Kunpeng BoostKit for Virtualization

User Guides

Issue	06
Date	2021-03-23

HUAWEI TECHNOLOGIES CO., LTD.

Trademarks and Permissions


and other Huawei trademarks are trademarks of Huawei Technologies Co., Ltd.
All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The purchased products, services and features are stipulated by the contract made between Huawei and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

Contents

1 Large-Scale Container Networking User Guide

1.1 Introduction

1.1.1 Container Network Overview

In container-based microservice service scenarios, one of the keys to deploying containers on the cloud is to manage the network between container clusters so that containers within a node or between physical nodes can communicate with each other.

Currently, there are two standard proposals for configuring Linux container network interfaces in the industry: Container Network Model (CNM) and Container Network Interface (CNI). This document describes how to verify large scale cluster deployment based on the container orchestration engine Kubernetes and network plane components Open vSwitch and Calico that are widely used in the industry. You can determine the most suitable network plane components based on the actual scale, function requirements, and performance requirements of your services.

For details about related commands and parameters, see the following official documents:

Open vSwitch official documentation: **https://docs.openvswitch.org/en/latest/**

Calico official documentation: **https://docs.projectcalico.org/about/about-calico**

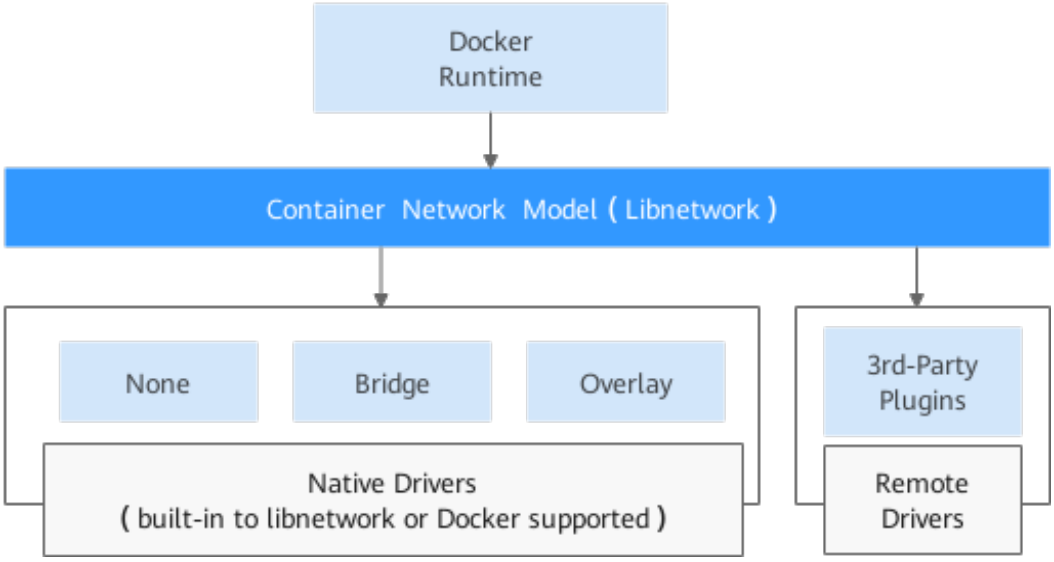
1.1.2 Concepts

Container Network Model (CNM)

CNM is proposed by Docker and adopted by the Libnetwork project as the network model standard. It is used by common open-source network components such as Kuryr, Open Virtual Networking, Calico, and Weave for container network interconnection.

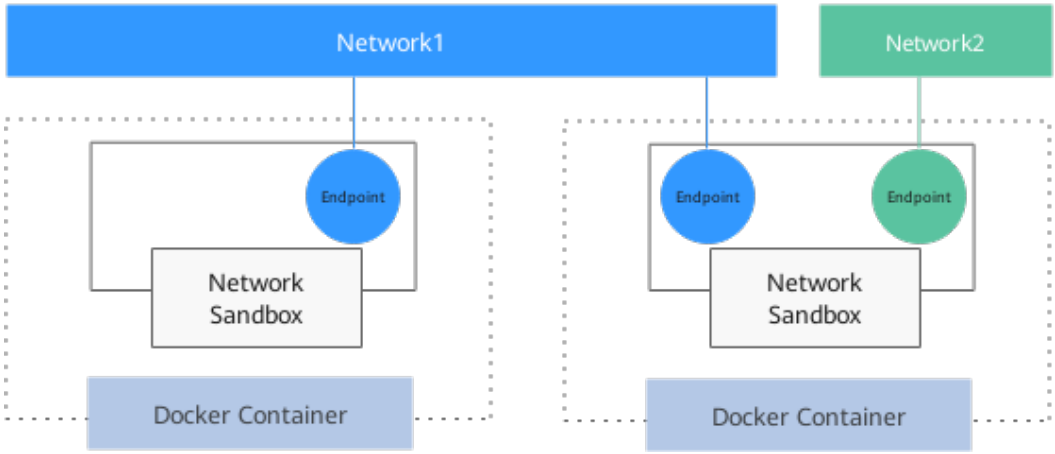
The CNM implementation Libnetwork provides interfaces between Docker daemons and network driver programs. The network controller is responsible for matching drivers with networks, and each driver is responsible for managing the network of the driver and provides services such as IP Address Management (IPAM) for the network. Drivers of the CNM can be native drivers (for built-in Libnetwork or network models supported by Docker) or third-party plugin drivers. The native drivers include None, Bridge, Overlay, and MACvlan. Third-party drivers provide more functions. In addition, the scope of a CNM driver can be defined as either a local scope (single-host mode) or a global scope (multi-host mode).

Figure 1-1 CNM drivers



Containers are connected through a series of network endpoints, as shown in **Figure 1-2**. A typical network interface exists in the form of a veth pair. One end of the veth pair is in the network sandbox of the container, and the other end is in the specified network. One network endpoint is added to only one network plane. Multiple network endpoints can exist in the network sandbox of a container.

Figure 1-2 CNM interfaces



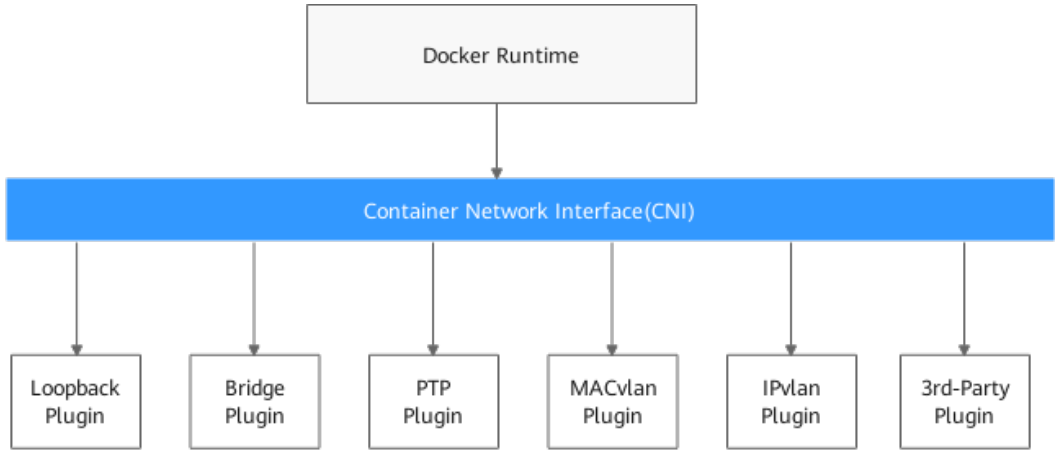
Container Network Interface (CNI)

Container Network Interface (CNI) is proposed by CoreOS and adopted by Apache Mesos, Cloud Foundry, Kubernetes, Kurma, and rkt as the network model standard. CNI is used by common open-source network components such as Contiv Networking, Calico, and Weave for container network interconnection.

As shown in **Figure 1-3**, CNI is implemented based on a simplest standard, enabling network development engineers to implement protocol communication between containers and network plugins in a simple way.

Multiple network plugins can run in a container so that the container can connect to different network planes driven by different plugins. The network is defined in a JSON configuration file and is instantiated as a new namespace when CNI plugins are called.

Figure 1-3 CNI drivers



Kubernetes

Kubernetes is an open-source container orchestration engine provided by Google. It supports automatic deployment, large-scale scalability, and containerized application management. Kubernetes classifies network scenarios into the following types: communication between containers in the same pod,

communication between pods, communication between pods and services, and communication between systems outside the cluster and services.

Pod and service resource objects use their own dedicated networks. The pod network is implemented through Kubernetes network plugin configuration (CNI model). The service network is specified by the Kubernetes cluster. The overall network model is implemented using external plugins. Therefore, you need to plan the network model and network deployment before deploying the Kubernetes cluster.

Open vSwitch

Open vSwitch (OVS) is a multi-layer switch software based on the Apache 2.0 open-source protocol. It aims to build a production environment quality switching platform that supports standard management interfaces, forward function interfaces, programmable plug-ins, and management control. Open Virtual Network (OVN) is a native virtualized network solution provided by OVS. It uses existing OVS functions to implement large-scale and high-quality cluster management.

Calico

Calico is an open-source network and network solution component developed by Tigera based on the Apache 2.0 open-source protocol. It can be used for containers, VMs, and native host machines. This component supports multiple platforms, including Kubernetes, OpenShift, Docker EE, OpenStack, and bare metal services. The Calico project aims to combine flexible network functions with security policy enforcement to provide solutions with native Linux kernel performance and cloud native scalability.

1.1.3 Application Scenarios

This document describes component selection and tuning for network plane establishment and deployment, and is applicable to service deployment such as Kubernetes clusters using container orchestration engines. This document also describes the network establishment and simulation tests of common components on the user network plane when the cluster is large (more than 100 physical nodes or Pod objects).

1.1.4 Implementation Principle

Open vSwitch

The OVS software architecture consists of the kernel-mode datapath and user-mode vswitchd and ovsdb, as shown in **Figure 1-4**.

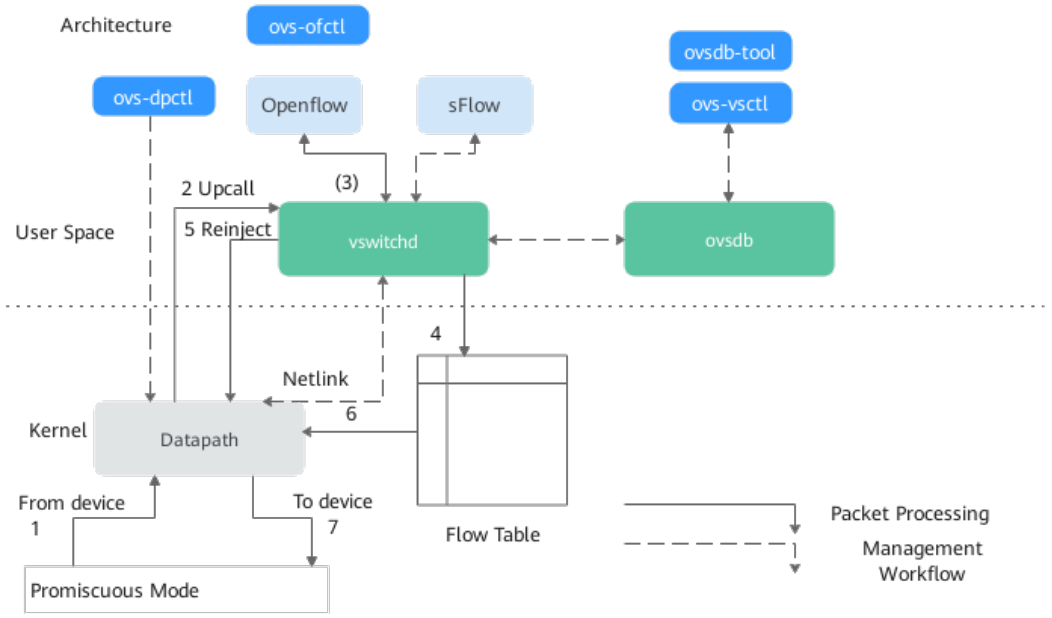
datapath is a kernel module responsible for data exchange. It reads data from the network port, quickly matches flow entries in the FlowTable, and directly forwards the data that is successfully matched or sends the data that fails to be matched to the vswitchd process for processing. The hook function is registered during OVS initialization to make the kernel module take over the packet processing on the port.

vswitchd is a daemon process for OVS management and control. It saves configuration information to OVSDB through the Unix socket and interacts with the kernel module through Netlink.

ovsdb is an OVS database that stores OVS configuration information.

In addition, the OVS's release package contains a series of management tools, such as ovs-vsctl, ovs-dpctl, ovs-ofctl, ovs-appctl and ovs-docker, facilitating OVS's configuration and use.

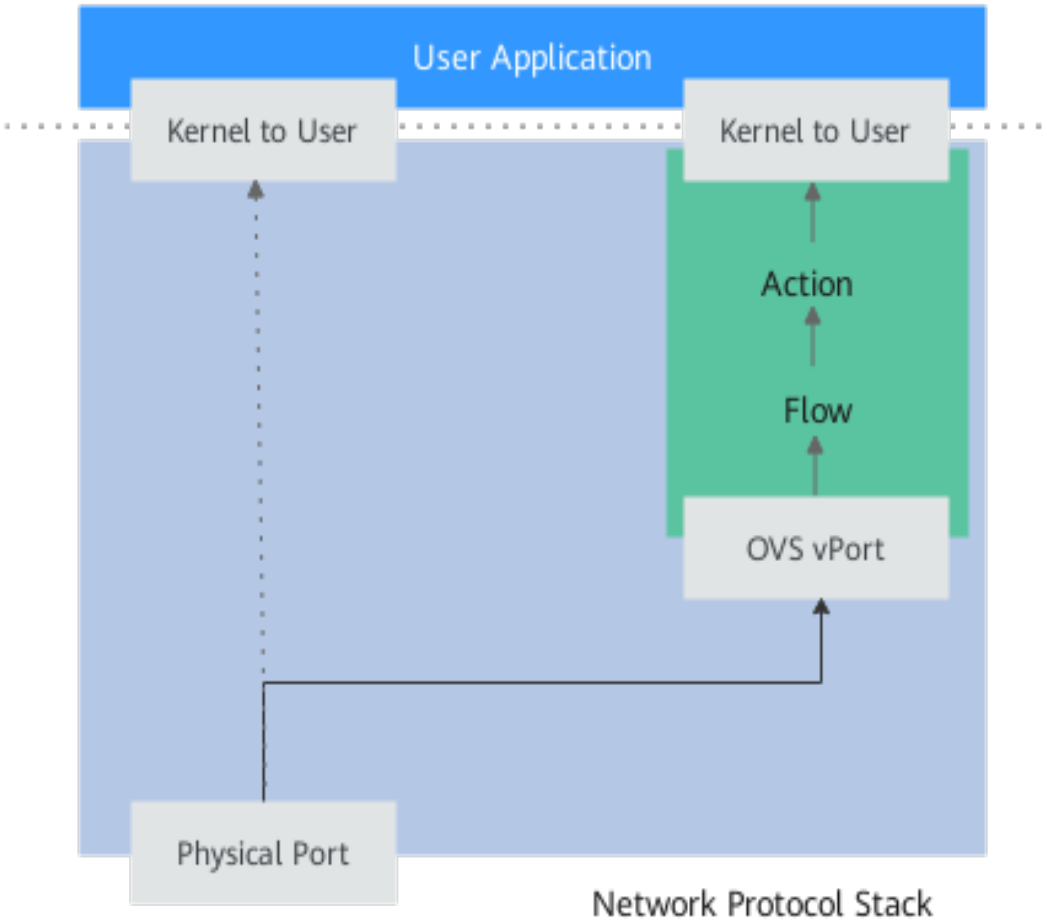
Figure 1-4 Open vSwitch software architecture



In **Figure 1-5**, dotted lines show the direction of data packets in the Linux network protocol before the OVS takes over network data. After receiving the data packets, physical NIC ports parse the packets layer by layer using the kernel protocol stack, exit the kernel mode, and transmits the data to the kernel mode.

After the OVS creates a bridge and binds the physical NIC, the data flow is received from the port of the physical NIC, enters the OVS through the vPort of the OVS in kernel mode, and matches the flow table based on the key value of the data packets. If the matching is successful, the subsequent flow table action is executed. If the operation fails, upcall is performed and the packets are processed by vswitchd.

Figure 1-5 Principles of OVS



Calico

Calico consists of Felix, Confd, and BIRD, as shown in **Figure 1-6**.

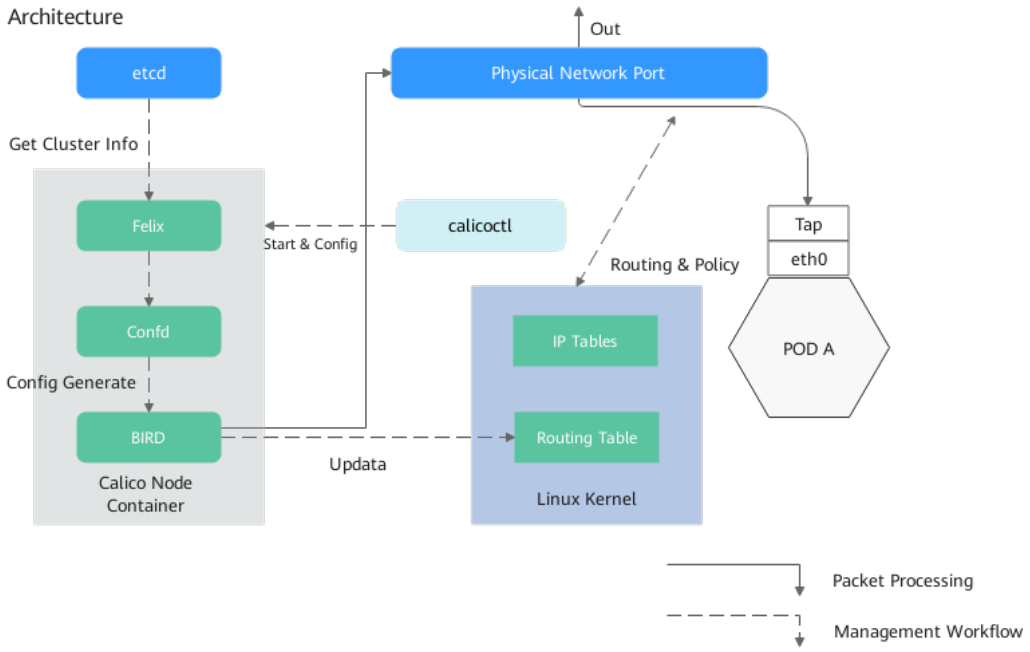
Felix is a daemon that runs Calico nodes and functions as the daemon of the endpoint of each node. It manages pod information on the current host, exchanges cluster pod information with the etcd service, and combines routing information and ACL policies.

