.037: kube-system/coredns-6d4b75cb6d-lbgjg:36430 -> 172.18.0.3:6443 to-stack FORWARDED (TCP Flags: ACK)
Jul 29 03:24:26.282: kube-system/kube-ovn-pinger-msvcn -> 172.18.0.2 to-stack FORWARDED (ICMPv4 EchoRequest)
```

After deploying Hubble Relay, Hubble can provide complete cluster-wide network traffic observation.

Configure port forwarding

In order to access the Hubble API normally, you need to create a port forwarding to forward the local request to the Hubble Service. You can execute the `kubectl port-forward deployment/hubble-relay -n kube-system 4245:4245` command to open the port forwarding in the current terminal.

The port forwarding configuration can refer to Port Forwarding.

`kubectl port-forward` is a blocking command, you can open a new terminal to execute the following command to observe the traffic information.

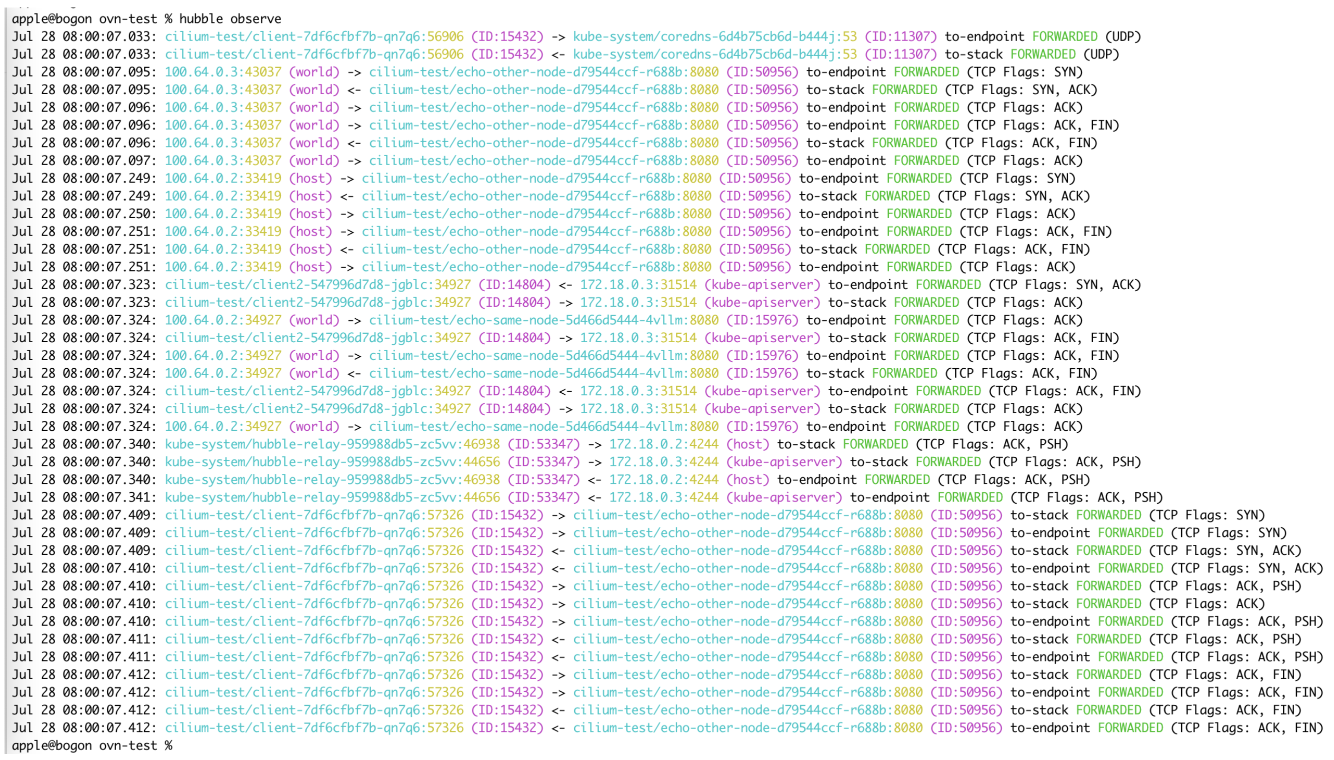
After configuring the port forwarding, execute the `hubble status` command in the terminal. If there is an output similar to the following, the port forwarding configuration is correct, and you can use the command line to observe the traffic.

```
# hubble status
Healthcheck (via localhost:4245): Ok
Current/Max Flows: 8,190/8,190 (100.00%)
Flows/s: 22.86
Connected Nodes: 2/2
```

Use the command line to observe traffic

Execute the `hubble observe` command in the terminal to view the traffic information of the cluster.

The traffic observed by the `cilium-test` namespace is as follows:



```
apple@bogon ovn-test % hubble observe
Jul 28 08:00:07.033: cilium-test/client-7df6cfbf7b-qn7q6:56906 (ID:15432) -> kube-system/coredns-6d4b75cb6d-b444j:53 (ID:11307) to-endpoint FORWARDED (UDP)
Jul 28 08:00:07.033: cilium-test/client-7df6cfbf7b-qn7q6:56906 (ID:15432) <- kube-system/coredns-6d4b75cb6d-b444j:53 (ID:11307) to-stack FORWARDED (UDP)
Jul 28 08:00:07.095: 100.64.0.3:43037 (world) -> cilium-test/echo-other-node-d79544ccf-r688b:8080 (ID:50956) to-endpoint FORWARDED (TCP Flags: SYN)
Jul 28 08:00:07.095: 100.64.0.3:43037 (world) <- cilium-test/echo-other-node-d79544ccf-r688b:8080 (ID:50956) to-stack FORWARDED (TCP Flags: SYN, ACK)
Jul 28 08:00:07.096: 100.64.0.3:43037 (world) -> cilium-test/echo-other-node-d79544ccf-r688b:8080 (ID:50956) to-endpoint FORWARDED (TCP Flags: ACK)
Jul 28 08:00:07.096: 100.64.0.3:43037 (world) -> cilium-test/echo-other-node-d79544ccf-r688b:8080 (ID:50956) to-endpoint FORWARDED (TCP Flags: ACK, FIN)
Jul 28 08:00:07.096: 100.64.0.3:43037 (world) <- cilium-test/echo-other-node-d79544ccf-r688b:8080 (ID:50956) to-stack FORWARDED (TCP Flags: ACK, FIN)
Jul 28 08:00:07.097: 100.64.0.3:43037 (world) -> cilium-test/echo-other-node-d79544ccf-r688b:8080 (ID:50956) to-endpoint FORWARDED (TCP Flags: ACK)
Jul 28 08:00:07.249: 100.64.0.2:33419 (host) -> cilium-test/echo-other-node-d79544ccf-r688b:8080 (ID:50956) to-endpoint FORWARDED (TCP Flags: SYN)
Jul 28 08:00:07.249: 100.64.0.2:33419 (host) <- cilium-test/echo-other-node-d79544ccf-r688b:8080 (ID:50956) to-stack FORWARDED (TCP Flags: SYN, ACK)
Jul 28 08:00:07.250: 100.64.0.2:33419 (host) -> cilium-test/echo-other-node-d79544ccf-r688b:8080 (ID:50956) to-endpoint FORWARDED (TCP Flags: ACK)
Jul 28 08:00:07.251: 100.64.0.2:33419 (host) -> cilium-test/echo-other-node-d79544ccf-r688b:8080 (ID:50956) to-endpoint FORWARDED (TCP Flags: ACK, FIN)
Jul 28 08:00:07.251: 100.64.0.2:33419 (host) <- cilium-test/echo-other-node-d79544ccf-r688b:8080 (ID:50956) to-stack FORWARDED (TCP Flags: ACK, FIN)
Jul 28 08:00:07.251: 100.64.0.2:33419 (host) -> cilium-test/echo-other-node-d79544ccf-r688b:8080 (ID:50956) to-endpoint FORWARDED (TCP Flags: ACK)
Jul 28 08:00:07.323: cilium-test/client2-547996d7d8-jgblc:34927 (ID:14804) <- 172.18.0.3:31514 (kube-apiserver) to-endpoint FORWARDED (TCP Flags: SYN, ACK)
Jul 28 08:00:07.323: cilium-test/client2-547996d7d8-jgblc:34927 (ID:14804) -> 172.18.0.3:31514 (kube-apiserver) to-stack FORWARDED (TCP Flags: ACK)
Jul 28 08:00:07.324: 100.64.0.2:34927 (world) -> cilium-test/echo-same-node-5d466d5444-4vllm:8080 (ID:15976) to-endpoint FORWARDED (TCP Flags: ACK)
Jul 28 08:00:07.324: cilium-test/client2-547996d7d8-jgblc:34927 (ID:14804) -> 172.18.0.3:31514 (kube-apiserver) to-stack FORWARDED (TCP Flags: ACK, FIN)
Jul 28 08:00:07.324: 100.64.0.2:34927 (world) -> cilium-test/echo-same-node-5d466d5444-4vllm:8080 (ID:15976) to-endpoint FORWARDED (TCP Flags: ACK, FIN)
Jul 28 08:00:07.324: 100.64.0.2:34927 (world) <- cilium-test/echo-same-node-5d466d5444-4vllm:8080 (ID:15976) to-stack FORWARDED (TCP Flags: ACK, FIN)
Jul 28 08:00:07.324: cilium-test/client2-547996d7d8-jgblc:34927 (ID:14804) <- 172.18.0.3:31514 (kube-apiserver) to-endpoint FORWARDED (TCP Flags: ACK, FIN)
Jul 28 08:00:07.324: cilium-test/client2-547996d7d8-jgblc:34927 (ID:14804) -> 172.18.0.3:31514 (kube-apiserver) to-stack FORWARDED (TCP Flags: ACK)
Jul 28 08:00:07.324: 100.64.0.2:34927 (world) -> cilium-test/echo-same-node-5d466d5444-4vllm:8080 (ID:15976) to-endpoint FORWARDED (TCP Flags: ACK)
Jul 28 08:00:07.340: kube-system/hubble-relay-959988db5-zc5vv:46938 (ID:53347) -> 172.18.0.2:4244 (host) to-stack FORWARDED (TCP Flags: ACK, PSH)
Jul 28 08:00:07.340: kube-system/hubble-relay-959988db5-zc5vv:44656 (ID:53347) -> 172.18.0.3:4244 (kube-apiserver) to-stack FORWARDED (TCP Flags: ACK, PSH)
Jul 28 08:00:07.340: kube-system/hubble-relay-959988db5-zc5vv:46938 (ID:53347) <- 172.18.0.2:4244 (host) to-endpoint FORWARDED (TCP Flags: ACK, PSH)
Jul 28 08:00:07.341: kube-system/hubble-relay-959988db5-zc5vv:44656 (ID:53347) <- 172.18.0.3:4244 (kube-apiserver) to-endpoint FORWARDED (TCP Flags: ACK, PSH)
Jul 28 08:00:07.409: cilium-test/client-7df6cfbf7b-qn7q6:57326 (ID:15432) -> cilium-test/echo-other-node-d79544ccf-r688b:8080 (ID:50956) to-stack FORWARDED (TCP Flags: SYN)
Jul 28 08:00:07.409: cilium-test/client-7df6cfbf7b-qn7q6:57326 (ID:15432) -> cilium-test/echo-other-node-d79544ccf-r688b:8080 (ID:50956) to-endpoint FORWARDED (TCP Flags: SYN)
Jul 28 08:00:07.409: cilium-test/client-7df6cfbf7b-qn7q6:57326 (ID:15432) <- cilium-test/echo-other-node-d79544ccf-r688b:8080 (ID:50956) to-stack FORWARDED (TCP Flags: SYN, ACK)
Jul 28 08:00:07.410: cilium-test/client-7df6cfbf7b-qn7q6:57326 (ID:15432) <- cilium-test/echo-other-node-d79544ccf-r688b:8080 (ID:50956) to-endpoint FORWARDED (TCP Flags: SYN, ACK)
Jul 28 08:00:07.410: cilium-test/client-7df6cfbf7b-qn7q6:57326 (ID:15432) -> cilium-test/echo-other-node-d79544ccf-r688b:8080 (ID:50956) to-stack FORWARDED (TCP Flags: ACK, PSH)
Jul 28 08:00:07.410: cilium-test/client-7df6cfbf7b-qn7q6:57326 (ID:15432) -> cilium-test/echo-other-node-d79544ccf-r688b:8080 (ID:50956) to-stack FORWARDED (TCP Flags: ACK)
Jul 28 08:00:07.410: cilium-test/client-7df6cfbf7b-qn7q6:57326 (ID:15432) -> cilium-test/echo-other-node-d79544ccf-r688b:8080 (ID:50956) to-endpoint FORWARDED (TCP Flags: ACK, PSH)
Jul 28 08:00:07.411: cilium-test/client-7df6cfbf7b-qn7q6:57326 (ID:15432) <- cilium-test/echo-other-node-d79544ccf-r688b:8080 (ID:50956) to-stack FORWARDED (TCP Flags: ACK, PSH)
Jul 28 08:00:07.411: cilium-test/client-7df6cfbf7b-qn7q6:57326 (ID:15432) <- cilium-test/echo-other-node-d79544ccf-r688b:8080 (ID:50956) to-endpoint FORWARDED (TCP Flags: ACK, PSH)
Jul 28 08:00:07.412: cilium-test/client-7df6cfbf7b-qn7q6:57326 (ID:15432) -> cilium-test/echo-other-node-d79544ccf-r688b:8080 (ID:50956) to-stack FORWARDED (TCP Flags: ACK, FIN)
Jul 28 08:00:07.412: cilium-test/client-7df6cfbf7b-qn7q6:57326 (ID:15432) -> cilium-test/echo-other-node-d79544ccf-r688b:8080 (ID:50956) to-endpoint FORWARDED (TCP Flags: ACK, FIN)
Jul 28 08:00:07.412: cilium-test/client-7df6cfbf7b-qn7q6:57326 (ID:15432) <- cilium-test/echo-other-node-d79544ccf-r688b:8080 (ID:50956) to-stack FORWARDED (TCP Flags: ACK, FIN)
Jul 28 08:00:07.412: cilium-test/client-7df6cfbf7b-qn7q6:57326 (ID:15432) <- cilium-test/echo-other-node-d79544ccf-r688b:8080 (ID:50956) to-endpoint FORWARDED (TCP Flags: ACK, FIN)
apple@bogon ovn-test %
```

Pay attention to the `hubble observe` command display result, which is the traffic information queried when the current command line is executed. Executing the command line multiple times can view different traffic information. For more detailed observation information, you can execute the `hubble help observe` command to view the detailed usage of Hubble CLI.

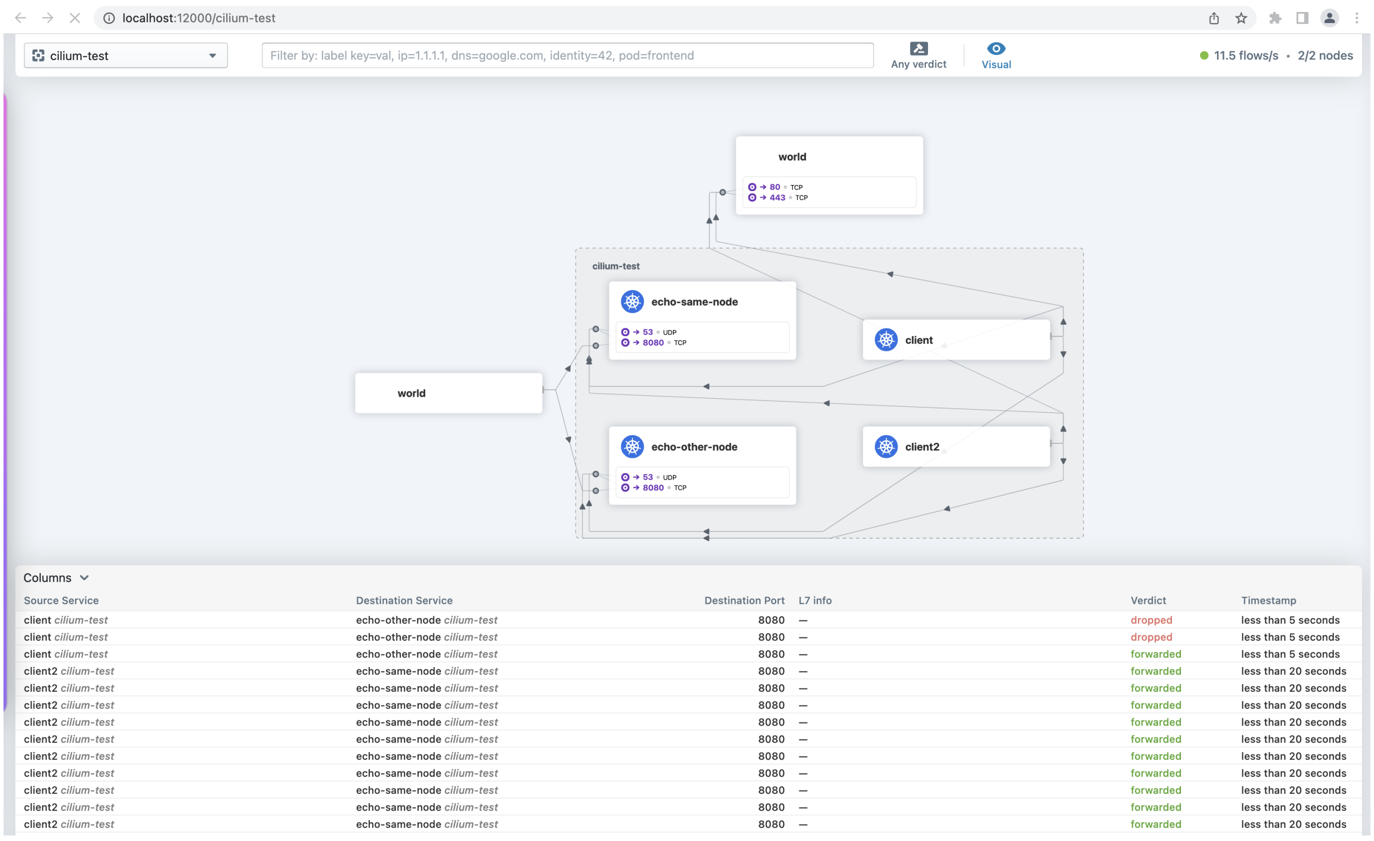
7.12.4 Use UI to observe traffic

Execute the `cilium status` command to confirm that the Hubble UI has been successfully installed. In the second step of the Hubble installation, the installation of the UI has been supplemented.

Execute the command `cilium hubble ui` to automatically create port forwarding and map the `hubble-ui service` to the local port.

When the command is executed normally, the local browser will be automatically opened and jump to the Hubble UI interface. If it does not jump automatically, enter `http://localhost:12000` in the browser to open the UI observation interface.

On the top left of the UI, select the `cilium-test` namespace to view the test traffic information provided by Cilium.



Source Service	Destination Service	Destination Port	L7 info	Verdict	Timestamp
client *cilium-test*	echo-other-node *cilium-test*	8080	—	dropped	less than 5 seconds
client *cilium-test*	echo-other-node *cilium-test*	8080	—	dropped	less than 5 seconds
client *cilium-test*	echo-other-node *cilium-test*	8080	—	forwarded	less than 5 seconds
client2 *cilium-test*	echo-same-node *cilium-test*	8080	—	forwarded	less than 20 seconds
client2 *cilium-test*	echo-same-node *cilium-test*	8080	—	forwarded	less than 20 seconds
client2 *cilium-test*	echo-same-node *cilium-test*	8080	—	forwarded	less than 20 seconds
client2 *cilium-test*	echo-same-node *cilium-test*	8080	—	forwarded	less than 20 seconds
client2 *cilium-test*	echo-same-node *cilium-test*	8080	—	forwarded	less than 20 seconds
client2 *cilium-test*	echo-same-node *cilium-test*	8080	—	forwarded	less than 20 seconds
client2 *cilium-test*	echo-same-node *cilium-test*	8080	—	forwarded	less than 20 seconds
client2 *cilium-test*	echo-same-node *cilium-test*	8080	—	forwarded	less than 20 seconds
client2 *cilium-test*	echo-same-node *cilium-test*	8080	—	forwarded	less than 20 seconds
client2 *cilium-test*	echo-same-node *cilium-test*	8080	—	forwarded	less than 20 seconds

7.12.5 Hubble Traffic Monitoring

Hubble component provides monitoring of Pod network behavior in the cluster. In order to support viewing the monitoring data provided by Hubble, you need to enable monitoring statistics.

Refer to the following command to supplement the `hubble.metrics.enabled` configuration item:

```
helm upgrade cilium cilium/cilium --version 1.11.6 \
   --namespace kube-system \
   --reuse-values \
   --set hubble.relay.enabled=true \
   --set hubble.ui.enabled=true \
   --set hubble.metrics.enabled="{dns,drop,tcp,flow,icmp,http}"
```

After the deployment is completed, you can view the monitoring data provided by Hubble through the `hubble-metrics` service. Execute the following command to view the monitoring data:

```
# curl 172.18.0.2:9091/metrics
# HELP hubble_drop_total Number of drops
# TYPE hubble_drop_total counter
hubble_drop_total{protocol="ICMPv6",reason="Unsupported L3 protocol"} 2
# HELP hubble_flows_processed_total Total number of flows processed
# TYPE hubble_flows_processed_total counter
hubble_flows_processed_total{protocol="ICMPv4",subtype="to-endpoint",type="Trace",verdict="FORWARDED"} 335
hubble_flows_processed_total{protocol="ICMPv4",subtype="to-stack",type="Trace",verdict="FORWARDED"} 335
hubble_flows_processed_total{protocol="ICMPv6",subtype="",type="Drop",verdict="DROPPED"} 2
hubble_flows_processed_total{protocol="TCP",subtype="to-endpoint",type="Trace",verdict="FORWARDED"} 8282
hubble_flows_processed_total{protocol="TCP",subtype="to-stack",type="Trace",verdict="FORWARDED"} 6767
hubble_flows_processed_total{protocol="UDP",subtype="to-endpoint",type="Trace",verdict="FORWARDED"} 1642
hubble_flows_processed_total{protocol="UDP",subtype="to-stack",type="Trace",verdict="FORWARDED"} 1642
# HELP hubble_icmp_total Number of ICMP messages
# TYPE hubble_icmp_total counter
hubble_icmp_total{family="IPv4",type="EchoReply"} 335
hubble_icmp_total{family="IPv4",type="EchoRequest"} 335
hubble_icmp_total{family="IPv4",type="RouterSolicitation"} 2
# HELP hubble_tcp_flags_total TCP flag occurrences
# TYPE hubble_tcp_flags_total counter
hubble_tcp_flags_total{family="IPv4",flag="FIN"} 2043
hubble_tcp_flags_total{family="IPv4",flag="RST"} 301
hubble_tcp_flags_total{family="IPv4",flag="SYN"} 1169
hubble_tcp_flags_total{family="IPv4",flag="SYN-ACK"} 1169
```

PDF Slack Support

June 20, 2023

August 2, 2022

GitHub

7.12.6 Comments

7.13 External Gateway

In some scenarios, all container traffic access to the outside needs to be managed and audited through an external gateway. Kube-OVN can forward outbound traffic to the corresponding external gateway by configuring the appropriate routes in the subnet.

7.13.1 Usage

```yaml
kind: Subnet
apiVersion: kubeovn.io/v1
metadata:
  name: external
spec:
  cidrBlock: 172.31.0.0/16
  gatewayType: centralized
  natOutgoing: false
  externalEgressGateway: 192.168.0.1
  policyRoutingTableID: 1000
  policyRoutingPriority: 1500
```

- `natOutgoing` : needs to be set to `false` .
- `externalEgressGateway` : Set to the address of the external gateway, which needs to be in the same Layer 2 reachable domain as the gateway node.
- `policyRoutingTableID` : The TableID of the local policy routing table used needs to be different for each subnet to avoid conflicts.
- `policyRoutingPriority` : Route priority, in order to avoid subsequent user customization of other routing operations conflict, here you can specify the route priority. If no special needs, you can fill in any value.

PDF Slack Support

July 3, 2022

May 24, 2022

GitHub

7.13.2 Comments

7.14 VIP reserved IP

VIP Virtual IP addresses are reserved for IP addresses. The reason for the design of VIP is that the IP and POD of kube-ovn are directly related in naming, so the function of reserving IP can not be realized directly based on IP. At the beginning of the design, VIP refers to the function of Openstack neutron Allowed-Address-Pairs(AAP), which can be used for Openstack octavia load balancer projects. It can also be used to provide in-machine application (POD) IP, as seen in the aliyun terway project. In addition, because neutron has the function of reserving IP, VIP has been extended to a certain extent, so that VIP can be directly used to reserve IP for POD, but this design will lead to the function of VIP and IP become blurred, which is not an elegant way to achieve, so it is not recommended to use in production. In addition, since the Switch LB of OVN can provide a function of using the internal IP address of the subnet as the front-end VIP of the LB, the scenario of using the OVN Switch LB Rule in the subnet for the VIP is extended. In short, there are only three use cases for VIP design at present:

- Allowed-Address-Pairs VIP
- Switch LB rule VIP
- Pod uses VIP to fix IP

7.14.1 1. Allowed-Address-Pairs VIP

In this scenario, we want to dynamically reserve a part of the IP but not allocate it to Pods but to other infrastructure enables, such as:

- Kubernetes nesting scenarios In which the upper-layer Kubernetes uses the Underlay network, the underlying Subnet addresses are used.
- LB or other network infrastructure needs to use an IP within a Subnet, but does not have a separate Pod.

In addition, VIP can reserve IP for Allowed-Address-Pairs to support the scenario in which a single NIC is configured with multiple IP addresses, for example:

- Keepalived can help with fast failover and flexible load balancing architecture by configuring additional IP address pairs

1.1 Automatically assign addresses to VIP

If you just want to reserve a number of IP addresses without requiring the IP address itself, you can use the following yaml to create:

```yaml
apiVersion: kubeovn.io/v1
kind: Vip
metadata:
  name: vip-dynamic-01
spec:
  subnet: ovn-default
  type: ""
```

- `subnet` : The IP address is reserved from the Subnet.
- `type` : Currently, two types of ip addresses are supported. If the value is empty, it indicates that the ip address is used only for ipam ip addresses. switch_lb_vip indicates that the IP address is used only for switch lb.

Query the VIP after it is created:

```
# kubectl get vip
NAME             V4IP         PV4IP   MAC                 PMAC   V6IP   PV6IP   SUBNET        READY
vip-dynamic-01   10.16.0.12           00:00:00:F0:DB:25                         ovn-default   true
```

It can be seen that the VIP is assigned an IP address of '10.16.0.12', which can be used by other network infrastructures later.

1.2 Use fixed address VIP

If there is a need for the reserved VIP IP address, the following yaml can be used for fixed allocation:

```yaml
apiVersion: kubeovn.io/v1
kind: Vip
metadata:
  name: static-vip01
spec:
  subnet: ovn-default
  v4ip: "10.16.0.121"
```

- `subnet`: The IP address is reserved from the Subnet.
- `v4ip`: Fixed assigned IP address, which must be within the CIDR range of 'subnet'.

Query the VIP after it is created:

```
# kubectl get vip
NAME             V4IP         PV4IP   MAC                 PMAC   V6IP   PV6IP   SUBNET        READY
static-vip01   10.16.0.121           00:00:00:F0:DB:26                         ovn-default   true
```

1.3 Pod Uses VIP to enable AAP

Pod can use annotation to specify VIP to enable AAP function. labels must meet the condition of node selector in VIP.

Pod annotation supports specifying multiple VIPs. The configuration format is: `ovn.kubernetes.io/aaps: vip-aap,vip-aap2,vip-aap3`

AAP support multi nic, if a Pod is configured with multiple nics, AAP will configure the Port corresponding to the same subnet of the Pod and VIP.

1.3.1 CREATE VIP SUPPORT AAP

```yaml
apiVersion: kubeovn.io/v1
kind: Vip
metadata:
  name: vip-aap
spec:
  subnet: ovn-default
  namespace: default
  selector:
    - "app: aap1"
```

VIP also supports the assignment of fixed and random addresses, as described above.

- `namespace`: In AAP scenarios, a VIP needs to specify a namespace explicitly. Only resources in the same namespace can enable the AAP function.
- `selector`: In the AAP scenario, the node selector used to select the Pod attached to the VIP has the same format as the NodeSelector format in Kubernetes.

Query the Port corresponding to the VIP:

```
# kubectl ko nbctl show ovn-default
switch e32e1d3b-c539-45f4-ab19-be4e33a061f6 (ovn-default)
    port aap-vip
        type: virtual
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: busybox
  annotations:
    ovn.kubernetes.io/aaps: vip-aap
  labels:
    app: aap1
spec:
  containers:
    - name: busybox
      image: busybox
      command: ["sleep", "3600"]
      securityContext:
        capabilities:
          add:
            - NET_ADMIN
```

Query the configuration of the AAP after the AAP is created:

```
# kubectl ko nbctl list logical_switch_port aap-vip
_uuid               : cd930750-0533-4f06-a6c0-217ddac73272
addresses           : []
```

```
dhcpv4_options      : []
dhcpv6_options      : []
dynamic_addresses   : []
enabled             : []
external_ids        : {ls=ovn-default, vendor=kube-ovn}
ha_chassis_group    : []
mirror_rules        : []
name                : aap-vip
options             : {virtual-ip="10.16.0.100", virtual-parents="busybox.default"}
parent_name         : []
port_security       : []
tag                 : []
tag_request         : []
type                : virtual
up                  : false
```

virtual-ip is set to the IP address reserved for the VIP, and virtual-parents is set to the Port of the Pod whose AAP function is enabled.

Query the configuration of the Pod after the POD is created:

```
# kubectl exec -it busybox -- ip addr add 10.16.0.100/16 dev eth0
# kubectl exec -it busybox01 -- ip addr show eth0
35: eth0@if36: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1400 qdisc noqueue
    link/ether 00:00:00:e2:ab:0c brd ff:ff:ff:ff:ff:ff
    inet 10.16.0.7/16 brd 10.16.255.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet 10.16.0.100/16 scope global secondary eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::200:ff:fee2:ab0c/64 scope link
       valid_lft forever preferred_lft forever
```

In addition to the IP assigned automatically when the Pod is created, the IP of the VIP is also successfully bound, and other Pods in the current subnet can communicate with these two IP addresses.

7.14.2 2. Switch LB rule vip

```yaml
apiVersion: kubeovn.io/v1
kind: Vip
metadata:
  name: slr-01
spec:
  subnet: ovn-default
  type: switch_lb_vip
```

- `subnet` : The IP address is reserved from the Subnet.
- `type` : Currently, two types of ip addresses are supported. If the value is empty, it indicates that the ip address is used only for ipam ip addresses. switch_lb_vip indicates that the IP address is used only for switch lb.

7.14.3 3. POD Use VIP to reserve IP address

It is not recommended to use this function in production because the distinction between this function and IP function is not clear.

```yaml
apiVersion: kubeovn.io/v1
kind: Vip
metadata:
  name: pod-use-vip
spec:
  subnet: ovn-default
  type: ""
```

This feature has been supported since v1.12.

You can use annotations to assign a VIP to a Pod, then the pod will use the vip's ip address:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: static-ip
  annotations:
    ovn.kubernetes.io/vip: pod-use-vip # use vip name
  namespace: default
spec:
  containers:
    - name: static-ip
      image: docker.io/library/nginx:alpine
```

3.1 StatefulSet and Kubevirt VM retain VIP

Due to the particularity of 'StatefulSet' and 'VM', after their Pod is destroyed and pulled up, it will re-use the previously set VIP.

VM retention VIP needs to ensure that 'kube-ovn-controller' 'keep-vm-ip' parameter is' true '. Please refer to Kubevirt VM enable keep its ip

PDF **Slack** **Support**

July 30, 2025

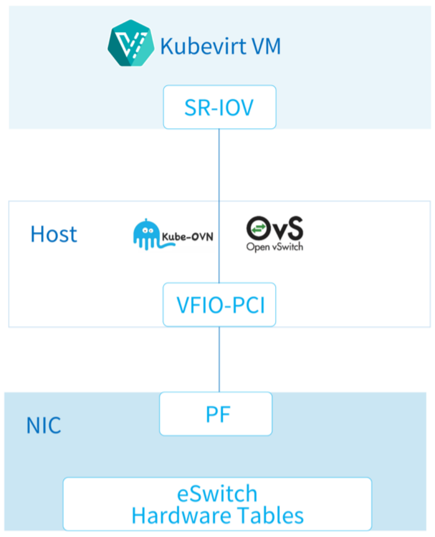
May 24, 2022

GitHub +1

7.14.4 Comments

7.15 Offload with Mellanox

Kube-OVN uses OVS for traffic forwarding in the final data plane, and the associated flow table matching, tunnel encapsulation and other functions are CPU-intensive, which consumes a lot of CPU resources and leads to higher latency and lower throughput under heavy traffic. Mellanox Accelerated Switching And Packet Processing (ASAP²) technology offloads OVS-related operations to an eSwitch within the eSwitch in the hardware. This technology can shorten the data path without modifying the OVS control plane, avoiding the use of host CPU resources, which dramatically reduce latency and significantly increase the throughput.



Note

The solution described in this article was verified in 2022. However, hardware NICs may now have new features, and some limitations mentioned may have been resolved. Please consult your hardware vendor for the latest technical constraints and capabilities.

7.15.1 Prerequisites

- Mellanox CX5/CX6/BlueField that support ASAP².
- CentOS 8 Stream or Linux 5.7 above.
- Since the current NIC does not support `dp_hash` and `hash` operation offload, OVN LB function should be disabled.
- In order to configure offload mode, the network card cannot be bound to a bond.

7.15.2 Configure SR-IOV and Device Plugin

Mellanox network card supports two ways to configure offload, one is to manually configure the network card SR-IOV and Device Plugin, and the other is to use sriov-network-operator Perform automatic configuration.

Manually configure SR-IOV and Device Plugin

CONFIGURE SR-IOV

Query the device ID of the network card, in the following example it is `84:00.0` and `84.00.1`:

```
# lspci -nn | grep ConnectX-5
84:00.0 Ethernet controller [0200]: Mellanox Technologies MT27800 Family [ConnectX-5] [15b3:1017]
84:00.1 Ethernet controller [0200]: Mellanox Technologies MT27800 Family [ConnectX-5] [15b3:1017]
```

Find the corresponding NIC by its device ID:

```
# ls -l /sys/class/net/ | grep 84:00.0
lrwxrwxrwx 1 root root 0 Feb 4 16:16 enp132s0f0np0 -> ../../devices/pci0000:80/0000:80:08.0/0000:84:00.0/net/enp132s0f0np0
# ls -l /sys/class/net/ | grep 84:00.1
lrwxrwxrwx 1 root root 0 Feb 4 16:16 enp132s0f1np1 -> ../../devices/pci0000:80/0000:80:08.0/0000:84:00.1/net/enp132s0f1np1
```

Check whether the network card is bound to bond:

In this example, the network cards enp132s0f0np0 and enp132s0f1np1 are bound to bond1

```
# ip link show enp132s0f0np0 | grep bond
160: enp132s0f0np0: <BROADCAST,MULTICAST,SLAVE,UP,LOWER_UP> mtu 1500 qdisc mq master bond1 state UP mode DEFAULT group default qlen 1000
# ip link show enp132s0f1np1 | grep bond
169: enp132s0f1np1: <BROADCAST,MULTICAST,SLAVE,UP,LOWER_UP> mtu 1500 qdisc mq master bond1 state UP mode DEFAULT group default qlen 1000
```

Remove bond and existing VF:

```
ifenslave -d bond1 enp132s0f0np0
ifenslave -d bond1 enp132s0f1np1
echo 0 > /sys/class/net/enp132s0f0np0/device/sriov_numvfs
echo 0 > /sys/class/net/enp132s0f1np1/device/sriov_numvfs
ip link set enp132s0f0np0 down
ip link set enp132s0f1np1 down
```

Steering Mode:

OVS-kernel supports two steering modes for rule insertion into hardware:

- SMFS (software-managed flow steering): default mode; rules are inserted directly to the hardware by the software (driver). This mode is optimized for rule insertion.
- DMFS (device-managed flow steering): rule insertion is done using firmware commands. This mode is optimized for throughput with a small amount of rules in the system.

The steering mode can be configured via sysfs or devlink API in kernels that support it:

```
# Configure via sysfs
echo <smfs|dmfs> > /sys/class/net/enp132s0f0np0/compat/devlink/steering_mode
echo <smfs|dmfs> > /sys/class/net/enp132s0f1np1/compat/devlink/steering_mode
# Configure via devlink
devlink dev param set pci/84:00.0 name flow_steering_mode value smfs cmode runtime
devlink dev param set pci/84:00.1 name flow_steering_mode value smfs cmode runtime
```

Note: If you don't know which mode to choose, you can use the default mode without configuration.

Check the number of available VFs:

```
# cat /sys/class/net/enp132s0f0np0/device/sriov_totalvfs
127
# cat /sys/class/net/enp132s1f0np1/device/sriov_totalvfs
127
```

Create VFs and do not exceeding the number found above:

```
# echo '4' > /sys/class/net/enp132s0f0np0/device/sriov_numvfs
# echo '4' > /sys/class/net/enp132s1f0np1/device/sriov_numvfs
# ip link show enp132s0f0np0
160: enp132s0f0np0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc mq state DOWN mode DEFAULT group default qlen 1000
    link/ether 08:c0:eb:74:c3:4a brd ff:ff:ff:ff:ff:ff
    vf 0 link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff, spoof checking off, link-state disable, trust off, query_rss off
    vf 1 link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff, spoof checking off, link-state disable, trust off, query_rss off
    vf 2 link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff, spoof checking off, link-state disable, trust off, query_rss off
    vf 3 link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff, spoof checking off, link-state disable, trust off, query_rss off
# ip link show enp132s0f1np1
```

```
169: enp132s0f1np1: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc mq state DOWN mode DEFAULT group default qlen 1000
    link/ether 08:c0:eb:74:c3:4b brd ff:ff:ff:ff:ff:ff
    vf 0 link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff, spoof checking off, link-state disable, trust off, query_rss off
    vf 1 link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff, spoof checking off, link-state disable, trust off, query_rss off
    vf 2 link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff, spoof checking off, link-state disable, trust off, query_rss off
    vf 3 link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff, spoof checking off, link-state disable, trust off, query_rss off
# ip link set enp132s0f0np0 up
# ip link set enp132s0f1np1 up
```

Find the device IDs corresponding to the above VFs:

```
# lspci -nn | grep ConnectX-5 | grep Virtual
84:00.2 Ethernet controller [0200]: Mellanox Technologies MT27800 Family [ConnectX-5 Virtual Function] [15b3:1018]
84:00.3 Ethernet controller [0200]: Mellanox Technologies MT27800 Family [ConnectX-5 Virtual Function] [15b3:1018]
84:00.4 Ethernet controller [0200]: Mellanox Technologies MT27800 Family [ConnectX-5 Virtual Function] [15b3:1018]
84:00.5 Ethernet controller [0200]: Mellanox Technologies MT27800 Family [ConnectX-5 Virtual Function] [15b3:1018]
84:00.6 Ethernet controller [0200]: Mellanox Technologies MT27800 Family [ConnectX-5 Virtual Function] [15b3:1018]
84:00.7 Ethernet controller [0200]: Mellanox Technologies MT27800 Family [ConnectX-5 Virtual Function] [15b3:1018]
84:01.0 Ethernet controller [0200]: Mellanox Technologies MT27800 Family [ConnectX-5 Virtual Function] [15b3:1018]
84:01.1 Ethernet controller [0200]: Mellanox Technologies MT27800 Family [ConnectX-5 Virtual Function] [15b3:1018]
```

Unbound the VFs from the driver:

```
echo 0000:84:00.2 > /sys/bus/pci/drivers/mlx5_core/unbind
echo 0000:84:00.3 > /sys/bus/pci/drivers/mlx5_core/unbind
echo 0000:84:00.4 > /sys/bus/pci/drivers/mlx5_core/unbind
echo 0000:84:00.5 > /sys/bus/pci/drivers/mlx5_core/unbind
echo 0000:84:00.6 > /sys/bus/pci/drivers/mlx5_core/unbind
echo 0000:84:00.7 > /sys/bus/pci/drivers/mlx5_core/unbind
echo 0000:84:01.0 > /sys/bus/pci/drivers/mlx5_core/unbind
echo 0000:84:01.1 > /sys/bus/pci/drivers/mlx5_core/unbind
```

Enable eSwitch mode and set up hardware offload:

```
devlink dev eswitch set pci/0000:84:00.0 mode switchdev
devlink dev eswitch set pci/0000:84:00.1 mode switchdev
ethtool -K enp132s0f0np0 hw-tc-offload on
ethtool -K enp132s0f1np1 hw-tc-offload on
```

SR-IOV VF LAG:

SR-IOV VF LAG allows the NIC's physical functions (PFs) to get the rules that the OVS tries to offload to the bond net-device, and to offload them to the hardware e-switch.The supported bond modes are as follows:

- Active-backup
- XOR
- LACP

SR-IOV VF LAG enables complete offload of the LAG functionality to the hardware. The bonding creates a single bonded PF port. Packets from the up-link can arrive from any of the physical ports and are forwarded to the bond device.When hardware offload is used, packets from both ports can be forwarded to any of the VFs. Traffic from the VF can be forwarded to both ports according to the bonding state. This means that when in active-backup mode, only one PF is up, and traffic from any VF goes through this PF. When in XOR or LACP mode, if both PFs are up, traffic from any VF is split between these two PFs.

In this example, LACP mode will be used, and the configuration is as follows:

```
modprobe bonding mode=802.3ad
ip link set enp132s0f0np0 master bond1
ip link set enp132s0f1np1 master bond1
ip link set enp132s0f0np0 up
ip link set enp132s0f1np1 up
ip link set bond1 up
```

Note: If you do not need to bind bond, please ignore the above operation.

Rebind the driver and complete the VF setup:

```
echo 0000:84:00.2 > /sys/bus/pci/drivers/mlx5_core/bind
echo 0000:84:00.3 > /sys/bus/pci/drivers/mlx5_core/bind
echo 0000:84:00.4 > /sys/bus/pci/drivers/mlx5_core/bind
echo 0000:84:00.5 > /sys/bus/pci/drivers/mlx5_core/bind
echo 0000:84:00.6 > /sys/bus/pci/drivers/mlx5_core/bind
echo 0000:84:00.7 > /sys/bus/pci/drivers/mlx5_core/bind
echo 0000:84:01.0 > /sys/bus/pci/drivers/mlx5_core/bind
echo 0000:84:01.1 > /sys/bus/pci/drivers/mlx5_core/bind
```

Some behaviors of `NetworkManager` may cause driver exceptions, if offloading problems occur we recommended to close `NetworkManager` and try again.

```
systemctl stop NetworkManager
systemctl disable NetworkManager
```

CONFIGURE DEVICE PLUGIN

Since each machine has a limited number of VFs and each Pod that uses acceleration will take up VF resources, we need to use the SR-IOV Device Plugin to manage the corresponding resources so that the scheduler knows how to schedule.

Create SR-IOV Configmap:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: sriovdp-config
  namespace: kube-system
data:
  config.json: |
    {
      "resourceList": [{
          "resourcePrefix": "mellanox.com",
          "resourceName": "cx5_sriov_switchdev",
          "selectors": {
                  "vendors": ["15b3"],
                  "devices": ["1018"],
                  "drivers": ["mlx5_core"]
              }
      }
      ]
    }
```

This plugin creates device plugin endpoints based on the configurations given in the config map associated with the SR-IOV Network Device Plugin.

- `selectors`: VF selectors
- `vendors`: Target device's vendor Hex code as string
- `devices`: Target Devices' device Hex code as string
- `drivers`: Target device driver names as string

`selectors` also supports VF selection based on `pciAddresses`, `acpiIndexes` and other parameters. For more detailed configuration, please refer to SR-IOV ConfigMap configuration

Please read the SR-IOV device plugin to deploy:

```
kubectl apply -f https://raw.githubusercontent.com/k8snetworkplumbingwg/sriov-network-device-plugin/v3.6.2/deployments/sriovdp-daemonset.yaml
```

Check if SR-IOV resources have been registered to Kubernetes Node:

```
kubectl describe node kube-ovn-01  | grep mellanox

mellanox.com/cx5_sriov_switchdev:  8
mellanox.com/cx5_sriov_switchdev:  8
mellanox.com/cx5_sriov_switchdev  0            0
```

Configure SR-IOV and Device Plugin using sriov-network-operator

Install node-feature-discovery to automatically detect hardware functions and system configuration:

```
kubectl apply -k https://github.com/kubernetes-sigs/node-feature-discovery/deployment/overlays/default?ref=v0.11.3
```

Or use the following command to manually add annotation to the network card with offload capability:

```
kubectl label nodes [offloadNicNode] feature.node.kubernetes.io/network-sriov.capable=true
```

Clone the code repository and install the Operator:

```
git clone --depth=1 https://github.com/kubeovn/sriov-network-operator.git
kubectl apply -k sriov-network-operator/deploy
```

Check if the Operator component is working properly:

```
# kubectl get -n kube-system all | grep sriov
NAME                                         READY   STATUS    RESTARTS   AGE
pod/sriov-network-config-daemon-bf9nt        1/1     Running   0          8s
pod/sriov-network-operator-54d7545f65-296gb   1/1     Running   0          10s

NAME                                        DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE
SELECTOR                                                AGE
daemonset.apps/sriov-network-config-daemon   1         1         1       1            1           beta.kubernetes.io/os=linux,feature.node.kubernetes.io/
network-sriov.capable=true   8s

NAME                                    READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/sriov-network-operator   1/1     1            1           10s

NAME                                                DESIRED   CURRENT   READY   AGE
replicaset.apps/sriov-network-operator-54d7545f65   1         1         1       10s
```

Check `SriovNetworkNodeState`, taking the `node1` node as an example, there are two Mellanox network cards on this node:

```
# kubectl get sriovnetworknodestates.sriovnetwork.openshift.io -n kube-system node1 -o yaml
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodeState
spec: ...
status:
  interfaces:
  - deviceID: "1017"
    driver: mlx5_core
    mtu: 1500
    pciAddress: "0000:5f:00.0"
    totalvfs: 8
    vendor: "15b3"
    linkSeed: 25000Mb/s
    linkType: ETH
    mac: 08:c0:eb:f4:85:bb
    name: ens41f0np0
  - deviceID: "1017"
    driver: mlx5_core
    mtu: 1500
    pciAddress: "0000:5f:00.1"
    totalvfs: 8
    vendor: "15b3"
    linkSeed: 25000Mb/s
    linkType: ETH
    mac: 08:c0:eb:f4:85:bb
    name: ens41f1np1
```

Create the `SriovNetworkNodePolicy` resource and select the network card to be managed through `nicSelector`:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: policy
  namespace: kube-system
spec:
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  eSwitchMode: switchdev
  numVfs: 3
  nicSelector:
    pfNames:
    - ens41f0np0
    - ens41f1np1
  resourceName: cx_sriov_switchdev
```

Check the `status` field of `SriovNetworkNodeState` again:

```
# kubectl get sriovnetworknodestates.sriovnetwork.openshift.io -n kube-system node1 -o yaml

...
spec:
  interfaces:
  - eSwitchMode: switchdev
    name: ens41f0np0
    numVfs: 3
    pciAddress: 0000:5f:00.0
    vfGroups:
    - policyName: policy
      vfRange: 0-2
      resourceName: cx_sriov_switchdev
  - eSwitchMode: switchdev
    name: ens41f1np1
    numVfs: 3
    pciAddress: 0000:5f:00.1
    vfGroups:
    - policyName: policy
      vfRange: 0-2
      resourceName: cx_sriov_switchdev
status:
  interfaces
  - Vfs:
    - deviceID: 1018
      driver: mlx5_core
      pciAddress: 0000:5f:00.2
      vendor: "15b3"
    - deviceID: 1018
      driver: mlx5_core
      pciAddress: 0000:5f:00.3
      vendor: "15b3"
    - deviceID: 1018
      driver: mlx5_core
      pciAddress: 0000:5f:00.4
      vendor: "15b3"
    deviceID: "1017"
    driver: mlx5_core
    linkSeed: 25000Mb/s
    linkType: ETH
    mac: 08:c0:eb:f4:85:ab
    mtu: 1500
    name: ens41f0np0
    numVfs: 3
    pciAddress: 0000:5f:00.0
    totalvfs: 3
    vendor: "15b3"
  - Vfs:
    - deviceID: 1018
      driver: mlx5_core
      pciAddress: 0000:5f:00.5
      vendor: "15b3"
    - deviceID: 1018
      driver: mlx5_core
      pciAddress: 0000:5f:00.6
      vendor: "15b3"
    - deviceID: 1018
      driver: mlx5_core
      pciAddress: 0000:5f:00.7
      vendor: "15b3"
    deviceID: "1017"
    driver: mlx5_core
    linkSeed: 25000Mb/s
    linkType: ETH
    mac: 08:c0:eb:f4:85:bb
    mtu: 1500
    name: ens41f1np1
    numVfs: 3
    pciAddress: 0000:5f:00.1
    totalvfs: 3
    vendor: "15b3"
```

Check the status of VF:

```
# lspci -nn | grep ConnectX
5f:00.0 Ethernet controller [0200]: Mellanox Technologies MT27800 Family [ConnectX-5] [15b3:1017]
5f:00.1 Ethernet controller [0200]: Mellanox Technologies MT27800 Family [ConnectX-5] [15b3:1017]
5f:00.2 Ethernet controller [0200]: Mellanox Technologies MT27800 Family [ConnectX-5 Virtual Function] [15b3:1018]
5f:00.3 Ethernet controller [0200]: Mellanox Technologies MT27800 Family [ConnectX-5 Virtual Function] [15b3:1018]
5f:00.4 Ethernet controller [0200]: Mellanox Technologies MT27800 Family [ConnectX-5 Virtual Function] [15b3:1018]
5f:00.5 Ethernet controller [0200]: Mellanox Technologies MT27800 Family [ConnectX-5 Virtual Function] [15b3:1018]
5f:00.6 Ethernet controller [0200]: Mellanox Technologies MT27800 Family [ConnectX-5 Virtual Function] [15b3:1018]
5f:00.7 Ethernet controller [0200]: Mellanox Technologies MT27800 Family [ConnectX-5 Virtual Function] [15b3:1018]
```

Check PF working mode:

```
# cat /sys/class/net/ens41f0np0/compat/devlink/mode
switchdev
```

7.15.3 Install Multus-CNI

The device IDs obtained during SR-IOV Device Plugin scheduling need to be passed to Kube-OVN via Multus-CNI, so Multus-CNI needs to be configured to perform the related tasks.

Please read Multus-CNI Document to deploy:

```
kubectl apply -f https://raw.githubusercontent.com/k8snetworkplumbingwg/multus-cni/v4.0.2/deployments/multus-daemonset-thick.yml
```

Note: multus provides Thin and Thick versions of the plug-in. To support SR-IOV, you need to install the Thick version.

Create `NetworkAttachmentDefinition`:

```yaml
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: sriov
  namespace: default
  annotations:
    k8s.v1.cni.cncf.io/resourceName: mellanox.com/cx5_sriov_switchdev
spec:
  config: '{
    "cniVersion": "0.3.1",
    "name": "kube-ovn",
    "plugins":[
        {
            "type":"kube-ovn",
            "server_socket":"/run/openvswitch/kube-ovn-daemon.sock",
            "provider": "sriov.default.ovn"
        },
        {
            "type":"portmap",
            "capabilities":{
                "portMappings":true
            }
        }
    ]
}'
```

- `provider` : the format should be {name}.{namespace}.ovn of related `NetworkAttachmentDefinition` .