Confd stores Calico configuration information generated by etcd and is provided for the BIRD layer.

BIRD (Bird Internet Routing Daemon) is a core component. BIRD in Calico refers to BIRD client and BIRD route reflector. BIRD proactively reads routing information configured by Felix on the local host and distributes routes in the data center through the Border Gateway Protocol (BGP).

In addition, the etcd component is a dependent component of Calico. You need to deploy the etcd service in the cluster in advance or reuse the etcd component of Kubernetes. Calico also provides the calicoctl management tool, which is used to confirm and configure the status of the Calico node.

Figure 1-6 Calico software architecture



1.1.5 Access and Usage Modes

You can use either of the following methods to access the OVS and Calico components:

1. Using built-in management tools such as ovs-vsctl (for OVS) and calicoctl (for Calico).
2. Using Kubernetes management tools such as kubectl to indirectly access, manage, and configure network components.

1.2 Component Selection Guide

1.2.1 Selection Procedure

Step 1 Determine whether the system requires L2 or L3 networking.

- If L2 networking is required, use OVS.
- If L3 networking is required, use Calico.
- If L2 and L3 networking are not required, go to **Step 2**.

Step 2 Determine whether multiple network planes are required for containers. (For example, multiple network components are used at the same time.)

- If yes, use Calico.
- If no, go to **Step 3**.

Step 3 Determine whether low computing resource usage of network components is required.

- If yes, use Calico.

- If no, go to **Step 4**.

Step 4 Determine whether multiple tenants use the same CIDR network.

- If yes, use OVS.
- If no, go to **Step 5**.

Step 5 Determine whether support for native VLAN configuration is required.

- If yes, use OVS.
- If no, go to **Step 6**.

Step 6 Determine whether support for advanced functions such as SDN integration, two-way rate limiting, and DPDK is required.

- If yes, use OVS.
- If no, go to **Step 7**.

Step 7 Determine whether hardware network isolation is required.

- If yes, use OVS.
- If no, go to **Step 8**.

Step 8 Determine whether the network bandwidth needs to be close to that of the host machine?

- If yes, use OVS.
- If no, go to **Step 9**.

Step 9 Determine whether the system is sensitive to latency in large-scale deployment?

- If yes, use Calico.
- If no, go to **Step 10**.

Step 10 Determine whether fixed pod IP addresses are required.

- If yes, use OVS.
- If no, no further action is required. You can select a component based on its usability and your familiarity with the component.

----End

1.2.2 Component Comparison

Both the OVS and Calico components support the ARM64 architecture, but they are different in terms of network model implementation. **Table 1-1** describes the differences of the two components. You can select a network component based on the following table.

Table 1-1 Component comparison

Component	Open vSwitch	Calico
Basic Network Model	L2 (Underlay/Overlay)	L3 BGP (Overlay)
Network Configuration Support	GRE/VxLAN/VLAN	BGP/IPIP/VxLAN

Ecosystem Maturity	Highly mature. Additional plug-ins, such as k-vswitch and kube-ovn, are required for integration with Kubernetes.	Mature. Works well with container orchestration engines such as Kubernetes.
Usability	Medium	Simplicity
VLAN Support	Supported by the native system.	Supported and implemented based on L3 routing.
Network Performance	Excellent There is a small amount of performance loss in GRE, VxLAN encapsulation and decapsulation, and flow table matching.	Excellent There is a small amount of performance loss in IPIP encapsulation and decapsulation.
Compute Resource	High resource usage when the network pressure is high.	Low. L3 routing direct connection, kernel routing table, and IP table are used. The computing resource overhead is low.
Accessing the Cluster from Outside	Direct connection based on L2 routing Cross-VLAN communication requires routing support.	Direct connection based on L3 routing
Accessing Systems Out of the Cluster	Direct connection based on L2 routing Cross-VLAN communication requires routing support.	Direct connection based on L3 routing
Number of Nodes in a Cluster	Unlimited	Unlimited
Single-cluster IP address space	Unlimited	Unlimited. The default value is 65535.
Single-node IP address space	Unlimited (single-node multi-VLAN)	Unlimited
Assigned Pod IP Address	Supported	Supported
Fixed Pod IP Address	Supported	Not supported. Customization is required.
Support for Multiple Network Planes	No supported version is available in the community.	Supported by the open-source community.

Network Isolation	Two-level isolation based on VLANs and network policies.	iptables software isolation.
Advanced Function Ecosystem Support	Strong (SDN integration, rate limiting, and other features)	Medium
Requirements	None	If more than 100 nodes are deployed, you are advised to change the Full Mesh network mode to Route-Reflector (RR) and configure one or two RR nodes for every 100 nodes.

1.3 Quick Start

1.3.1 Installation Guide

To manage and use large-scale container network planes, you need to select proper network plane components and configurations based on the planned physical cluster and container service scale and actual service requirements on network functions, bandwidth latency, computing resources, and management customization.

This chapter describes how to select and operate network components.

For details about the installation, see the following documents:

Open vSwitch Installation Guide (CentOS 7.6)

Calico Installation Guide (CentOS 7.6)

1.3.2 Typical Network Topology

This section describes the typical topology when the OVS and Calico components are used as network planes in a container cluster.

1.3.2.1 Typical OVS Network Structure

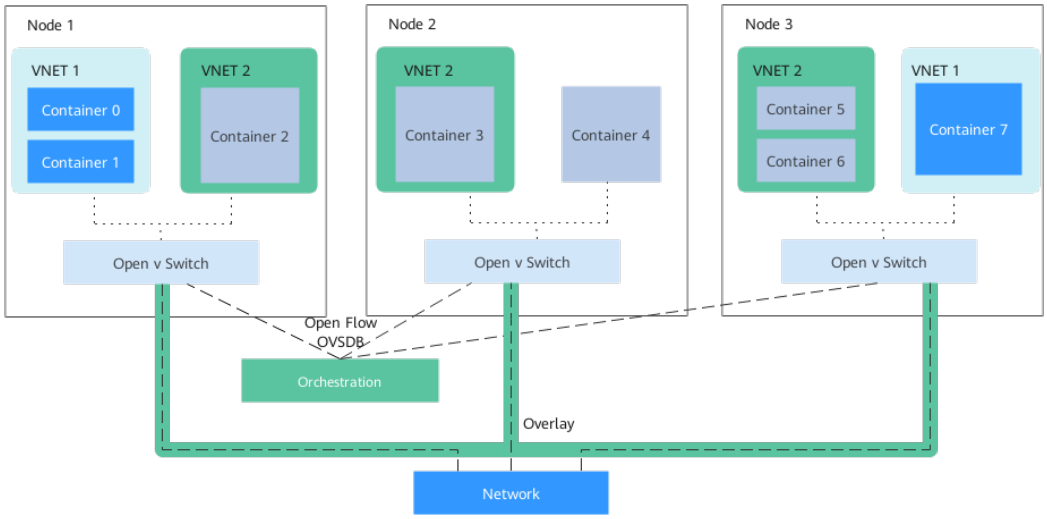
The OVS uses a virtual bridge to take over data of physical ports and performs flow table configuration and bridge configuration to implement end-to-end data switching, as shown in **Figure 1-4** and **Figure 1-5**.

Figure 1-7 shows the principles of the OVS cluster network topology:

1. Physical nodes in a cluster can communicate with each other. The OVS uses the VxLAN or GRE overlay mode to implement network plane communication between nodes.
2. In a physical cluster node, the OVS uses a virtual bridge to manage the container port network and its VLAN information.

3. Between physical cluster nodes, different OVS components use cluster orchestration tools such as OVN to orchestrate networks and configure routing and communication.

Figure 1-7 Typical OVS network topology



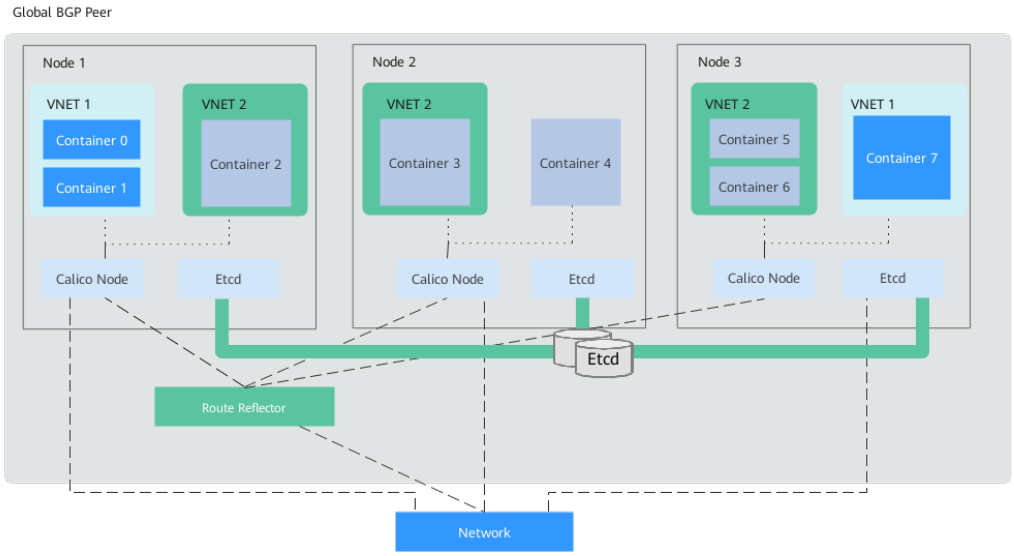
1.3.2.2 Typical Calico Network Structure

Calico uses IP Tunnel and VxLAN to encapsulate and decapsulate data, defines ACLs, plans network policies using iptables, and uses the kernel routing table to implement end-to-end data exchange.

Figure 1-8 shows the principles of the Calico cluster network topology:

1. Physical nodes in a cluster can communicate with each other. Calico uses the BMS network, IPIP, or VxLAN host network overlay mode to implement network plane communication between nodes.
2. In a physical cluster node, Calico maps the network to the container in veth pair mode and manages the container port network information through the routing table and iptables.
3. Between physical cluster nodes, Calico components exchange cluster topology information through the etcd cluster and obtain the routing information of container services in the cluster through BGP, implementing cross-node container network interconnection.

Figure 1-8 Typical Calico network topology



1.3.3 Networking Operation Practice

This section describes how to deploy and verify network planes using Kubernetes cluster, OVS components, and Calico components as an example.

1.3.3.1 OVS Component Networking Operations

Prerequisites

1. The Docker and Kubernetes components (kubeadm, kubectl, and kubelet) have been installed on the node to be deployed.
2. The OVS component has been installed on the node to be deployed.
3. The node to be deployed can properly pull Docker images.

Procedure

Step 1 Start the OVS service on all nodes to be deployed.

NOTE

This section uses the default installation path **/usr** as an example. If the OVS configuration has been changed, you need to change the commands accordingly. For details, see **Open vSwitch Installation Guide (CentOS 7.6)**.

```
export PATH=$PATH:/usr/share/openvswitch/scripts
ovs-ctl start
```

Figure 1-9 shows the startup process. After the startup is complete, check the OVS virtual bridge version information.

Figure 1-9 OVS startup process

```
[                    RPMS]# ovs-ctl start
/etc/openvswitch/conf.db does not exist ... (warning).
Creating empty database /etc/openvswitch/conf.db           [  OK  ]
Starting ovsdb-server                                      [  OK  ]
system ID not configured, please use --system-id ... failed!
Configuring Open vSwitch system IDs                        [  OK  ]
Inserting openvswitch module                               [  OK  ]
Starting ovs-vswitchd                                      [  OK  ]
Enabling remote OVSDB managers                             [  OK  ]
[                    RPMS]# ovs-vsctl show
e85bad1f-cf13-4834-b35e-cca17990b0c6
    ovs_version: "2.12.0"
```

Step 2 Set the Kubernetes master node during initialization.

NOTE

In this section, the **10.244.0.0/16** network segment is used as the network driver CIDR and the default gateway is used as the network broadcast address. If you need to specify another network segment, modify the command accordingly.

```
kubeadm init --pod-network-cidr=10.244.0.0/16
mkdir -p $HOME/.kube
cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
chown $(id -u):$(id -g) $HOME/.kube/config
```

After the initialization is complete, information in **Figure 1-10** and **Figure 1-11** are displayed. Check whether the pods information of the Kubernetes cluster is normal and whether the node status is **NotReady**. Back up the **kubeadm join** command in the output for future use. Then, deploy the network plane.

Figure 1-10 Successful initialization of the active Kubernetes node

```
Your Kubernetes control-plane has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

  mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config

You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
  https://kubernetes.io/docs/concepts/cluster-administration/addons/

Then you can join any number of worker nodes by running the following on each as root:

kubeadm join [                  ]:6443 --token [                    ] \
    --discovery-token-ca-cert-hash sha256:[                                                                ]
```

Figure 1-11 Initialization status of the active Kubernetes node

```
[                    RPMS]# kubectl get pods --all-namespaces -o wide
NAMESPACE     NAME                                      READY   STATUS    RESTARTS   AGE     IP              NODE             NOMINATED NODE   READINESS GATES
kube-system   coredns-5c98db65d4-9qnkm                  0/1     Pending   0          2m34s   <none>          <none>           <none>           <none>
kube-system   coredns-5c98db65d4-xlr5l                  0/1     Pending   0          2m34s   <none>          <none>           <none>           <none>
kube-system   etcd-container-1-liu                      1/1     Running   0          106s    90.91.16.150    container-1-liu  <none>           <none>
kube-system   kube-apiserver-container-1-liu            1/1     Running   0          93s     90.91.16.150    container-1-liu  <none>           <none>
kube-system   kube-controller-manager-container-1-liu   1/1     Running   0          91s     90.91.16.150    container-1-liu  <none>           <none>
kube-system   kube-proxy-cfvwd                          1/1     Running   0          2m34s   90.91.16.150    container-1-liu  <none>           <none>
kube-system   kube-scheduler-container-1-liu            1/1     Running   0          118s    90.91.16.150    container-1-liu  <none>           <none>
[                    RPMS]# kubectl get nodes -o wide
NAME             STATUS     ROLES    AGE    VERSION   INTERNAL-IP     EXTERNAL-IP   OS-IMAGE                KERNEL-VERSION                CONTAINER-RUNTIME
container-1-liu  NotReady   master   3m7s   v1.15.1   90.91.16.150    <none>        CentOS Linux 7 (AltArch)  4.14.0-115.el7a.0.1.aarch64   docker://18.9.9
```

Step 3 Edit the deployment YAML file of k-vswitch (OVS Kubernetes component).

Download the **k-vswitch.yaml** file and edit the **clusterCIDA**, **serviceCIDA**, and **overlayType** fields in the file.

The values of **clusterCIDA** and **serviceCIDA** must be the same as the IP address segments planned for the Kubernetes cluster. The value of **overlayType** can be set to **vxlan** or **gre** based on user requirements.

Figure 1-12 Modifying the k-vswitch configuration file

```
#
# clusterCIDR should be updated to the same CIDR configured on your
#    Kubernetes components
# serviceCIDR should be updated to the same CIDR configured on your
#    Kubernetes components
# overlayType should be updated based on the overlay type you want.
#    Currently 'vxlan' and 'gre' are supported. 'gre' is recommended
#    but some cloud providers may not allow gre traffic over your network.
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: k-vswitch
  namespace: kube-system
data:
  clusterCIDR: "10.244.0.0/16"   # change this depending on your cluster, e.g. "100.96.0.0/11"
  serviceCIDR: "192.168.0.0/16"   # change this depending on your cluster, e.g. "100.64.0.0/13"
  overlayType: "gre"          # change this depending on your cluster, can be "vxlan" or "gre"
```

Step 4 Deploy cluster network components.

Use kubectl to deploy cluster network components.

```
kubectl apply -f k-vswitch.yaml
```

After the deployment is complete, the coredns service is in the **Running** state and the node is in the **Ready** state, as shown in **Figure 1-13** and **Figure 1-14**.

Figure 1-13 Installing the k-vswitch component

```
[root@[redacted] k-vswitch]# kubectl apply -f k-vswitch.yaml
configmap/k-vswitch created
customresourcedefinition.apiextensions.k8s.io/vswitchconfigs.kvswitch.io created
serviceaccount/k-vswitch created
clusterrole.rbac.authorization.k8s.io/k-vswitch created
clusterrolebinding.rbac.authorization.k8s.io/k-vswitch created
statefulset.apps/k-vswitch-controller created
daemonset.apps/k-vswitchd created
```

Figure 1-14 k-vswitch component deployment status

```
[root@[redacted] k-vswitch]# kubectl get pods --all-namespaces
NAMESPACE     NAME                                  READY   STATUS    RESTARTS   AGE
kube-system   coredns-5c98db65d4-9qnkm              1/1     Running   0          56m
kube-system   coredns-5c98db65d4-xlr5l              1/1     Running   0          56m
kube-system   etcd-container-1-liu                  1/1     Running   0          55m
kube-system   k-vswitch-controller-0                1/1     Running   0          41s
kube-system   k-vswitchd-chnhd                      1/1     Running   0          41s
kube-system   kube-apiserver-container-1-liu        1/1     Running   0          55m
kube-system   kube-controller-manager-container-1-liu  1/1     Running   0          55m
kube-system   kube-proxy-cfvwd                      1/1     Running   0          56m
kube-system   kube-scheduler-container-1-liu        1/1     Running   0          56m
[root@[redacted] k-vswitch]# kubectl get nodes -o wide
NAME            STATUS   ROLES    AGE   VERSION   INTERNAL-IP     EXTERNAL-IP   OS-IMAGE                KERNEL-VERSION                CONTAINER-RUNTIME
[redacted]      Ready    master   57m   v1.15.1   90.91.16.150    <none>        CentOS Linux 7 (AltArch)   4.14.0-115.el7a.0.1.aarch64   docker://18.9.9
```

Step 5 Add nodes to the cluster.

On other Kubernetes nodes to be deployed, run the **kubeadm join** command backed up in Step 2 to add the nodes to be deployed to the Kubernetes cluster.

```
kubeadm join <master-ip:port> --token <your-token> \
--discovery-token-ca-cert-hash sha256:<your-sha256-ca>
```

After the cluster nodes are added, "This node has joined the cluster" is displayed, as shown in **Figure 1-15**. The OVS component networking procedure is complete.