7.15.4 Overlay offload

Enable Offload in Kube-OVN

Download the scripts:

```
wget https://raw.githubusercontent.com/kubeovn/kube-ovn/release-1.14/dist/images/install.sh
```

Change the related options, `IFACE` should be the physic NIC and has an IP:

```
ENABLE_MIRROR=${ENABLE_MIRROR:-false}
HW_OFFLOAD=${HW_OFFLOAD:-true}
ENABLE_LB=${ENABLE_LB:-false}
IFACE="bond1"
# Take manual configuration of the network card in SR-IOV and Device Plugin as an example. If bond is bound, set IFACE to bond1. If bond is not bound, set
IFACE to enp132s0f0np0 or enp132s0f1np1.
```

Install Kube-OVN:

```
bash install.sh
```

Create Pods with VF NICs

Pods that use VF for network offload acceleration can be created using the following yaml:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx-overlay
  annotations:
    v1.multus-cni.io/default-network: default/sriov
    sriov.default.ovn.kubernetes.io/logical_switch: ovn-default
spec:
  containers:
  - name: nginx-overlay
    image: docker.io/library/nginx:alpine
    resources:
      requests:
        mellanox.com/cx5_sriov_switchdev: '1'
      limits:
        mellanox.com/cx5_sriov_switchdev: '1'
```

- `v1.multus-cni.io/default-network` : is the {namespace}/{name} of `NetworkAttachmentDefinition` in the previous step.
- `sriov.default.ovn.kubernetes.io/logical_switch` : Specify the Subnet to which the Pod belongs. If you want the subnet to which the Pod belongs to be the default subnet, this line annotation can be omitted.

7.15.5 Underlay offload

Enable Offload in Kube-OVN

Download the scripts:

```
wget https://raw.githubusercontent.com/kubeovn/kube-ovn/release-1.14/dist/images/install.sh
```

Change the related options, `IFACE` should be the physic NIC and has an IP:

```
ENABLE_MIRROR=${ENABLE_MIRROR:-false}
HW_OFFLOAD=${HW_OFFLOAD:-true}
ENABLE_LB=${ENABLE_LB:-false}
IFACE=""
# If Underlay uninstallation is required, IFACE needs to be set to other non-PF network cards. (When IFACE is empty, the K8s cluster communication network card will be used by default. Note that this network card cannot be a PF network card)
```

Install Kube-OVN:

```
bash install.sh
```

Create Pods with VF NICs

Pods that use VF for network offload acceleration can be created using the following yaml:

```
apiVersion: kubeovn.io/v1
kind: ProviderNetwork
metadata:
  name: underlay-offload
spec:
  defaultInterface: bond1

---
apiVersion: kubeovn.io/v1
kind: Vlan
metadata:
  name: vlan0
spec:
  id: 0
  provider: underlay-offload

---
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
  name: vlan0
spec:
  protocol: IPv4
  provider: ovn
  cidrBlock: 10.10.204.0/24
  gateway: 10.10.204.254
  vlan: vlan0
  excludeIps:
  - 10.10.204.1..10.10.204.100

---
apiVersion: v1
kind: Pod
metadata:
  name: nginx-underlay
  annotations:
    k8s.v1.cni.cncf.io/networks: '[{
      "name": "sriov",
      "namespace": "default",
      "default-route": ["10.10.204.254"]
    }]'
    sriov.default.ovn.kubernetes.io/logical_switch: vlan0
spec:
  containers:
  - name: nginx-underlay
    image: docker.io/library/nginx:alpine
    resources:
      requests:
        mellanox.com/cx5_sriov_switchdev: '1'
      limits:
        mellanox.com/cx5_sriov_switchdev: '1'
```

- `v1.multus-cni.io/default-network` : is the {namespace}/{name} of `NetworkAttachmentDefinition` in the previous step.

Note: In the above example, multus is used to create a Pod using VF as the secondary network card, and VF is used as the default route of the Pod. You can also use VF as the main network card of the Pod. For more details on multus configuration, see Multiple Network Card Management.

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-underlay-noVF
  annotations:
    ovn.kubernetes.io/logical_switch: vlan0
spec:
  containers:
  - name: nginx-underlay-noVF
    image: docker.io/library/nginx:alpine
```

The above example will create a Pod that does not use VF for network offload acceleration, and its flow table will still be delivered to ovs-kernel but not to e-switch.

7.15.6 offload verification

Running the following command in the `ovs-ovn` container of the Pod run node to observe if offload success.

```
# ovs-appctl dpctl/dump-flows -m type=offloaded
ufid:91cc45de-e7e9-4935-8f82-1890430b0f66, skb_priority(0/0),skb_mark(0/0),ct_state(0/0x23),ct_zone(0/0),ct_mark(0/0),ct_label(0/0x1),recirc_id(0),dp_hash(0/
0),in_port(5b45c61b307e_h),packet_type(ns=0/0,id=0/0),eth(src=00:00:00:c5:6d:4e,dst=00:00:00:e7:16:ce),eth_type(0x0800),ipv4(src=0.0.0.0/0.0.0.0,dst=0.
0.0.0/0.0.0.0,proto=0/0,tos=0/0,ttl=0/0,frag=no), packets:941539, bytes:62142230, used:0.260s, offloaded:yes, dp:tc, actions:54235e5753b8_h
ufid:e00768d7-e652-4d79-8182-3291d852b791, skb_priority(0/0),skb_mark(0/0),ct_state(0/0x23),ct_zone(0/0),ct_mark(0/0),ct_label(0/0x1),recirc_id(0),dp_hash(0/
0),in_port(54235e5753b8_h),packet_type(ns=0/0,id=0/0),eth(src=00:00:00:e7:16:ce,dst=00:00:00:c5:6d:4e),eth_type(0x0800),ipv4(src=0.0.0.0/0.0.0.0,dst=0.
0.0.0/0.0.0.0,proto=0/0,tos=0/0,ttl=0/0,frag=no), packets:82386659, bytes:115944854173, used:0.260s, offloaded:yes, dp:tc, actions:5b45c61b307e_h
```

If there is `offloaded:yes, dp:tc` content, the offloading is successful.

PDF **Slack** **Support**

July 30, 2025

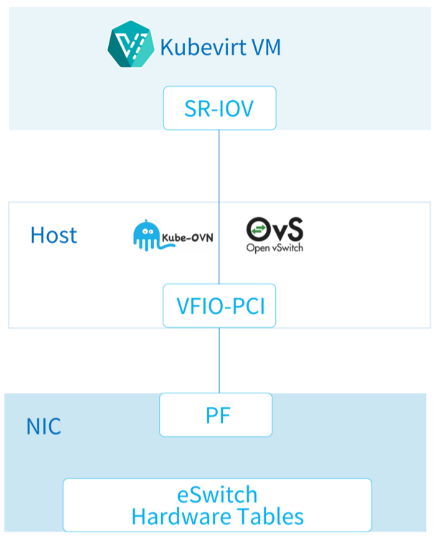
May 24, 2022

GitHub

7.15.7 Comments

7.16 Offload with Corigine

Kube-OVN uses OVS for traffic forwarding in the final data plane, and the associated flow table matching, tunnel encapsulation and other functions are CPU-intensive, which consumes a lot of CPU resources and leads to higher latency and lower throughput under heavy traffic. Corigine Agilio CX series SmartNIC can offload OVS-related operations to the hardware. This technology can shorten the data path without modifying the OVS control plane, avoiding the use of host CPU resources, which dramatically reduce latency and significantly increase the throughput.



Note

The solution described in this article was verified in 2022. However, hardware NICs may now have new features, and some limitations mentioned may have been resolved. Please consult your hardware vendor for the latest technical constraints and capabilities.

7.16.1 Prerequisites

- Corigine Agilio CX series SmartNIC.
- CentOS 8 Stream or Linux 5.7 above.
- Since the current NIC does not support `dp_hash` and `hash` operation offload, OVN LB function should be disabled.

7.16.2 Setup SR-IOV

Please read Agilio Open vSwitch TC User Guide for the detail usage of this SmartNIC.

The following scripts are saved for subsequent execution of firmware-related operations:

```
#!/bin/bash
DEVICE=${1}
DEFAULT_ASSY=scan
ASSY=${2:-${DEFAULT_ASSY}}
APP=${3:-flower}
```

```
if [ "x${DEVICE}" = "x" -o ! -e /sys/class/net/${DEVICE} ]; then
    echo Syntax: ${0} device [ASSY] [APP]
    echo
    echo This script associates the TC Offload firmware
    echo with a Netronome SmartNIC.
    echo
    echo device: is the network device associated with the SmartNIC
    echo ASSY: defaults to ${DEFAULT_ASSY}
    echo APP: defaults to flower. flower-next is supported if updated
    echo      firmware has been installed.
    exit 1
fi

# It is recommended that the assembly be determined by inspection
# The following code determines the value via the debug interface
if [ "${ASSY}x" = "scanx" ]; then
    ethtool -W ${DEVICE} 0
    DEBUG=$(ethtool -w ${DEVICE} data /dev/stdout | strings)
    SERIAL=$(echo "${DEBUG}" | grep "^SN:")
    ASSY=$(echo ${SERIAL} | grep -oE AMDA[0-9]{4})
fi

PCIADDR=$(basename $(readlink -e /sys/class/net/${DEVICE}/device))
FWDIR="/lib/firmware/netronome"

# AMDA0081 and AMDA0097 uses the same firmware
if [ "${ASSY}" = "AMDA0081" ]; then
    if [ ! -e ${FWDIR}/${APP}/nic_AMDA0081.nffw ]; then
       ln -sf nic_AMDA0097.nffw ${FWDIR}/${APP}/nic_AMDA0081.nffw
   fi
fi

FW="${FWDIR}/pci-${PCIADDR}.nffw"
ln -sf "${APP}/nic_${ASSY}.nffw" "${FW}"

# insert distro-specific initramfs section here...
```

Switching firmware options and reloading the driver:

```
./agilio-tc-fw-select.sh ens47np0 scan
rmmod nfp
modprobe nfp
```

Check the number of available VFs and create VFs.

```
# cat /sys/class/net/ens3/device/sriov_totalvfs
65

# echo 4 > /sys/class/net/ens47/device/sriov_numvfs
```

7.16.3 Install SR-IOV Device Plugin

Since each machine has a limited number of VFs and each Pod that uses acceleration will take up VF resources, we need to use the SR-IOV Device Plugin to manage the corresponding resources so that the scheduler knows how to schedule.

Create SR-IOV Configmap:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: sriovdp-config
  namespace: kube-system
data:
  config.json: |
    {
      "resourceList": [{
          "resourcePrefix": "corigine.com",
          "resourceName": "agilio_sriov",
          "selectors": {
                  "vendors": ["19ee"],
                  "devices": ["6003"],
                  "drivers": ["nfp_netvf"]
              }
      }
      ]
    }
```

Please read the SR-IOV device plugin to deploy:

```
kubectl apply -f https://raw.githubusercontent.com/intel/sriov-network-device-plugin/master/deployments/k8s-v1.16/sriovdp-daemonset.yaml
```

Check if SR-IOV resources have been registered to Kubernetes Node:

```
kubectl describe no containerserver  | grep corigine

corigine.com/agilio_sriov:  4
corigine.com/agilio_sriov:  4
corigine.com/agilio_sriov  0           0
```

7.16.4 Install Multus-CNI

The device IDs obtained during SR-IOV Device Plugin scheduling need to be passed to Kube-OVN via Multus-CNI, so Multus-CNI needs to be configured to perform the related tasks.

Please read Multus-CNI Document to deploy:

```
kubectl apply -f https://raw.githubusercontent.com/k8snetworkplumbingwg/multus-cni/master/deployments/multus-daemonset.yml
```

Create `NetworkAttachmentDefinition`:

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: default
  namespace: default
  annotations:
    k8s.v1.cni.cncf.io/resourceName: corigine.com/agilio_sriov
spec:
  config: '{
    "cniVersion": "0.3.1",
    "name": "kube-ovn",
    "plugins":[
        {
            "type":"kube-ovn",
            "server_socket":"/run/openvswitch/kube-ovn-daemon.sock",
            "provider": "default.default.ovn"
        },
        {
            "type":"portmap",
            "capabilities":{
                "portMappings":true
            }
        }
    ]
}'
```

- `provider`: the format should be {name}.{namespace}.ovn of related `NetworkAttachmentDefinition`.

7.16.5 Enable Offload in Kube-OVN

Download the scripts:

```
wget https://raw.githubusercontent.com/kubeovn/kube-ovn/release-1.14/dist/images/install.sh
```

Change the related options, `IFACE` should be the physic NIC and has an IP:

```
ENABLE_MIRROR=${ENABLE_MIRROR:-false}
HW_OFFLOAD=${HW_OFFLOAD:-true}
ENABLE_LB=${ENABLE_LB:-false}
IFACE="ensp01"
```

Install Kube-OVN:

```
bash install.sh
```

7.16.6 Create Pods with VF NICs

Pods that use VF for network offload acceleration can be created using the following yaml:

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  namespace: default
  annotations:
    v1.multus-cni.io/default-network: default/default
spec:
  containers:
    - name: nginx
      image: docker.io/library/nginx:alpine
      resources:
        requests:
          corigine.com/agilio_sriov: '1'
        limits:
          corigine.com/agilio_sriov: '1'
```

- `v1.multus-cni.io/default-network`: should be the {namespace}/{name} of related `NetworkAttachmentDefinition`.

Running the following command in the `ovs-ovn` container of the Pod run node to observe if offload success.

```
# ovs-appctl dpctl/dump-flows -m type=offloaded
ufid:91cc45de-e7e9-4935-8f82-1890430b0f66, skb_priority(0/0),skb_mark(0/0),ct_state(0/0x23),ct_zone(0/0),ct_mark(0/0),ct_label(0/0x1),recirc_id(0),dp_hash(0/
0),in_port(5b45c61b307e_h),packet_type(ns=0/0,id=0/0),eth(src=00:00:00:c5:6d:4e,dst=00:00:00:e7:16:ce),eth_type(0x0800),ipv4(src=0.0.0.0/0.0.0.0,dst=0.
0.0.0/0.0.0.0,proto=0/0,tos=0/0,ttl=0/0,frag=no), packets:941539, bytes:62142230, used:0.260s, offloaded:yes, dp:tc, actions:54235e5753b8_h
ufid:e00768d7-e652-4d79-8182-3291d852b791, skb_priority(0/0),skb_mark(0/0),ct_state(0/0x23),ct_zone(0/0),ct_mark(0/0),ct_label(0/0x1),recirc_id(0),dp_hash(0/
0),in_port(54235e5753b8_h),packet_type(ns=0/0,id=0/0),eth(src=00:00:00:e7:16:ce,dst=00:00:00:c5:6d:4e),eth_type(0x0800),ipv4(src=0.0.0.0/0.0.0.0,dst=0.
0.0.0/0.0.0.0,proto=0/0,tos=0/0,ttl=0/0,frag=no), packets:82386659, bytes:115944854173, used:0.260s, offloaded:yes, dp:tc, actions:5b45c61b307e_h
```

If there is `offloaded:yes, dp:tc` content, the offloading is successful.

PDF **Slack** **Support**

July 30, 2025

May 24, 2022

GitHub

7.16.7 Comments

7.17 Hardware Offload for Yunsilicon

The OVS software based solution is CPU intensive, affecting system performance and preventing full utilization of the available bandwidth.

Yunsilicon metaScale SmartNICs provide a drop-in accelerator for OVS which can support very high flow and policy capacities without degradation in performance. By taking use of SR-IOV technology we can achieve low network latency and high throughput.

Note

1. The solution described in this article was verified in 2024. However, hardware NICs may now have new features, and some limitations mentioned may have been resolved. Please consult your hardware vendor for the latest technical constraints and capabilities.
2. Currently, Yunsilicon only supports the v1.11 series version of Kube-OVN, and some of the latest features cannot be used.

7.17.1 Prerequisites

- MCR Allinone Packages
- Yunsilicon metaScale family NICs
- Enable SR-IOV and VT-d in BIOS

7.17.2 Installation Guide

Install Kube-OVN with hw-offload mode enabled

1. Download the install script:

```
wget https://github.com/yunsilicon/kube-ovn/blob/release-1.11/dist/images/install.sh
```

1. Configure node

Edit the configuration file named `ovs-dpdk-config` on the node that needs to run ovs-dpdk. The configuration file needs to be placed in the `/opt/ovs-config` directory.

```
# specify log level for ovs dpdk, the value is info or dbg, default is info
VLOG=info
# specify nic offload, the value is true or false, default is true
HW_OFFLOAD=true
# specify cpu mask for ovs dpdk, not specified by default
CPU_MASK=0x02
# specify socket memory, not specified by default
SOCKET_MEM="2048,2048"
# specify encap IP
ENCAP_IP=6.6.6.208/24
# specify pci device
DPDK_DEV=0000:b3:00.0
# specify mtu, default is 1500
PF_MTU=1500
# specify bond name if bond enabled, not specified by default
BR_PHY_BOND_NAME=bond0
```

1. Install Kube-OVN

`NOTICE`: We need to manually modify the openvswitch image in the script, please contact the technical support of yunsilicon to obtain the supporting version.

```
bash install.sh
```

Setting Up SR-IOV

1. Find the device id of metaScale device, below is `b3:00.0`

```
[root@k8s-master ~]# lspci -d 1f67:
b3:00.0 Ethernet controller: Device 1f67:1111 (rev 02)
b3:00.1 Ethernet controller: Device 1f67:1111 (rev 02)
```

1. Find the related interface with device id, below is `p3p1`

```
ls -l /sys/class/net/ | grep b3:00.0
lrwxrwxrwx 1 root root 0 May  7 16:30 p3p1 -> ../../devices/pci0000:b2/0000:b2:00.0/0000:b3:00.0/net/p3p1
```

1. Check available VF number

```
cat /sys/class/net/p3p1/device/sriov_totalvfs
512
```

1. Create VFs

```
echo '10' > /sys/class/net/p3p1/device/sriov_numvfs
```

1. Find the device ids of VFs created above

```
lspci -d 1f67:
b3:00.0 Ethernet controller: Device 1f67:1111 (rev 02)
b3:00.1 Ethernet controller: Device 1f67:1111 (rev 02)
b3:00.2 Ethernet controller: Device 1f67:1112
b3:00.3 Ethernet controller: Device 1f67:1112
b3:00.4 Ethernet controller: Device 1f67:1112
b3:00.5 Ethernet controller: Device 1f67:1112
b3:00.6 Ethernet controller: Device 1f67:1112
b3:00.7 Ethernet controller: Device 1f67:1112
b3:01.0 Ethernet controller: Device 1f67:1112
b3:01.1 Ethernet controller: Device 1f67:1112
b3:01.2 Ethernet controller: Device 1f67:1112
b3:01.3 Ethernet controller: Device 1f67:1112
```

1. Enable switchdev mode by device id of PF

```
devlink dev eswitch set pci/0000:b3:00.0 mode switchdev
```

1. Disable NetworkManager if it's running

```
systemctl stop NetworkManager
systemctl disable NetworkManager
```

Install SR-IOV Device Plugin

1. Create a ConfigMap that defines SR-IOV resource pool configuration

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: sriovdp-config
  namespace: kube-system
data:
  config.json: |
    {
        "resourceList": [{
                "resourceName": "xsc_sriov",
                "resourcePrefix": "yunsilicon.com",
                "selectors": {
                    "vendors": ["1f67"],
                    "devices": ["1012", "1112"]
                }}
        ]
    }
```

1. Follow SR-IOV Device Plugin to deploy device plugin.
2. Check if SR-IOV devices have been discovered by device plugin

```
# kubectl describe node <node name> | grep yunsilicon.com/xsc_sriov
  yunsilicon.com/xsc_sriov:  10
  yunsilicon.com/xsc_sriov:  10
  yunsilicon.com/xsc_sriov  0              0
```

Install Multus-CNI

1. Follow Multus-CNI to deploy Multus-CNI

```
kubectl apply -f https://raw.githubusercontent.com/k8snetworkplumbingwg/multus-cni/master/deployments/multus-daemonset.yml
```

1. Create a NetworkAttachmentDefinition

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: sriov-net1
  namespace: default
  annotations:
    k8s.v1.cni.cncf.io/resourceName: yunsilicon.com/xsc_sriov
spec:
  config: '{
    "cniVersion": "0.3.1",
    "name": "kube-ovn",
    "plugins":[
        {
            "type":"kube-ovn",
            "server_socket":"/run/openvswitch/kube-ovn-daemon.sock",
            "provider": "sriov-net1.default.ovn"
        },
        {
            "type":"portmap",
            "capabilities":{
                "portMappings":true
            }
        }
    ]
}'
```

Create Pod with SR-IOV

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  annotations:
    v1.multus-cni.io/default-network: default/sriov-net1
spec:
  containers:
    - name: nginx
      image: nginx:alpine
      resources:
        requests:
          yunsilicon.com/xsc_sriov: '1'
        limits:
          yunsilicon.com/xsc_sriov: '1'
```

Verify If Offload Works

```
ovs-appctl dpctl/dump-flows type=offloaded
flow-dump from pmd on cpu core: 9
ct_state(-new+est-rel+rpl+trk),ct_mark(0/0x3),recirc_id(0x2d277),in_port(15),packet_type(ns=0,id=0),eth(src=00:00:00:9d:fb:1a,dst=00:
00:00:ce:cf:b9),eth_type(0x0800),ipv4(dst=10.16.0.14,frag=no), packets:6, bytes:588, used:7.276s, actions:ct(zone=4,nat),recirc(0x2d278)
ct_state(-new+est-rel-rpl+trk),ct_mark(0/0x3),recirc_id(0x2d275),in_port(8),packet_type(ns=0,id=0),eth(src=00:00:00:ce:cf:b9,dst=00:00:00:9d:fb:
1a),eth_type(0x0800),ipv4(dst=10.16.0.18,frag=no), packets:5, bytes:490, used:7.434s, actions:ct(zone=6,nat),recirc(0x2d276)
ct_state(-new+est-rel-rpl+trk),ct_mark(0/0x1),recirc_id(0x2d276),in_port(8),packet_type(ns=0,id=0),eth(src=00:00:00:ce:cf:b9,dst=00:00:00:9d:fb:1a/
01:00:00:00:00:00),eth_type(0x0800),ipv4(frag=no), packets:5, bytes:490, used:7.434s, actions:15
recirc_id(0),in_port(15),packet_type(ns=0,id=0),eth(src=00:00:00:9d:fb:1a/01:00:00:00:00:00,dst=00:00:00:ce:cf:b9),eth_type(0x0800),ipv4(dst=10.
16.0.14/255.192.0.0,frag=no), packets:6, bytes:588, used:7.277s, actions:ct(zone=6,nat),recirc(0x2d277)
recirc_id(0),in_port(8),packet_type(ns=0,id=0),eth(src=00:00:00:ce:cf:b9/01:00:00:00:00:00,dst=00:00:00:9d:fb:1a),eth_type(0x0800),ipv4(dst=10.
16.0.18/255.192.0.0,frag=no), packets:6, bytes:588, used:7.434s, actions:ct(zone=4,nat),recirc(0x2d275)
ct_state(-new+est-rel+rpl+trk),ct_mark(0/0x1),recirc_id(0x2d278),in_port(15),packet_type(ns=0,id=0),eth(dst=00:
00:00:ce:cf:b9/01:00:00:00:00:00),eth_type(0x0800),ipv4(frag=no), packets:6, bytes:588, used:7.277s, actions:8
```

You can find some flows if all works well.

PDF Slack Support

July 21, 2025

May 29, 2024

GitHub

7.17.3 Comments

7.18 Offload with YUSUR

Kube-OVN uses OVS for traffic forwarding in the final data plane, and the associated flow table matching, tunnel encapsulation and other functions are CPU-intensive, which consumes a lot of CPU resources and leads to higher latency and lower throughput under heavy traffic.YUSUR CONFLUX-22OOE series SmartNIC can offload OVS-related operations to the hardware. This technology can shorten the data path without modifying the OVS control plane, avoiding the use of host CPU resources, which dramatically reduce latency and significantly increase the throughput.

Note

The solution described in this article was verified in 2024. However, hardware NICs may now have new features, and some limitations mentioned may have been resolved. Please consult your hardware vendor for the latest technical constraints and capabilities.

7.18.1 Prerequisites

- YUSUR CONFLUX-22OOE series SmartNIC.
- ensure hados(Heterogeneous Agile Developing & Operating System) installed.
- Enable SR-IOV in BIOS.

7.18.2 Installation Guide

Setting Up SR-IOV

1. Based on the vendor ID (1f47) of the YUSUR CONFLUX-22OOE series SmartNIC, identify the device IDs of the network card on the host, such as (00:0a.0) and (00:0b.0), which correspond to the two physical ports on the 2200E. You can select one according to the fiber connection status.

```
lspci | grep 1f47
00:0a.0 Ethernet controller: Device 1f47:1001 (rev 10)
00:0b.0 Ethernet controller: Device 1f47:1001 (rev 10)
```

1. Check available VF number:

```
cat /sys/bus/pci/devices/0000\:00\:0a.0/sriov_totalvfs
256
```

1. Create VFs and do not exceeding the number found above:

```
echo 7 > /sys/bus/pci/devices/0000\:00\:0a.0/sriov_numvfs
```

1. Find the device IDs corresponding to the above VFs:

```
lspci | grep 1f47
00:0a.0 Ethernet controller: Device 1f47:1001 (rev 10)
00:0a.1 Ethernet controller: Device 1f47:110f (rev 10)
00:0a.2 Ethernet controller: Device 1f47:110f (rev 10)
00:0a.3 Ethernet controller: Device 1f47:110f (rev 10)
00:0a.4 Ethernet controller: Device 1f47:110f (rev 10)
00:0a.5 Ethernet controller: Device 1f47:110f (rev 10)
00:0a.6 Ethernet controller: Device 1f47:110f (rev 10)
00:0a.7 Ethernet controller: Device 1f47:110f (rev 10)
00:0b.0 Ethernet controller: Device 1f47:1001 (rev 10)
```

Configure and install SR-IOV Device Plugin

1. Create an SR-IOV related ConfigMap to facilitate the SR-IOV Device Plugin installation, enabling it to locate VF resources on nodes based on this configuration and provide them for Pod usage:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: sriovdp-config
  namespace: kube-system
data:
  config.json: |
    {
        "resourceList": [{
                "resourceName": "sriov_dpu",
                "resourcePrefix": "yusur.tech",
                "selectors": {
                    "vendors": ["1f47"],
                    "devices": ["110f"]
                }}
        ]
    }
```

1. Install and run the SR-IOV Device Plugin.

```
kubectl apply -f https://raw.githubusercontent.com/k8snetworkplumbingwg/sriov-network-device-plugin/v3.6.2/deployments/sriovdp-daemonset.yaml
```

1. Check if SR-IOV resources have been registered to Kubernetes Node:

```
kubectl describe node node1 | grep yusur
  yusur.tech/sriov_dpu:  7
  yusur.tech/sriov_dpu:  7
  yusur.tech/sriov_dpu  0            0
```

7.18.3 Install Multus-CNI

The device IDs obtained during SR-IOV Device Plugin scheduling need to be passed to Kube-OVN via Multus-CNI, so Multus-CNI needs to be configured to perform the related tasks.

```
kubectl apply -f https://raw.githubusercontent.com/k8snetworkplumbingwg/multus-cni/v4.0.2/deployments/multus-daemonset-thick.yml
```

Create `NetworkAttachmentDefinition`:

```
apiVersion:
  "k8s.cni.cncf.io/v1"
kind:
  NetworkAttachmentDefinition
metadata:
  name: test
  namespace: kube-system
  annotations:
    k8s.v1.cni.cncf.io/resourceName: yusur.tech/sriov_dpu
spec:
  config: '{
    "cniVersion": "0.3.1",
    "name": "kube-ovn",
    "plugins":[
        {
            "type":"kube-ovn",
            "server_socket":"/run/openvswitch/kube-ovn-daemon.sock",
            "provider": "test.kube-system.ovn"
        },
        {
            "type":"portmap",
            "capabilities":{
                "portMappings":true
            }
        }
    ]
}
```

- `provider`: the format should be `{name}.{namespace}.ovn` of related NetworkAttachmentDefinition.

7.18.4 Enable Offload in Kube-OVN

1. Download the scripts:

```
wget https://github.com/kubeovn/kube-ovn/blob/release-1.12/dist/images/install.sh
```

Change the related options, `IFACE` should be the physic NIC and has an IP:

```
ENABLE_MIRROR=${ENABLE_MIRROR:-false}
HW_OFFLOAD=${HW_OFFLOAD:-true}
ENABLE_LB=${ENABLE_LB:-false}
IFACE="p0"
```

1. Install Kube-OVN:

```
bash install.sh
```

Create Pods with VF NICsCreate Pods with VF NICs

Pods that use VF for network offload acceleration can be created using the following yaml:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  namespace: default
  annotations:
    v1.multus-cni.io/default-network: kube-system/test
spec:
  containers:
    - name: nginx
      image: docker.io/library/nginx:alpine
      resources:
        requests:
          yusur.tech/sriov_dpu: '1'
        limits:
          yusur.tech/sriov_dpu: '1'
```

- `v1.multus-cni.io/default-network` : should be the `{namespace}/{name}` of related NetworkAttachmentDefinition.