Figure 1-15 Adding nodes on the OVS network

```
[root@container-2-liu ~]# kubeadm join 90.91.16.150:6443 --token 2e74pm.s8xsiebaya5y1afv \
>     --discovery-token-ca-cert-hash sha256:4baffa1fdb1170fd68713fdb829c8b0cd9c1ba33e33cc9c61f1629bf7942880d
[preflight] Running pre-flight checks
        [WARNING IsDockerSystemdCheck]: detected "cgroupfs" as the Docker cgroup driver. The recommended driver is "systemd". Please
        [WARNING Hostname]: hostname "container-2-liu" could not be reached
        [WARNING Hostname]: hostname "container-2-liu": lookup container-2-liu on [::1]:53: read udp [::1]:51756->[::1]:53: read: co
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -oyaml'
[kubelet-start] Downloading configuration for the kubelet from the "kubelet-config-1.15" ConfigMap in the kube-system namespace
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Activating the kubelet service
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap...

This node has joined the cluster:
* Certificate signing request was sent to apiserver and a response was received.
* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the control-plane to see this node join the cluster.
```

----End

Network Verification

Step 1 Copy the following content and edit the Nginx deployment test YAML file to test the intra-node and cross-node communication capability.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-arm-deployment
spec:
  selector:
    matchLabels:
      app: arm64v8_nginx
  replicas: 5
  template:
    metadata:
      labels:
        app: arm64v8_nginx
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        ports:
        - containerPort: 80
```

Step 2 Run the **kubectl** command to deploy the Nginx service. If information shown in **Figure 1-16** is displayed, the Nginx service is running properly and the IP address has been allocated.

```
kubectl apply -f nginx.yaml
```

Figure 1-16 OVS networking service deployment test

```
[root@container-1-liu ~]# kubectl apply -f nginx.yaml
deployment.apps/nginx-arm-deployment created
[root@container-1-liu ~]#
```

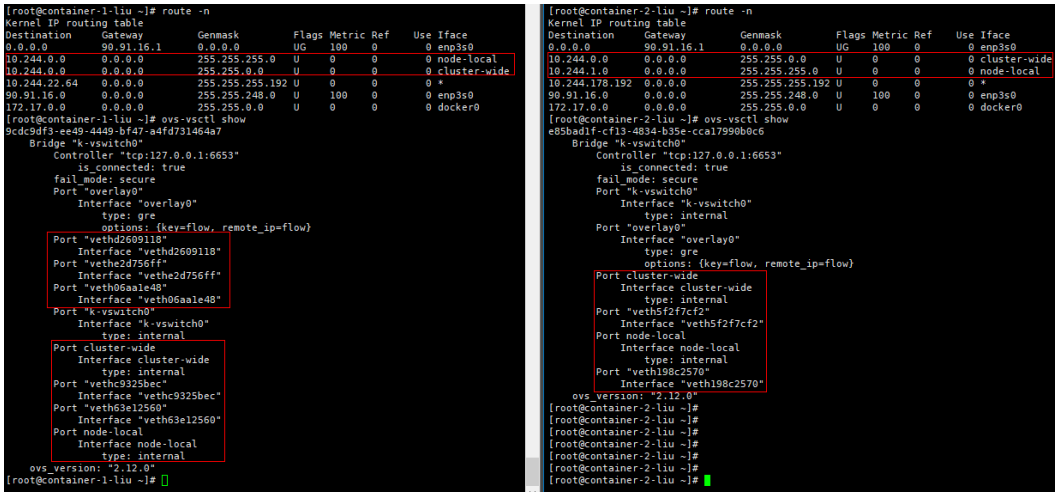
1 root@container-1-liu:~/k-vs...

```
Every 2.0s: kubectl get pods --all-namespaces -o wide

NAMESPACE     NAME                                        READY   STATUS    RESTARTS   AGE   IP              NODE              NOMINATED NODE   READINESS GATES
default       nginx-arm-deployment-67447f4448-6ddz2       1/1     Running   0          49s   10.244.1.59     container-2-liu   <none>           <none>
default       nginx-arm-deployment-67447f4448-92xt6       1/1     Running   0          49s   10.244.0.19     container-1-liu   <none>           <none>
default       nginx-arm-deployment-67447f4448-bw66v       1/1     Running   0          49s   10.244.0.18     container-1-liu   <none>           <none>
default       nginx-arm-deployment-67447f4448-tbdcq       1/1     Running   0          49s   10.244.0.20     container-1-liu   <none>           <none>
default       nginx-arm-deployment-67447f4448-wdict       1/1     Running   0          49s   10.244.1.58     container-2-liu   <none>           <none>
kube-system   coredns-5c98db65d4-9qnkm                    1/1     Running   0          16h   10.244.0.11     container-1-liu   <none>           <none>
kube-system   coredns-5c98db65d4-xlr5l                    1/1     Running   0          16h   10.244.0.10     container-1-liu   <none>           <none>
kube-system   etcd-container-1-liu                        1/1     Running   0          16h   90.91.16.150    container-1-liu   <none>           <none>
kube-system   k-vswitch-controller-0                      1/1     Running   0          15h   90.91.16.150    container-1-liu   <none>           <none>
kube-system   k-vswitchd-5lvs2                            1/1     Running   0          15h   90.91.16.151    container-2-liu   <none>           <none>
kube-system   k-vswitchd-chnhd                            1/1     Running   0          15h   90.91.16.150    container-1-liu   <none>           <none>
kube-system   kube-apiserver-container-1-liu              1/1     Running   0          16h   90.91.16.150    container-1-liu   <none>           <none>
kube-system   kube-controller-manager-container-1-liu     1/1     Running   0          16h   90.91.16.150    container-1-liu   <none>           <none>
kube-system   kube-proxy-cfvwd                            1/1     Running   0          16h   90.91.16.150    container-1-liu   <none>           <none>
kube-system   kube-proxy-mm77l                            1/1     Running   0          15h   90.91.16.151    container-2-liu   <none>           <none>
kube-system   kube-scheduler-container-1-liu              1/1     Running   0          16h   90.91.16.150    container-1-liu   <none>           <none>
```

Step 3 Check that the node routing information and OVS bridge status are normal, as shown in **Figure 1-17**. The OVS has configured the cluster-wide and node-local port information on the k-vswitch0 bridge. All service ports are mounted to the

bridge in veth pair mode. Intra-node and inter-node communication can be implemented through routes and the OVS bridge.

Figure 1-17 Routing and bridge information of OVS networking nodes



----End

1.3.3.2 Calico Component Networking Operations

Prerequisites

1. The Docker and Kubernetes components (kubeadm, kubectl, and kubelet) have been installed on the node to be deployed.
2. The node to be deployed can properly pull Docker images.

Procedure

Step 1 Set the Kubernetes master node during initialization.

NOTE

In this section, the **10.244.0.0/16** network segment is used as the network driver CIDR and the default gateway is used as the network broadcast address. If you need to specify another network segment, modify the command accordingly.

```
kubeadm init --pod-network-cidr=10.244.0.0/16
mkdir -p $HOME/.kube
cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
chown $(id -u):$(id -g) $HOME/.kube/config
```

After the initialization is complete, information in **Figure 1-18** and **Figure 1-19** are displayed. Check whether the pods information of the Kubernetes cluster is normal and whether the node status is **NotReady**. Back up the **kubeadm join** command in the output for future use. Then, deploy the network plane.

Figure 1-18 Successful initialization of the active Kubernetes node

```
Your Kubernetes control-plane has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

  mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config

You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
  https://kubernetes.io/docs/concepts/cluster-administration/addons/

Then you can join any number of worker nodes by running the following on each as root:

kubeadm join [redacted]:6443 --token [redacted] \
    --discovery-token-ca-cert-hash sha256:[redacted]
```

Figure 1-19 Initialization status of the active Kubernetes node

```
[redacted] RPMS]# kubectl get pods --all-namespaces -o wide
NAMESPACE     NAME                                      READY   STATUS    RESTARTS   AGE     IP              NODE             NOMINATED NODE   READINESS GATES
kube-system   coredns-5c98db65d4-9qnkm                  0/1     Pending   0          2m34s   <none>          <none>           <none>           <none>
kube-system   coredns-5c98db65d4-xlr5l                  0/1     Pending   0          2m34s   <none>          <none>           <none>           <none>
kube-system   etcd-container-1-liu                      1/1     Running   0          106s    90.91.16.150    container-1-liu  <none>           <none>
kube-system   kube-apiserver-container-1-liu            1/1     Running   0          93s     90.91.16.150    container-1-liu  <none>           <none>
kube-system   kube-controller-manager-container-1-liu   1/1     Running   0          91s     90.91.16.150    container-1-liu  <none>           <none>
kube-system   kube-proxy-cfvwd                          1/1     Running   0          2m34s   90.91.16.150    container-1-liu  <none>           <none>
kube-system   kube-scheduler-container-1-liu            1/1     Running   0          118s    90.91.16.150    container-1-liu  <none>           <none>
[redacted] RPMS]# kubectl get nodes -o wide
NAME              STATUS     ROLES    AGE    VERSION   INTERNAL-IP    EXTERNAL-IP   OS-IMAGE                KERNEL-VERSION                CONTAINER-RUNTIME
container-1-liu   NotReady   master   3m7s   v1.15.1   90.91.16.150   <none>        CentOS Linux 7 (AltArch)   4.14.0-115.el7a.0.1.aarch64   docker://18.9.9
```

Step 2 Edit the Calico deployment YAML file.

Download the **calico.yaml** and **calicoctl.yaml** deployment files. The BGP IPIP mode recommended by Calico is deployed by default and you do not need to modify the configuration file. If you need to deploy the Calico component in VxLAN Only mode, see **1.4 Open vSwitch Network Plane Management**.

Step 3 Deploy cluster network components.

Use kubectl to deploy cluster network components.

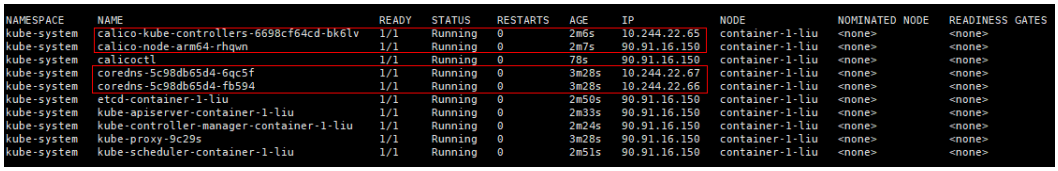
```
kubectl apply -f calico.yaml
kubectl apply -f calicoctl.yaml
alias calicoctl="kubectl exec -i -n kube-system calicoctl -- /calicoctl"
```

After the deployment is complete, the coredns service is in the **Running** state and the node is in the **Ready** state, as shown in **Figure 1-20** and **Figure 1-21**.

Figure 1-20 Calico component installation

```
[root@container-1-liu ~]# kubectl apply -f calico.yaml
configmap/calico-config created
customresourcedefinition.apiextensions.k8s.io/bgpconfigurations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/bgppeers.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/blockaffinities.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/clusterinformations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/felixconfigurations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/globalnetworkpolicies.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/globalnetworksets.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/hostendpoints.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamblocks.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamconfigs.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamhandles.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ippools.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/networkpolicies.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/networksets.crd.projectcalico.org created
clusterrole.rbac.authorization.k8s.io/calico-kube-controllers created
clusterrolebinding.rbac.authorization.k8s.io/calico-kube-controllers created
clusterrole.rbac.authorization.k8s.io/calico-node created
clusterrolebinding.rbac.authorization.k8s.io/calico-node created
daemonset.apps/calico-node-arm64 created
daemonset.apps/calico-node-amd64 created
serviceaccount/calico-node created
deployment.apps/calico-kube-controllers created
serviceaccount/calico-kube-controllers created
```

Figure 1-21 Calico component deployment status



```
NAMESPACE     NAME                                       READY   STATUS    RESTARTS   AGE     IP              NODE              NOMINATED NODE   READINESS GATES
kube-system   calico-kube-controllers-6698cf64cd-bk6lv   1/1     Running   0          2m6s    10.244.22.65    container-1-liu   <none>           <none>
kube-system   calico-node-arm64-rhqwn                    1/1     Running   0          2m7s    90.91.16.150    container-1-liu   <none>           <none>
kube-system   calicoctl                                  1/1     Running   0          78s     90.91.16.150    container-1-liu   <none>           <none>
kube-system   coredns-5c98db65d4-6qc5f                   1/1     Running   0          3m28s   10.244.22.67    container-1-liu   <none>           <none>
kube-system   coredns-5c98db65d4-fb594                   1/1     Running   0          3m28s   10.244.22.66    container-1-liu   <none>           <none>
kube-system   etcd-container-1-liu                       1/1     Running   0          2m50s   90.91.16.150    container-1-liu   <none>           <none>
kube-system   kube-apiserver-container-1-liu             1/1     Running   0          2m33s   90.91.16.150    container-1-liu   <none>           <none>
kube-system   kube-controller-manager-container-1-liu    1/1     Running   0          2m24s   90.91.16.150    container-1-liu   <none>           <none>
kube-system   kube-proxy-9c29s                           1/1     Running   0          3m28s   90.91.16.150    container-1-liu   <none>           <none>
kube-system   kube-scheduler-container-1-liu             1/1     Running   0          2m51s   90.91.16.150    container-1-liu   <none>           <none>
```

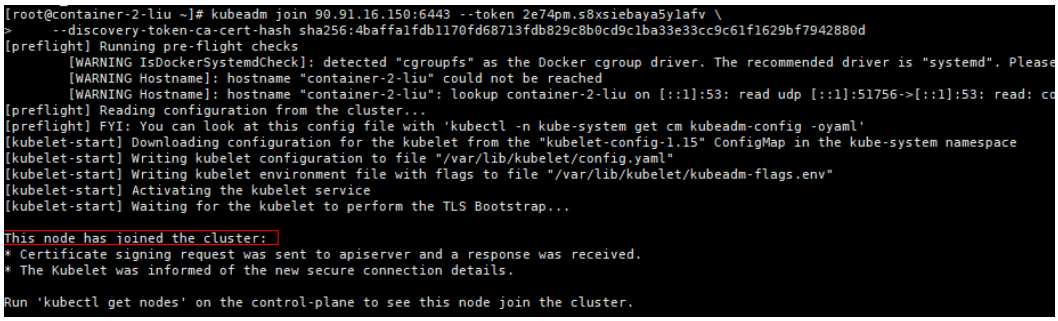
Step 4 Add nodes to the cluster.

On other Kubernetes nodes to be deployed, run the **kubeadm join** command backed up in Step 2 to add the nodes to be deployed to the Kubernetes cluster.

```
kubeadm join <master-ip:port> --token <your-token> \
--discovery-token-ca-cert-hash sha256:<your-sha256-ca>
```

After the cluster nodes are added, "This node has joined the cluster" is displayed, as shown in **Figure 1-22**. The Calico component networking procedure is complete.

Figure 1-22 Adding nodes on the Calico network



```
[root@container-2-liu ~]# kubeadm join 90.91.16.150:6443 --token 2e74pm.s8xsiebaya5y1afv \
>     --discovery-token-ca-cert-hash sha256:4baffa1fdb1170fd68713fdb829c8b0cd9c1ba33e33cc9c61f1629bf7942880d
[preflight] Running pre-flight checks
        [WARNING IsDockerSystemdCheck]: detected "cgroupfs" as the Docker cgroup driver. The recommended driver is "systemd". Please
        [WARNING Hostname]: hostname "container-2-liu" could not be reached
        [WARNING Hostname]: hostname "container-2-liu": lookup container-2-liu on [::1]:53: read udp [::1]:51756->[::1]:53: read: co
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -oyaml'
[kubelet-start] Downloading configuration for the kubelet from the "kubelet-config-1.15" ConfigMap in the kube-system namespace
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Activating the kubelet service
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap...

This node has joined the cluster:
* Certificate signing request was sent to apiserver and a response was received.
* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the control-plane to see this node join the cluster.
```

----End

Network Verification

Step 1 Copy the following content and edit the Nginx deployment test YAML file to test the intra-node and cross-node communication capability.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-arm-deployment
spec:
  selector:
    matchLabels:
      app: arm64v8_nginx
  replicas: 5
  template:
    metadata:
      labels:
        app: arm64v8_nginx
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        ports:
        - containerPort: 80
```

Step 2 Run the **kubectl** command to deploy the Nginx service. If information shown in **Figure 1-23** is displayed, the Nginx service is running properly and the IP address has been allocated.


```
kubectl apply -f nginx.yaml
```

Figure 1-23 Calico networking service deployment test

```
NAMESPACE     NAME                                       READY   STATUS    RESTARTS   AGE     IP               NODE              NOMINATED NODE   READINESS GATES
default       nginx-arm-deployment-67447f4448-4jqlh      1/1     Running   0          21s     10.244.22.69     container-1-liu   <none>           <none>
default       nginx-arm-deployment-67447f4448-5pzkt      1/1     Running   0          21s     10.244.22.70     container-1-liu   <none>           <none>
default       nginx-arm-deployment-67447f4448-hrhvk      1/1     Running   0          21s     10.244.178.193   container-2-liu   <none>           <none>
default       nginx-arm-deployment-67447f4448-ktw7g      1/1     Running   0          21s     10.244.178.194   container-2-liu   <none>           <none>
default       nginx-arm-deployment-67447f4448-v6tnx      1/1     Running   0          21s     10.244.22.68     container-1-liu   <none>           <none>
kube-system   calico-kube-controllers-6698cf64cd-bk6lv   1/1     Running   0          4m59s   10.244.22.65     container-1-liu   <none>           <none>
kube-system   calico-node-arm64-rhqwn                    1/1     Running   0          5m      90.91.16.150     container-1-liu   <none>           <none>
kube-system   calico-node-arm64-w4g7n                    1/1     Running   0          65s     90.91.16.151     container-2-liu   <none>           <none>
kube-system   calicoctl                                  1/1     Running   0          4m11s   90.91.16.150     container-1-liu   <none>           <none>
kube-system   coredns-5c98db65d4-6qc5f                   1/1     Running   0          6m21s   10.244.22.67     container-1-liu   <none>           <none>
kube-system   coredns-5c98db65d4-fb594                   1/1     Running   0          6m21s   10.244.22.66     container-1-liu   <none>           <none>
kube-system   etcd-container-1-liu                       1/1     Running   0          5m43s   90.91.16.150     container-1-liu   <none>           <none>
kube-system   kube-apiserver-container-1-liu             1/1     Running   0          5m26s   90.91.16.150     container-1-liu   <none>           <none>
kube-system   kube-controller-manager-container-1-liu    1/1     Running   0          5m17s   90.91.16.150     container-1-liu   <none>           <none>
kube-system   kube-proxy-9c29s                           1/1     Running   0          6m21s   90.91.16.150     container-1-liu   <none>           <none>
kube-system   kube-proxy-kgmsk                           1/1     Running   0          65s     90.91.16.151     container-2-liu   <none>           <none>
kube-system   kube-scheduler-container-1-liu             1/1     Running   0          5m44s   90.91.16.150     container-1-liu   <none>           <none>
```

Check that the node routing information and OVS bridge status are normal, as shown in **Figure 1-24**. Calico has configured the addresses in the IP pool for the container space. All service ports are mapped to Calico components in veth pair mode. Intra-node and inter-node communication can be implemented through routes.

Figure 1-24 Routing and workload of Calico network nodes

```
[root@container-1-liu ~]# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         90.91.16.1      0.0.0.0         UG    100    0        0 enp3s0
10.244.22.64    0.0.0.0         255.255.255.192 U     0      0        0 *
10.244.22.65    0.0.0.0         255.255.255.255 UH    0      0        0 cali924e079cfa6
10.244.22.66    0.0.0.0         255.255.255.255 UH    0      0        0 caliba7968916b4
10.244.22.67    0.0.0.0         255.255.255.255 UH    0      0        0 cali483aa0dbe95
10.244.22.68    0.0.0.0         255.255.255.255 UH    0      0        0 cali44047625f64
10.244.22.69    0.0.0.0         255.255.255.255 UH    0      0        0 cali7780df574b2
10.244.22.70    0.0.0.0         255.255.255.255 UH    0      0        0 cali7fc6af55f2c
10.244.178.192  90.91.16.151    255.255.255.192 UG    0      0        0 tunl0
90.91.16.0      0.0.0.0         255.255.248.0   U     100    0        0 enp3s0
172.17.0.0      0.0.0.0         255.255.0.0     U     0      0        0 docker0
[root@container-1-liu ~]# calicoctl get ipPool
NAME                  CIDR            SELECTOR
default-ipv4-ippool   10.244.0.0/16   all()

[root@container-1-liu ~]# calicoctl get workloadEndpoint
WORKLOAD                                  NODE              NETWORKS             INTERFACE
nginx-arm-deployment-67447f4448-4jqlh     container-1-liu   10.244.22.69/32      cali7780df574b2
nginx-arm-deployment-67447f4448-5pzkt     container-1-liu   10.244.22.70/32      cali7fc6af55f2c
nginx-arm-deployment-67447f4448-hrhvk     container-2-liu   10.244.178.193/32    cali40a0e55f13e
nginx-arm-deployment-67447f4448-ktw7g     container-2-liu   10.244.178.194/32    cali1fd70543ae4
nginx-arm-deployment-67447f4448-v6tnx     container-1-liu   10.244.22.68/32      cali44047625f64
```

----End

1.4 Open vSwitch Network Plane Management

For OVS, network plane management is to manage the networking mode. The management tasks are as follows:

Managing the networking mode

- Configuring the VxLAN network
- Configuring the GRE tunnel network

For details about the OVS networking mode, see **1.3.3.1 OVS Component Networking Operations**. After the value of **overlayType** is changed to **gre** or **vxlan**, run the **kubectl apply** command to enable the configuration file to complete the networking mode configuration.

1.5 Calico Network Plane Management

1.5.1 IP Address Pool Management

For Calico, network plane management covers IP address pool and BGP configuration. The IP pool management tasks are as follows:

- Managing an IP Address Pool
  - Viewing IP address pool information
  - Adding an IP address pool
  - Modifying an IP address pool
- Changing the IPIP mode
- Changing the VxLAN mode
- Disabling or enabling an IP address pool
  - Deleting an IP address pool

1.5.1.1 Viewing IP Address Pool Information

calicoctl get ipPool -o wide

```
[root@container-1-liu calico]# calicoctl get ipPool -o wide
NAME                  CIDR             NAT    IPIPMODE   VXLANMODE   DISABLED   SELECTOR
default-ipv4-ippool   10.244.0.0/16    true   Always     Never       false      all()
ippool-vxlan          192.144.0.0/16   true   Never      Always      false      all()
```

1.5.1.2 Adding an IP Address Pool

Step 1 Edit the YAML file of the new IP address pool.

Configure a new IP address pool. Ensure that the name and CIDR of the IP address pool cannot conflict with those of existing IP address pools.

```
apiVersion: projectcalico.org/v3
kind: IPPool
metadata:
name: ippool-vxlan
spec:
cidr: 192.144.0.0/16
vxlanMode: Always
natOutgoing: true
```

For details about parameter settings, see **https://docs.projectcalico.org/reference/resources/ippool**.

Step 2 Add an IP address pool.

$ calicoctl apply -f - < ipPool.yaml

```
[root@container-1-liu calico]# calicoctl apply -f - < ipPool_vxlan.yaml
Successfully applied 1 'IPPool' resource(s)
[root@container-1-liu calico]# calicoctl get ipPool -o wide
NAME                  CIDR             NAT    IPIPMODE   VXLANMODE   DISABLED   SELECTOR
default-ipv4-ippool   10.244.0.0/16    true   Always     Never       false      all()
ippool-vxlan          192.144.0.0/16   true   Never      Always      false      all()
```

----End

1.5.1.3 Modifying an IP Address Pool

Changing the IPIP Mode

This section describes how to change the IPIP mode of the default-ipv4-ippool IP address pool to **CrossSubnet** as an example.

```
calicoctl patch ipPool default-ipv4-ippool -p '{"spec": {"ipipMode": "CrossSubnet"}}'
```

For details about the parameter settings, see **https://docs.projectcalico.org/reference/resources/ippool**.

Changing the VxLAN Mode

This section describes how to change the VxLAN mode of the ippool-vxlan IP address pool to **CrossSubnet** as an example.

```
calicoctl patch ipPool ippool-vxlan -p '{"spec": {"vxlanMode": "CrossSubnet"}}'
```

For details about the parameter settings, see **https://docs.projectcalico.org/reference/resources/ippool**.

Disabling or Enabling an IP Address Pool

This section uses the ippool-vxlan IP address pool as an example.

```
calicoctl patch ipPool ippool-vxlan -p '{"spec": {"disabled": true}}'
calicoctl patch ipPool ippool-vxlan -p '{"spec": {"disabled": false}}'
```

1.5.1.4 Deleting an IP Address Pool

This section describes how to delete the ippool-vxlan IP address pool as an example. You can use either of the following commands.

```
calicoctl delete ipPool ippool-vxlan
calicoctl delete -f - < ipPool_vxlan.yaml
```

1.5.2 Managing BGP Configuration

For Calico, network plane management covers IP address pool and BGP configuration. The BGP pool management tasks are as follows:

- Managing BGP configuration
  - Adding BGP configuration
  - Modifying BGP configuration
- Changing the BGP mode to Full Mesh
- Changing the BGP mode to Route Reflector
  - Deleting BGP configuration

1.5.2.1 Checking the BGP Configuration

```
calicoctl get bgpConfiguration -o wide
```

By default, new clusters do not contain any BGP configuration and the default Calico BGP configuration is used. To change the BGP configuration, you need to the BGP configuration of the default domain.

1.5.2.2 Adding BGP Configuration

Step 1 Edit the YAML file for adding BGP configuration.

Configure a new BGP. Ensure that the name and ASNumber of the BGP do not conflict with those of existing BGPs.

```
apiVersion: projectcalico.org/v3
kind: BGPConfiguration
metadata:
name: default
spec:
logSeverityScreen: Info
nodeToNodeMeshEnabled: true
asNumber: 63400
serviceClusterIPs:
- cidr: 10.244.0.0/16
```

For details about parameter settings, see **https://docs.projectcalico.org/reference/resources/bgpconfig**.

Step 2 Add a BGP.

```
calicoctl apply -f - < bgp.yaml
```

```
[root@container-1-liu calico]# calicoctl apply -f - < bgp.yaml
Successfully applied 1 'BGPConfiguration' resource(s)
[root@container-1-liu calico]# calicoctl get bgpConfiguration -o wide
NAME      LOGSEVERITY   MESHENABLED   ASNUMBER
default   Info          true          63400
```

----End

1.5.2.3 Modifying BGP Configuration

Change the BGP mode to Full Mesh.

This section describes how to change the default BGP configuration as an example.

```
calicoctl patch bgpconfiguration default -p '{"spec": {"nodeToNodeMeshEnabled": true}}'
```

Changing the BGP Mode to Route Reflector

This section uses the default BGP configuration as an example to describe how to configure a node as a route reflector.

Step 1 Disable the full mesh mode between nodes.

After this step is complete, cross-node pod communication is interrupted.

```
calicoctl patch bgpconfiguration default -p '{"spec": {"nodeToNodeMeshEnabled": false}}'
```

Step 2 Configure a route reflector node.

```
kubectl label node <node-name> route-reflector=true
```

Step 3 Interconnecting the route reflector node and common nodes.

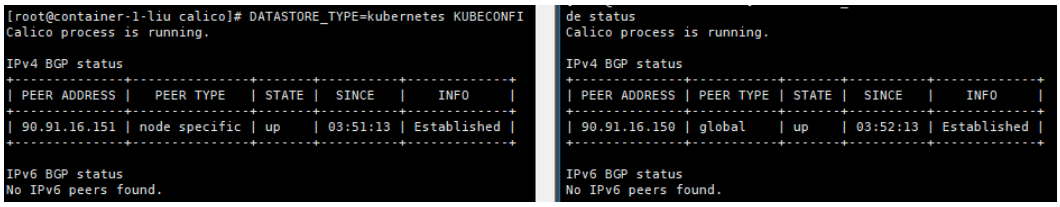
Edit the BGPPeer YAML file and use calicoctl to load the file.

```
kind: BGPPeer
apiVersion: projectcalico.org/v3
```



```
metadata:
name: peer-with-route-reflectors
spec:
nodeSelector: all()
peerSelector: route-reflector == true
calicoctl apply -f - < bgpPeer.yaml
```

After the configuration is complete, check the node status on each node. All nodes managed in the local domain can be viewed on the route reflector node, and only the route reflector node can be viewed on common nodes.



----End

1.5.2.4 Deleting BGP Configuration

This section describes how to delete the default BGP as an example. You can use either of the following commands.

```
calicoctl delete bgpConfiguration default
calicoctl delete -f - < bgp.yaml
```

1.6 Troubleshooting

Problem 1: "ovs-ctl command not found" During OVS Startup

Symptom

When the **ovs-ctl start** command is run, the message "ovs-ctl command not found" is displayed.

Possible Causes

The tool path is not set into the environment variable. For details about how to set the environment variable, see **Start with the Default Configurations**.

Procedure

1. Check the Open vSwitch installation path, for example, **/usr/local/share**.
2. Set the environment variable.
   ```
   export PATH=$PATH:/usr/local/share/openvswitch/scripts
   ```
3. Restart OVS components.
   ```
   ovs-ctl start
   ```

Problem 2: Abnormal Inter-Node Network Communication After Kubernetes OVS Plane Deployment

Symptom

After the OVS plane is deployed on Kubernetes, the displayed network bandwidth is 0 when iperf/qperf services are running.

Possible Causes

By default, the network interface MTU in the container is 1500 bytes, which is the same as that of the host. After data packets are sent from the container, the data packets are encapsulated in the GRE/VxLAN overlay mode on the host OVS bridge. After the encapsulation, the data packets exceed 1500 bytes. As a result, the host machine fails to send network packets.

Procedure

You are advised to add network port MTU configuration in the Docker startup command and set the network interface MTU in the container to 1400.

Problem 3: Calico-node in the Running Instead of Ready State After Kubernetes Calico Deployment

Symptom

After Calico is deployed on Kubernetes, the calico-node is in the Running state but cannot be Ready.

Possible Causes

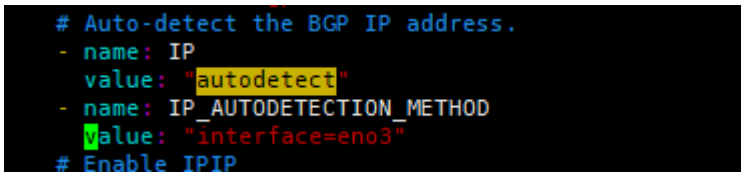
As shown in the following figure, docker logs show that the dataplane updates operation is being performed and the resync connection cannot be set up. The calicoctl tool is used to query node information on the master node. It is found that some nodes have dual network planes. However, the calico component uses AUTO_DETECTION to discover BGP network interfaces by default. The external network interfaces are incorrectly selected on these nodes. As a result, the network cannot communicate with each other.

```
2020-05-09 07:12:43.840 [INFO][75] table.go 497: Loading current iptables state and checking it is correct. ipVersion=0x4 table="mangle"
2020-05-09 07:12:43.844 [INFO][75] int_dataplane.go 1176: Finished applying updates to dataplane. msecToApply=6.49271
2020-05-09 07:12:43.845 [INFO][75] int_dataplane.go 1162: Applying dataplane updates
2020-05-09 07:12:43.845 [INFO][75] table.go 810: Invalidating dataplane cache ipVersion=0x4 reason="post update" table="raw"
2020-05-09 07:12:43.848 [INFO][75] table.go 497: Loading current iptables state and checking it is correct. ipVersion=0x4 table="raw"
2020-05-09 07:12:43.854 [INFO][75] int_dataplane.go 1176: Finished applying updates to dataplane. msecToApply=8.71458
2020-05-09 07:12:44.151 [INFO][75] int_dataplane.go 1162: Applying dataplane updates
2020-05-09 07:12:44.151 [INFO][75] table.go 810: Invalidating dataplane cache ipVersion=0x4 reason="post update" table="filter"
2020-05-09 07:12:44.151 [INFO][75] table.go 810: Invalidating dataplane cache ipVersion=0x4 reason="post update" table="nat"
2020-05-09 07:12:44.154 [INFO][75] table.go 497: Loading current iptables state and checking it is correct. ipVersion=0x4 table="filter"
2020-05-09 07:12:44.157 [INFO][75] table.go 497: Loading current iptables state and checking it is correct. ipVersion=0x4 table="nat"
2020-05-09 07:12:44.162 [INFO][75] int_dataplane.go 1176: Finished applying updates to dataplane. msecToApply=10.781870000000001
2020-05-09 07:12:51.991 [INFO][75] int_dataplane.go 1162: Applying dataplane updates
2020-05-09 07:12:51.991 [INFO][75] ipsets.go 223: Asked to resync with the dataplane on next update. family="inet"
2020-05-09 07:12:51.991 [INFO][75] ipsets.go 306: Resyncing ipsets with dataplane. family="inet"
2020-05-09 07:12:51.994 [INFO][75] ipsets.go 356: Finished resync family="inet" numInconsistenciesFound=0 resyncDuration=2.36858ms
2020-05-09 07:12:51.994 [INFO][75] int_dataplane.go 1176: Finished applying updates to dataplane. msecToApply=3.28108
2020-05-09 07:13:02.254 [INFO][75] int_dataplane.go 1162: Applying dataplane updates
2020-05-09 07:13:02.254 [INFO][75] ipsets.go 223: Asked to resync with the dataplane on next update. family="inet"
2020-05-09 07:13:02.254 [INFO][75] ipsets.go 306: Resyncing ipsets with dataplane. family="inet"
2020-05-09 07:13:02.256 [INFO][75] ipsets.go 356: Finished resync family="inet" numInconsistenciesFound=0 resyncDuration=2.36425ms
2020-05-09 07:13:02.257 [INFO][75] int_dataplane.go 1176: Finished applying updates to dataplane. msecToApply=3.24192
2020-05-09 07:13:13.230 [INFO][75] int_dataplane.go 1162: Applying dataplane updates
2020-05-09 07:13:13.230 [INFO][75] ipsets.go 223: Asked to resync with the dataplane on next update. family="inet"
2020-05-09 07:13:13.230 [INFO][75] ipsets.go 306: Resyncing ipsets with dataplane. family="inet"
2020-05-09 07:13:13.233 [INFO][75] ipsets.go 356: Finished resync family="inet" numInconsistenciesFound=0 resyncDuration=2.44248ms
2020-05-09 07:13:13.234 [INFO][75] int_dataplane.go 1176: Finished applying updates to dataplane. msecToApply=3.29682
```

```
NAME              ASN       IPV4                IPV6
container-1-liu   (64512)   90.91.16.150/21
k8s-x86-master    (64512)   126.26.161.160/24
```

Procedure

You are advised to use nodeSelector + IP_AUTODETECTION_METHOD to distinguish these nodes and manually identify BGP network interfaces.



Problem 4: Service Node Network Interruption After Kubernetes Calico Deployment

Symptom

After Calico is deployed on Kubernetes, the network of the service node is interrupted.

Possible Causes

Check whether the Hi1822 NIC is used on the network plane.

The Calico component uses IPIP (IP tunnel mode) by default. However, the Hi1822 NIC firmware does not support the verification and unloading of IP tunnel packets and the TSO function. If the verification and offload functions in the transmit direction and the TSO function are enabled on the NIC, and the system sends IP tunnel packets, the NIC becomes abnormal. As a result, the NIC hardware does not obtain packets from the host and sends them to the network side, and the driver reports transmit timeout.

Specifically, the service network is interrupted and disconnected from the gateway. The service network can be recovered only after the gateway is restarted.