Offload verification

Running the following command in the `ovs-ovn` container of the Pod run node to observe if offload success.

```
# ovs-appctl dpctl/dump-flows -m type=offloaded
ufid:67c2e10f-92d4-4574-be70-d072815ff166, skb_priority(0/0),skb_mark(0/0),ct_state(0/0x23),ct_zone(0/0),ct_mark(0/0),ct_label(0/0),recirc_id(0),dp_hash(0/
0),in_port(d85b161b6840_h),packet_type(ns=0/0,id=0/0),eth(src=0a:c9:1c:70:01:09,dst=8a:18:a4:22:b7:7d),eth_type(0x0800),ipv4(src=10.0.1.10,dst=10.
0.1.6,proto=6,tos=0/0x3,ttl=0/0,frag=no),tcp(src=60774,dst=9001), packets:75021, bytes:109521630, offload_packets:75019, offload_bytes:109521498, used:
3.990s,offloaded:yes,dp:tc, actions:set(tunnel(tun_id=0x5,dst=192.168.201.12,ttl=64,tp_dst=6081,geneve({class=0x102,type=0x80,len=4,0xa0006}),flags(csum|
key))),genev_sys_6081
ufid:7940666e-a0bd-42a5-8116-1e84e81bb338, skb_priority(0/0),tunnel(tun_id=0x5,src=192.168.201.12,dst=192.168.201.11,ttl=0/
0,tp_dst=6081,geneve({class=0x102,type=0x80,len=4,0x6000a}),flags(+key)),skb_mark(0/0),ct_state(0/0),ct_zone(0/0),ct_mark(0/0),ct_label(0/
0),recirc_id(0),dp_hash(0/0),in_port(genev_sys_6081),packet_type(ns=0/0,id=0/0),eth(src=8a:18:a4:22:b7:7d,dst=0a:c9:1c:70:01:09),eth_type(0x0800),ipv4(src=10.
0.1.6,dst=10.0.1.10,proto=6,tos=0/0,ttl=0/0,frag=no),tcp(src=9001,dst=60774), packets:6946, bytes:459664, offload_packets:6944, offload_bytes:459532, used:
4.170s, dp:tc,offloaded:yes,actions:d85b161b6840_h
```

PDF Slack Support

July 30, 2025

August 13, 2024

GitHub

7.18.5 Comments

7.19 DPDK Support

This document describes how Kube-OVN combines with OVS-DPDK to provide a DPDK-type network interface to KubeVirt's virtual machines.

Upstream KubeVirt does not currently support OVS-DPDK, users need to use the downstream patch Vhostuser implementation to build KubeVirt by themselves or KVM Device Plugin to use OVS-DPDK.

7.19.1 Prerequisites

- The node needs to provide a dedicated NIC for the DPDK driver to run.
- The node needs to have Hugepages enabled.

7.19.2 Set DPDK driver

Here we use `driverctl` for example, please refer to the DPDK documentation for specific parameters and other driver usage:

```
driverctl set-override 0000:00:0b.0 uio_pci_generic
```

7.19.3 Configure Nodes

Labeling OVS-DPDK-enabled nodes for Kube-OVN to recognize:

```
kubectl label nodes <node> ovn.kubernetes.io/ovs_dp_type="userspace"
```

Create the configuration file `ovs-dpdk-config` in the `/opt/ovs-config` directory on nodes that support DPDK.

```
ENCAP_IP=192.168.122.193/24
DPDK_DEV=0000:00:0b.0
```

- `ENCAP_IP` : The tunnel endpoint address.
- `DPDK_DEV` : The PCI ID of the device.

7.19.4 Install Kube-OVN

Download scripts:

```
wget https://raw.githubusercontent.com/kubeovn/kube-ovn/release-1.14/dist/images/install.sh
```

Enable the DPDK installation option:

```
bash install.sh --with-hybrid-dpdk
```

7.19.5 Usage

Here we verify the OVS-DPDK functionality by creating a virtual machine with a vhostuser type NIC.

Here we use the KVM Device Plugin to create virtual machines. For more information on how to use it, please refer to [KVM Device Plugin].(https://github.com/kubevirt/kubernetes-device-plugins/blob/master/docs/README.kvm.md).

```
kubectl apply -f https://raw.githubusercontent.com/kubevirt/kubernetes-device-plugins/master/manifests/kvm-ds.yml
```

Create NetworkAttachmentDefinition:

```
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  name: ovn-dpdk
  namespace: default
```

```
spec:
  config: >-
    {
        "cniVersion": "0.3.0",
        "type": "kube-ovn",
        "server_socket": "/run/openvswitch/kube-ovn-daemon.sock",
        "provider": "ovn-dpdk.default.ovn",
        "vhost_user_socket_volume_name": "vhostuser-sockets",
        "vhost_user_socket_name": "sock"
    }
```

Create a VM image using the following Dockerfile:

```
FROM quay.io/kubevirt/virt-launcher:v0.46.1

# wget http://cloud.centos.org/centos/7/images/CentOS-7-x86_64-GenericCloud.qcow2
COPY CentOS-7-x86_64-GenericCloud.qcow2 /var/lib/libvirt/images/CentOS-7-x86_64-GenericCloud.qcow2
```

Create a virtual machine:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: vm-config
data:
  start.sh: |
    chmod u+w /etc/libvirt/qemu.conf
    echo "hugetlbfs_mount = \"/dev/hugepages\"" >> /etc/libvirt/qemu.conf
    virtlogd &
    libvirtd &

    mkdir /var/lock

    sleep 5

    virsh define /root/vm/vm.xml
    virsh start vm

    tail -f /dev/null
  vm.xml: |
    <domain type='kvm'>
      <name>vm</name>
      <uuid>4a9b3f53-fa2a-47f3-a757-dd87720d9d1d</uuid>
      <memory unit='KiB'>2097152</memory>
      <currentMemory unit='KiB'>2097152</currentMemory>
      <memoryBacking>
        <hugepages>
          <page size='2' unit='M' nodeset='0'/>
        </hugepages>
      </memoryBacking>
      <vcpu placement='static'>2</vcpu>
      <cputune>
        <shares>4096</shares>
        <vcpupin vcpu='0' cpuset='4'/>
        <vcpupin vcpu='1' cpuset='5'/>
        <emulatorpin cpuset='1,3'/>
      </cputune>
      <os>
        <type arch='x86_64' machine='pc'>hvm</type>
        <boot dev='hd'/>
      </os>
      <features>
        <acpi/>
        <apic/>
      </features>
      <cpu mode='host-model'>
        <model fallback='allow'/>
        <topology sockets='1' cores='2' threads='1'/>
        <numa>
          <cell id='0' cpus='0-1' memory='2097152' unit='KiB' memAccess='shared'/>
        </numa>
      </cpu>
      <on_reboot>restart</on_reboot>
      <devices>
        <emulator>/usr/libexec/qemu-kvm</emulator>
        <disk type='file' device='disk'>
          <driver name='qemu' type='qcow2' cache='none'/>
          <source file='/var/lib/libvirt/images/CentOS-7-x86_64-GenericCloud.qcow2'/>
          <target dev='vda' bus='virtio'/>
        </disk>

        <interface type='vhostuser'>
          <mac address='00:00:00:0A:30:89'/>
          <source type='unix' path='/var/run/vm/sock' mode='server'/>
           <model type='virtio'/>
          <driver queues='2'>
            <host mrg_rxbuf='off'/>
          </driver>
        </interface>
        <serial type='pty'>
          <target type='isa-serial' port='0'>
```

```
            <model name='isa-serial'/>
          </target>
        </serial>
        <console type='pty'>
          <target type='serial' port='0'/>
        </console>
        <channel type='unix'>
          <source mode='bind' path='/var/lib/libvirt/qemu/channel/target/domain-1-vm/org.qemu.guest_agent.0'/>
          <target type='virtio' name='org.qemu.guest_agent.0' state='connected'/>
          <alias name='channel0'/>
          <address type='virtio-serial' controller='0' bus='0' port='1'/>
        </channel>

      </devices>
    </domain>
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: vm-deployment
  labels:
    app: vm
spec:
  replicas: 1
  selector:
    matchLabels:
      app: vm
  template:
    metadata:
      labels:
        app: vm
      annotations:
        k8s.v1.cni.cncf.io/networks: default/ovn-dpdk
        ovn-dpdk.default.ovn.kubernetes.io/ip_address: 10.16.0.96
        ovn-dpdk.default.ovn.kubernetes.io/mac_address: 00:00:00:0A:30:89
    spec:
      nodeSelector:
        ovn.kubernetes.io/ovs_dp_type: userspace
      securityContext:
        runAsUser: 0
      volumes:
        - name: vhostuser-sockets
          emptyDir: {}
        - name: xml
          configMap:
            name: vm-config
        - name: hugepage
          emptyDir:
            medium: HugePages-2Mi
        - name: libvirt-runtime
          emptyDir: {}
      containers:
        - name: vm
          image: vm-vhostuser:latest
          command: ["bash", "/root/vm/start.sh"]
          securityContext:
            capabilities:
              add:
                - NET_BIND_SERVICE
                - SYS_NICE
                - NET_RAW
                - NET_ADMIN
            privileged: false
            runAsUser: 0
          resources:
            limits:
              cpu: '2'
              devices.kubevirt.io/kvm: '1'
              memory: '8784969729'
              hugepages-2Mi: 2Gi
            requests:
              cpu: 666m
              devices.kubevirt.io/kvm: '1'
              ephemeral-storage: 50M
              memory: '4490002433'
          volumeMounts:
            - name: vhostuser-sockets
              mountPath: /var/run/vm
            - name: xml
              mountPath: /root/vm/
            - mountPath: /dev/hugepages
              name: hugepage
            - name: libvirt-runtime
              mountPath: /var/run/libvirt
```

Wait for the virtual machine to be created successfully and then go to the Pod to configure the virtual machine:

```
# virsh set-user-password vm root 12345
Password set successfully for root in vm

# virsh console vm
Connected to domain 'vm'
```

```
Escape character is ^] (Ctrl + ])

CentOS Linux 7 (Core)
Kernel 3.10.0-1127.el7.x86_64 on an x86_64

localhost login: root
Password:
Last login: Fri Feb 25 09:52:54 on ttyS0
```

Next, you can log into the virtual machine for network configuration and test:

```
ip link set eth0 mtu 1400
ip addr add 10.16.0.96/16 dev eth0
ip ro add default via 10.16.0.1
ping 114.114.114.114
```

PDF **Slack** **Support**

February 15, 2023

May 24, 2022

GitHub

7.19.6 Comments

7.20 Integration with OpenStack

In some cases, users need to run virtual machines with OpenStack and containers with Kubernetes, and need the network to interoperate between containers and virtual machines and be under a unified control plane. If the OpenStack Neutron side also uses OVN as the underlying network, then Kube-OVN can use either cluster interconnection or shared underlying OVN to connect the OpenStack and Kubernetes networks.

7.20.1 Cluster Interconnection

This pattern is similar to Cluster Inter-Connection with OVN-IC to connect two Kubernetes cluster networks, except that the two ends of the cluster are replaced with OpenStack and Kubernetes.

Prerequisites

1. The subnet CIDRs within OpenStack and Kubernetes cannot overlap with each other in auto-route mode.
2. A set of machines needs to exist that can be accessed by each cluster over the network and used to deploy controllers that interconnect across clusters.
3. Each cluster needs to have a set of machines that can access each other across clusters via IP as the gateway nodes.
4. This solution only connects to the Kubernetes default subnet with selected VPC in OpenStack.

Deploy OVN-IC DB

Start the `OVN-IC` DB with the following command:

```
docker run --name=ovn-ic-db -d --network=host -v /etc/ovn/:/etc/ovn -v /var/run/ovn:/var/run/ovn -v /var/log/ovn:/var/log/ovn kubeovn/kube-ovn:v1.14.4 bash
start-ic-db.sh
```

Kubernetes Side Operations

Create `ovn-ic-config` ConfigMap in `kube-system` Namespace:

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: ovn-ic-config
  namespace: kube-system
data:
  enable-ic: "true"
  az-name: "az1"
  ic-db-host: "192.168.65.3"
  ic-nb-port: "6645"
  ic-sb-port: "6646"
  gw-nodes: "az1-gw"
  auto-route: "true"
```

- `enable-ic` : Whether to enable cluster interconnection.
- `az-name` : Distinguish the cluster names of different clusters, each interconnected cluster needs to be different.
- `ic-db-host` : Address of the node where the `OVN-IC` DB is deployed.
- `ic-nb-port` : `OVN-IC` Northbound Database port, default 6645.
- `ic-sb-port` : `OVN-IC` Southbound Database port, default 6645.
- `gw-nodes` : The name of the nodes in the cluster interconnection that takes on the work of the gateways, separated by commas.
- `auto-route` : Whether to automatically publish and learn routes.

OpenStack Side Operations

Create logical routers that interconnect with Kubernetes:

```
# openstack router create router0
# openstack router list
+--------------------------------------+---------+--------+-------+----------------------------------+
| ID                                   | Name    | Status | State | Project                          |
+--------------------------------------+---------+--------+-------+----------------------------------+
| d5b38655-249a-4192-8046-71aa4d2b4af1 | router0 | ACTIVE | UP    | 98a29ab7388347e7b5ff8bdd181ba4f9 |
+--------------------------------------+---------+--------+-------+----------------------------------+
```

Set the availability zone name in the OVN northbound database within OpenStack, which needs to be different from the other interconnected clusters:

```
ovn-nbctl set NB_Global . name=op-az
```

Start the `OVN-IC` controller at a node that has access to the `OVN-IC` DB:

```
/usr/share/ovn/scripts/ovn-ctl --ovn-ic-nb-db=tcp:192.168.65.3:6645 \
  --ovn-ic-sb-db=tcp:192.168.65.3:6646 \
  --ovn-northd-nb-db=unix:/run/ovn/ovnnb_db.sock \
  --ovn-northd-sb-db=unix:/run/ovn/ovnsb_db.sock \
  start_ic
```

- `ovn-ic-nb-db`, `ovn-ic-sb-db`: OVN-IC Northbound database and southbound database addresses.
- `ovn-northd-nb-db`, `ovn-northd-sb-db`: Current cluster OVN northbound database and southbound data address.

Configuration gateway nodes:

```
ovs-vsctl set open_vswitch . external_ids:ovn-is-interconn=true
```

The next step is to create a logical topology by operating the OVN in OpenStack.

Connect the `ts` interconnect switch and the `router0` logical router, and set the relevant rules:

```
ovn-nbctl lrp-add router0 lrp-router0-ts 00:02:ef:11:39:4f 169.254.100.73/24
ovn-nbctl lsp-add ts lsp-ts-router0 -- lsp-set-addresses lsp-ts-router0 router \
  -- lsp-set-type lsp-ts-router0 router \
  -- lsp-set-options lsp-ts-router0  router-port=lrp-router0-ts
ovn-nbctl lrp-set-gateway-chassis lrp-router0-ts {gateway chassis} 1000
ovn-nbctl set NB_Global . options:ic-route-adv=true options:ic-route-learn=true
```

Verify that OpenStack has learned the Kubernetes routing rules:

```
# ovn-nbctl lr-route-list router0
IPv4 Routes
                10.0.0.22            169.254.100.34 dst-ip (learned)
             10.16.0.0/16            169.254.100.34 dst-ip (learned)
```

Next, you can create a virtual machine under the `router0` network to verify that it can interconnect with Pods under Kubernetes.

7.20.2 Shared Underlay OVN

In this scenario, OpenStack and Kubernetes share the same OVN, so concepts such as VPC and Subnet can be pulled together for better control and interconnection.

In this mode we deploy the OVN normally using Kube-OVN, and OpenStack modifies the Neutron configuration to connect to the same OVN DB. OpenStack requires networking-ovn as a Neutron backend implementation.

Neutron Modification

Modify the Neutron configuration file `/etc/neutron/plugins/ml2/ml2_conf.ini`:

```
[ovn]
...
ovn_nb_connection = tcp:[192.168.137.176]:6641,tcp:[192.168.137.177]:6641,tcp:[192.168.137.178]:6641
ovn_sb_connection = tcp:[192.168.137.176]:6642,tcp:[192.168.137.177]:6642,tcp:[192.168.137.178]:6642
ovn_l3_scheduler = OVN_L3_SCHEDULER
```

- `ovn_nb_connection`, `ovn_sb_connection`: The address needs to be changed to the address of the `ovn-central` nodes deployed by Kube-OVN.

Modify the OVS configuration for each node:

```
ovs-vsctl set open . external-ids:ovn-remote=tcp:[192.168.137.176]:6642,tcp:[192.168.137.177]:6642,tcp:[192.168.137.178]:6642
ovs-vsctl set open . external-ids:ovn-encap-type=geneve
ovs-vsctl set open . external-ids:ovn-encap-ip=192.168.137.200
```

- `external-ids:ovn-remote` : The address needs to be changed to the address of the `ovn-central` nodes deployed by Kube-OVN.
- `ovn-encap-ip` : Change to the IP address of the current node.

Using OpenStack Internal Resources in Kubernetes

The next section describes how to query OpenStack's network resources in Kubernetes and create Pods in the subnet from OpenStack.

Query the existing network resources in OpenStack for the following resources that have been pre-created.

```
# openstack router list
+--------------------------------------+---------+--------+-------+----------------------------------+
| ID                                   | Name    | Status | State | Project                          |
+--------------------------------------+---------+--------+-------+----------------------------------+
| 22040ed5-0598-4f77-bffd-e7fd4db47e93 | router0 | ACTIVE | UP    | 62381a21d569404aa236a5dd8712449c |
+--------------------------------------+---------+--------+-------+----------------------------------+
# openstack network list
+--------------------------------------+----------+--------------------------------------+
| ID                                   | Name     | Subnets                              |
+--------------------------------------+----------+--------------------------------------+
| cd59e36a-37db-4c27-b709-d35379a7920f | provider | 01d73d9f-fdaa-426c-9b60-aa34abbfacae |
+--------------------------------------+----------+--------------------------------------+
# openstack subnet list
+--------------------------------------+-------------+--------------------------------------+----------------+
| ID                                   | Name        | Network                              | Subnet         |
+--------------------------------------+-------------+--------------------------------------+----------------+
| 01d73d9f-fdaa-426c-9b60-aa34abbfacae | provider-v4 | cd59e36a-37db-4c27-b709-d35379a7920f | 192.168.1.0/24 |
+--------------------------------------+-------------+--------------------------------------+----------------+
# openstack server list
+--------------------------------------+-------------------+--------+-----------------------+--------+--------+
| ID                                   | Name              | Status | Networks              | Image  | Flavor |
+--------------------------------------+-------------------+--------+-----------------------+--------+--------+
| 8433d622-a8d6-41a7-8b31-49abfd64f639 | provider-instance | ACTIVE | provider=192.168.1.61 | ubuntu | m1     |
+--------------------------------------+-------------------+--------+-----------------------+--------+--------+
```

On the Kubernetes side, query the VPC resources from OpenStack:

```
# kubectl get vpc
NAME                                           STANDBY   SUBNETS
neutron-22040ed5-0598-4f77-bffd-e7fd4db47e93   true      ["neutron-cd59e36a-37db-4c27-b709-d35379a7920f"]
ovn-cluster                                    true      ["join","ovn-default"]
```

`neutron-22040ed5-0598-4f77-bffd-e7fd4db47e93` is the VPC resources synchronized from OpenStack.

Next, you can create Pods and run them according to Kube-OVN's native VPC and Subnet operations.

Bind VPC, Subnet to Namespace `net2` and create Pod:

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: net2
---
apiVersion: kubeovn.io/v1
kind: Vpc
metadata:
  creationTimestamp: "2021-06-20T13:34:11Z"
  generation: 2
  labels:
    ovn.kubernetes.io/vpc_external: "true"
  name: neutron-22040ed5-0598-4f77-bffd-e7fd4db47e93
  resourceVersion: "583728"
  uid: 18d4c654-f511-4def-a3a0-a6434d237c1e
spec:
  namespaces:
  - net2
---
kind: Subnet
apiVersion: kubeovn.io/v1
metadata:
  name: net2
spec:
  vpc: neutron-22040ed5-0598-4f77-bffd-e7fd4db47e93
  namespaces:
    - net2
  cidrBlock: 12.0.1.0/24
  natOutgoing: false
---
apiVersion: v1
kind: Pod
metadata:
  name: ubuntu
  namespace: net2
spec:
  containers:
    - image: docker.io/kubeovn/kube-ovn:v1.8.0
      command:
        - "sleep"
        - "604800"
      imagePullPolicy: IfNotPresent
      name: ubuntu
  restartPolicy: Always
```

PDF Slack Support

July 30, 2025

May 24, 2022

GitHub

7.20.3 Comments

7.21 Use IPsec to encrypt communication between nodes

This function is supported from v1.13.0 onwards, and the host UDP 500 and 4500 ports need to be available.

7.21.1 Encryption process

kube-ovn-cni is responsible for applying for certificates and will create a certificate signing request to kube-ovn-controller. kube-ovn-controller will automatically approve the certificate application, and then kube-ovn-cni will generate an ipsec configuration file based on the certificate and finally start the ipsec process.

7.21.2 Configure IPsec

Change the args `--enable-ovn-ipsec=false` in kube-ovn-controller and kube-ovn-cni to `--enable-ovn-ipsec=true`.

PDF **Slack** **Support**

August 12, 2024

April 18, 2023

GitHub

7.21.3 Comments

7.22 OVN Remote Port Mirroring

This feature provides ability to mirror the traffic of the specified Pod and direction, and to send the mirrored traffic to a remote destination.

This feature requires Kube-OVN version not lower than v1.12.

7.22.1 Install Multus-CNI

Install Multus-CNI by referring the Multus-CNI Document.

7.22.2 Create NetworkAttachmentDefinition

Create the following NetworkAttachmentDefinition:

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: attachnet
  namespace: default
spec:
  config: |
    {
      "cniVersion": "0.3.1",
      "type": "kube-ovn",
      "server_socket": "/run/openvswitch/kube-ovn-daemon.sock",
      "provider": "attachnet.default.ovn"
    }
```

Format of the `provider` field is `<NAME>.<NAMESPACE>.ovn`.

7.22.3 Create Underlay Network

The mirrored traffic is encapsulated before transmition, so MTU of the network used to transmit the traffic should be greater than the mirrored LSP/Pod. Here we are using an underlay network.

Create the following underlay network:

```
apiVersion: kubeovn.io/v1
kind: ProviderNetwork
metadata:
  name: net1
spec:
  defaultInterface: eth1
---
apiVersion: kubeovn.io/v1
kind: Vlan
metadata:
  name: vlan1
spec:
  id: 0
  provider: net1
---
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
  name: subnet1
spec:
  protocol: IPv4
  cidrBlock: 172.19.0.0/16
  excludeIps:
  - 172.19.0.2..172.19.0.20
  gateway: 172.19.0.1
  vlan: vlan1
  provider: attachnet.default.ovn
```

The subnet's `provider` MUST be the same as the `provider` of the NetworkAttachmentDefinition created above.

7.22.4 Create Receiving Pod

Create the following Pod:

```
apiVersion: v1
kind: Pod
metadata:
  name: pod1
  annotations:
    k8s.v1.cni.cncf.io/networks: default/attachnet
spec:
  containers:
  - name: bash
    image: docker.io/kubeovn/kube-ovn:v1.14.4
    args:
    - bash
    - -c
    - sleep infinity
    securityContext:
      privileged: true
```

After the Pod has been created, checkout the IP addresses:

```
$ kubectl get ips | grep pod1
pod1.default                      10.16.0.12   00:00:00:FF:34:24  kube-ovn-worker  ovn-default
pod1.default.attachnet.default.ovn  172.19.0.21  00:00:00:A0:30:68  kube-ovn-worker  subnet1
```

The IP address `172.19.0.21` will be used later.

7.22.5 Create OVN Remote Port Mirroring

Create the following OVN remote port mirroring:

```
kubectl ko nbctl mirror-add mirror1 gre 99 from-lport 172.19.0.21
kubectl ko nbctl lsp-attach-mirror coredns-787d4945fb-gpnkb.kube-system mirror1
```

`coredns-787d4945fb-gpnkb.kube-system` is the OVN LSP name with a format `<POD_NAME>.<POD_NAMESPACE>`.

Here is the OVN command usage:

```
ovn-nbctl mirror-add <NAME> <TYPE> <INDEX> <FILTER> <IP>

NAME   - add a mirror with given name
TYPE   - specify TYPE 'gre' or 'erspan'
INDEX  - specify the tunnel INDEX value
         (indicates key if GRE, erpsan_idx if ERSPAN)
FILTER - specify FILTER for mirroring selection
         ('to-lport' / 'from-lport')
IP     - specify Sink / Destination i.e. Remote IP

ovn-nbctl mirror-del [NAME]         remove mirrors
ovn-nbctl mirror-list               print mirrors

ovn-nbctl lsp-attach-mirror PORT MIRROR   attach source PORT to MIRROR
ovn-nbctl lsp-detach-mirror PORT MIRROR   detach source PORT from MIRROR
```

7.22.6 Configure Receiving Pod

Execute the following commands in the Pod:

```
root@pod1:/kube-ovn# ip link add mirror1 type gretap local 172.19.0.21 key 99 dev net1
root@pod1:/kube-ovn# ip link set mirror1 up
```

Now you can capture the mirrored packets:

```
root@pod1:/kube-ovn# tcpdump -i mirror1 -nnve
tcpdump: listening on mirror1, link-type EN10MB (Ethernet), snapshot length 262144 bytes
05:13:30.328808 00:00:00:a3:f5:e2 > 00:00:00:97:0f:6e, ethertype ARP (0x0806), length 42: Ethernet (len 6), IPv4 (len 4), Request who-has 10.16.0.7 tell 10.
16.0.4, length 28
05:13:30.559167 00:00:00:a3:f5:e2 > 00:00:00:89:d5:cc, ethertype IPv4 (0x0800), length 212: (tos 0x0, ttl 64, id 57364, offset 0, flags [DF], proto UDP (17),
length 198)
    10.16.0.4.53 > 10.16.0.6.50472: 34511 NXDomain*- 0/1/1 (170)
05:13:30.559343 00:00:00:a3:f5:e2 > 00:00:00:89:d5:cc, ethertype IPv4 (0x0800), length 212: (tos 0x0, ttl 64, id 57365, offset 0, flags [DF], proto UDP (17),
length 198)
    10.16.0.4.53 > 10.16.0.6.45177: 1659 NXDomain*- 0/1/1 (170)
05:13:30.560625 00:00:00:a3:f5:e2 > 00:00:00:89:d5:cc, ethertype IPv4 (0x0800), length 200: (tos 0x0, ttl 64, id 57367, offset 0, flags [DF], proto UDP (17),
length 186)
    10.16.0.4.53 > 10.16.0.6.43848: 2636*- 0/1/1 (158)
05:13:30.562774 00:00:00:a3:f5:e2 > 00:00:00:89:d5:cc, ethertype IPv4 (0x0800), length 191: (tos 0x0, ttl 64, id 57368, offset 0, flags [DF], proto UDP (17),
length 177)
    10.16.0.4.53 > 10.16.0.6.37755: 48737 NXDomain*- 0/1/1 (149)
05:13:30.563523 00:00:00:a3:f5:e2 > 00:00:00:89:d5:cc, ethertype IPv4 (0x0800), length 187: (tos 0x0, ttl 64, id 57369, offset 0, flags [DF], proto UDP (17),
length 173)
```

```
    10.16.0.4.53 > 10.16.0.6.53887: 45519 NXDomain*- 0/1/1 (145)
05:13:30.564940 00:00:00:a3:f5:e2 > 00:00:00:89:d5:cc, ethertype IPv4 (0x0800), length 201: (tos 0x0, ttl 64, id 57370, offset 0, flags [DF], proto UDP (17),
length 187)
    10.16.0.4.53 > 10.16.0.6.40846: 25745 NXDomain*- 0/1/1 (159)
05:13:30.565140 00:00:00:a3:f5:e2 > 00:00:00:89:d5:cc, ethertype IPv4 (0x0800), length 201: (tos 0x0, ttl 64, id 57371, offset 0, flags [DF], proto UDP (17),
length 187)
    10.16.0.4.53 > 10.16.0.6.45214: 61875 NXDomain*- 0/1/1 (159)
05:13:30.566023 00:00:00:a3:f5:e2 > 00:00:00:55:e4:4e, ethertype IPv4 (0x0800), length 80: (tos 0x0, ttl 64, id 45937, offset 0, flags [DF], proto UDP (17),
length 66)
    10.16.0.4.44116 > 172.18.0.1.53: 16025+ [1au] AAAA? kube-ovn.io. (38)
```

7.22.7 Notice

1. If you are using ERSPAN as the encapsulation protocol, the Linux kernel version of the OVN nodes and remote devices must not be lower than 4.14. If you are using ERSPAN as the encapsulation protocol and using IPv6 as the transport network, the Linux kernel version must not be lower than 4.16.
2. The transmission of mirrored traffic is unidirectional, so you only need to ensure that the OVN node can access the remote device.