Procedure

You are advised to check the NIC first. If the Hi1822 NIC is used, you are advised to run the **ethtool** command to disable the verification and unload in the transmit direction.

```
ethtool -K <eth-port> tx off
```

1.7 References

1.7.1 OVS Kubernetes Deployment YAML File

```
# Configmap 'k-vswitch' is the only resource in this file that requires
# updates based on your cluster configuration.
#
# clusterCIDR should be updated to the same CIDR configured on your
#    Kubernetes components
# serviceCIDR should be updated to the same CIDR configured on your
#    Kubernetes components
# overlayType should be updated based on the overlay type you want.
#    Currently 'vxlan' and 'gre' are supported. 'gre' is recommended
#    but some cloud providers may not allow gre traffic over your network.
---
apiVersion: v1
kind: ConfigMap
metadata:
name: k-vswitch
namespace: kube-system
data:
```



```
clusterCIDR: "<clusterCIDR>"   # change this depending on your cluster, e.g. "100.96.0.0/11"
serviceCIDR: "<serviceCIDR>"   # change this depending on your cluster, e.g. "100.64.0.0/13"
overlayType: "<overlayType>"         # change this depending on your cluster, can be "vxlan" or "gre"

---
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
name: vswitchconfigs.kvswitch.io
spec:
group: kvswitch.io
version: v1alpha1
names:
kind: VSwitchConfig
plural: vswitchconfigs
scope: Cluster

---
apiVersion: v1
kind: ServiceAccount
metadata:
name: k-vswitch
namespace: kube-system

---
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
name: k-vswitch
rules:
- apiGroups:
- ""
resources:
- services
- nodes
- endpoints
- pods
- namespaces
verbs:
- list
- get
- watch
- apiGroups:
- "networking.k8s.io"
resources:
- "networkpolicies"
verbs:
- get
- list
- watch
- apiGroups:
- "kvswitch.io"
resources:
- vswitchconfigs
verbs:
- list
- get
- watch
- create
- update
- patch
- delete

---
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
name: k-vswitch
roleRef:
```

```
apiGroup: rbac.authorization.k8s.io
kind: ClusterRole
name: k-vswitch
subjects:
- kind: ServiceAccount
name: k-vswitch
namespace: kube-system

---
apiVersion: apps/v1
kind: StatefulSet
metadata:
name: k-vswitch-controller
namespace: kube-system
spec:
replicas: 1
updateStrategy:
type: RollingUpdate
serviceName: k-vswitch-controller
selector:
matchLabels:
k8s-app: k-vswitch-controller
template:
metadata:
labels:
k8s-app: k-vswitch-controller
spec:
tolerations:
- key: "node-role.kubernetes.io/master"
effect: NoSchedule
hostNetwork: true
serviceAccountName: k-vswitch
containers:
- name: k-vswitch-controller
image: kvswitch/k-vswitch:latest
imagePullPolicy: IfNotPresent
command:
- "/bin/k-vswitch-controller"
- "--cluster-cidr=$(K_VSWITCH_CLUSTER_CIDR)"
- "--service-cidr=$(K_VSWITCH_SERVICE_CIDR)"
- "--overlay-type=$(K_VSWITCH_OVERLAY_TYPE)"
env:
- name: K_VSWITCH_CLUSTER_CIDR
valueFrom:
configMapKeyRef:
name: k-vswitch
key: clusterCIDR
- name: K_VSWITCH_SERVICE_CIDR
valueFrom:
configMapKeyRef:
name: k-vswitch
key: serviceCIDR
- name: K_VSWITCH_OVERLAY_TYPE
valueFrom:
configMapKeyRef:
name: k-vswitch
key: overlayType

---
apiVersion: apps/v1
kind: DaemonSet
metadata:
name: k-vswitchd
namespace: kube-system
labels:
k8s-app: k-vswitchd
spec:
selector:
matchLabels:
```

```
k8s-app: k-vswitchd
template:
metadata:
labels:
k8s-app: k-vswitchd
annotations:
scheduler.alpha.kubernetes.io/critical-pod: ''
spec:
serviceAccountName: k-vswitch
containers:
- name: k-vswitchd
image: kvswitch/k-vswitch:latest
imagePullPolicy: IfNotPresent
env:
- name: NODE_NAME
valueFrom:
fieldRef:
fieldPath: spec.nodeName
securityContext:
privileged: true
volumeMounts:
- mountPath: /etc/cni/net.d
name: cni-conf
- mountPath: /etc/openvswitch
name: ovs-etc
- mountPath: /var/run/openvswitch
name: ovs-run
- mountPath: /var/log/openvswitch
name: ovs-log
- mountPath: /lib/modules
name: lib-modules
initContainers:
- name: install-cni
image: kvswitch/k-vswitch:latest
imagePullPolicy: IfNotPresent
command:
- /bin/sh
- -c
- |
set -e -x;
cp /bin/k-vswitch-cni /opt/cni/bin/
volumeMounts:
- mountPath: /opt/cni/bin
name: cni-bin-dir
hostNetwork: true
tolerations:
- key: CriticalAddonsOnly
operator: Exists
- effect: NoSchedule
key: node-role.kubernetes.io/master
operator: Exists
- effect: NoSchedule
key: node.kubernetes.io/not-ready
operator: Exists
volumes:
- name: cni-bin-dir
hostPath:
path: /opt/cni/bin
- name: cni-conf
hostPath:
path: /etc/cni/net.d
- name: ovs-run
hostPath:
path: /var/run/openvswitch
- name: ovs-etc
hostPath:
path: /etc/openvswitch
- name: ovs-log
hostPath:
```



```
path: /var/log/openvswitch
- name: lib-modules
hostPath:
path: /lib/modules
```

1.7.2 Calico Deployment YAML File

```
---
# Source: calico/templates/calico-config.yaml
# This ConfigMap is used to configure a self-hosted Calico installation.
kind: ConfigMap
apiVersion: v1
metadata:
name: calico-config
namespace: kube-system
data:
# Typha is disabled.
typha_service_name: "none"
# Configure the backend to use.
calico_backend: "bird"
# Configure the MTU to use for workload interfaces and the
# tunnels.  For IPIP, set to your network MTU - 20; for VXLAN
# set to your network MTU - 50.
veth_mtu: "1440"

# The CNI network configuration to install on each node.  The special
# values in this config will be automatically populated.
cni_network_config: |-
{
"name": "k8s-pod-network",
"cniVersion": "0.3.1",
"plugins": [
{
"type": "calico",
"log_level": "info",
"datastore_type": "kubernetes",
"nodename": "__KUBERNETES_NODE_NAME__",
"mtu": __CNI_MTU__,
"ipam": {
"type": "calico-ipam"
},
"policy": {
"type": "k8s"
},
"kubernetes": {
"kubeconfig": "__KUBECONFIG_FILEPATH__"
}
},
{
"type": "portmap",
"snat": true,
"capabilities": {"portMappings": true}
},
{
"type": "bandwidth",
"capabilities": {"bandwidth": true}
}
]
}

---
# Source: calico/templates/kdd-crds.yaml

apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
name: bgpconfigurations.crd.projectcalico.org
spec:
scope: Cluster
```



```
group: crd.projectcalico.org
version: v1
names:
kind: BGPConfiguration
plural: bgpconfigurations
singular: bgpconfiguration

---
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
name: bgppeers.crd.projectcalico.org
spec:
scope: Cluster
group: crd.projectcalico.org
version: v1
names:
kind: BGPPeer
plural: bgppeers
singular: bgppeer

---
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
name: blockaffinities.crd.projectcalico.org
spec:
scope: Cluster
group: crd.projectcalico.org
version: v1
names:
kind: BlockAffinity
plural: blockaffinities
singular: blockaffinity

---
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
name: clusterinformations.crd.projectcalico.org
spec:
scope: Cluster
group: crd.projectcalico.org
version: v1
names:
kind: ClusterInformation
plural: clusterinformations
singular: clusterinformation

---
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
name: felixconfigurations.crd.projectcalico.org
spec:
scope: Cluster
group: crd.projectcalico.org
version: v1
names:
kind: FelixConfiguration
plural: felixconfigurations
singular: felixconfiguration

---
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
name: globalnetworkpolicies.crd.projectcalico.org
spec:
```



```
scope: Cluster
group: crd.projectcalico.org
version: v1
names:
kind: GlobalNetworkPolicy
plural: globalnetworkpolicies
singular: globalnetworkpolicy
shortNames:
- gnp

---
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
name: globalnetworksets.crd.projectcalico.org
spec:
scope: Cluster
group: crd.projectcalico.org
version: v1
names:
kind: GlobalNetworkSet
plural: globalnetworksets
singular: globalnetworkset

---
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
name: hostendpoints.crd.projectcalico.org
spec:
scope: Cluster
group: crd.projectcalico.org
version: v1
names:
kind: HostEndpoint
plural: hostendpoints
singular: hostendpoint

---
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
name: ipamblocks.crd.projectcalico.org
spec:
scope: Cluster
group: crd.projectcalico.org
version: v1
names:
kind: IPAMBlock
plural: ipamblocks
singular: ipamblock

---
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
name: ipamconfigs.crd.projectcalico.org
spec:
scope: Cluster
group: crd.projectcalico.org
version: v1
names:
kind: IPAMConfig
plural: ipamconfigs
singular: ipamconfig

---
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
```



```
metadata:
name: ipamhandles.crd.projectcalico.org
spec:
scope: Cluster
group: crd.projectcalico.org
version: v1
names:
kind: IPAMHandle
plural: ipamhandles
singular: ipamhandle

---
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
name: ippools.crd.projectcalico.org
spec:
scope: Cluster
group: crd.projectcalico.org
version: v1
names:
kind: IPPool
plural: ippools
singular: ippool

---
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
name: kubecontrollersconfigurations.crd.projectcalico.org
spec:
scope: Cluster
group: crd.projectcalico.org
version: v1
names:
kind: KubeControllersConfiguration
plural: kubecontrollersconfigurations
singular: kubecontrollersconfiguration
---
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
name: networkpolicies.crd.projectcalico.org
spec:
scope: Namespaced
group: crd.projectcalico.org
version: v1
names:
kind: NetworkPolicy
plural: networkpolicies
singular: networkpolicy

---
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
name: networksets.crd.projectcalico.org
spec:
scope: Namespaced
group: crd.projectcalico.org
version: v1
names:
kind: NetworkSet
plural: networksets
singular: networkset

---
---
# Source: calico/templates/rbac.yaml
```

```
# Include a clusterrole for the kube-controllers component,
# and bind it to the calico-kube-controllers serviceaccount.
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
name: calico-kube-controllers
rules:
# Nodes are watched to monitor for deletions.
- apiGroups: [""]
resources:
- nodes
verbs:
- watch
- list
- get
# Pods are queried to check for existence.
- apiGroups: [""]
resources:
- pods
verbs:
- get
# IPAM resources are manipulated when nodes are deleted.
- apiGroups: ["crd.projectcalico.org"]
resources:
- ippools
verbs:
- list
- apiGroups: ["crd.projectcalico.org"]
resources:
- blockaffinities
- ipamblocks
- ipamhandles
verbs:
- get
- list
- create
- update
- delete
# kube-controllers manages hostendpoints.
- apiGroups: ["crd.projectcalico.org"]
resources:
- hostendpoints
verbs:
- get
- list
- create
- update
- delete
# Needs access to update clusterinformations.
- apiGroups: ["crd.projectcalico.org"]
resources:
- clusterinformations
verbs:
- get
- create
- update
# KubeControllersConfiguration is where it gets its config
- apiGroups: ["crd.projectcalico.org"]
resources:
- kubecontrollersconfigurations
verbs:
# read its own config
- get
# create a default if none exists
- create
# update status
- update
# watch for changes
```



```
- watch
---
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
name: calico-kube-controllers
roleRef:
apiGroup: rbac.authorization.k8s.io
kind: ClusterRole
name: calico-kube-controllers
subjects:
- kind: ServiceAccount
name: calico-kube-controllers
namespace: kube-system
---
# Include a clusterrole for the calico-node DaemonSet,
# and bind it to the calico-node serviceaccount.
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
name: calico-node
rules:
# The CNI plugin needs to get pods, nodes, and namespaces.
- apiGroups: [""]
resources:
- pods
- nodes
- namespaces
verbs:
- get
- apiGroups: [""]
resources:
- endpoints
- services
verbs:
# Used to discover service IPs for advertisement.
- watch
- list
# Used to discover Typhas.
- get
# Pod CIDR auto-detection on kubeadm needs access to config maps.
- apiGroups: [""]
resources:
- configmaps
verbs:
- get
- apiGroups: [""]
resources:
- nodes/status
verbs:
# Needed for clearing NodeNetworkUnavailable flag.
- patch
# Calico stores some configuration information in node annotations.
- update
# Watch for changes to Kubernetes NetworkPolicies.
- apiGroups: ["networking.k8s.io"]
resources:
- networkpolicies
verbs:
- watch
- list
# Used by Calico for policy information.
- apiGroups: [""]
resources:
- pods
- namespaces
- serviceaccounts
verbs:
- list
```



```
- watch
# The CNI plugin patches pods/status.
- apiGroups: [""]
resources:
- pods/status
verbs:
- patch
# Calico monitors various CRDs for config.
- apiGroups: ["crd.projectcalico.org"]
resources:
- globalfelixconfigs
- felixconfigurations
- bgppeers
- globalbgpconfigs
- bgpconfigurations
- ippools
- ipamblocks
- globalnetworkpolicies
- globalnetworksets
- networkpolicies
- networksets
- clusterinformations
- hostendpoints
- blockaffinities
verbs:
- get
- list
- watch
# Calico must create and update some CRDs on startup.
- apiGroups: ["crd.projectcalico.org"]
resources:
- ippools
- felixconfigurations
- clusterinformations
verbs:
- create
- update
# Calico stores some configuration information on the node.
- apiGroups: [""]
resources:
- nodes
verbs:
- get
- list
- watch
# These permissions are only requried for upgrade from v2.6, and can
# be removed after upgrade or on fresh installations.
- apiGroups: ["crd.projectcalico.org"]
resources:
- bgpconfigurations
- bgppeers
verbs:
- create
- update
# These permissions are required for Calico CNI to perform IPAM allocations.
- apiGroups: ["crd.projectcalico.org"]
resources:
- blockaffinities
- ipamblocks
- ipamhandles
verbs:
- get
- list
- create
- update
- delete
- apiGroups: ["crd.projectcalico.org"]
resources:
- ipamconfigs
```



```
verbs:
- get
# Block affinities must also be watchable by confd for route aggregation.
- apiGroups: ["crd.projectcalico.org"]
resources:
- blockaffinities
verbs:
- watch
# The Calico IPAM migration needs to get daemonsets. These permissions can be
# removed if not upgrading from an installation using host-local IPAM.
- apiGroups: ["apps"]
resources:
- daemonsets
verbs:
- get

---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
name: calico-node
roleRef:
apiGroup: rbac.authorization.k8s.io
kind: ClusterRole
name: calico-node
subjects:
- kind: ServiceAccount
name: calico-node
namespace: kube-system

---
# Source: calico/templates/calico-node.yaml
# This manifest installs the calico-node container, as well
# as the CNI plugins and network config on
# each master and worker node in a Kubernetes cluster.
kind: DaemonSet
apiVersion: apps/v1
metadata:
name: calico-node
namespace: kube-system
labels:
k8s-app: calico-node
spec:
selector:
matchLabels:
k8s-app: calico-node
updateStrategy:
type: RollingUpdate
rollingUpdate:
maxUnavailable: 1
template:
metadata:
labels:
k8s-app: calico-node
annotations:
# This, along with the CriticalAddonsOnly toleration below,
# marks the pod as a critical add-on, ensuring it gets
# priority scheduling and that its resources are reserved
# if it ever gets evicted.
scheduler.alpha.kubernetes.io/critical-pod: ''
spec:
nodeSelector:
kubernetes.io/os: linux
hostNetwork: true
tolerations:
# Make sure calico-node gets scheduled on all nodes.
- effect: NoSchedule
operator: Exists
# Mark the pod as a critical add-on for rescheduling.
```



```
- key: CriticalAddonsOnly
operator: Exists
- effect: NoExecute
operator: Exists
serviceAccountName: calico-node
# Minimize downtime during a rolling upgrade or deletion; tell Kubernetes to do a "force
# deletion": https://kubernetes.io/docs/concepts/workloads/pods/pod/#termination-of-pods.
terminationGracePeriodSeconds: 0
priorityClassName: system-node-critical
initContainers:
# This container performs upgrade from host-local IPAM to calico-ipam.
# It can be deleted if this is a fresh installation, or if you have already
# upgraded to use calico-ipam.
- name: upgrade-ipam
image: calico/cni:v3.13.1
command: ["/opt/cni/bin/calico-ipam", "-upgrade"]
env:
- name: KUBERNETES_NODE_NAME
valueFrom:
fieldRef:
fieldPath: spec.nodeName
- name: CALICO_NETWORKING_BACKEND
valueFrom:
configMapKeyRef:
name: calico-config
key: calico_backend
volumeMounts:
- mountPath: /var/lib/cni/networks
name: host-local-net-dir
- mountPath: /host/opt/cni/bin
name: cni-bin-dir
securityContext:
privileged: true
# This container installs the CNI binaries
# and CNI network config file on each node.
- name: install-cni
image: calico/cni:v3.13.1
command: ["/install-cni.sh"]
env:
# Name of the CNI config file to create.
- name: CNI_CONF_NAME
value: "10-calico.conflist"
# The CNI network config to install on each node.
- name: CNI_NETWORK_CONFIG
valueFrom:
configMapKeyRef:
name: calico-config
key: cni_network_config
# Set the hostname based on the k8s node name.
- name: KUBERNETES_NODE_NAME
valueFrom:
fieldRef:
fieldPath: spec.nodeName
# CNI MTU Config variable
- name: CNI_MTU
valueFrom:
configMapKeyRef:
name: calico-config
key: veth_mtu
# Prevents the container from sleeping forever.
- name: SLEEP
value: "false"
volumeMounts:
- mountPath: /host/opt/cni/bin
name: cni-bin-dir
- mountPath: /host/etc/cni/net.d
name: cni-net-dir
securityContext:
privileged: true
```



```
# Adds a Flex Volume Driver that creates a per-pod Unix Domain Socket to allow Dikastes
# to communicate with Felix over the Policy Sync API.
- name: flexvol-driver
image: calico/pod2daemon-flexvol:v3.13.1
volumeMounts:
- name: flexvol-driver-host
mountPath: /host/driver
securityContext:
privileged: true
containers:
# Runs calico-node container on each Kubernetes node.  This
# container programs network policy and routes on each
# host.
- name: calico-node
image: calico/node:v3.13.1
env:
# Use Kubernetes API as the backing datastore.
- name: DATASTORE_TYPE
value: "kubernetes"
# Wait for the datastore.
- name: WAIT_FOR_DATASTORE
value: "true"
# Set based on the k8s node name.
- name: NODENAME
valueFrom:
fieldRef:
fieldPath: spec.nodeName
# Choose the backend to use.
- name: CALICO_NETWORKING_BACKEND
valueFrom:
configMapKeyRef:
name: calico-config
key: calico_backend
# Cluster type to identify the deployment type
- name: CLUSTER_TYPE
value: "k8s,bgp"
# Auto-detect the BGP IP address.
- name: IP
value: "autodetect"
# Enable IPIP
- name: CALICO_IPV4POOL_IPIP
value: "Always"
# Enable or Disable VXLAN on the default IP pool.
- name: CALICO_IPV4POOL_VXLAN
value: "Never"
# Set MTU for tunnel device used if ipip is enabled
- name: FELIX_IPINIPMTU
valueFrom:
configMapKeyRef:
name: calico-config
key: veth_mtu
# Set MTU for the VXLAN tunnel device.
- name: FELIX_VXLANMTU
valueFrom:
configMapKeyRef:
name: calico-config
key: veth_mtu
# The default IPv4 pool to create on startup if none exists. Pod IPs will be
# chosen from this range. Changing this value after installation will have
# no effect. This should fall within `--cluster-cidr`.
# - name: CALICO_IPV4POOL_CIDR
#   value: "192.168.0.0/16"
# Disable file logging so `kubectl logs` works.
- name: CALICO_DISABLE_FILE_LOGGING
value: "true"
# Set Felix endpoint to host default action to ACCEPT.
- name: FELIX_DEFAULTENDPOINTTOHOSTACTION
value: "ACCEPT"
# Disable IPv6 on Kubernetes.
```



```
- name: FELIX_IPV6SUPPORT
value: "false"
# Set Felix logging to "info"
- name: FELIX_LOGSEVERITYSCREEN
value: "info"
- name: FELIX_HEALTHENABLED
value: "true"
securityContext:
privileged: true
resources:
requests:
cpu: 250m
livenessProbe:
exec:
command:
- /bin/calico-node
- -felix-live
- -bird-live
periodSeconds: 10
initialDelaySeconds: 10
failureThreshold: 6
readinessProbe:
exec:
command:
- /bin/calico-node
- -felix-ready
- -bird-ready
periodSeconds: 10
volumeMounts:
- mountPath: /lib/modules
name: lib-modules
readOnly: true
- mountPath: /run/xtables.lock
name: xtables-lock
readOnly: false
- mountPath: /var/run/calico
name: var-run-calico
readOnly: false
- mountPath: /var/lib/calico
name: var-lib-calico
readOnly: false
- name: policysync
mountPath: /var/run/nodeagent
volumes:
# Used by calico-node.
- name: lib-modules
hostPath:
path: /lib/modules
- name: var-run-calico
hostPath:
path: /var/run/calico
- name: var-lib-calico
hostPath:
path: /var/lib/calico
- name: xtables-lock
hostPath:
path: /run/xtables.lock
type: FileOrCreate
# Used to install CNI.
- name: cni-bin-dir
hostPath:
path: /opt/cni/bin
- name: cni-net-dir
hostPath:
path: /etc/cni/net.d
# Mount in the directory for host-local IPAM allocations. This is
# used when upgrading from host-local to calico-ipam, and can be removed
# if not using the upgrade-ipam init container.
- name: host-local-net-dir
```

```
hostPath:
path: /var/lib/cni/networks
# Used to create per-pod Unix Domain Sockets
- name: policysync
hostPath:
type: DirectoryOrCreate
path: /var/run/nodeagent
# Used to install Flex Volume Driver
- name: flexvol-driver-host
hostPath:
type: DirectoryOrCreate
path: /usr/libexec/kubernetes/kubelet-plugins/volume/exec/nodeagent~uds
---

apiVersion: v1
kind: ServiceAccount
metadata:
name: calico-node
namespace: kube-system

---
# Source: calico/templates/calico-kube-controllers.yaml
# See https://github.com/projectcalico/kube-controllers
apiVersion: apps/v1
kind: Deployment
metadata:
name: calico-kube-controllers
namespace: kube-system
labels:
k8s-app: calico-kube-controllers
spec:
# The controllers can only have a single active instance.
replicas: 1
selector:
matchLabels:
k8s-app: calico-kube-controllers
strategy:
type: Recreate
template:
metadata:
name: calico-kube-controllers
namespace: kube-system
labels:
k8s-app: calico-kube-controllers
annotations:
scheduler.alpha.kubernetes.io/critical-pod: ''
spec:
nodeSelector:
kubernetes.io/os: linux
tolerations:
# Mark the pod as a critical add-on for rescheduling.
- key: CriticalAddonsOnly
operator: Exists
- key: node-role.kubernetes.io/master
effect: NoSchedule
serviceAccountName: calico-kube-controllers
priorityClassName: system-cluster-critical
containers:
- name: calico-kube-controllers
image: calico/kube-controllers:v3.13.1
env:
# Choose which controllers to run.
- name: ENABLED_CONTROLLERS
value: node
- name: DATASTORE_TYPE
value: kubernetes
readinessProbe:
exec:
command:
```



```
- /usr/bin/check-status
- -r

---

apiVersion: v1
kind: ServiceAccount
metadata:
name: calico-kube-controllers
namespace: kube-system

---
# Source: calico/templates/calico-etcd-secrets.yaml

---
# Source: calico/templates/calico-typha.yaml

---
# Source: calico/templates/configure-canal.yaml
```

1.7.3 Calicoctl Deployment YAML File

```
# Calico Version v3.13.1
# https://docs.projectcalico.org/releases#v3.13.1
# This manifest includes the following component versions:
#   calico/ctl:v3.13.1

apiVersion: v1
kind: ServiceAccount
metadata:
name: calicoctl
namespace: kube-system

---

apiVersion: v1
kind: Pod
metadata:
name: calicoctl
namespace: kube-system
spec:
nodeSelector:
kubernetes.io/os: linux
hostNetwork: true
serviceAccountName: calicoctl
containers:
- name: calicoctl
image: calico/ctl:v3.13.1
command: ["/bin/sh", "-c", "while true; do sleep 3600; done"]
env:
- name: DATASTORE_TYPE
value: kubernetes

---

kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
name: calicoctl
rules:
- apiGroups: [""]
resources:
- namespaces
- nodes
verbs:
- get
- list
- update
- apiGroups: [""]
```



```
resources:
- nodes/status
verbs:
- update
- apiGroups: [""]
resources:
- pods
- serviceaccounts
verbs:
- get
- list
- apiGroups: [""]
resources:
- pods/status
verbs:
- update
- apiGroups: ["crd.projectcalico.org"]
resources:
- bgppeers
- bgpconfigurations
- clusterinformations
- felixconfigurations
- globalnetworkpolicies
- globalnetworksets
- ippools
- kubecontrollersconfigurations
- networkpolicies
- networksets
- hostendpoints
- ipamblocks
- blockaffinities
- ipamhandles
- ipamconfigs
verbs:
- create
- get
- list
- update
- delete
- apiGroups: ["networking.k8s.io"]
resources:
- networkpolicies
verbs:
- get
- list

---

apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
name: calicoctl
roleRef:
apiGroup: rbac.authorization.k8s.io
kind: ClusterRole
name: calicoctl
subjects:
- kind: ServiceAccount
name: calicoctl
namespace: kube-system
```

2 Kube-OVN User Guide

2.1 Introduction

2.1.1 Kube-OVN Overview

Kube-OVN integrates the OVN-based (OVN is short for open virtual network) network virtualization with Kubernetes. It offers an advanced container network fabric for enterprises with the most functions and the easiest operation. The software is open sourced based on Apache 2.0. Kube-OVN 1.2.0 officially supports the ARM64 architecture and can natively run on TaiShan servers.

Kube-OVN supports the following functions:

- Binding subnets to namespaces: Each namespace can have a unique subnet (backed by an independent logical switch). Pods within a namespace will have IP addresses allocated from the current subnet. Multiple namespaces can share a subnet.
- Subnet isolation: Kube-OVN allows you to configure a subnet to deny any traffic from source IP addresses not within the same subnet. You can also filter traffic based on the IP address or IP address segment whitelist.
- Network policies: Kube-OVN supports Kubernetes network policies in high-performance OVN access control list (ACL) mode.
- Static IP addresses for workloads: Random or static IP addresses are allocated to workloads.
- Static IP addresses for pods: Static IP addresses and MAC addresses are allocated to pods.

- IP address reuse supported by StatefulSets: Within the lifecycle of a StatefulSet, pods reuse IP addresses by name. (The IP addresses can be randomly allocated for the first time and are fixed within the lifecycle.)
- Multi-NIC IP address management (IPAM): The IPAM container network interface (CNI) plugin within the cluster is supported. In addition to Kube-OVN, MACvlan, VLAN, and host-device can be deployed to make full use of the advantages of subnets and static IP address allocation.
- Dynamic and bidirectional QoS: Bidirectional bandwidth QoS management is supported, and the ingress/egress bandwidths of pods can be dynamically changed.
- Embedded load balancers: The high-performance distributed L2 load balancer embedded in the OVN is used to replace kube-proxy.
- Distributed gateways: Each node can function as a gateway to provide external network connections.
- Namespaced gateways: Each namespace can function as a gateway to provide external network connections.
- Direct connection to the external network: The IP addresses of pods are directly exposed to the external network. You can add a static route to an external router to divert the traffic of the container network segments to any host in the cluster.
- BGP support: The IP addresses of pods can be exposed to the external Border Gateway Protocol (BGP).
- Traffic mirror: Traffic between containers can be duplicated to facilitate monitoring, auditing, and fault diagnosis.
- VLAN support: The underlay virtual local area network (VLAN) mode is supported to obtain better performance and throughput.
- DPDK support: The Open vSwitch with the Data Plane Development Kit (OvS-DPDK) can run in pods.
- IPv6 support: Pods can be deployed in IPv6-only mode.
- Problem locating tool: Tools are provided to locate, monitor, and dump network problems.

2.1.2 Concepts

Kubernetes

Kubernetes is an open-source container orchestration engine provided by Google. It supports automatic deployment, large-scale scalability, and containerized application management. Kubernetes classifies network scenarios into the following types: communication between containers in the same pod, communication between pods, communication between pods and services, and communication between systems outside the cluster and services.

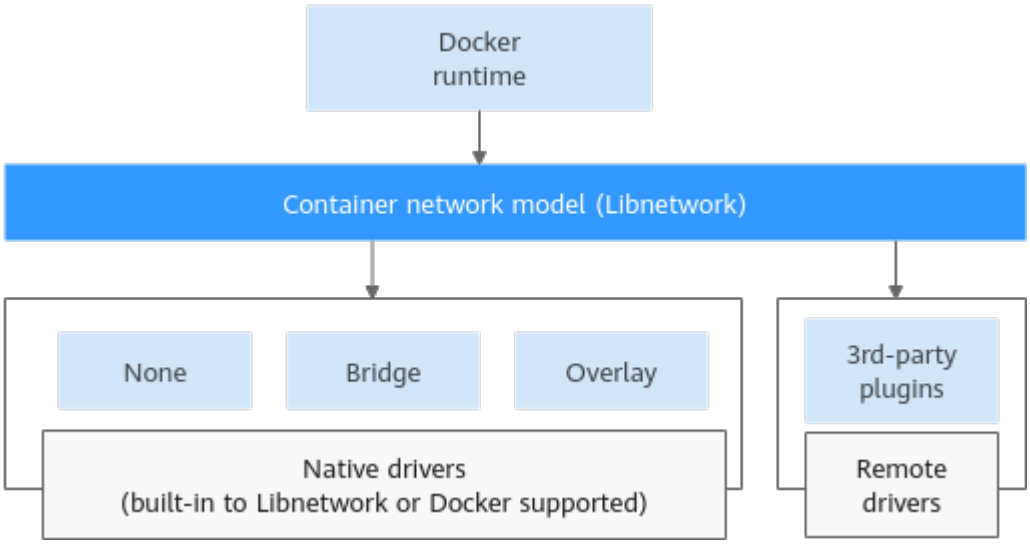
Pod and service resource objects use their own dedicated networks. The pod network is implemented through Kubernetes network plugin configuration (CNI model). The service network is specified by the Kubernetes cluster. The overall network model is implemented using external plugins. Therefore, you need to plan the network model and network deployment before deploying the Kubernetes cluster.

Container Network Model (CNM)

CNM is proposed by Docker and adopted by the Libnetwork project as the network model standard. It is used by common open-source network components such as Kuryr, OVN, Calico, and Weave for container network interconnection.

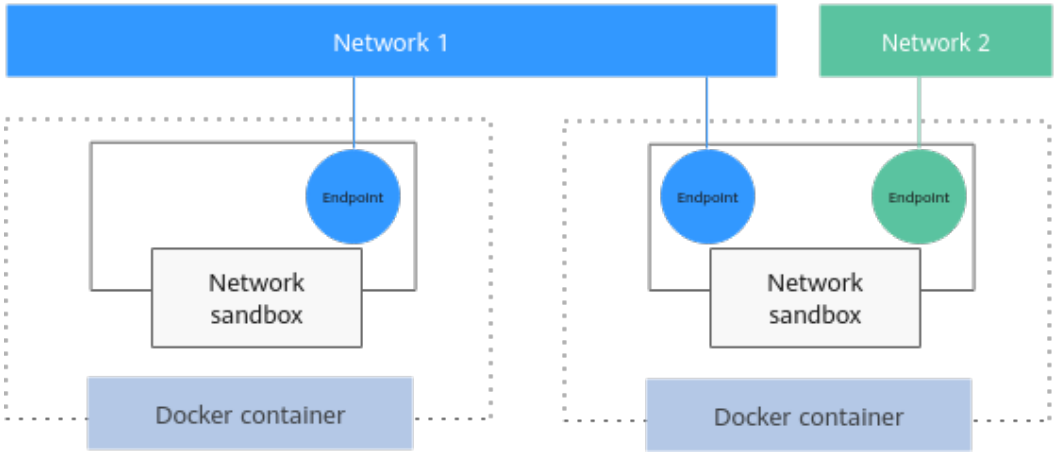
As shown in **Figure 2-1**, the CNM implementation Libnetwork provides interfaces between Docker daemons and network driver programs. The network controller is responsible for matching drivers with networks, and each driver is responsible for managing the network of the driver and provides services such as IPAM for the network. Drivers of the CNM can be native drivers (for built-in Libnetwork or network models supported by Docker) or third-party plugin drivers. The native drivers include None, Bridge, Overlay, and MACvlan. Third-party drivers provide more functions. In addition, the scope of a CNM driver can be defined as either a local scope (single-host mode) or a global scope (multi-host mode).

Figure 2-1 CNM drivers



Containers are connected through a series of network endpoints, as shown in **Figure 2-2**. A typical network interface exists in the form of a veth pair. One end of the veth pair is in the network sandbox of the container, and the other end is in the specified network. One network endpoint is added to only one network plane. Multiple network endpoints can exist in the network sandbox of a container.

Figure 2-2 CNM interfaces



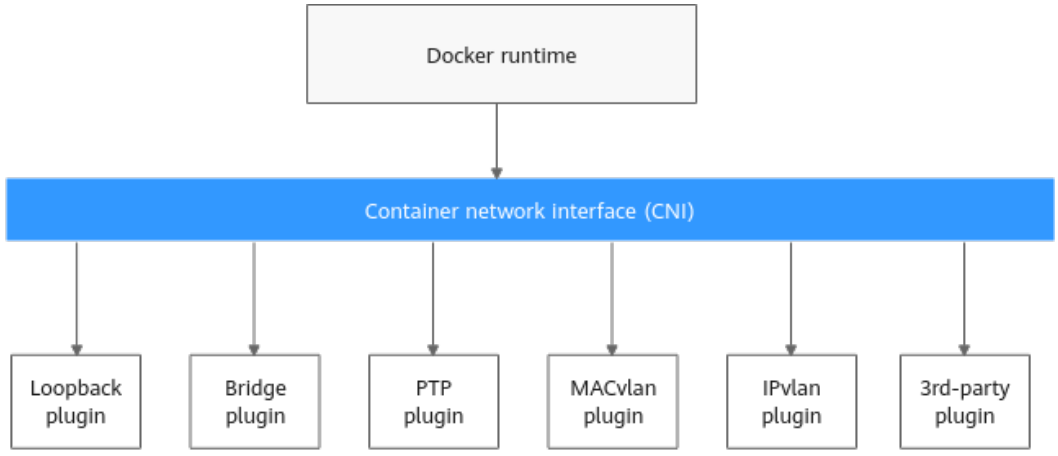
Container Network Interface (CNI)

CNI is proposed by CoreOS and adopted by Apache Mesos, Cloud Foundry, Kubernetes, Kurma, and rkt as the network model standard. CNI is used by common open-source network components such as Contiv Networking, Calico, and Weave for container network interconnection.

As shown in **Figure 2-3**, CNI is implemented based on a simplest standard, enabling network development engineers to implement protocol communication between containers and network plugins in a simple way.

Multiple network plugins can run in a container so that the container can connect to different network planes driven by different plugins. The network is defined in a JSON configuration file and is instantiated as a new namespace when CNI plugins are called.

Figure 2-3 CNI drivers



Open vSwitch

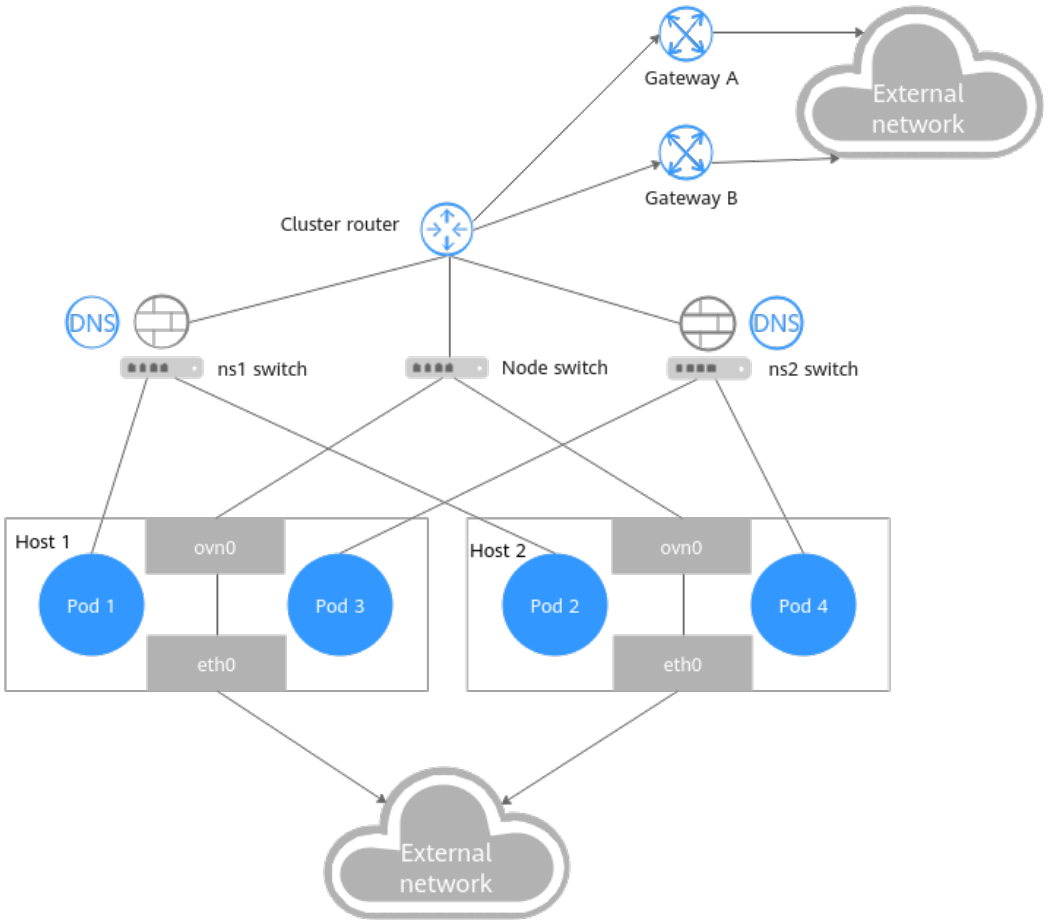
Open vSwitch (OVS) is a multi-layer switch software based on the Apache 2.0 open-source protocol. It aims to build a production environment quality switching platform that supports standard management interfaces, forward function interfaces, programmable plugins, and management control. OVN is a native

virtualized network solution provided by OVS. It uses existing OVS functions to implement large-scale and high-quality cluster management.

Kube-OVN network topology

Figure 2-4 shows the switches, cluster routers, and firewalls of Kube-OVN. They are deployed on all nodes in the cluster in distributed mode, and there is no single point of failure in the network topology of the cluster.

Figure 2-4 Kube-OVN network topology



2.1.3 Application Scenarios

When using the Kubernetes container orchestration engine to deploy services, you need to use the open-source Kube-OVN component to plan and deploy network planes. This document provides a Quick Start and detailed guidance for component deployment.