PDF Slack Support

July 30, 2025

April 20, 2023

GitHub

7.22.8 Comments

7.23 NodeLocal DNSCache and Kube-OVN adaptation

NodeLocal DNSCache improves cluster DNS performance by running DNS cache as a DaemonSet on cluster nodes. This function can also be adapted to Kube-OVN.

7.23.1 Nodelocal DNSCache deployment

Deploy Kubernetes NodeLocal DNScache

This step refers to Kubernetes official website configuration nodelocaldnscache.

Deploy with the following script:

```
#!bin/bash

localdns=169.254.20.10
domain=cluster.local
kubedns=10.96.0.10

wget https://raw.githubusercontent.com/kubernetes/kubernetes/master/cluster/addons/dns/nodelocaldns/nodelocaldns.yaml
sed -i "s/__PILLAR__LOCAL__DNS__/$localdns/g; s/__PILLAR__DNS__DOMAIN__/$domain/g; s/,__PILLAR__DNS__SERVER__//g; s/__PILLAR__CLUSTER__DNS__/$kubedns/g"
nodelocaldns.yaml

kubectl apply -f nodelocaldns.yaml
```

Modify the kubelet configuration file on each node, modify the clusterDNS field in `/var/lib/kubelet/config.yaml` to the local DNS IP 169.254.20.10, and then restart the kubelet service.

Kube-OVN corresponding DNS configuration

After deploying the Nodelocal DNScache component of Kubernetes, Kube-OVN needs to make the following modifications:

UNDERLAY SUBNET ENABLE U2O SWITCH

If the underlay subnet needs to use the local DNS function, you need to enable the U2O function, that is, configure `spec.u2oInterconnection = true` in `kubectl edit subnet {your subnet}`. If it is an overlay subnet, this step is not required.

SPECIFY THE CORRESPONDING LOCAL DNS IP FOR KUBE-OVN-CONTROLLER

```
kubectl edit deployment kube-ovn-controller -n kube-system
```

Add field to spec.template.spec.containers.args `--node-local-dns-ip=169.254.20.10`

REBUILD THE CREATED PODS

The reason for this step is to let the Pod regenerate `/etc/resolv.conf` so that the nameserver points to the local DNS IP. If the nameserver of the Pod is not rebuilt, it will still use the DNS ClusterIP of the cluster. At the same time, if the u2o switch is turned on, the Pod needs to be rebuilt to regenerate the Pod gateway.

7.23.2 Validator local DNS cache function

After the above configuration is completed, you can find the Pod verification as follows. You can see that the Pod's DNS server points to the local 169.254.20.10 and successfully resolves the domain name:

```
# kubectl exec -it pod1 -- nslookup github.com
Server:         169.254.20.10
Address:        169.254.20.10:53


Name:   github.com
Address: 20.205.243.166
```

You can also capture packets at the node and verify as follows. You can see that the DNS query message reaches the local DNS service through the ovn0 network card, and the DNS response message returns in the same way:

```
# tcpdump -i any port 53

06:20:00.441889 659246098c56_h P   ifindex 17 00:00:00:73:f1:06 ethertype IPv4 (0x0800), length 75: 10.16.0.2.40230 > 169.254.20.10.53: 1291+ A? baidu.com.
(27)
06:20:00.441889 ovn0  In  ifindex 7 00:00:00:50:32:cd ethertype IPv4 (0x0800), length 75: 10.16.0.2.40230 > 169.254.20.10.53: 1291+ A? baidu.com. (27)
06:20:00.441950 659246098c56_h P   ifindex 17 00:00:00:73:f1:06 ethertype IPv4 (0x0800), length 75: 10.16.0.2.40230 > 169.254.20.10.53: 1611+ AAAA?
baidu.com. (27)
06:20:00.441950 ovn0  In  ifindex 7 00:00:00:50:32:cd ethertype IPv4 (0x0800), length 75: 10.16.0.2.40230 > 169.254.20.10.53: 1611+ AAAA? baidu.com. (27)
06:20:00.442203 ovn0  Out ifindex 7 00:00:00:52:99:d8 ethertype IPv4 (0x0800), length 145: 169.254.20.10.53 > 10.16.0.2.40230: 1611* 0/1/0 (97)
06:20:00.442219 659246098c56_h Out ifindex 17 00:00:00:ea:b3:5e ethertype IPv4 (0x0800), length 145: 169.254.20.10.53 > 10.16.0.2.40230: 1611* 0/1/0 (97)
06:20:00.442273 ovn0  Out ifindex 7 00:00:00:52:99:d8 ethertype IPv4 (0x0800), length 125: 169.254.20.10.53 > 10.16.0.2.40230: 1291* 2/0/0 A 39.156.66.10, A
110.242.68.66 (77)
06:20:00.442278 659246098c56_h Out ifindex 17 00:00:00:ea:b3:5e ethertype IPv4 (0x0800), length 125: 169.254.20.10.53 > 10.16.0.2.40230: 1291* 2/0/0 A 39.
156.66.10, A 110.242.68.66 (77)
```

7.23.3 Note

⚠️ **Note:**
If NetworkPolicy is configured in your environment, make sure to explicitly allow traffic to the local DNS IP (such as 169.254.20.10) and the node's CIDR in your NetworkPolicy. This prevents DNS requests and responses from being blocked by NetworkPolicy, which could cause Pods to fail DNS resolution.

NetworkPolicy Example

Below is an example NetworkPolicy that allows Pods to access the local DNS cache and node network:

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-local-dns-and-node-cidr
  namespace: default  # Change the namespace as needed
spec:
  podSelector: {}  # Apply to all Pods; add label selectors as needed
  policyTypes:
  - Ingress
  - Egress
  egress:
  # Allow access to local DNS cache
  - to:
    - ipBlock:
        cidr: 169.254.20.10/32
  # Allow access to node CIDR (modify according to your actual node network CIDR)
  - to:
    - ipBlock:
        cidr: 10.0.0.0/8  # Example node CIDR; modify as needed
  ingress:
  # Allow responses from local DNS cache
  - from:
    - ipBlock:
        cidr: 169.254.20.10/32
  # Allow traffic from node CIDR (modify according to your actual node network CIDR)
  - from:
    - ipBlock:
        cidr: 10.0.0.0/8  # Example node CIDR; modify as needed
```

Notes:

- `169.254.20.10/32` : The IP address of the local DNS cache
- `10.0.0.0/8` : Example node CIDR; please modify according to your actual node network range

PDF **Slack** **Support**

July 3, 2025

May 5, 2023

GitHub

7.23.4 Comments

7.24 Default VPC NAT Policy Rule

7.24.1 Purpose

In the Overlay Subnet under the default VPC, when the `natOutgoing` switch is turned on, all Pods in the subnet need to do SNAT to access the external network, but in some scenarios we do not want all Pods in the subnet to access the external network by SNAT.

So the NAT Policy Rule is to provide a way for users to decide which CIDRs or IPs in the subnet to access the external network need SNAT.

7.24.2 How to use NAT Policy Rules

Enable the `natOutgoing` switch in `subnet.Spec`, and add the field `natOutgoingPolicyRules` as follows:

```yaml
spec:
  natOutgoing: true
  natOutgoingPolicyRules:
    - action: forward
      match:
        srcIPs: 10.0.11.0/30,10.0.11.254
    - action: nat
      match:
        srcIPs: 10.0.11.128/26
        dstIPs: 114.114.114.114,8.8.8.8
```

The above case shows that there are two NAT policy rules:

1. Packets with source IP 10.0.11.0/30 or 10.0.11.254 will not perform SNAT when accessing the external network.
2. When a packet with source IP 10.0.11.128/26 and destination IP 114.114.114.114 or 8.8.8.8 accesses the external network, SNAT will be performed.

Field description:

`action`: The action that will be executed for packets that meets the corresponding conditions of the `match`. The action is divided into two types: `forward` and `nat`. When natOutgoingPolicyRules is not configured, packets are still SNAT by default.

`match`: Indicates the matching segment of the message, the matching segment includes `srcIPs` and `dstIPs`, here indicates the source IP and destination IP of the message from the subnet to the external network. `match.srcIPs` and `match.dstIPs` support multiple cidr and ip, separated by commas. If multiple match rules overlap, the action that is matched first will be executed according to the order of the `natOutgoingPolicyRules` array.

PDF **Slack** **Support**

September 1, 2023

June 5, 2023

GitHub

7.24.3 Comments

8. Reference

8.1 Architecture

This document describes the general architecture of Kube-OVN, the functionality of each component and how they interact with each other.

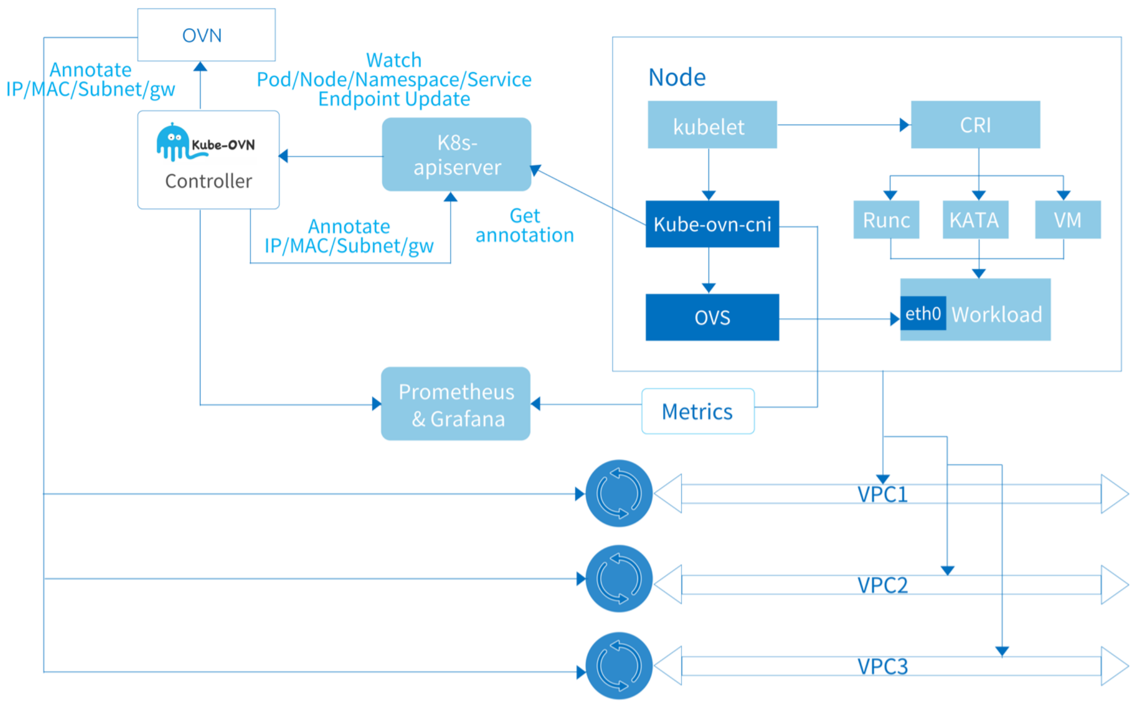
Overall, Kube-OVN serves as a bridge between Kubernetes and OVN, combining proven SDN with Cloud Native. This means that Kube-OVN not only implements network specifications under Kubernetes, such as CNI, Service and Networkpolicy, but also brings a large number of SDN domain capabilities to cloud-native, such as logical switches, logical routers, VPCs, gateways, QoS, ACLs and traffic mirroring.

Kube-OVN also maintains a good openness to integrate with many technology solutions, such as Cilium, Submariner, Prometheus, KubeVirt, etc.

8.1.1 Component Introduction

The components of Kube-OVN can be broadly divided into three categories.

- Upstream OVN/OVS components.
- Core Controller and Agent.
- Monitoring, operation and maintenance tools and extension components.



Upstream OVN/OVS Components

This type of component comes from the OVN/OVS community with specific modifications for Kube-OVN usage scenarios. OVN/OVS itself is a mature SDN system for managing virtual machines and containers, and we strongly recommend that users interested in the Kube-OVN implementation read ovn-architecture(7) first to understand what OVN is and how to integrate with it. Kube-OVN uses the northbound interface of OVN to create and coordinate virtual networks and map the network concepts into Kubernetes.

All OVN/OVS-related components have been packaged into images and are ready to run in Kubernetes.

OVN-CENTRAL

The `ovn-central` Deployment runs the control plane components of OVN, including `ovn-nb`, `ovn-sb`, and `ovn-northd`.

- `ovn-nb`: Saves the virtual network configuration and provides an API for virtual network management. `kube-ovn-controller` will mainly interact with `ovn-nb` to configure the virtual network.
- `ovn-sb`: Holds the logical flow table generated from the logical network of `ovn-nb`, as well as the actual physical network state of each node.
- `ovn-northd`: translates the virtual network of `ovn-nb` into a logical flow table in `ovn-sb`.

Multiple instances of `ovn-central` will synchronize data via the Raft protocol to ensure high availability.

OVS-OVN

`ovs-ovn` runs as a DaemonSet on each node, with `openvswitch`, `ovsdb`, and `ovn-controller` running inside the Pod. These components act as agents for `ovn-central` to translate logical flow tables into real network configurations.

Core Controller and Agent

This part is the core component of Kube-OVN, serving as a bridge between OVN and Kubernetes, bridging the two systems and translating network concepts between them. Most of the core functions are implemented in these components.

KUBE-OVN-CONTROLLER

This component performs the translation of all resources within Kubernetes to OVN resources and acts as the control plane for the entire Kube-OVN system. The `kube-ovn-controller` listens for events on all resources related to network functionality and updates the logical network within the OVN based on resource changes. The main resources listened including:

Pod, Service, Endpoint, Node, NetworkPolicy, VPC, Subnet, Vlan, ProviderNetwork.

Taking the Pod event as an example, `kube-ovn-controller` listens to the Pod creation event, allocates the address via the built-in in-memory IPAM function, and calls `ovn-central` to create logical ports, static routes and possible ACL rules. Next, `kube-ovn-controller` writes the assigned address and subnet information such as CIDR, gateway, route, etc. to the annotation of the Pod. This annotation is then read by `kube-ovn-cni` and used to configure the local network.

KUBE-OVN-CNI

This component runs on each node as a DaemonSet, implements the CNI interface, and operates the local OVS to configure the local network.

This DaemonSet copies the `kube-ovn` binary to each machine as a tool for interaction between `kubelet` and `kube-ovn-cni`. This binary sends the corresponding CNI request to `kube-ovn-cni` for further operation. The binary will be copied to the `/opt/cni/bin` directory by default.

`kube-ovn-cni` will configure the specific network to perform the appropriate traffic operations, and the main tasks including:

1. Config `ovn-controller` and `vswitchd`.
2. Handle CNI Add/Del requests:

a. Create or delete veth pair and bind or unbind to OVS ports.

b. Configure OVS ports

c. Update host iptables/ipset/route rules.

3. Dynamically update the network QoS.
4. Create and configure the `ovn0` NIC to connect the container network and the host network.
5. Configure the host NIC to implement Vlan/Underlay/EIP.
6. Dynamically config inter-cluster gateways.

Monitoring, Operation and Maintenance Tools and Extension Components

These components provide monitoring, diagnostics, operations tools, and external interface to extend the core network capabilities of Kube-OVN and simplify daily operations and maintenance.

KUBE-OVN-SPEAKER

This component is a DaemonSet running on a specific labeled nodes that publish routes to the external, allowing external access to the container directly through the Pod IP.

For more information on how to use it, please refer to BGP Support.

KUBE-OVN-PINGER

This component is a DaemonSet running on each node to collect OVS status information, node network quality, network latency, etc. The monitoring metrics collected can be found in Metrics.

KUBE-OVN-MONITOR

This component collects OVN status information and the monitoring metrics, all metrics can be found in Metrics.

KUBECTL-KO

This component is a kubectl plugin, which can quickly run common operations, for more usage, please refer to kubectl plugin.

PDF Slack Support

July 30, 2025

May 24, 2022

GitHub

8.1.2 Comments

8.2 Kube-OVN RoadMap

This document defines high level goals for Kube-OVN project. We welcome community contributors to discuss and update this Roadmap through Issues.

8.2.1 Network Datapath

Kube-OVN currently supports two network modes, Overlay and Underlay. We hope to improve the stability, performance, and compatibility with the ecosystem of these two network modes in Kubernetes.

- Improved Datapath network performance
- Keeping up with the latest network API features in the community
- Enhanced network monitoring and visualization capabilities
- Addition of automated test cases for various scenarios

8.2.2 VPC Network

VPC network is a key feature of Kube-OVN, many functions have been used in production environment, and we hope to increase the maturity of these functions and improve the user experiences.

- Standardize multiple gateway solutions and provide the best egress practice
- Provide more VPC internal basic network capabilities and solutions, such as DNS, DHCP, LoadBalancer, etc.
- Simplify VPC operation complexity and provide a more comprehensive CLI
- Supplement automated test cases for various scenarios

8.2.3 User Experience

Improve the user experience of Kubernetes cni, making container networking simpler, more reliable, and efficient.

- Helm/Operator to automate daily operations
- More organized metrics and grafana dashboard
- Troubleshooting tools that can automatically find known issues
- Integrated with other projects like kubeaz, kubekey, sealos etc.

PDF **Slack** **Support**

2024 5 16

2024 5 9

GitHub

8.2.4 Comments

8.3 Release Management

Kube-OVN currently mainly releases Minor and Patch versions. Minor versions include the addition of new features, major OVN/OVS upgrades, internal architecture adjustments, and API changes. Patch versions focus primarily on bug fixes, security vulnerability repairs, dependency upgrades, and are backward compatible with previous APIs.

8.3.1 Maintenance Strategy

Kube-OVN currently continuously maintains the main branch and the two most recent release branches, such as `master`, `release-1.12`, and `release-1.11`. The latest release branch (e.g., `release-1.12`) will undergo more frequent iterations and releases, with all bug fixes, security vulnerabilities, and dependency upgrades being backported to this branch as much as possible.

The previous release branch (e.g., `release-1.11`) will backport significant bug fixes and security vulnerability repairs.

8.3.2 Release Cycle

Minor versions are released as needed, based on whether there are significant new features or major architectural adjustments completed in the main branch, currently about once every six months. Patch versions are triggered based on the bug fix status of the branch, generally within a week after bug fixes are merged.

8.3.3 Patch Version Release Method

Currently, most of the work for Patch versions can be automated using the hack/release.sh script, with the main steps described as follows:

1. Check the current branch build status (automated)
2. Push the new tag image to Docker Hub (automated)
3. Push the new tag code to GitHub (automated)
4. Update the version information in the code (automated)
5. Update the version information in the documentation repository (automated)
6. Generate Release Note PR (automated)
7. Merge Release Note (manual)
8. Manually merge the GitHub action generated Release Note PR
9. Modify the GitHub Release information (manual)
10. Edit the newly created Release on the GitHub Release page, change the title to the corresponding version number (e.g., `v1.12.12`), and copy the Release Note generated in the previous step into the Release details

8.3.4 Minor Version Release Method

Currently, the main tasks for Minor branches still need to be completed manually, with the main steps described as follows:

1. Push a new release branch on GitHub, e.g., `release-1.13` (manual)
2. Update the version information in the `VERSION`, `dist/images/install.sh`, `charts/kube-ovn/values.yaml`, and `charts/kube-ovn/Chart.yaml` from the main branch to the next Minor version, e.g., `v1.14.0` (manual)
3. Push the new tag image to Docker Hub (manual)
4. Push the new tag code to GitHub in the release branch (manual)
5. Create a new release branch in the documentation repository, e.g., `v1.13`, and modify the `version` and `branch` information in the `mkdocs.yml` file (manual)
6. Generate Release Note PR (automated)
7. Merge Release Note (manual)
8. Manually merge the GitHub action generated Release Note PR
9. Modify the GitHub Release information (manual)
10. Edit the newly created Release on the GitHub Release page, change the title to the corresponding version number (e.g., `v1.13.0`), and copy the Release Note generated in the previous step into the Release details
11. Update the `VERSION` file in the release branch to the next Patch version, e.g., `v1.13.1`

PDF **Slack** **Support**

May 16, 2024

May 8, 2024

GitHub

8.3.5 Comments

8.4 Feature Stage

In Kube-OVN, feature stage is classified into **Alpha**, **Beta** and **GA**, based on the degree of feature usage, documentation and test coverage.

8.4.1 Definition of Stage

For **Alpha** stage functions:

- The feature is not fully documented and well tested.
- This feature may change or even be removed in the future.
- This feature API is not guaranteed to be stable and may be removed.
- Community provides low priority support for this feature and long-term support cannot be guaranteed.
- Since feature stability and long-term support cannot be guaranteed, it can be tested and verified, but is not recommended for production use.

For **Beta** stage functions:

- This feature is partially documented and tested, but complete coverage is not guaranteed.
- This feature may change in the future and the upgrade may affect the network, but it will not be removed as a whole.
- This feature API may change in the future and the fields may be adjusted, but not removed as a whole.
- This feature will be supported by the community in the long term.
- It can be used on non-critical services as the functionality will be supported for a long time, but it is not recommended for critical production service as there is a possibility of changes in functionality and APIs that may break the network.

For **GA** stage functions:

- The feature has full documentation and test coverage.
- The feature will remain stable and upgrades will be guaranteed to be smooth.
- This feature API is not subject to disruptive changes.
- This feature will be supported with high priority by the community and long-term support will be guaranteed.

8.4.2 Feature Stage List

This list records the feature stages from the 1.8 release.

Feature	Default	Stage	Since	Until
Namespaced Subnet	true	GA	1.8	
Distributed Gateway	true	GA	1.8	
Active-backup Centralized Gateway	true	GA	1.8	
ECMP Centralized Gateway	false	Beta	1.8	
Subnet ACL	true	Alpha	1.9	
Subnet Isolation (Will be replaced by ACL later)	true	Beta	1.8	
Underlay Subnet	true	GA	1.8	
Multiple Pod Interface	true	Beta	1.8	
Subnet DHCP	false	Alpha	1.10	
Subnet with External Gateway	false	Alpha	1.8	
Cluster Inter-Connection with OVN-IC	false	Beta	1.8	
Cluster Inter-Connection with Submariner	false	Alpha	1.9	
VIP Reservation	true	Alpha	1.10	
Create Custom VPC	true	Beta	1.8	
Custom VPC Floating IP/SNAT/DNAT	true	Alpha	1.10	
Custom VPC Static Route	true	Alpha	1.10	
Custom VPC Policy Route	true	Alpha	1.10	
Custom VPC Security Group	true	Alpha	1.10	
Container Bandwidth QoS	true	GA	1.8	
linux-netem QoS	true	Alpha	1.9	
Prometheus Integration	false	GA	1.8	
Grafana Integration	false	GA	1.8	
IPv4/v6 DualStack	false	GA	1.8	
Default VPC EIP/SNAT	false	Beta	1.8	
Traffic Mirroring	false	GA	1.8	
NetworkPolicy	true	Beta	1.8	
Webhook	false	Alpha	1.10	
Performance Tuning	false	Beta	1.8	
Interconnection with Routes in Overlay Mode	false	Alpha	1.8	
BGP Support	false	Alpha	1.9	
Cilium Integration	false	Alpha	1.10	
Custom VPC Peering	false	Alpha	1.10	

Feature	Default	Stage	Since	Until
Mellanox Offload	false	Alpha	1.8	
Corigine Offload	false	Alpha	1.10	
Windows Support	false	Alpha	1.10	
DPDK Support	false	Alpha	1.10	
OpenStack Integration	false	Alpha	1.9	
Single Pod Fixed IP/Mac	true	GA	1.8	
Workload with Fixed IP	true	GA	1.8	
StatefulSet with Fixed IP	true	GA	1.8	
VM with Fixed IP	false	Beta	1.9	
Load Balancer Type Service in Default VPC	false	Alpha	1.11	
Load Balance in Custom VPC	false	Alpha	1.11	
DNS in Custom VPC	false	Alpha	1.11	
Underlay and Overlay Interconnection	false	Beta	1.12	
VPC Egress Gateway	true	Alpha	1.14	

PDF **Slack** **Support**

July 30, 2025

September 6, 2022

GitHub

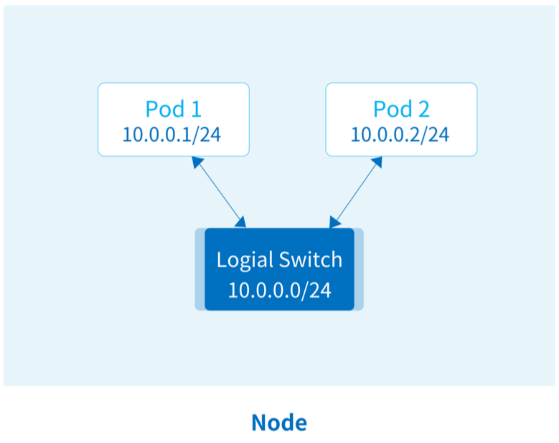
8.4.3 Comments

8.5 Underlay Traffic Topology

This document describes the forwarding path of traffic in Underlay mode under different scenarios.

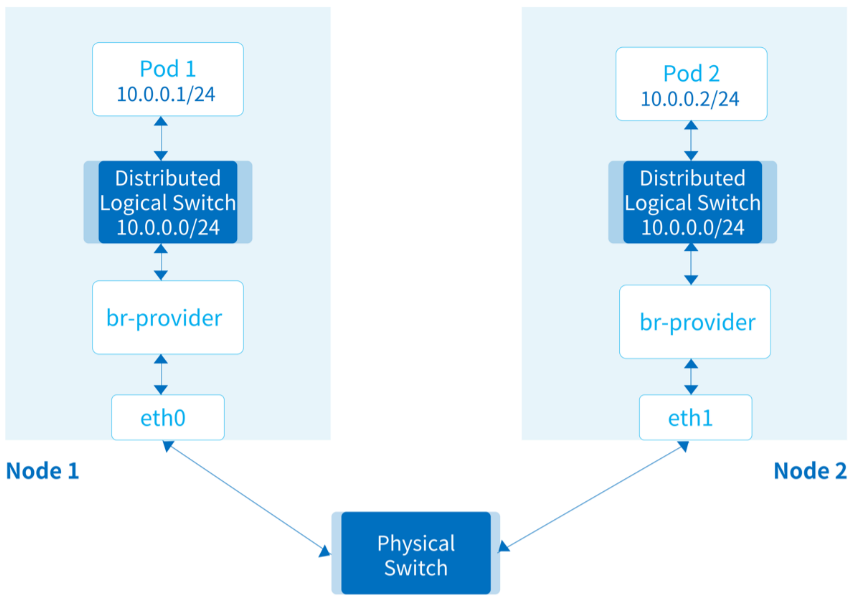
8.5.1 Pods in Same Node and Same Subnet

Internal logical switches exchange packets directly, without access to the external network.



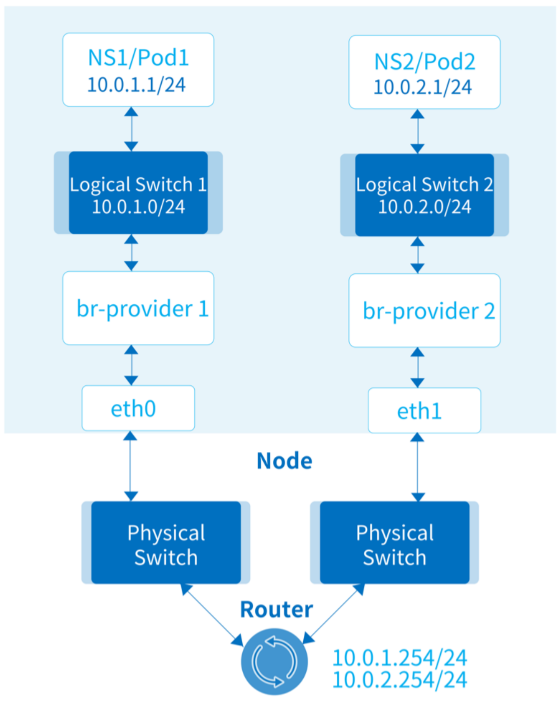
8.5.2 Pods in Different Nodes and Same Subnet

Packets enter the physic switch via the node NIC and are exchanged by the physic switch.



8.5.3 Pods in Same Node and Different Subnets

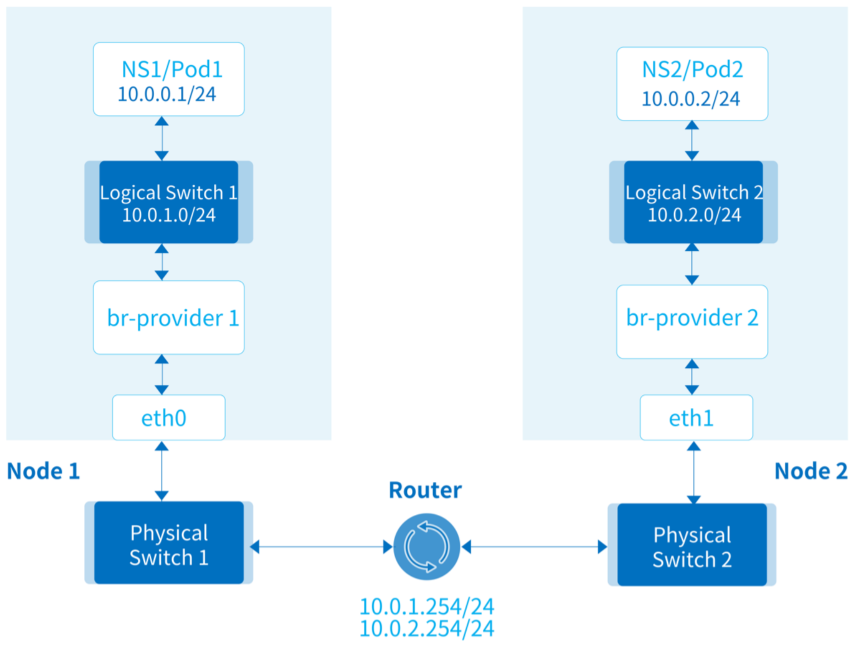
Packets enter the physic network via the node NIC and are exchanged and routed and forwarded by physic switches and routers.



Here br-provider-1 and br-provider-2 can be the same OVS bridge,multiple subnet can share a Provider Network.

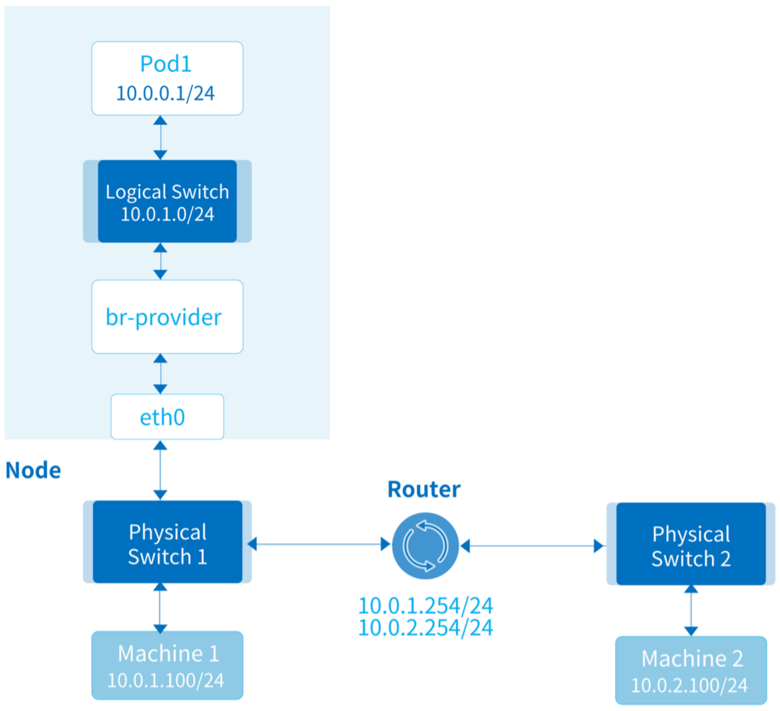
8.5.4 Pods in Different Nodes and Different Subnets

Packets enter the physic network via the node NIC and are exchanged and routed and forwarded by physic switches and routers.



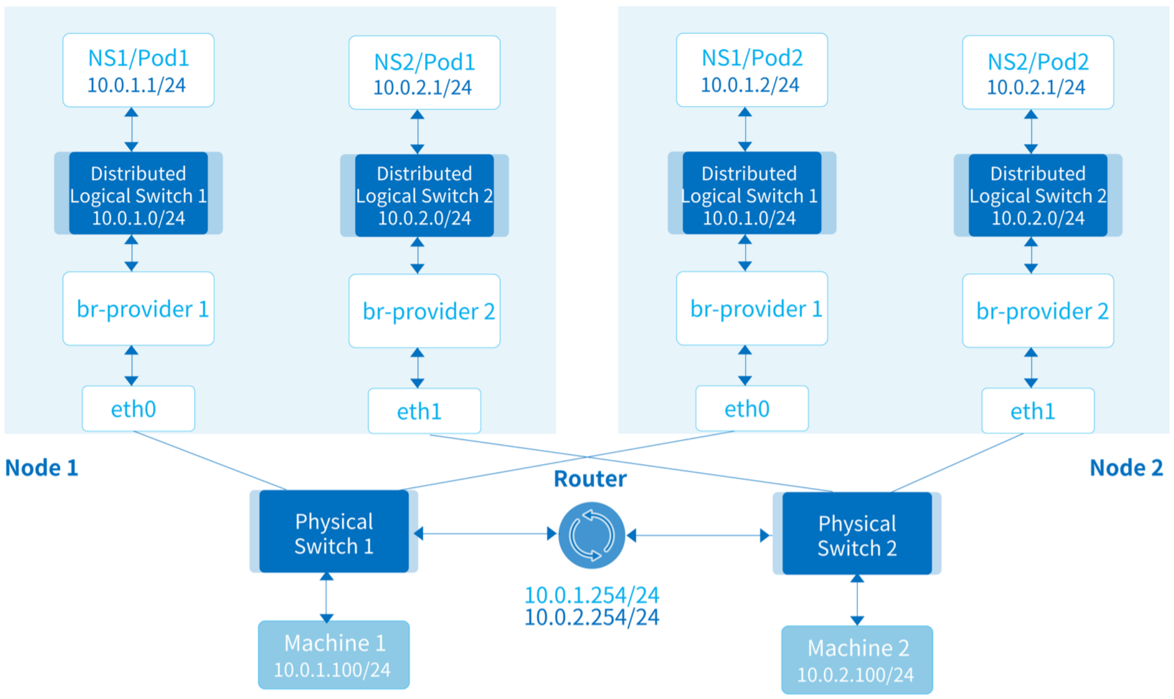
8.5.5 Access to External

Packets enter the physic network via the node NIC and are exchanged and routed and forwarded by physic switches and routers.

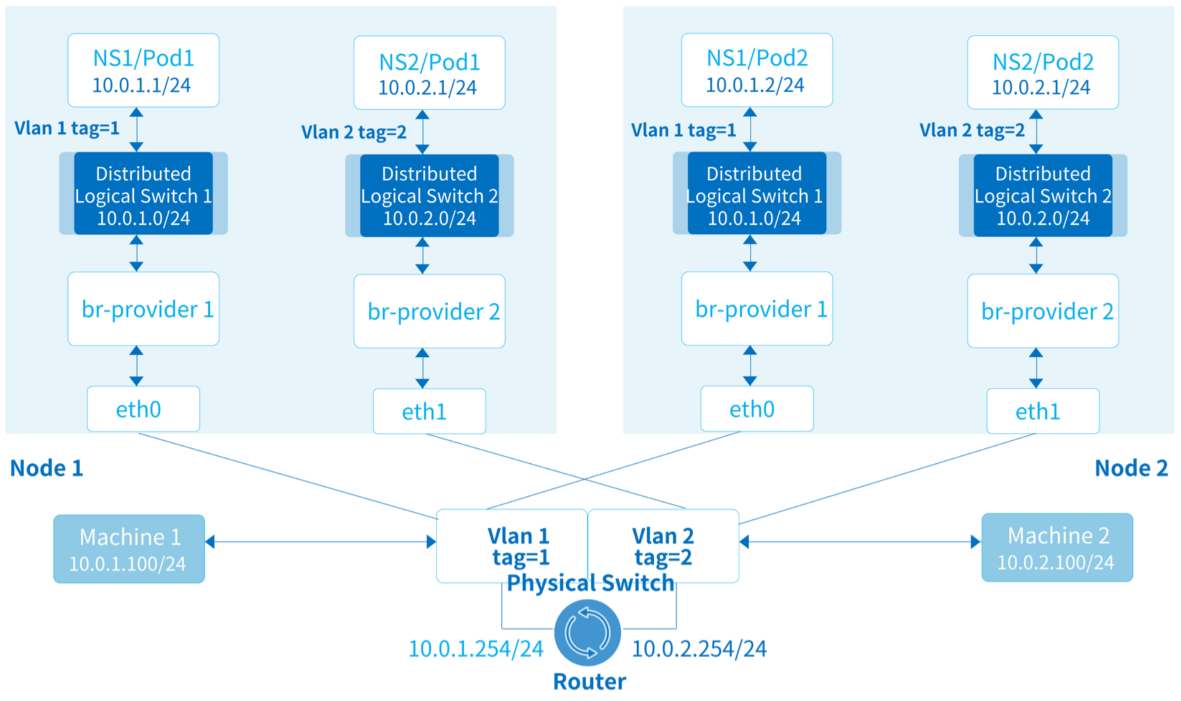


The communication between nodes and Pods follows the same logic.

8.5.6 Overview without Vlan Tag



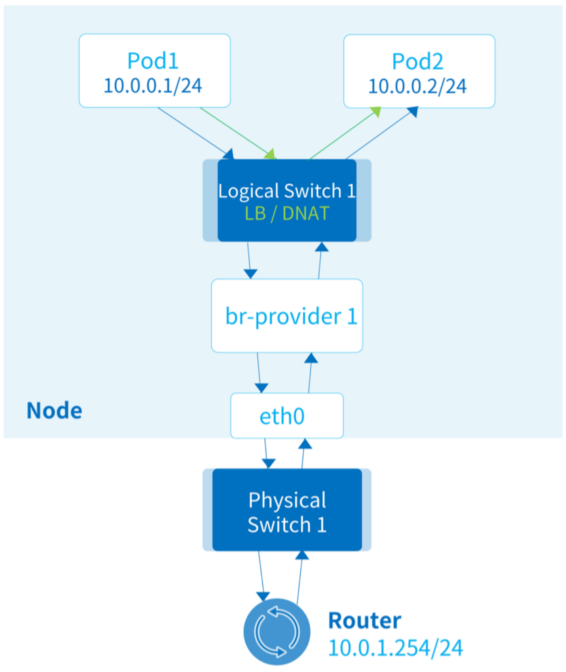
8.5.7 Overview with Vlan Tag



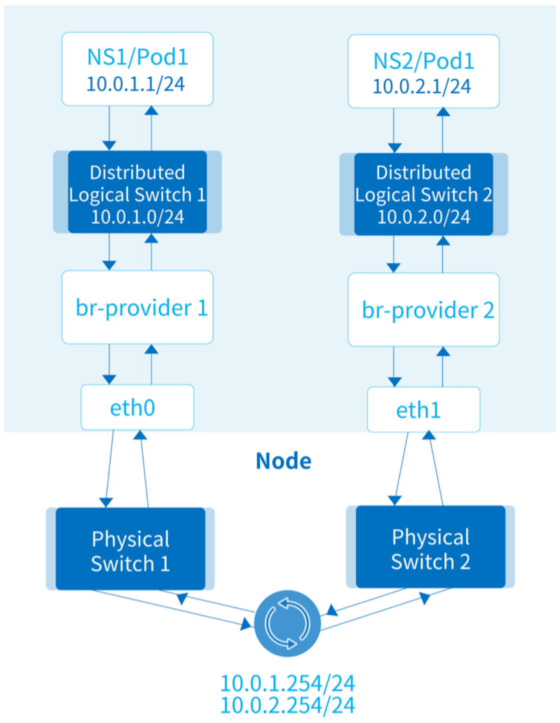
8.5.8 Pod visit Service IP

Kube-OVN configures load balancing for each Kubernetes Service on a logical switch on each subnet. When a Pod accesses other Pods by accessing the Service IP, a network packet is constructed with the Service IP as the destination address and the MAC address of the gateway as the destination MAC address. After the network packet enters the logical switch, load balancing will intercept and DNAT the network packet to modify the destination IP and port to the IP and port of one of the Endpoint corresponding to the Service. Since the logical switch does not modify the Layer 2 destination MAC address of the network packet, the network packet will still be delivered to the physic gateway after entering the physic switch, and the physic gateway will be required to forward the network packet.

Service Backend is the Same Node and Same Subnet Pod



Service Backend is the Same Node and Different Subnets Pod



PDF Slack Support

July 30, 2025

May 20, 2022

GitHub

8.5.9 Comments

8.6 Iptables Rules

Kube-OVN uses `ipset` and `iptables` to implement gateway NAT functionality in the default VPC overlay Subnets.

The ipset used is shown in the following table:

Name(IPv4/IPv6)	Type	Usage
ovn40services/ovn60services	hash:net	Service CIDR
ovn40subnets/ovn60subnets	hash:net	Overlay Subnet CIDR and NodeLocal DNS IP address
ovn40subnets-nat/ovn60subnets-nat	hash:net	Overlay Subnet CIDRs that enable `NatOutgoing`
ovn40subnets-distributed-gw/ovn60subnets-distributed-gw	hash:net	Overlay Subnet CIDRs that use distributed gateway
ovn40other-node/ovn60other-node	hash:net	Internal IP addresses for other Nodes
ovn40local-pod-ip-nat/ovn60local-pod-ip-nat	hash:ip	Deprecated
ovn40subnets-nat-policy	hash:net	All subnet cidrs configured with natOutgoingPolicyRules
ovn40natpr-418e79269dc5-dst	hash:net	The dstIPs corresponding to the rule in natOutgoingPolicyRules
ovn40natpr-418e79269dc5-src	hash:net	The srcIPs corresponding to the rule in natOutgoingPolicyRules

The iptables rules (IPv4) used are shown in the following table:

Table	Chain	Rule	Usage	Note
filter	INPUT	-m set --match-set ovn40services src -j ACCEPT	Allow k8s service and pod traffic to pass through	--
filter	INPUT	-m set --match-set ovn40services dst -j ACCEPT	Allow k8s service and pod traffic to pass through	--
filter	INPUT	-m set --match-set ovn40subnets src -j ACCEPT	Allow k8s service and pod traffic to pass through	--
filter	INPUT	-m set --match-set ovn40subnets dst -j ACCEPT	Allow k8s service and pod traffic to pass through	--
filter	FORWARD	-m set --match-set ovn40services src -j ACCEPT	Allow k8s service and pod traffic to pass through	--
filter	FORWARD	-m set --match-set ovn40services dst -j ACCEPT	Allow k8s service and pod traffic to pass through	--
filter	FORWARD	-m set --match-set ovn40subnets src -j ACCEPT	Allow k8s service and pod traffic to pass through	--
filter	FORWARD	-m set --match-set ovn40subnets dst -j ACCEPT	Allow k8s service and pod traffic to pass through	--
filter	FORWARD	-s 10.16.0.0/16 -m comment --comment "ovn-subnet-gateway,ovn-default"	Used to count packets from the subnet to the external network	"10.16.0.0/16" is the cidr of the "ovn-subnet-gateway" before th comment is used to identify the used to count the subnet inbou outbound gateway packets, an default" after the "," is the nam subnet
filter	FORWARD	-d 10.16.0.0/16 -m comment --comment "ovn-subnet-gateway,ovn-default"	Used to count packets from the external network accessing the subnet	"10.16.0.0/16" is the cidr of the "ovn-subnet-gateway" before th comment is used to identify the used to count the subnet inbou outbound gateway packets, an default" after the "," is the nam subnet
filter	OUTPUT	-p udp -m udp --dport 6081 -j MARK --set-xmark 0x0	Clear traffic tag to prevent SNAT	UDP: bad checksum on VXLAN
nat	PREROUTING	-m comment --comment "kube-ovn prerouting rules" -j OVN-PREROUTING	Enter OVN-PREROUTING chain processing	--
nat	POSTROUTING	-m comment --comment "kube-ovn postrouting rules" -j OVN-POSTROUTING	Enter OVN-POSTROUTING chain processing	--

Table	Chain	Rule	Usage	Note
nat	OVN-PREROUTING	-i ovn0 -m set --match-set ovn40subnets src -m set --match-set ovn40services dst -j MARK --set-xmark 0x4000/0x4000	Adding masquerade tags to Pod access service traffic	Used when the built-in LB is tu
nat	OVN-PREROUTING	-p tcp -m addrtype --dst-type LOCAL -m set --match-set KUBE-NODE-PORT-LOCAL-TCP dst -j MARK --set-xmark 0x80000/0x80000	Add specific tags to ExternalTrafficPolicy for Local's Service traffic (TCP)	Only used when kube-proxy is mode
nat	OVN-PREROUTING	-p udp -m addrtype --dst-type LOCAL -m set --match-set KUBE-NODE-PORT-LOCAL-UDP dst -j MARK --set-xmark 0x80000/0x80000	Add specific tags to ExternalTrafficPolicy for Local's Service traffic (UDP)	Only used when kube-proxy is mode
nat	OVN-POSTROUTING	-m set --match-set ovn40services src -m set --match-set ovn40subnets dst -m mark --mark 0x4000/0x4000 -j SNAT --to-source	Use node IP as the source address for access from node to overlay Pods via service IP.	Works only when kube-proxy is mode
nat	OVN-POSTROUTING	-m mark --mark 0x4000/0x4000 -j MASQUERADE	Perform SNAT for specific tagged traffic	--
nat	OVN-POSTROUTING	-m set --match-set ovn40subnets src -m set --match-set ovn40subnets dst -j MASQUERADE	Perform SNAT for Service traffic between Pods passing through the node	--
nat	OVN-POSTROUTING	-m mark --mark 0x80000/0x80000 -m set --match-set ovn40subnets-distributed-gw dst -j RETURN	For Service traffic where ExternalTrafficPolicy is Local, if the Endpoint uses a distributed gateway, SNAT is not required.	--
nat	OVN-POSTROUTING	-m mark --mark 0x80000/0x80000 -j MASQUERADE	For Service traffic where ExternalTrafficPolicy is Local, if the Endpoint uses a centralized gateway, SNAT is required.	--
nat	OVN-POSTROUTING	-p tcp -m tcp --tcp-flags SYN NONE -m conntrack --ctstate NEW -j RETURN	No SNAT is performed when the Pod IP is exposed to the outside world	--
nat	OVN-POSTROUTING	-s 10.16.0.0/16 -m set ! --match-set ovn40subnets	When the Pod accesses the network outside the cluster, if the subnet is	10.16.0.0/16 is the Subnet CID 192.168.0.101 is the specified node

Table	Chain	Rule	Usage	Note
		dst -j SNAT --to-source 192.168.0.101	NatOutgoing and a centralized gateway with the specified IP is used, perform SNAT	
nat	OVN-POSTROUTING	-m set --match-set ovn40subnets-nat src -m set ! --match-set ovn40subnets dst -j MASQUERADE	When the Pod accesses the network outside the cluster, if NatOutgoing is enabled on the subnet, perform SNAT	--
nat	OVN-POSTROUTING	-m set --match-set ovn40subnets-nat-policy src -m set ! --match-set ovn40subnets dst -j OVN-NAT-POLICY	When Pod accesses the network outside the cluster, if natOutgoingPolicyRules is enabled on the subnet, the packet with the specified policy will perform SNAT	ovn40subnets-nat-policy is all s segments configured with natOutgoingPolicyRules
nat	OVN-POSTROUTING	-m mark --mark 0x90001/0x90001 -j MASQUERADE --random-fully	When Pod accesses the network outside the cluster, if natOutgoingPolicyRules is enabled on the subnet, the packet with the specified policy will perform SNAT	After coming out of OVN-NAT- tagged with 0x90001/0x90001, SNAT
nat	OVN-POSTROUTING	-m mark --mark 0x90002/0x90002 -j RETURN	When Pod accesses the network outside the cluster, if natOutgoingPolicyRules is enabled on the subnet, the packet with the specified policy will perform SNAT	After coming out of OVN-NAT- tagged with 0x90002/0x90002, SNAT
nat	OVN-NAT-POLICY	-s 10.0.11.0/24 -m comment --comment natPolicySubnet-net1 -j OVN-NAT-PSUBNET-aa98851157c5	When Pod accesses the network outside the cluster, if natOutgoingPolicyRules is enabled on the subnet, the packet with the specified policy will perform SNAT	10.0.11.0/24 represents the CI subnet net1, and the rules und NAT-PSUBNET-aa98851157c5 correspond to the natOutgoing configuration of this subnet
nat	OVN-NAT-PSUBNET-xxxxxxxxxxxx	-m set --match-set ovn40natpr-418e79269dc5-src src -m set --match-set ovn40natpr-418e79269dc5-dst dst -j MARK --set-xmark 0x90002/0x90002	When Pod accesses the network outside the cluster, if natOutgoingPolicyRules is enabled on the subnet, the packet with the specified policy will perform SNAT	418e79269dc5 indicates the ID natOutgoingPolicyRules, which viewed through status.natOutgoingPolicyRules[indicating that srcIPs meets ovn40natpr-418e79269dc5-src, meets ovn40natpr-418e79269d marked with tag 0x90002
mangle	OVN-OUTPUT	-d 10.241.39.2/32 -p tcp -m tcp --dport 80 -j MARK --set-xmark 0x90003/0x90003	Introduce kubelet's detection traffic to tproxy with a specific mark	

Table	Chain	Rule	Usage	Note
mangle	OVN-PREROUTING	-d 10.241.39.2/32 -p tcp -m tcp --dport 80 -j TPROXY --on-port 8102 --on-ip 172.18.0.3 --tproxy-mark 0x90004/0x90004	Introduce kubelet's detection traffic to tproxy with a specific mark	

8.6.1 Comments

8.7 Kube-OVN-Pinger args Reference

Based on the Kube-OVN v1.12.0 version, We have compiled the parameters supported by Kube-ovn-pinger, and listed the value types, meanings, and default values of each field defined by the parameters for reference

8.7.1 Args Describeption

Arg Name	Type	Description	Default Value
port	Int	metrics port	8080
kubeconfig	String	Path to kubeconfig file with authorization and master location information. If not set use the inCluster token.	""
ds-namespace	String	kube-ovn-pinger daemonset namespace	"kube-system"
ds-name	String	kube-ovn-pinger daemonset name	"kube-ovn-pinger"
interval	Int	interval seconds between consecutive pings	5
mode	String	server or job Mode	"server"
exit-code	Int	exit code when failure happens	0
internal-dns	String	check dns from pod	"kubernetes.default"
external-dns	String	check external dns resolve from pod	""
external-address	String	check ping connection to an external address	"114.114.114.114"
network-mode	String	The cni plugin current cluster used	"kube-ovn"
enable-metrics	Bool	Whether to support metrics query	true
ovs.timeout	Int	Timeout on JSON-RPC requests to OVS.	2
system.run.dir	String	OVS default run directory.	"/var/run/openvswitch"
database.vswitch.name	String	The name of OVS db.	"Open_vSwitch"
database.vswitch.socket.remote	String	JSON-RPC unix socket to OVS db.	"unix:/var/run/openvswitch/db.sock"
database.vswitch.file.data.path	String	OVS db file.	"/etc/openvswitch/conf.db"
database.vswitch.file.log.path	String	OVS db log file.	"/var/log/openvswitch/ovsdb-server.log"
database.vswitch.file.pid.path	String	OVS db process id file.	"/var/run/openvswitch/ovsdb-server.pid"
database.vswitch.file.system.id.path	String	OVS system id file.	"/etc/openvswitch/system-id.conf"
service.vswitchd.file.log.path	String	OVS vswitchd daemon log file.	"/var/log/openvswitch/ovs-vswitchd.log"
service.vswitchd.file.pid.path	String	OVS vswitchd daemon process id file.	"/var/run/openvswitch/ovs-vswitchd.pid"
service.ovncontroller.file.log.path	String	OVN controller daemon log file.	"/var/log/ovn/ovn-controller.log"

Arg Name	Type	Description	Default Value
service.ovncontroller.file.pid.path	String	OVN controller daemon process id file.	"/var/run/ovn/ovn-controller.pid"

PDF Slack Support

February 23, 2023

February 23, 2023

GitHub

8.7.2 Comments

8.8 Development and Contribution Guide

8.8.1 Contribution Process

Kube-OVN does not have a complex contribution process—all work happens on GitHub. If you want to submit a new feature or fix a bug, simply create an Issue and Pull Request (PR) on GitHub. After maintainers review and all GitHub Actions pass, the code will be merged.

8.8.2 Environmental Preparation

Kube-OVN uses Golang to develop and Go Modules to manage dependency, please check env `GO111MODULE="on"`.

golangci-lint is used to scan code for compliance issues. It needs to be installed in the development environment. Please refer to local-installation.

To reduce the size of the final generated image, Kube-OVN uses some of the Docker buildx experimental features, please update Docker to the latest version and enable buildx:

```
docker buildx create --use
```

8.8.3 Build Image

Use the following command to download the code and generate the image required to run Kube-OVN:

```
git clone https://github.com/kubeovn/kube-ovn.git
cd kube-ovn
make release
```

To build an image to run in an ARM environment, run the following command:

```
make release-arm
```

8.8.4 Building the Base Image

If you need to change the operating system version, dependencies, OVS/OVN code, etc., you need to rebuild the base image.

The Dockerfile used for the base image is `dist/images/Dockerfile.base`.

Build instructions:

```
# build x86 base image
make base-amd64

# build arm base image
make base-arm64
```

8.8.5 Run E2E

Kube-OVN uses:

- KIND to build local Kubernetes cluster: `go install sigs.k8s.io/kind@latest`
- jinjanator to render templates: `pip install jinjanator`
- Ginkgo to run test cases: `go install github.com/onsi/ginkgo/v2/ginkgo; go get github.com/onsi/gomega/...`

Please refer to the relevant documentation for dependency installation.

Run E2E locally:

```
make kind-init
make kind-install
make e2e
```

To run the Underlay E2E test, run the following commands:

```
make kind-init
make kind-install-underlay
make e2e-underlay-single-nic
```

To run the ovn vpc nat gw eip, fip, snat, dnat E2E test, run the following commands:

```
make kind-init
make kind-install
make ovn-vpc-nat-gw-conformance-e2e
```

To run the iptables vpc nat gw eip, fip, snat, dnat E2E test, run the following commands:

```
make kind-init
make kind-install
make iptables-vpc-nat-gw-conformance-e2e
```

To run the loadbalancer service E2E test, run the following commands:

```
make kind-init
make kind-install
make kube-ovn-lb-svc-conformance-e2e
```

To clean, run the following commands:

```
make kind-clean
```

PDF **Slack** **Support**

July 30, 2025

May 20, 2022

GitHub

8.8.6 Comments

8.9 OVS/OVN Customization

Upstream OVN/OVS was originally designed with the goal of a general purpose SDN controller and data plane. Due to some specific usage of the Kubernetes network,Kube-OVN only focused on part of the features. In order to achieve better performance, stability and specific features, Kube-OVN has made some modifications to the upstream OVN/OVS. Users using their own OVN/OVS with Kube-OVN controllers need to be aware of the possible impact of the following changes:

Did not merge into the upstream modification.

- 38df6fa3f7 Adjust the election timer to avoid large-scale cluster election jitter.
- d4888c4e75 add fdb update logging.
- d4888c4e75 fdb: fix mac learning in environments with hairpin enabled.
- 9a81b91368 ovsdb-tool: add optional server id parameter for "join-cluster" command.
- 0700cb90f9 Destination non-service traffic bypasses conntrack to improve performance on a particular data path.
- c48049a64f ECMP algorithm is adjusted from `dp_hash` to `hash` to avoid the hash error problem in some kernels.
- 64383c14a9 Fix kernel Crash issue under Windows.
- 08a95db2ca Support for github action builds on Windows.
- 680e77a190 Windows uses tcp listening by default.
- 05e57b3227 add support for windows.
- 0181b68be1 br-int controller: listen on 127.0.0.1:6653 by default.
- b3801ecb73 modify src route priority.
- 977e569539 fix reaching resubmit limit in underlay.
- 45a4a22161 ovn-nbctl: do not remove LB if vips is empty.
- 540592b9ff Replaces the Mac address as the destination address after DNAT to reduce additional performance overhead.
- 10972d9632 Fix vswitchd ofport_usage memory leak.

Merged into upstream modification:

- 20626ea909 Multicast traffic bypasses LB and ACL processing stages to improve specific data path performance.
- a2d9ff3ccd Deb build adds compile optimization options.

PDF Slack Support

February 16, 2023

May 24, 2022

GitHub

8.9.1 Comments

8.10 Tunnel Protocol Selection

Kube-OVN uses OVN/OVS as the data plane implementation and currently supports `Geneve`, `Vxlan` and `STT` tunnel encapsulation protocols. These three protocols differ in terms of functionality, performance and ease of use. This document will describe the differences in the use of the three protocols so that users can choose according to their situation. OVN Architecture Design Decision can be referenced for the differences in design among these three protocols in OVN.

8.10.1 Geneve

The `Geneve` protocol is the default tunneling protocol selected during Kube-OVN deployment and is also the default recommended tunneling protocol for OVN. This protocol is widely supported in the kernel and can be accelerated using the generic offload capability of modern NICs. Since `Geneve` has a variable header, it is possible to use 24bit space to mark different datapaths users can create a larger number of virtual networks and a single datapath can support 32767 ports.

If you are using Mellanox or Corigine SmartNIC OVS offload, `Geneve` requires a higher kernel version. Upstream kernel of 5.4 or higher, or other compatible kernels that backports this feature.

Due to the use of UDP encapsulation, this protocol does not make good use of the TCP-related offloads of modern NICs when handling TCP over UDP, and consumes more CPU resources when handling large packets.

8.10.2 Vxlan

`Vxlan` is a recently supported protocol in the upstream OVN, which is widely supported in the kernel and can be accelerated using the common offload capabilities of modern NICs. Due to the limited length of the protocol header and the additional space required for OVN orchestration, there is a limit to the number of datapaths that can be created, with a maximum of 4096 datapaths and a maximum of 4096 ports under each datapath. Also, `inport`-based ACLs are not supported due to header length limitations.

`Vxlan` offloading is supported in common kernels if using Mellanox or Corigine SmartNIC.

Due to the use of UDP encapsulation, this protocol does not make good use of the TCP-related offloads of modern NICs when handling TCP over UDP, and consumes more CPU resources when handling large packets.

8.10.3 STT

The `STT` protocol is an early tunneling protocol supported by the OVN that uses TCP-like headers to take advantage of the TCP offload capabilities common to modern NICs and significantly increase TCP throughput. The protocol also has a long header to support full OVN capabilities and large-scale datapaths.

This protocol is not supported in the kernel. To use it, you need to compile an additional OVS kernel module and recompile the new version of the kernel module when upgrading the kernel.

This protocol is not currently supported by the SmartNic and cannot use the offloading capability of OVS offloading.

8.10.4 References

- VXLAN vs GENEVE: Understand The Difference
- OVN FAQ
- What is Geneve

PDF Slack Support

July 15, 2025

June 17, 2022

GitHub

8.10.5 Comments

8.11 Metrics

This document lists all the monitoring metrics provided by Kube-OVN.

8.11.1 ovn-monitor

OVN status metrics:

Type	Metric	Description
Gauge	kube_ovn_ovn_status	OVN Health Status. The values are: (2) for standby or follower, (1) for active or leader, (0) for unhealthy.
Gauge	kube_ovn_failed_req_count	The number of failed requests to OVN stack.
Gauge	kube_ovn_log_file_size	The size of a log file associated with an OVN component.
Gauge	kube_ovn_db_file_size	The size of a database file associated with an OVN component.
Gauge	kube_ovn_chassis_info	Whether the OVN chassis is up (1) or down (0), together with additional information about the chassis.
Gauge	kube_ovn_db_status	The status of OVN NB/SB DB, (1) for healthy, (0) for unhealthy.
Gauge	kube_ovn_logical_switch_info	The information about OVN logical switch. This metric is always up (1).
Gauge	kube_ovn_logical_switch_external_id	Provides the external IDs and values associated with OVN logical switches. This metric is always up (1).
Gauge	kube_ovn_logical_switch_port_binding	Provides the association between a logical switch and a logical switch port. This metric is always up (1).
Gauge	kube_ovn_logical_switch_tunnel_key	The value of the tunnel key associated with the logical switch.
Gauge	kube_ovn_logical_switch_ports_num	The number of logical switch ports connected to the OVN logical switch.
Gauge	kube_ovn_logical_switch_port_info	The information about OVN logical switch port. This metric is always up (1).
Gauge	kube_ovn_logical_switch_port_tunnel_key	The value of the tunnel key associated with the logical switch port.
Gauge	kube_ovn_cluster_enabled	Is OVN clustering enabled (1) or not (0).
Gauge	kube_ovn_cluster_role	A metric with a constant '1' value labeled by server role.
Gauge	kube_ovn_cluster_status	A metric with a constant '1' value labeled by server status.
Gauge	kube_ovn_cluster_term	The current raft term known by this server.
Gauge	kube_ovn_cluster_leader_self	Is this server consider itself a leader (1) or not (0).
Gauge	kube_ovn_cluster_vote_self	Is this server voted itself as a leader (1) or not (0).
Gauge	kube_ovn_cluster_election_timer	The current election timer value.
Gauge	kube_ovn_cluster_log_not_committed	The number of log entries not yet committed by this server.
Gauge	kube_ovn_cluster_log_not_applied	

Type	Metric	Description
		The number of log entries not yet applied by this server.
Gauge	kube_ovn_cluster_log_index_start	The log entry index start value associated with this server.
Gauge	kube_ovn_cluster_log_index_next	The log entry index next value associated with this server.
Gauge	kube_ovn_cluster_inbound_connections_total	The total number of inbound connections to the server.
Gauge	kube_ovn_cluster_outbound_connections_total	The total number of outbound connections from the server.
Gauge	kube_ovn_cluster_inbound_connections_error_total	The total number of failed inbound connections to the server.
Gauge	kube_ovn_cluster_outbound_connections_error_total	The total number of failed outbound connections from the server.

8.11.2 ovs-monitor

`ovsdb` and `vswitchd` status metrics:

Type	Metric	Description
Gauge	ovs_status	OVS Health Status. The values are: health(1), unhealthy(0).
Gauge	ovs_info	This metric provides basic information about OVS. It is always set to 1.
Gauge	failed_req_count	The number of failed requests to OVS stack.
Gauge	log_file_size	The size of a log file associated with an OVS component.
Gauge	db_file_size	The size of a database file associated with an OVS component.
Gauge	datapath	Represents an existing datapath. This metrics is always 1.
Gauge	dp_total	Represents total number of datapaths on the system.
Gauge	dp_if	Represents an existing datapath interface. This metrics is always 1.
Gauge	dp_if_total	Represents the number of ports connected to the datapath.
Gauge	dp_flows_total	The number of flows in a datapath.
Gauge	dp_flows_lookup_hit	The number of incoming packets in a datapath matching existing flows in the datapath.
Gauge	dp_flows_lookup_missed	The number of incoming packets in a datapath not matching any existing flow in the datapath.
Gauge	dp_flows_lookup_lost	The number of incoming packets in a datapath destined for userspace process but subsequently dropped before reaching userspace.
Gauge	dp_masks_hit	The total number of masks visited for matching incoming packets.
Gauge	dp_masks_total	The number of masks in a datapath.
Gauge	dp_masks_hit_ratio	The average number of masks visited per packet. It is the ration between hit and total number of packets processed by a datapath.
Gauge	interface	Represents OVS interface. This is the primary metric for all other interface metrics. This metrics is always 1.
Gauge	interface_admin_state	The administrative state of the physical network link of OVS interface. The values are: down(0), up(1), other(2).
Gauge	interface_link_state	The state of the physical network link of OVS interface. The values are: down(0), up(1), other(2).
Gauge	interface_mac_in_use	The MAC address in use by OVS interface.
Gauge	interface_mtu	The currently configured MTU for OVS interface.
Gauge	interface_of_port	Represents the OpenFlow port ID associated with OVS interface.
Gauge	interface_if_index	Represents the interface index associated with OVS interface.
Gauge	interface_tx_packets	Represents the number of transmitted packets by OVS interface.
Gauge	interface_tx_bytes	Represents the number of transmitted bytes by OVS interface.
Gauge	interface_rx_packets	Represents the number of received packets by OVS interface.
Gauge	interface_rx_bytes	Represents the number of received bytes by OVS interface.
Gauge	interface_rx_crc_err	Represents the number of CRC errors for the packets received by OVS interface.
Gauge	interface_rx_dropped	Represents the number of input packets dropped by OVS interface.
Gauge	interface_rx_errors	

Type	Metric	Description
		Represents the total number of packets with errors received by OVS interface.
Gauge	interface_rx_frame_err	Represents the number of frame alignment errors on the packets received by OVS interface.
Gauge	interface_rx_missed_err	Represents the number of packets with RX missed received by OVS interface.
Gauge	interface_rx_over_err	Represents the number of packets with RX overrun received by OVS interface.
Gauge	interface_tx_dropped	Represents the number of output packets dropped by OVS interface.
Gauge	interface_tx_errors	Represents the total number of transmit errors by OVS interface.
Gauge	interface_collisions	Represents the number of collisions on OVS interface.

8.11.3 kube-ovn-pinger

Network quality related metrics:

Type	Metric	Description
Gauge	pinger_ovs_up	If the ovs on the node is up
Gauge	pinger_ovs_down	If the ovs on the node is down
Gauge	pinger_ovn_controller_up	If the ovn_controller on the node is up
Gauge	pinger_ovn_controller_down	If the ovn_controller on the node is down
Gauge	pinger_inconsistent_port_binding	The number of mismatch port bindings between ovs and ovn-sb
Gauge	pinger_apiserver_healthy	If the apiserver request is healthy on this node
Gauge	pinger_apiserver_unhealthy	If the apiserver request is unhealthy on this node
Histogram	pinger_apiserver_latency_ms	The latency ms histogram the node request apiserver
Gauge	pinger_internal_dns_healthy	If the internal dns request is unhealthy on this node
Gauge	pinger_internal_dns_unhealthy	If the internal dns request is unhealthy on this node
Histogram	pinger_internal_dns_latency_ms	The latency ms histogram the node request internal dns
Gauge	pinger_external_dns_health	If the external dns request is healthy on this node
Gauge	pinger_external_dns_unhealthy	If the external dns request is unhealthy on this node
Histogram	pinger_external_dns_latency_ms	The latency ms histogram the node request external dns
Histogram	pinger_pod_ping_latency_ms	The latency ms histogram for pod peer ping
Gauge	pinger_pod_ping_lost_total	The lost count for pod peer ping
Gauge	pinger_pod_ping_count_total	The total count for pod peer ping
Histogram	pinger_node_ping_latency_ms	The latency ms histogram for pod ping node
Gauge	pinger_node_ping_lost_total	The lost count for pod ping node
Gauge	pinger_node_ping_count_total	The total count for pod ping node
Histogram	pinger_external_ping_latency_ms	The latency ms histogram for pod ping external address
Gauge	pinger_external_lost_total	The lost count for pod ping external address

8.11.4 kube-ovn-controller

`kube-ovn-controller` status metrics:

Type	Metric	Description
Histogram	rest_client_request_latency_seconds	Request latency in seconds. Broken down by verb and URL
Counter	rest_client_requests_total	Number of HTTP requests, partitioned by status code, method, and host
Counter	lists_total	Total number of API lists done by the reflectors
Summary	list_duration_seconds	How long an API list takes to return and decode for the reflectors
Summary	items_per_list	How many items an API list returns to the reflectors
Counter	watches_total	Total number of API watches done by the reflectors
Counter	short_watches_total	Total number of short API watches done by the reflectors
Summary	watch_duration_seconds	How long an API watch takes to return and decode for the reflectors
Summary	items_per_watch	How many items an API watch returns to the reflectors
Gauge	last_resource_version	Last resource version seen for the reflectors
Histogram	ovs_client_request_latency_milliseconds	The latency histogram for ovs request
Gauge	subnet_available_ip_count	The available num of ip address in subnet
Gauge	subnet_used_ip_count	The used num of ip address in subnet

8.11.5 kube-ovn-cni

`kube-ovn-cni` status metrics:

Type	Metric	Description
Histogram	cni_op_latency_seconds	The latency seconds for cni operations
Counter	cni_wait_address_seconds_total	Latency that cni wait controller to assign an address
Counter	cni_wait_connectivity_seconds_total	Latency that cni wait address ready in overlay network
Counter	cni_wait_route_seconds_total	Latency that cni wait controller to add routed annotation to pod
Histogram	rest_client_request_latency_seconds	Request latency in seconds. Broken down by verb and URL
Counter	rest_client_requests_total	Number of HTTP requests, partitioned by status code, method, and host
Counter	lists_total	Total number of API lists done by the reflectors
Summary	list_duration_seconds	How long an API list takes to return and decode for the reflectors
Summary	items_per_list	How many items an API list returns to the reflectors
Counter	watches_total	Total number of API watches done by the reflectors
Counter	short_watches_total	Total number of short API watches done by the reflectors
Summary	watch_duration_seconds	How long an API watch takes to return and decode for the reflectors
Summary	items_per_watch	How many items an API watch returns to the reflectors
Gauge	last_resource_version	Last resource version seen for the reflectors
Histogram	ovs_client_request_latency_milliseconds	The latency histogram for ovs request

PDF Slack Support

July 30, 2025

June 21, 2022

GitHub

8.11.6 Comments

8.12 Kube-OVN API Reference

Based on Kube-OVN v1.12.0, we have compiled a list of CRD resources supported by Kube-OVN, listing the types and meanings of each field of CRD definition for reference.

8.12.1 Generic Condition Definition

Property Name	**Type**	**Description**
type	String	Type of status
status	String	The value of status, in the range of `True`, `False` or `Unknown`
reason	String	The reason for the status change
message	String	The specific message of the status change
lastUpdateTime	Time	The last time the status was updated
lastTransitionTime	Time	Time of last status type change

In each CRD definition, the Condition field in Status follows the above format, so we explain it in advance.

8.12.2 Subnet Definition

Subnet

Property Name	**Type**	**Description**
apiVersion	String	Standard Kubernetes version information field, all custom resources have this value as kubeovn.io/v1
kind	String	Standard Kubernetes resource type field, all instances of this resource will have the value `Subnet`
metadata	ObjectMeta	Standard Kubernetes resource metadata information
spec	SubnetSpec	Subnet specific configuration information
status	SubnetStatus	Subnet status information

SUBNETSPEC

Property Name	Type	Description
default	Bool	Whether this subnet is the default subnet
vpc	String	The vpc which the subnet belongs to, default is ovn-cluster
protocol	String	IP protocol, the value is in the range of `IPv4`, `IPv6` or `Dual`
namespaces	[]String	The list of namespaces bound to this subnet
cidrBlock	String	The range of the subnet, e.g. 10.16.0.0/16
gateway	String	The gateway address of the subnet, the default value is the first available address under the CIDRBlock of the subnet
excludeIps	[]String	The range of addresses under this subnet that will not be automatically assigned
provider	String	Default value is `ovn`. In the case of multiple NICs, the value is `<name>.<namespace>` of the NetworkAttachmentDefinition, Kube-OVN will use this information to find the corresponding subnet resource
gatewayType	String	The gateway type in overlay mode, either `distributed` or `centralized`
gatewayNode	String	The gateway node when the gateway mode is centralized, node names can be comma-separated
natOutgoing	Bool	Whether the outgoing traffic is NAT
externalEgressGateway	String	The address of the external gateway. This parameter and the natOutgoing parameter cannot be set at the same time
policyRoutingPriority	Uint32	Policy route priority. Used to control the forwarding of traffic to the external gateway address after the subnet gateway
policyRoutingTableID	Uint32	The TableID of the local policy routing table, should be different for each subnet to avoid conflicts
private	Bool	Whether the subnet is a private subnet, which denies access to addresses inside the subnet if the subnet is private
allowSubnets	[]String	If the subnet is a private subnet, the set of addresses that are allowed to access the subnet
vlan	String	The name of vlan to which the subnet is bound
vips	[]String	The virtual-ip parameter information for virtual type lsp on the subnet
logicalGateway	Bool	Whether to enable logical gateway
disableGatewayCheck	Bool	Whether to skip the gateway connectivity check when creating a pod
disableInterConnection	Bool	Whether to enable subnet interconnection across clusters
enableDHCP	Bool	Whether to configure dhcp configuration options for lsps belong this subnet
dhcpV4Options	String	The DHCP_Options record associated with lsp dhcpv4_options on the subnet
dhcpV6Options	String	The DHCP_Options record associated with lsp dhcpv6_options on the subnet
enableIPv6RA	Bool	Whether to configure the ipv6_ra_configs parameter for the lrp port of the router connected to the subnet
ipv6RAConfigs	String	

Property Name	Type	Description
		The ipv6_ra_configs parameter configuration for the lrp port of the router connected to the subnet
acls	[]Acl	The acls record associated with the logical-switch of the subnet
u2oInterconnection	Bool	Whether to enable interconnection mode for Overlay/Underlay
enableLb	*Bool	Whether the logical-switch of the subnet is associated with load-balancer records
enableEcmp	Bool	Centralized subnet, whether to enable ECMP routing

Acl

Property Name	Type	Description
direction	String	Restrict the direction of acl, which value is `from-lport` or `to-lport`
priority	Int	Acl priority, in the range 0 to 32767
match	String	Acl rule match expression
action	String	The action of the rule, which value is in the range of `allow-related`, `allow-stateless`, `allow`, `drop`, `reject`

SUBNETSTATUS

Property Name	Type	Description
conditions	[]SubnetCondition	Subnet status change information, refer to the beginning of the document for the definition of Condition
v4AvailableIPs	Float64	Number of available IPv4 IPs
v4availableIPrange	String	The available range of IPv4 addresses on the subnet
v4UsingIPs	Float64	Number of used IPv4 IPs
v4usingIPrange	String	Used IPv4 address ranges on the subnet
v6AvailableIPs	Float64	Number of available IPv6 IPs
v6availableIPrange	String	The available range of IPv6 addresses on the subnet
v6UsingIPs	Float64	Number of used IPv6 IPs
v6usingIPrange	String	Used IPv6 address ranges on the subnet
activateGateway	String	The currently working gateway node in centralized subnet of master-backup mode
dhcpV4OptionsUUID	String	The DHCP_Options record identifier associated with the lsp dhcpv4_options on the subnet
dhcpV6OptionsUUID	String	The DHCP_Options record identifier associated with the lsp dhcpv6_options on the subnet
u2oInterconnectionIP	String	The IP address used for interconnection when Overlay/Underlay interconnection mode is enabled

8.12.3 IP Definition

IP

Property Name	Type	Description
apiVersion	String	Standard Kubernetes version information field, all custom resources are kubeovn.io/v1
kind	String	Standard Kubernetes resource type field, all instances of this resource have the value `IP`
metadata	ObjectMeta	Standard Kubernetes resource metadata information
spec	IPSpec	IP specific configuration information

IPSEPC

Property Name	Type	Description
podName	String	Pod name which assigned with this IP
namespace	String	The name of the namespace where the pod is bound
subnet	String	The subnet which the ip belongs to
attachSubnets	[]String	The name of the other subnets attached to this primary IP (field deprecated)
nodeName	String	The name of the node where the pod is bound
ipAddress	String	IP address, in `v4IP,v6IP` format for dual-stack cases
v4IPAddress	String	IPv4 IP address
v6IPAddress	String	IPv6 IP address
attachIPs	[]String	Other IP addresses attached to this primary IP (field is deprecated)
macAddress	String	The Mac address of the bound pod
attachMacs	[]String	Other Mac addresses attached to this primary IP (field deprecated)
containerID	String	The Container ID corresponding to the bound pod
podType	String	Special workload pod, can be `StatefulSet`, `VirtualMachine` or empty

8.12.4 Underlay configuration

Vlan

Property Name	Type	Description
apiVersion	String	Standard Kubernetes version information field, all instances of this resource will be kubeovn.io/v1
kind	String	Standard Kubernetes resource type field, all instances of this resource will have the value `Vlan`
metadata	ObjectMeta	Standard Kubernetes resource metadata information
spec	VlanSpec	Vlan specific configuration information
status	VlanStatus	Vlan status information

VLANSPEC

Property Name	Type	Description
id	Int	Vlan tag number, in the range of 0~4096
provider	String	The name of the ProviderNetwork to which the vlan is bound

VLANSTATUS

Property Name	Type	Description
subnets	[]String	The list of subnets to which the vlan is bound
conditions	[]VlanCondition	Vlan status change information, refer to the beginning of the document for the definition of Condition

ProviderNetwork

Property Name	Type	Description
apiVersion	String	Standard Kubernetes version information field, all custom resources are kubeovn.io/v1
kind	String	Standard Kubernetes resource type field, all instances of this resource will have the value `ProviderNetwork`
metadata	ObjectMeta	Standard Kubernetes resource metadata information
spec	ProviderNetworkSpec	ProviderNetwork specific configuration information
status	ProviderNetworkStatus	ProviderNetwork status information

PROVIDERNETWORKSPEC

Property Name	Type	Description
defaultInterface	String	The name of the NIC interface used by default for this bridge network
customInterfaces	[]CustomInterface	The special NIC configuration used by this bridge network
excludeNodes	[]String	The names of the nodes that will not be bound to this bridge network
exchangeLinkName	Bool	Whether to exchange the bridge NIC and the corresponding OVS bridge name

CustomInterface

Property Name	Type	Description
interface	String	NIC interface name used for underlay
nodes	[]String	List of nodes using the custom NIC interface

PROVIDERNETWORKSTATUS

Property Name	Type	Description
ready	Bool	Whether the current bridge network is in the ready state
readyNodes	[]String	The name of the node whose bridge network is ready
notReadyNodes	[]String	The name of the node whose bridge network is not ready
vlans	[]String	The name of the vlan to which the bridge network is bound
conditions	[]ProviderNetworkCondition	ProviderNetwork status change information, refer to the beginning of the document for the definition of Condition

8.12.5 Vpc Definition

Vpc

Property Name	Type	Description
apiVersion	String	Standard Kubernetes version information field, all custom resources have this value as kubeovn.io/v1
kind	String	Standard Kubernetes resource type field, all instances of this resource will have the value `Vpc`
metadata	ObjectMeta	Standard Kubernetes resource metadata information
spec	VpcSpec	Vpc specific configuration information
status	VpcStatus	Vpc status information

VPCSPEC

Property Name	Type	Description
namespaces	[]String	List of namespaces bound by Vpc
staticRoutes	[]*StaticRoute	The static route information configured under Vpc
policyRoutes	[]*PolicyRoute	The policy route information configured under Vpc
vpcPeerings	[]*VpcPeering	Vpc interconnection information
enableExternal	Bool	Whether vpc is connected to an external switch
defaultSubnet	String	Name of the subnet that should be used by custom Vpc as the default one

StaticRoute

Property Name	Type	Description
policy	String	Routing policy, takes the value of `policySrc` or `policyDst`
cidr	String	Routing cidr value
nextHopIP	String	The next hop information of the route

PolicyRoute

Property Name	Type	Description
priority	Int32	Priority for policy route
match	String	Match expression for policy route
action	String	Action for policy route, the value is in the range of `allow`, `drop`, `reroute`
nextHopIP	String	The next hop of the policy route, separated by commas in the case of ECMP routing

VpcPeering

Property Name	Type	Description
remoteVpc	String	Name of the interconnected peering vpc
localConnectIP	String	The local ip for vpc used to connect to peer vpc

VPCSTATUS

Property Name	Type	Description
conditions	[]VpcCondition	Vpc status change information, refer to the beginning of the documentation for the definition of Condition
standby	Bool	Whether the vpc creation is complete, the subnet under the vpc needs to wait for the vpc creation to complete other proceeding
default	Bool	Whether it is the default vpc
defaultLogicalSwitch	String	The default subnet under vpc
router	String	The logical-router name for the vpc
tcpLoadBalancer	String	TCP LB information for vpc
udpLoadBalancer	String	UDP LB information for vpc
tcpSessionLoadBalancer	String	TCP Session Hold LB Information for Vpc
udpSessionLoadBalancer	String	UDP session hold LB information for Vpc
subnets	[]String	List of subnets for vpc
vpcPeerings	[]String	List of peer vpcs for vpc interconnection
enableExternal	Bool	Whether the vpc is connected to an external switch

VpcNatGateway

Property Name	Type	Description
apiVersion	String	Standard Kubernetes version information field, all custom resources are kubeovn.io/v1
kind	String	Standard Kubernetes resource type field, all instances of this resource will have the value `VpcNatGateway`
metadata	ObjectMeta	Standard Kubernetes resource metadata information
spec	VpcNatSpec	Vpc gateway specific configuration information

VPCNATSPEC

Property Name	Type	Description
vpc	String	Vpc name which the vpc gateway belongs to
subnet	String	The name of the subnet to which the gateway pod belongs
lanIp	String	The IP address assigned to the gateway pod
selector	[]String	Standard Kubernetes selector match information
tolerations	[]VpcNatToleration	Standard Kubernetes tolerance information

VpcNatToleration

Property Name	Type	Description
key	String	The key information of the taint tolerance
operator	String	Takes the value of `Exists` or `Equal`
value	String	The value information of the taint tolerance
effect	String	The effect of the taint tolerance, takes the value of `NoExecute`, `NoSchedule`, or `PreferNoSchedule`
tolerationSeconds	Int64	The amount of time the pod can continue to run on the node after the taint is added

The meaning of the above tolerance fields can be found in the official Kubernetes documentation Taint and Tolerance.

IptablesEIP

Property Name	Type	Description
apiVersion	String	Standard Kubernetes version information field, all custom resources have this value as kubeovn.io/v1
kind	String	Standard Kubernetes resource type field, all instances of this resource have the value `IptablesEIP`
metadata	ObjectMeta	Standard Kubernetes resource metadata information
spec	IptablesEipSpec	IptablesEIP specific configuration information used by vpc gateway
status	IptablesEipStatus	IptablesEIP status information used by vpc gateway

IPTABLESEIPSPEC

Property Name	Type	Description
v4ip	String	IptablesEIP v4 address
v6ip	String	IptablesEIP v6 address
macAddress	String	The assigned mac address, not actually used
natGwDp	String	Vpc gateway name

IPTABLESEIPSTATUS

Property Name	Type	Description
ready	Bool	Whether IptablesEIP is configured complete
ip	String	The IP address used by IptablesEIP, currently only IPv4 addresses are supported
redo	String	IptablesEIP crd creation or update time
nat	String	The type of IptablesEIP, either `fip`, `snat`, or `dnat`
conditions	[]IptablesEIPCondition	IptablesEIP status change information, refer to the beginning of the documentation for the definition of Condition

IptablesFIPRule

Property Name	Type	Description
apiVersion	String	Standard Kubernetes version information field, all custom resources have this value as kubeovn.io/v1
kind	String	Standard Kubernetes resource type field, all instances of this resource have the value `IptablesFIPRule`
metadata	ObjectMeta	Standard Kubernetes resource metadata information
spec	IptablesFIPRuleSpec	The IptablesFIPRule specific configuration information used by vpc gateway
status	IptablesFIPRuleStatus	IptablesFIPRule status information used by vpc gateway

IPTABLESFIPRULESPEC

Property Name	Type	Description
eip	String	Name of the IptablesEIP used for IptablesFIPRule
internalIp	String	The corresponding internal IP address

IPTABLESFIPRULESTATUS

Property Name	Type	Description
ready	Bool	Whether IptablesFIPRule is configured or not
v4ip	String	The v4 IP address used by IptablesEIP
v6ip	String	The v6 IP address used by IptablesEIP
natGwDp	String	Vpc gateway name
redo	String	IptablesFIPRule crd creation or update time
conditions	[]IptablesFIPRuleCondition	IptablesFIPRule status change information, refer to the beginning of the documentation for the definition of Condition

IptablesSnatRule

Property Name	Type	Description
apiVersion	String	Standard Kubernetes version information field, all custom resources have this value as kubeovn.io/v1
kind	String	Standard Kubernetes resource type field, all instances of this resource have the value `IptablesSnatRule`
metadata	ObjectMeta	Standard Kubernetes resource metadata information
spec	IptablesSnatRuleSpec	The IptablesSnatRule specific configuration information used by the vpc gateway
status	IptablesSnatRuleStatus	IptablesSnatRule status information used by vpc gateway

IPTABLESSNATRULESPEC

Property Name	Type	Description
eip	String	Name of the IptablesEIP used by IptablesSnatRule
internalIp	String	IptablesSnatRule's corresponding internal IP address

IPTABLESSNATRULESTATUS

Property Name	Type	Description
ready	Bool	Whether the configuration is complete
v4ip	String	The v4 IP address used by IptablesSnatRule
v6ip	String	The v6 IP address used by IptablesSnatRule
natGwDp	String	Vpc gateway name
redo	String	IptablesSnatRule crd creation or update time
conditions	[]IptablesSnatRuleCondition	IptablesSnatRule status change information, refer to the beginning of the documentation for the definition of Condition

IptablesDnatRule

Property Name	Type	Description
apiVersion	String	Standard Kubernetes version information field, all custom resources have this value as kubeovn.io/v1
kind	String	Standard Kubernetes resource type field, all instances of this resource have the value `IptablesDnatRule`
metadata	ObjectMeta	Standard Kubernetes resource metadata information
spec	IptablesDnatRuleSpec	The IptablesDnatRule specific configuration information used by vpc gateway
status	IptablesDnatRuleStatus	IptablesDnatRule status information used by vpc gateway

IPTABLESDNATRULESPEC

Property Name	Type	Description
eip	Sting	Name of IptablesEIP used by IptablesDnatRule
externalPort	Sting	External port used by IptablesDnatRule
protocol	Sting	Vpc gateway dnat protocol type
internalIp	Sting	Internal IP address used by IptablesDnatRule
internalPort	Sting	Internal port used by IptablesDnatRule

IPTABLESDNATRULESTATUS

Property Name	Type	Description
ready	Bool	Whether the configuration is complete
v4ip	String	The v4 IP address used by IptablesDnatRule
v6ip	String	The v6 IP address used by IptablesDnatRule
natGwDp	String	Vpc gateway name
redo	String	IptablesDnatRule crd creation or update time
conditions	[]IptablesDnatRuleCondition	IptablesDnatRule Status change information, refer to the beginning of the documentation for the definition of Condition

VpcDns

Property Name	Type	Description
apiVersion	String	Standard Kubernetes version information field, all custom resources have kubeovn.io/v1
kind	String	Standard Kubernetes resource type field, all instances of this resource will have the value `VpcDns`
metadata	ObjectMeta	Standard Kubernetes resource metadata information
spec	VpcDnsSpec	VpcDns specific configuration information
status	VpcDnsStatus	VpcDns status information

VPCDNSSPEC

Property Name	Type	Description
vpc	String	Name of the vpc where VpcDns is located
subnet	String	The subnet name of the address assigned to the VpcDns pod

VPCDNSSTATUS

Property Name	Type	Description
conditions	[]VpcDnsCondition	VpcDns status change information, refer to the beginning of the document for the definition of Condition
active	Bool	Whether VpcDns is in use

For detailed documentation on the use of VpcDns, see Customizing VPC DNS.

SwitchLBRule

Property Name	Type	Description
apiVersion	String	Standard Kubernetes version information field, all custom resources have this value as kubeovn.io/v1
kind	String	Standard Kubernetes resource type field, all instances of this resource will have the value `SwitchLBRule`
metadata	ObjectMeta	Standard Kubernetes resource metadata information
spec	SwitchLBRuleSpec	SwitchLBRule specific configuration information
status	SwitchLBRuleStatus	SwitchLBRule status information

SWITCHLBRULESPEC

Property Name	Type	Description
vip	String	Vip address of SwitchLBRule
namespace	String	SwitchLBRule's namespace
selector	[]String	Standard Kubernetes selector match information
sessionAffinity	String	Standard Kubernetes service sessionAffinity value
ports	[]SlrPort	List of SwitchLBRule ports

For detailed configuration information of SwitchLBRule, you can refer to Customizing VPC Internal Load Balancing health check.

SlrPort

Property Name	Type	Description
name	String	Port name
port	Int32	Port number
targetPort	Int32	Target port of SwitchLBRule
protocol	String	Protocol type

SWITCHLBRULESTATUS

Property Name	Type	Description
conditions	[]SwitchLBRuleCondition	SwitchLBRule status change information, refer to the beginning of the document for the definition of Condition
ports	String	Port information
service	String	Name of the service

8.12.6 Security Group and Vip

SecurityGroup

Property Name	Type	Description
apiVersion	String	Standard Kubernetes version information field, all custom resources are kubeovn.io/v1
kind	String	Standard Kubernetes resource type field, all instances of this resource will have a value of `SecurityGroup`
metadata	ObjectMeta	Standard Kubernetes resource metadata information
spec	SecurityGroupSpec	Security Group specific configuration information
status	SecurityGroupStatus	Security group status information

SECURITYGROUPSPEC

Property Name	Type	Description
ingressRules	[]*SgRule	Inbound security group rules
egressRules	[]*SgRule	Outbound security group rules
allowSameGroupTraffic	Bool	Whether lsps in the same security group can interoperate and whether traffic rules need to be updated

SgRule

Property Name	Type	Description
ipVersion	String	IP version number, `ipv4` or `ipv6`
protocol	String	The value of `icmp`, `tcp`, or `udp`
priority	Int	Acl priority. The value range is 1-200, the smaller the value, the higher the priority.
remoteType	String	The value is either `address` or `securityGroup`
remoteAddress	String	The address of the other side
remoteSecurityGroup	String	The name of security group on the other side
portRangeMin	Int	The starting value of the port range, the minimum value is 1.
portRangeMax	Int	The ending value of the port range, the maximum value is 65535.
policy	String	The value is `allow` or `drop`

SECURITYGROUPSTATUS

Property Name	Type	Description
portGroup	String	The name of the port-group for the security group
allowSameGroupTraffic	Bool	Whether lsps in the same security group can interoperate, and whether the security group traffic rules need to be updated
ingressMd5	String	The MD5 value of the inbound security group rule
egressMd5	String	The MD5 value of the outbound security group rule
ingressLastSyncSuccess	Bool	Whether the last synchronization of the inbound rule was successful
egressLastSyncSuccess	Bool	Whether the last synchronization of the outbound rule was successful

Vip

Property Name	Type	Description
apiVersion	String	Standard Kubernetes version information field, all custom resources are kubeovn.io/v1
kind	String	Standard Kubernetes resource type field, all instances of this resource will have the value `Vip`
metadata	ObjectMeta	Standard Kubernetes resource metadata information
spec	VipSpec	Vip specific configuration information
status	VipStatus	Vip status information

VIPSPEC

Property Name	Type	Description
namespace	String	Vip's namespace
subnet	String	Vip's subnet
type	String	The type of Vip, either `switch_lb_vip`, or empty
v4ip	String	Vip IPv4 ip address
v6ip	String	Vip IPv6 ip address
macAddress	String	Vip mac address
parentV4ip	String	Not currently in use
parentV6ip	String	Not currently in use
parentMac	String	Not currently in use
selector	[]String	Standard Kubernetes selector match information
attachSubnets	[]String	This field is deprecated and no longer used

VIPSTATUS

Property Name	Type	Description
conditions	[]VipCondition	Vip status change information, refer to the beginning of the documentation for the definition of Condition
ready	Bool	Vip is ready or not
v4ip	String	Vip IPv4 ip address, should be the same as the spec field
v6ip	String	Vip IPv6 ip address, should be the same as the spec field
mac	String	The vip mac address, which should be the same as the spec field
pv4ip	String	Not currently used
pv6ip	String	Not currently used
pmac	String	Not currently used

OvnEip

Property Name	Type	Description
apiVersion	String	Standard Kubernetes version information field, all custom resources are kubeovn.io/v1
kind	String	Standard Kubernetes resource type field, all instances of this resource will have the value `OvnEip`
metadata	ObjectMeta	Standard Kubernetes resource metadata information
spec	OvnEipSpec	OvnEip specific configuration information for default vpc
status	OvnEipStatus	OvnEip status information for default vpc

OVNEIPSPEC

Property Name	Type	Description
externalSubnet	String	OvnEip's subnet name
v4Ip	String	OvnEip IPv4 address
v6Ip	String	OvnEip IPv6 address
macAddress	String	OvnEip Mac address
type	String	OvnEip use type, the value can be `lrp`, `lsp` or `nat`

OVNEIPSTATUS

Property Name	Type	Description
conditions	[]OvnEipCondition	OvnEip status change information, refer to the beginning of the documentation for the definition of Condition
type	String	OvnEip use type, the value can be `lrp`, `lsp` or `nat`
nat	String	dnat snat fip
v4Ip	String	The IPv4 ip address used by ovnEip
v6Ip	String	The IPv4 ip address used by ovnEip
macAddress	String	Mac address used by ovnEip

OvnFip

Property Name	Type	Description
apiVersion	String	Standard Kubernetes version information field, all custom resources are kubeovn.io/v1
kind	String	Standard Kubernetes resource type field, all instances of this resource will have the value `OvnFip`
metadata	ObjectMeta	Standard Kubernetes resource metadata information
spec	OvnFipSpec	OvnFip specific configuration information in default vpc
status	OvnFipStatus	OvnFip status information in default vpc

OVNFIPSPEC

Property Name	Type	Description
ovnEip	String	Name of the bound ovnEip
ipType	String	vip crd or ip crd ("" means ip crd)
ipName	String	The IP crd name corresponding to the bound Pod
vpc	String	The vpc crd name corresponding to the bound Pod
V4Ip	String	The IPv4 ip addresss corresponding to vip or the bound Pod

OVNFIPSTATUS

Property Name	Type	Description
ready	Bool	OvnFip is ready or not
v4Eip	String	Name of the ovnEip to which ovnFip is bound
v4Ip	String	The ovnEip address currently in use
vpc	String	The name of the vpc where ovnFip is located
conditions	[]OvnFipCondition	OvnFip status change information, refer to the beginning of the document for the definition of Condition

OvnSnatRule

Property Name	Type	Description
apiVersion	String	Standard Kubernetes version information field, all custom resources have kubeovn.io/v1
kind	String	Standard Kubernetes resource type field, all instances of this resource will have the value `OvnSnatRule`
metadata	ObjectMeta	Standard Kubernetes resource metadata information
spec	OvnSnatRuleSpec	OvnSnatRule specific configuration information in default vpc
status	OvnSnatRuleStatus	OvnSnatRule status information in default vpc

OVNSNATRULESPEC

Property Name	Type	Description
ovnEip	String	Name of the ovnEip to which ovnSnatRule is bound
vpcSubnet	String	The name of the subnet of the vpc configured by ovnSnatRule
vpc	String	The vpc crd name corresponding to the ovnSnatRule bound Pod
ipName	String	The IP crd name corresponding to the ovnSnatRule bound Pod
v4IpCidr	String	The IPv4 cidr of the vpc subnet

OVNSNATRULESTATUS

Property Name	Type	Description
ready	Bool	OvnSnatRule is ready or not
v4Eip	String	The ovnEip address to which ovnSnatRule is bound
v4IpCidr	String	The cidr address used to configure snat in the logical-router
vpc	String	The name of the vpc where ovnSnatRule is located
conditions	[]OvnSnatRuleCondition	OvnSnatRule status change information, refer to the beginning of the document for the definition of Condition

PDF Slack Support

June 24, 2025

February 16, 2023

GitHub +3

8.12.7 Comments

8.13 Annotation Usage

Kube-OVN uses a large number of Pod and Node Annotations for configuring functionality and transferring information. Users can refer to this document to understand the usage of each Annotation, to better troubleshooting and information retrieval.

Note: Some Annotations may change as the code is adjusted.

8.13.1 Pod Annotation

Key	Value	Description
ovn.kubernetes.io/allocated	`true` or `false`	If the Pod primary interface has already been allocated an address
ovn.kubernetes.io/routed	`true` or `false`	If the Pod primary interface has already been allocated a route
ovn.kubernetes.io/mac_address	String	MAC address allocated to Pod primary interface, when creating a Pod, you can set a fixed MAC address by this Annotation
ovn.kubernetes.io/ip_address	String	IP address allocated to Pod primary interface, when creating a Pod, you can set a fixed IP address by this Annotation
ovn.kubernetes.io/cidr	String	Subnet CIDR that the Pod primary interface belongs to
ovn.kubernetes.io/gateway	String	Subnet Gateway address that the Pod primary interface belongs to
ovn.kubernetes.io/ip_pool	IP list, separated by comma	Pod primary interface will choose address from this list, used for workload fix address
ovn.kubernetes.io/bgp	`true` , `cluster` , `local`	Enable Pod address BGP advertisement
ovn.kubernetes.io/snat	String	SNAT address for accessing external address
ovn.kubernetes.io/eip	String	EIP address that Pod accesses external clusters and is accessed from external.
ovn.kubernetes.io/vip	String	VIP allocated to Pod primary interface
ovn.kubernetes.io/virtualmachine	String	The VirtualMachineInstance that the Pod primary interface belongs to
ovn.kubernetes.io/logical_router	String	The VPC that the Pod primary interface belongs to
ovn.kubernetes.io/layer2_forward	`true` or `false`	Enable add `unknown` address to Pod primary interface in OVN NorthboundDB LSP
ovn.kubernetes.io/port_security	`true` or `false`	Enable Pod primary interface port security
ovn.kubernetes.io/logical_switch	String	The Subnet that the Pod primary interface belongs to
ovn.kubernetes.io/vlan_id	Int	The VlanID that the Pod primary interface belongs to
ovn.kubernetes.io/ingress_rate	Int	Pod primary interface ingress rate limit, measured in Mbits/s
ovn.kubernetes.io/egress_rate	Int	Pod primary interface egress rate limit, measured in Mbits/s
ovn.kubernetes.io/security_groups	String list, separated by comma	The SecurityGroup that the Pod primary interface belongs to
ovn.kubernetes.io/allow_live_migration	`true` or `false`	Allow live migration for Pod primary interface, used by KubeVirt
ovn.kubernetes.io/default_route	`true` or `false`	Set the default route to the Pod primary interface.
ovn.kubernetes.io/provider_network	String	The ProviderNetwork that the Pod primary interface belongs to
ovn.kubernetes.io/mirror	`true` or `false`	Enable Pod primary interface traffic mirror

Key	Value	Description
ovn.kubernetes.io/latency	Int	The delay injected to the Pod primary interface card, measured in milliseconds
ovn.kubernetes.io/limit	Int	Maximum number of packets that the qdisc queue of the primary interface of the Pod
ovn.kubernetes.io/loss	Float	The probability of packet loss on the Pod primary interface
ovn.kubernetes.io/jitter	Int	The jitter of packet latency on the Pod primary interface, measured in milliseconds

8.13.2 Node Annotation

Key	Value	Description
ovn.kubernetes.io/allocated	`true` or `false`	If the `ovn0` interface has already been allocated an address
ovn.kubernetes.io/ip_address	String	IP address allocated to `ovn0` interface
ovn.kubernetes.io/mac_address	String	MAC address allocated to `ovn0` interface
ovn.kubernetes.io/cidr	String	Subnet CIDR that the node `ovn0` interface belongs to
ovn.kubernetes.io/gateway	String	Subnet gateway that the node `ovn0` interface belongs to
ovn.kubernetes.io/chassis	String	The Chassis ID in OVN-SouthBoundDB that the node belongs to
ovn.kubernetes.io/port_name	String	The LSP name in OVN-NorthboundDB that the node `ovn0` interface belongs to
ovn.kubernetes.io/logical_switch	String	Subnet that the node `ovn0` interface belongs to
ovn.kubernetes.io/tunnel_interface	String	Network interface used for tunnel encapsulation

8.13.3 Namespace Annotation

Key	Value	Description
ovn.kubernetes.io/cidr	CIDR list, separated by comma	The CIDRs of subnets bound by this Namespace
ovn.kubernetes.io/exclude_ips	excludeIPs list, separated by semicolon	The excludeIPs of subnets bound by this Namespace

8.13.4 Subnet Annotation

Key	Value	Description
ovn.kubernetes.io/bgp	`true`, `cluster`, `local`	Enable Subnet address BGP advertisement

8.13.5 Service Annotation

Key	Value	Description
ovn.kubernetes.io/bgp	`true` or `false`	Enable Service address BGP advertisement
ovn.kubernetes.io/switch_lb_vip	String	Additional VIP addresses assigned to Service in Kube-OVN.
ovn.kubernetes.io/vpc	String	The VPC that the Service belongs to

8.13.6 Networkpolicy Annotation

Key	Value	Description
ovn.kubernetes.io/enable_log	`true` or `false`	Enable NetworkPolicy log
ovn.kubernetes.io/log_acl_actions	One or more combinations of "allow,drop,pass"	Print ACL logs that match ACL action

PDF Slack Support

July 30, 2025

June 12, 2024

GitHub

8.13.7 Comments

8.14 Document Specification

In order to ensure a consistent document style, please follow the following style guidelines when submitting documents.

8.14.1 Punctuation

All punctuation in the text content in Chinese documents should use Chinese format punctuation, and all text content in English documents should use English punctuation.

Bad	Good
Here is a one-click installation script that can help you quickly install a highly available, production-ready container network.	Here is a one-click installation script that can help you quickly install a highly available, production-ready container network.

English numbers and Chinese characters should be separated by spaces.

Bad	Good
Kube-OVN provides a one-click installation script to install version 1.10 of Kube-OVN.	Kube-OVN provides a one-click installation script to install version 1.10 of Kube-OVN.

Example content should start with `：`, other sentences should end with `.` End.

Bad	Good
Please confirm that the environment configuration is correct before installation Download the installation script using the command below. `wget 127.0.0.1`	Please confirm that the environment configuration is correct before installation. Download the installation script using the following command: `wget 127.0.0.1`

8.14.2 Code Block

yaml code blocks need to be identified as yaml.

Bad	Good
\`\`\`\` apiVersion: kubeovn.io/v1 kind: Subnet metadata: name: attach-subnet \`\`\`\`	\`\`\`\`yaml apiVersion: kubeovn.io/v1 kind: Subnet metadata: name: attach-subnet \`\`\`\`

Command-line manipulation example code blocks need to be identified as bash.

Bad	Good
\`\`\`\` wget 127.0.0.1 \`\`\`\`	\`\`\`\`bash wget 127.0.0.1 \`\`\`\`

If the command line operation example contains output content, the executed command needs to start with `#` to distinguish input from output.

Bad

```
oilbeater@macdeMac-3 ~ ping 114.114.114.114 -c 3
PING 114.114.114.114 (114.114.114.114): 56 data bytes
64 bytes from 114.114.114.114: icmp_seq=0 ttl=83 time=10.429 ms
64 bytes from 114.114.114.114: icmp_seq=1 ttl=79 time=11.360 ms
64 bytes from 114.114.114.114: icmp_seq=2 ttl=76 time=10.794 ms

--- 114.114.114.114 ping statistics ---
3 packets transmitted, 3 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 10.429/10.861/11.360/0.383 ms
```

Good

```
# ping 114.114.114.114 -c 3
PING 114.114.114.114 (114.114.114.114): 56 data bytes
64 bytes from 114.114.114.114: icmp_seq=0 ttl=83 time=10.429 ms
64 bytes from 114.114.114.114: icmp_seq=1 ttl=79 time=11.360 ms
64 bytes from 114.114.114.114: icmp_seq=2 ttl=76 time=10.794 ms

--- 114.114.114.114 ping statistics ---
3 packets transmitted, 3 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 10.429/10.861/11.360/0.383 ms
```

If the command line operation example only contains execution commands and no output results, multiple commands do not need to start with `#`.

Bad

```
# mv /etc/origin/ovn/ovnnb_db.db /tmp
# mv /etc/origin/ovn/ovnsb_db.db /tmp
```

Good

```
mv /etc/origin/ovn/ovnnb_db.db /tmp
mv /etc/origin/ovn/ovnsb_db.db /tmp
```

8.14.3 Link

Links in the site use the corresponding `md` file path.

Bad

```
Please refer to [Preparation](http://kubeovn.github.io/prepare) before
installation.
```

Good

```
Please refer to [Preparation](./prepare.md) before
installation.
```

Bad

```
If you have any questions, please refer to [Kubernetes
Documentation](http://kubernetes.io).
```

Good

```
If you have any questions, please refer to [Kubernetes Documentation](http://
kubernetes.io){: target="_blank" }.
```

8.14.4 Empty Line

Different logical blocks, such as title and text, text and code, text and number need to be separated by blank lines.

Bad

~~~
Download the script below to install it:
```bash
wget 127.0.0.1
```
~~~

Good

~~~
Download the script below to install it:

```bash
wget 127.0.0.1
```
~~~

Separate logical blocks with only *one* blank line.

Bad	Good

`Download the script below to install it:`
 	
`
```
bash`
`wget 127.0.0.1`
`
```
 `
	`Download the script below to install it:`
	
	`
	```
	bash`
	`wget 127.0.0.1`
	`
	```
	 `

PDF **Slack** **Support**

August 23, 2023

July 19, 2022

GitHub

8.14.5 Comments

9. Contact US

PDF Slack Support

June 30, 2022

June 30, 2022

GitHub

9.1 Comments