2.1.4 Accessing and Using Kube-OVN

You can access and use Kube-OVN by using the following tools during deployment and O&M:

- Plugin management tool provided by Kube-OVN, that is, kubectl ko

- Kubernetes management tool, that is, kubectl, which enables you to indirectly access, manage, and configure network components

2.2 Quick Start

2.2.1 Basic Installation

2.2.1.1 Environment Requirements

OS and Software Versions

Kube-OVN is a CNI-compliant network component that runs in the Kubernetes environment. **Table 2-1** lists the supported OS and software versions.

Table 2-1 Environment requirements

Item	Name	Version
OS	CentOS	7.5 or later
	Ubuntu	16.04 or later
Kernel	Kernel	3.10.0-898 or later
Docker	Docker Community	1.12.6 or later
	Docker CE	18.09.0 or later
Kubernetes	Kubernetes	1.11 or later

Network Plugins

Kube-OVN contains all the required CNI plugins. You do not need to install the CNI plugin separately.

To prevent the network from being affected by the rules and routes of other network plugins, you are advised to install Kube-OVN in a Kubernetes cluster that has no other network plugins installed.

If other network plugins have been installed in the cluster, delete them and the corresponding network configurations such as NICs, iptables rules, and routing rules. Ensure that no configuration files of other network plugins exist in the **/etc/cni/net.d/** directory.

2.2.1.2 One-Click Installation (Recommended)

Prerequisites

1. The Docker and Kubernetes components (kubeadm, kubectl, and kubelet) have been installed on the node to be deployed.

2. The node to be deployed can properly pull Docker images.

Procedure

Step 1 Set the master Kubernetes nodes during initialization.

NOTE

In this section, the **10.16.0.0/16** network segment is used as the network driver classless inter-domain routing (CIDR) and the default gateway is used as the network broadcast address. If you need to specify another network segment, modify the commands accordingly.

```
kubeadm init --pod-network-cidr=10.16.0.0/16
mkdir -p $HOME/.kube
cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
chown $(id -u):$(id -g) $HOME/.kube/config
```

After the initialization is complete, information in **Figure 2-5** and **Figure 2-6** are displayed. Check whether the pod information of the Kubernetes cluster is normal and whether the node status is **NotReady**. Back up the **kubeadm join** command in the output for future use. Then, deploy the network plane.

Figure 2-5 Successful initialization of the master Kubernetes nodes

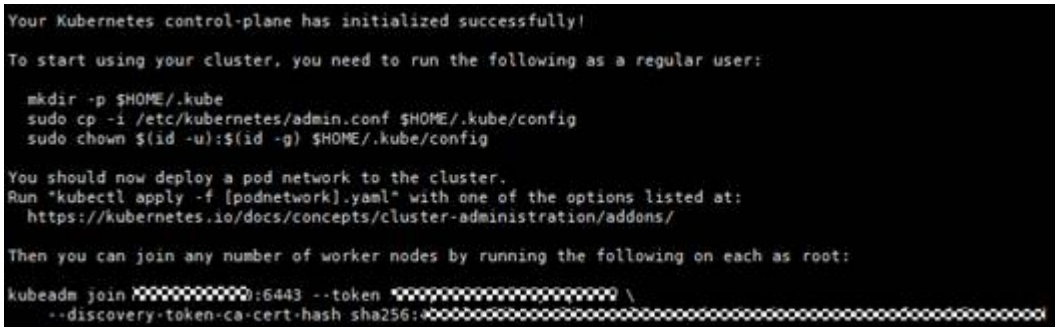
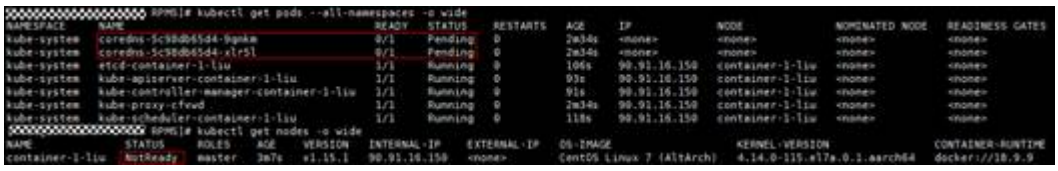


Figure 2-6 Initialization status of the master Kubernetes nodes



Step 2 Obtain and edit the one-click installation script. For details about how to obtain the script, see One-Click Installation Script.

Download and edit the Kube-OVN one-click installation script **install.sh** and deploy it on the Kunpeng processor platform. You need to add the suffix **-arm** to the **VERSION** field to support proper running on the ARM64 platform. Set the following variables in the file to expected values based on the site requirements. Set **NETWORK_TYPE** to **geneve** or **vlan** based on site requirements.

CAUTION

The Kube-OVN version and features are evolving rapidly. It is recommended that the Kube-OVN version obtained by the one-click installation script be the same as the image version. Otherwise, Kube-OVN may fail to be deployed and started in one-click mode.

```
REGISTRY="kubeovn"
NAMESPACE="kube-system"        # The ns to deploy kube-ovn
POD_CIDR="10.16.0.0/16"        # Do NOT overlap with NODE/SVC/JOIN CIDR
SVC_CIDR="10.96.0.0/12"        # Do NOT overlap with NODE/POD/JOIN CIDR
JOIN_CIDR="100.64.0.0/16"      # Do NOT overlap with NODE/POD/SVC CIDR
LABEL="node-role.kubernetes.io/master" # The node label to deploy OVN DB
IFACE=""                       # The nic to support container network, if empty will use the nic that the default
route use
NETWORK_TYPE="geneve"          # geneve or vlan
VERSION="v1.2.1-arm"
```

Step 3 Use the **install.sh** script to deploy cluster network components.

```
sh install.sh
```

After the deployment is complete, the coredns service is in the **Running** state and nodes are in the **Ready** state, as shown in **Figure 2-7** and **Figure 2-8**.

Figure 2-7 Pods after Kube-OVN is deployed

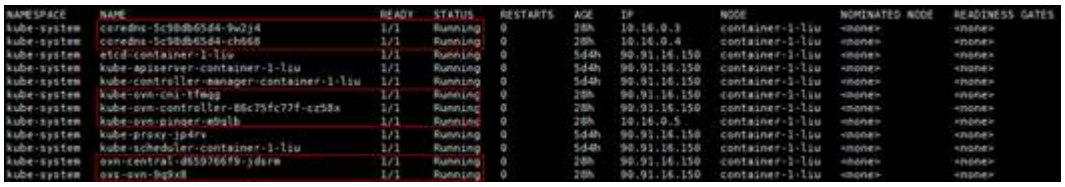


Figure 2-8 Node status after Kube-OVN is deployed

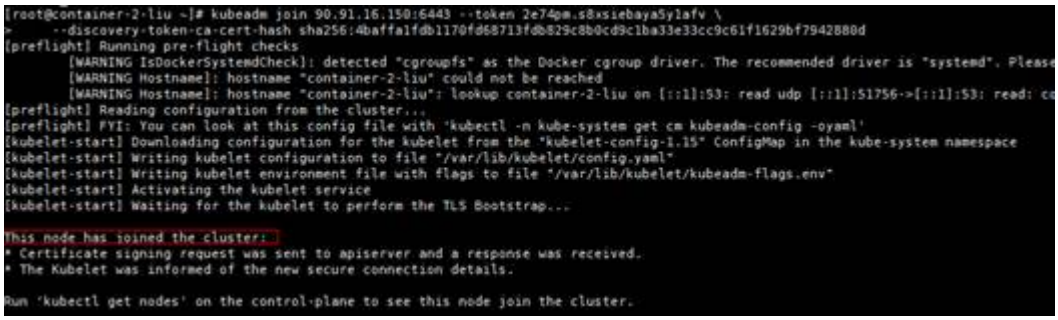
Step 4 Add nodes to the cluster.

On other Kubernetes nodes to be deployed, run the **kubeadm join** command backed up in **Step 1** to add the nodes to be deployed to the Kubernetes cluster.

```
kubeadm join <master-ip:port> --token <your-token> \
--discovery-token-ca-cert-hash sha256:<your-sha256-ca>
```

After the cluster nodes are added, "This node has joined the cluster" is displayed, as shown in **Figure 2-9**. The Kube-OVN networking procedure is complete.

Figure 2-9 Successful adding of cluster nodes



----End

2.2.1.3 Manual Installation

Prerequisites

1. The Docker and Kubernetes components (kubeadm, kubectl, and kubelet) have been installed on the node to be deployed.
2. The node to be deployed can properly pull Docker images.

Procedure

Step 1 Set the master Kubernetes nodes during initialization.

NOTE

In this section, the **10.16.0.0/16** network segment is used as the network driver classless CIDR and the default gateway is used as the network broadcast address. If you need to specify another network segment, modify the commands accordingly.

```
kubeadm init --pod-network-cidr=10.16.0.0/16
mkdir -p $HOME/.kube
cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
chown $(id -u):$(id -g) $HOME/.kube/config
```

After the initialization is complete, information in **Figure 2-10** and **Figure 2-11** are displayed. Check whether the pod information of the Kubernetes cluster is normal and whether the node status is **NotReady**. Back up the **kubeadm join** command in the output for future use. Then, deploy the network plane.

Figure 2-10 Successful initialization of the master Kubernetes nodes

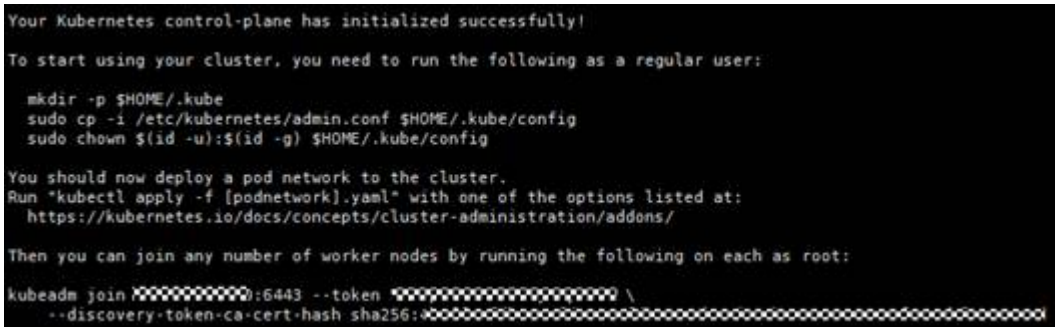
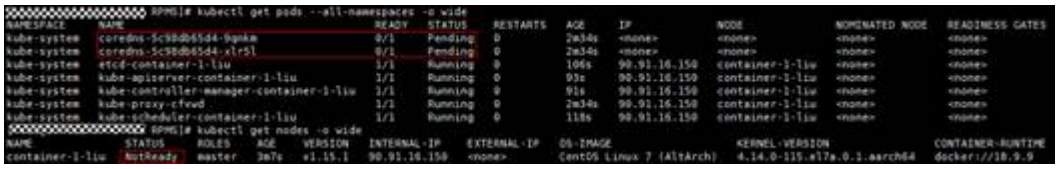


Figure 2-11 Initialization status of the master Kubernetes nodes



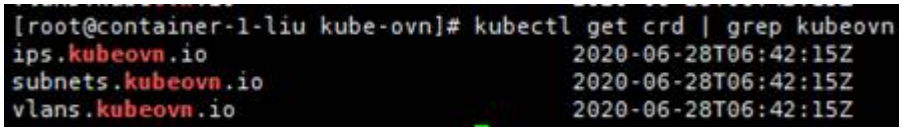
Step 2 Deploy custom resource definition (CRD) resources.

Kube-OVN creates three types of CRD resources, that is, subnets, IP addresses, and VLANs, to facilitate network management. Create the **crd.yaml** file by referring to **YAML File for CRD Deployment**. Then, run the **kubectl** command to create CRD resources.

```
kubectl apply -f crd.yaml
```

After the execution is complete, run the **kubectl get crd** command to view the result. The information shown in **Figure 2-12** is displayed.

Figure 2-12 Successful deployment of CRD resources



Step 3 Label the node where the ovndb resides.

Specify the master OVN node for Kube-OVN to deploy the ovndb for storing data persistently on the hard drive of the host machine.

```
kubectl label node <node-name to deploy ovndb> kube-ovn/role=master
```

After specifying the master OVN node, you can run the **kubectl get node --show-labels** command to view the label.

Figure 2-13 Node labels

Step 4 Deploy the OVN.

The underlying network of Kube-OVN depends on the OVS and OVN provided by the Open vSwitch community. Create the **ovn.yaml** file by referring to **YAML File for OVN Deployment**. Then, run the **kubectl** command to deploy the OVS and OVN.

```
kubectl apply -f ovn.yaml
```

After the execution is complete, run the **kubectl get pods -A** command to view pod status. The information shown in **Figure 2-14** is displayed.

Figure 2-14 Successful deployment of the OVN

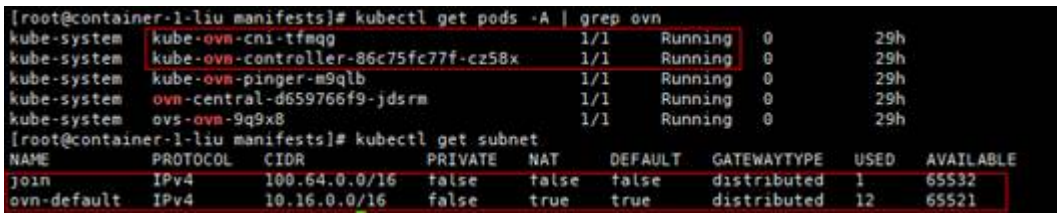
Step 5 Deploy the Kube-OVN-Controller and CNI Server.

By default, Kube-OVN uses 10.16.0.0/16 as the default subnet. 100.64.0.1/16 is used as the subnet for communication between hosts and pods, the master NIC of Kubernetes nodes is used for pod traffic, and the traffic mirroring mode is enabled. Before the deployment, obtain and edit the **kube-ovn.yaml** file by referring to **YAML File for Kube-OVN Deployment**. Then, run the **kubectl** command to deploy Kube-OVN.

```
kubectl apply -f kube-ovn.yaml
```

After the execution is complete, run the **kubectl get pods -A** command to view pod status, and run the **kubectl get subnet** command to view subnets automatically created. The information shown in **Figure 2-15** is displayed.

Figure 2-15 Successful deployment of Kube-OVN



Step 6 Install the kubectl plugin.

Kube-OVN provides the kubectl plugin for better monitoring network quality and diagnosing faults. You are advised to install the plugin.

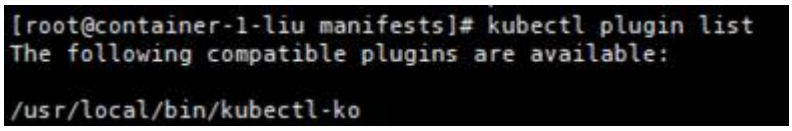
Save the code provided in **kubectl Plugin** as the **kubectl-ko** file. Then, copy the file to a directory (for example, **/usr/local/bin**) in **$PATH**, and grant the execute permission on the file.

```
cp kubectl-ko /usr/local/bin/kubectl-ko
chmod +x /usr/local/bin/kubectl-ko
```

After the execution is complete, run the **kubectl** command to check the plugin status.

```
kubectl plugin list
```

The command output is as follows:



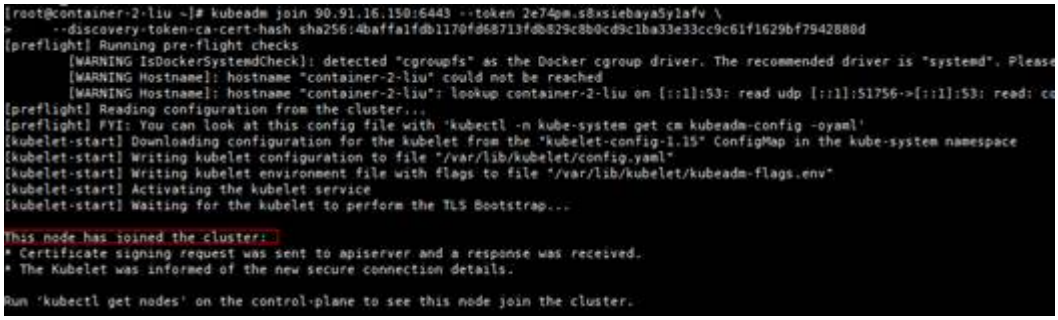
Step 7 Add nodes to the cluster.

On other Kubernetes nodes to be deployed, run the **kubeadm join** command backed up in **Step 1** to add the nodes to be deployed to the Kubernetes cluster.

```
kubeadm join <master-ip:port> --token <your-token> \
--discovery-token-ca-cert-hash sha256:<your-sha256-ca>
```

After the cluster nodes are added, "This node has joined the cluster" is displayed, as shown in **Figure 2-16**. The Kube-OVN networking procedure is complete.

Figure 2-16 Successful adding of cluster nodes



----End

2.2.2 Advanced Installation

2.2.2.1 Configuring VLAN Support

By default, Kube-OVN uses Geneve to encapsulate cross-host traffic and abstracts a virtual overlay network from the infrastructure. In addition, Kube-OVN supports the VLAN function since version 1.2.0. In scenarios sensitive to performance and throughput, an underlay network in VLAN mode is supported. The container network can be directly connected to physical switches through the VLAN to achieve better performance and throughput.

To use the VLAN mode, a host must have an NIC dedicated for the container network. The NIC port on the switch must work in trunk mode to allow 802.1Q data packets to pass through. Currently, Geneve or VLAN is a global option, and all containers must work in the same mode.

NOTE

The VLAN mode requires that an independent NIC be provided for the Kube-OVN container network. The VLAN mode is not recommended for a single network plane.

The following is an example of configuring VLAN support:

Step 1 Modify the installation script.

In the script, set **NETWORK_TYPE** to **VLAN** and **VLAN_INTERFACE_NAME** to the corresponding host NIC, and deploy the cluster in the original mode.

Step 2 Create a VLAN.

```
kubectl create
```

Create the following VLAN:

```
apiVersion: kubeovn.io/v1
kind: Vlan
metadata:
  name: product
spec:
  vlanId: 10
```

Step 3 Create a namespace.

```
kubectl create
```

Create the following namespace:


```
apiVersion: v1
kind: Namespace
metadata:
  name: product
  labels:
    name: product
```

Step 4 Create subnets and bind them to the VLAN.

```
kubectl create
```

The command will create the following subnets. Multiple subnets can be bound to the same VLAN.

```
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
  name: product
spec:
  cidrBlock: 10.100.0.0/16
  default: false
  gateway: 10.100.0.1
  gatewayType: distributed
  natOutgoing: true
  vlan: product
  namespaces:
    - product
```

Step 5 Create a pod.

Run the following command to deploy the nginx pod in the namespace created in **Step 3** and allocate IP addresses to the pod in the corresponding VLAN:

```
kubectl run samplepod --image=nginx --namespace=product
```

----End

2.2.2.2 Deploying HA

The Kube-OVN HA involves two components: ovndb and Kube-OVN-Controller.

The HA modes of the two components are different. The ovndb uses Raft for distributed consistency to achieve the cluster HA of the active-active mode. The Kube-OVN-Controller needs to process the status and events in the cluster. Each event can have only one working instance. Therefore, the HA of the leader-election mode is used.

Deploying HA for the ovndb

Step 1 Add nodes for deploying the ovndb. An odd number (1, 3, 5...) of nodes are recommended.

```
kubectl label node <node-name to deploy ovndb> kube-ovn/role=master
```

Step 2 Change the value of **replicas** in the **ovn-central deployment** section in the **ovn.yaml** file to the number of nodes configured in the previous step.

```
---
kind: Deployment
apiVersion: apps/v1
metadata:
  name: ovn-central
  namespace: kube-system
  annotations:
    kubernetes.io/description: |
      OVN components: northd, nb and sb.
spec:
  replicas: 1
  strategy:
    rollingUpdate:
      maxSurge: 0%
      maxUnavailable: 100%
    type: RollingUpdate
```

Step 3 Deploy the modified **ovn.yaml** file.

```
kubectl apply -f ovn.yaml
```

After ovn-central pods enter the **Ready** state, HA deployment for the ovndb is complete.

----End

Deploying HA for the Kube-OVN-Controller

The Kube-OVN-Controller implements the leader-election process. You only need to increase the value of **replicas** to achieve HA.

Change the value of **replicas** in the **kube-ovn-controller deployment** section in the **kube-ovn.yaml** file and run the **kubectl apply** command.

```
kubectl apply -f kube-ovn.yaml
```

After the Kube-OVN-Controller enters the **Ready** state, HA deployment for the Kube-OVN-Controller is complete.


```
---
kind: Deployment
apiVersion: apps/v1
metadata:
  name: ovn-central
  namespace: kube-system
  annotations:
    kubernetes.io/description: |
      OVN components: northd, nb and sb.
spec:
  replicas: 1
  strategy:
    rollingUpdate:
      maxSurge: 0%
      maxUnavailable: 100%
    type: RollingUpdate
```

2.2.2.3 Configuring the Built-in Subnets

Two built-in subnets are configured during Kube-OVN installation, as shown in **Figure 2-17**.

- Default subnet: default subnet used for allocating IP addresses to pods. The default CIDR is 10.16.0.0/16.
- Node subnet: special subnet for communication between nodes and pods. The default CIDR is 100.64.0.1/16.

Figure 2-17 Built-in subnets of Kube-OVN

```
[root@container-1-liu manifests]# kubectl get subnet
NAME          PROTOCOL   CIDR            PRIVATE   NAT     DEFAULT   GATEWAYTYPE   USED   AVAILABLE
join          IPv4       100.64.0.0/16   false     false   false     distributed   1      65532
ovn-default   IPv4       10.16.0.0/16    false     true    true      distributed   12     65521
```

During the installation, you can modify the Kube-OVN-Controller configuration in the **kube-ovn.yaml** file. Note that the two subnets cannot conflict with the CIDRs of the existing host network and service (SVC) network.

```
containers:
  - name: kube-ovn-controller
    image: "kubeovn/kube-ovn:v1.2.1-arm"
    imagePullPolicy: IfNotPresent
    command:
    - /kube-ovn/start-controller.sh
    args:
    - --default-cidr=10.16.0.0/16
    - --default-gateway=10.16.0.1
    - --node-switch-cidr=100.64.0.0/16
```

2.2.2.4 Selecting the Host NIC, MTU, and Traffic Mirrors

If the host machine has multiple NICs, Kube-OVN selects the NIC of the default route as the NIC for cross-node communication between containers by default and establishes the corresponding Geneve tunnel.

Extra payloads are occupied when the Geneve tunnel is used for overlay network encapsulation. When creating a container NIC, Kube-OVN adjusts the maximum transmission unit (MTU) of the container NIC based on the MTU of the selected NIC. The default MTU is the MTU of the host NIC minus 100.

By default, Kube-OVN creates a mirror0 NIC on each node to copy the network traffic of all containers on the current node. You can use tcpdump and other tools to analyze the traffic.

If you need to customize the preceding functions, you can configure **cni-server** in the **kube-ovn.yaml** file.

```
      containers:
      - name: cni-server
        image: "kubeovn/kube-ovn:v1.2.1-arm"
        imagePullPolicy: IfNotPresent
        command:
          - sh
          - /kube-ovn/start-cniserver.sh
        args:
          - --iface=eth1
          - --mtu=1333
          - --enable-mirror=true
```


2.2.3 Uninstallation

If you need to delete Kube-OVN and replace other network plugins, perform the following steps to delete Kube-OVN components and OVS configuration to prevent interference to other network plugins.

Step 1 Use the cleanup script to uninstall the resources created by Kubernetes.

Obtain the script described in Cleanup Script, grant the execute permission on the script, and run the following command on the master Kubernetes nodes:

```
bash cleanup.sh
```

Step 2 Delete the configuration of the ovsdb and Open vSwitch.

```
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
```

Step 3 Restart the nodes and ensure that the corresponding NIC information, iptables rules, and ipset rules are deleted.

----End

2.3 Service Deployment

2.3.1 Recommended Deployment Modes

Deployment Mode	Features and Application Scenarios	Kube-OVN Behavior
Pod	Minimum deployment unit.	Kube-OVN randomly allocates an IP address in a subnet. You can also specify a static IP address and a MAC address in the YAML configuration file.
Deployment	Pods and ReplicaSets.	Kube-OVN randomly allocates an IP address in a subnet. You can also specify a static IP address in the YAML configuration file.

Deployment Mode	Features and Application Scenarios	Kube-OVN Behavior
StatefulSet	Manages workloads of stateful applications and provides static IDs and uniqueness assurance. The IDs cannot be replaced with each other. After scheduling, the IDs remain unchanged. StatefulSets are applicable to scenarios that require stable network IDs, persistent storage, orderly deployment, and automatic rolling update.	IP addresses are allocated in sequence in a subnet. You can specify static IP addresses in the YAML configuration file. The network information remains unchanged within the lifecycle of a StatefulSet. During pod update or deletion, the logical switch port of the OVN is not deleted, and the new pod reuses the old interface information. Therefore, the IP address, MAC address, and other network information can be reused.
DaemonSet	Ensures that a pod copy runs on all or certain nodes. When a node is added or deleted, the pod is automatically added or deleted. DaemonSet is applicable to storage, logs, and monitoring services.	Kube-OVN randomly allocates an IP address in a subnet. You can also specify a static IP address in the YAML configuration file.
Job	Creates one or more pods to run a task, and deletes all pods after the task is detected as completed.	Kube-OVN randomly allocates an IP address in a subnet. You can also specify a static IP address in the YAML configuration file.
CronJob	Time-based scheduling task	Kube-OVN randomly allocates an IP address in a subnet. You can also specify a static IP address in the YAML configuration file.

2.3.2 Static IP Addresses and MAC Addresses for Pods

By default, Kube-OVN allocates an IP address and an MAC addresses to a pod based on the subnet to which the namespace where the pod resides belongs. You can specify the IP address and MAC address by setting the **ovn.kubernetes.io/ip_address** and **ovn.kubernetes.io/mac_address** fields in the **annotations** section in the YAML configuration file when creating the pod.

Example:

```
apiVersion: v1
kind: Pod
metadata:
  name: static-ip
```



```
  namespace: ls1
  annotations:
    ovn.kubernetes.io/ip_address: 10.16.0.15
    ovn.kubernetes.io/mac_address: 00:00:00:53:6B:B6
spec:
  containers:
  - name: static-ip
    image: nginx:alpine
```

NOTE

When specifying the IP address and MAC address for a pod, note that:

- The specified IP address must be within the CIDR of the subnet to which the namespace where the pod resides belongs.
- The specified IP address and MAC address cannot conflict with used IP addresses and MAC addresses.
- You can specify only the IP address or the MAC address.

2.3.3 Static IP Addresses for Workloads

Kube-OVN allows you to specify static IP addresses for workloads (Deployment, StatefulSet, DaemonSet, Job, and CronJob) by setting the **ovn.kubernetes.io/ip_pool** field in the **annotations** section in the YAML configuration file. A pod under a workload automatically selects the IP addresses specified by the **ovn.kubernetes.io/ip_pool** field. In addition, Kube-OVN can ensure that the IP addresses do not conflict with other IP addresses.

Example:

```
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
    spec:
      containers:
      - name: backend
        image: nginx:alpine
```

NOTE

When specifying IP addresses for a workload, note that:

- The specified IP addresses must be within the CIDR of the subnet to which the namespace where the pod resides belongs.
- The specified IP addresses cannot conflict with used IP addresses.
- If the number of specified IP addresses is less than the value of **replicas**, extra pods cannot be created. In this case, adjust the number of IP addresses specified in the **ovn.kubernetes.io/ip_pool** field based on the workload update policy and capacity expansion plan.

2.3.4 Static IP Addresses for StatefulSets

Same as other workloads, you can specify the IP address used by a pod by setting the **ovn.kubernetes.io/ip_pool** field. StatefulSets are mainly used for stateful services and have higher requirements on network fixation. Therefore, Kube-OVN provides the following enhancements:

1. IP addresses specified in the **ovn.kubernetes.io/ip_pool** field are allocated to pods in sequence.
2. During the update or deletion of a StatefulSet, **Logical_switch_port** stored in the OVN is not deleted. The newly generated pod directly reuses the old interface information. Therefore, this pod can reuse IP addresses, MAC addresses, and other network information to achieve the state retention function similar to that of StatefulSet Volumn.
3. If **ovn.kubernetes.io/ip_pool** is not specified for a StatefulSet during the update or deletion of this StatefulSet, an IP address and a MAC address are randomly allocated to a pod when the pod is created for the first time. After that, the network information remains fixed throughout the lifecycle of the StatefulSet.

Example:

```
apiVersion: apps/v1beta1
kind: StatefulSet
metadata:
  name: web
  namespace: ls1
spec:
  serviceName: "nginx"
  replicas: 2
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:alpine
```

NOTE

When specifying IP addresses for a StatefulSet, note that:

- The specified IP addresses must be within the CIDR of the subnet to which the namespace where the pod resides belongs.
- The specified IP addresses cannot conflict with used IP addresses.
- If the number of specified IP addresses is less than the value of **replicas**, extra pods cannot be created. In this case, adjust the number of IP addresses specified in the **ovn.kubernetes.io/ip_pool** field based on the workload update policy and capacity expansion plan.

2.4 Subnet Management

2.4.1 Default Subnet

Kube-OVN organizes IP addresses by subnet. Each namespace can belong to different subnets. Pods under a namespace automatically obtain IP addresses from

the corresponding subnet and share the network configuration (such as the CIDR, gateway type, access control, and NAT control) of the subnet.

Kube-OVN has a built-in default subnet. IP addresses are automatically allocated from the default subnet to all the namespaces without explicit declaration of subnet belonging, and the namespaces use the network configuration of the default subnet. The default subnet uses a distributed gateway and performs network address translation (NAT) on the outbound traffic. The behavior is the same as the default behavior of Flannel. Users can use most network functions without additional configurations.

You can change the configuration of the default subnet during the installation. For details, see **2.2.2.3 Configuring the Built-in Subnets**.

Run the following command to view the default subnet:

```
kubectl get subnet ovn-default -o yaml
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
  creationTimestamp: "2020-06-28T06:42:32Z"
  finalizers:
  - kube-ovn-controller
  generation: 1
  name: ovn-default
  resourceVersion: "750199"
  selfLink: /apis/kubeovn.io/v1/subnets/ovn-default
  uid: e3a9162e-ad11-4bd0-9962-db6d6d26b01e
spec:
  cidrBlock: 10.16.0.0/16
  default: true
  excludeIps:
  - 10.16.0.1
  gateway: 10.16.0.1
  gatewayNode: ""
  gatewayType: distributed
  natOutgoing: true
  private: false
  protocol: IPv4
  provider: ovn
  underlayGateway: false
status:
  activateGateway: ""
  availableIPs: 65517
  conditions:
  - lastTransitionTime: "2020-06-28T06:42:32Z"
    lastUpdateTime: "2020-06-28T06:42:33Z"
    reason: ResetLogicalSwitchAclSuccess
    status: "True"
    type: Validated
  - lastTransitionTime: "2020-06-28T06:42:33Z"
    lastUpdateTime: "2020-06-28T06:42:33Z"
    reason: ResetLogicalSwitchAclSuccess
    status: "True"
    type: Ready
  - lastTransitionTime: "2020-06-28T06:42:32Z"
    lastUpdateTime: "2020-06-28T06:42:32Z"
    message: Not Observed
    reason: Init
    status: Unknown
    type: Error
  usingIPs: 16
```


2.4.2 Node Subnet

According to the Kubernetes network specifications, a node can directly communicate with all pods. To achieve this goal, Kube-OVN creates a join subnet and creates a virtual NIC ovn0 on each node to connect to the join subnet. Hosts can communicate with pods through the subnet.

You can change the configuration of the node subnet during the installation. For details, see **2.2.2.3 Configuring the Built-in Subnets**.

- Run the following command to view the node subnet:

```
kubectl get subnet join -o yaml
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
  creationTimestamp: "2020-06-28T06:42:32Z"
  finalizers:
  - kube-ovn-controller
  generation: 2
  name: join
  resourceVersion: "749434"
  selfLink: /apis/kubeovn.io/v1/subnets/join
  uid: 3179fdc6-56a9-4211-a4c2-d922f4463adb
spec:
  cidrBlock: 100.64.0.0/16
  default: false
  excludeIps:
  - 100.64.0.1
  gateway: 100.64.0.1
  gatewayNode: ""
  gatewayType: distributed
  natOutgoing: false
  private: false
  protocol: IPv4
  provider: ovn
  underlayGateway: false
status:
  activateGateway: ""
  availableIPs: 65530
  conditions:
  - lastTransitionTime: "2020-06-28T06:42:33Z"
    lastUpdateTime: "2020-06-28T06:42:33Z"
    reason: ResetLogicalSwitchAclSuccess
    status: "True"
    type: Validated
  - lastTransitionTime: "2020-06-28T06:42:33Z"
    lastUpdateTime: "2020-06-28T06:42:33Z"
    reason: ResetLogicalSwitchAclSuccess
    status: "True"
    type: Ready
  - lastTransitionTime: "2020-06-28T06:42:33Z"
    lastUpdateTime: "2020-06-28T06:42:33Z"
    message: Not Observed
    reason: Init
    status: Unknown
    type: Error
  usingIPs: 3
```

- Run the following command on each node to view the ovn0 NIC:

```
ifconfig ovn0
ovn0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1400
        inet 100.64.0.2  netmask 255.255.0.0  broadcast 100.64.255.255
        inet6 fe80::200:ff:fec6:936d  prefixlen 64  scopeid 0x20<link>
        ether 00:00:00:c6:93:6d  txqueuelen 1000  (Ethernet)
        RX packets 5673864  bytes 530785977 (506.1 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 5243874  bytes 8069050878 (7.5 GiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```


2.4.3 Customized Subnet

In Kube-OVN, IP addresses are managed by subnet network segment. One or more namespaces can be bound to a subnet. Pods under these namespaces automatically obtain IP addresses from the subnet and share the network configuration (such as CIDR, gateway type, access control, and NAT control) of the subnet.

Run the following command to create a subnet:

```
cat <<EOF | kubectl create -f -
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
  name: ls1
spec:
  protocol: IPv4
  cidrBlock: 10.66.0.0/16
  excludeIps:
  - 10.66.0.1..10.66.0.10
  gateway: 10.66.0.1
  namespaces:
  - ls1
EOF
```

2.4.4 Subnet Access Control

By default, subnets created by Kube-OVN can communicate with each other, and pods can access external networks through gateways.

To control access between subnets, set **private** to **true** in the subnet CRD. In this way, the subnet is isolated from other subnets and external networks, and can only communicate with internal subnets. You can configure the whitelist by setting the **allSubnets** field in the YAML configuration file of Kube-OVN.

Access control can be further implemented using the network policy of Kubernetes. Kube-OVN implements network policy rules, which have higher priority than the access control settings in the subnet CRD.

The following is an example of configuring access control for a subnet.

```
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
  name: subnet-acl
spec:
  protocol: IPv4
  default: false
  namespaces:
  - ns1
  - ns2
  cidrBlock: 10.69.0.0/16
  gateway: 100.64.0.1
  excludeIps:
  - 100.64.0.1
  private: true
  allowSubnets:
  - 10.16.0.0/16
  - 10.18.0.0/16
```

2.4.5 Egress Gateway Configuration

Pods in the Kube-OVN network access the network outside the cluster through gateways. Currently, two types of gateways are supported. You can adjust the gateway type in the subnet.

- Distributed gateway

  Distributed gateways are the default gateway type of subnets. Each node functions as the gateway for the pods on the current node to access the external network. Data packets are routed to the host network stack through the ovn0 NIC of the local host, and then to the external network based on the routing rules of the host. If **natOutgoing** is set to **true**, a pod uses the IP address of the current host machine to access the external network.

  The following is an example of configuration on a subnet.

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

- Centralized gateway

  If you want to use a static IP address to access the external network from a subnet for security operations such as auditing and whitelisting, you can configure a centralized gateway in the subnet. In centralized gateway mode, data packets of a pod are first routed to the ovn0 NIC of a specified node and then to the external network based on the routing rules of the host. If **natOutgoing** is set to **true**, a pod uses the IP address of a specified host machine to access the external network.

  The following is an example of configuration on a subnet.

  ```
  apiVersion: kubeovn.io/v1
  kind: Subnet
  metadata:
    name: centralized
  spec:
    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
  ```

2.4.6 IP Addresses of Pods Exposed to the External Network

In Kube-OVN, the IP address of a pod can be directly exposed to the external network through a static route. In this case, the **natOutgoing** field of the subnet where the pod is located must be set to **false** to disable the NAT mapping for outbound traffic.

In addition, check whether there is a drop rule in the Forward link of iptables on the host node. For the ovn0 NIC and the default outbound NIC, enable the Forward rule in the Forward link of iptables.

- Physical environment

  To expose the IP address of a pod to the external network so that the external network can access containers using the IP address, set **natOutgoing** of the corresponding subnet to **false**. In addition, you need to add a static route on the external router and set the next hop of the data packets whose destination address is the subnet CIDR to any host in the cluster.

  The following is an example of configuration on a subnet.

```
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
  name: subnet-gateway
spec:
  protocol: IPv4
  default: false
  namespaces:
  - ns1
  - ns2
  cidrBlock: 100.64.0.0/16
  gateway: 100.64.0.1
  excludeIps:
  - 100.64.0.1
  private: false
  gatewayType: distributed
  natOutgoing: false
```

- Virtual network environment

  If the next hop of the container network is set to a host in the cluster due to restrictions of security groups or conntrack, data packets may be dropped due to asymmetric routing.

  In this case, you are advised to set **gatewayType** to **centralized** and set the corresponding **gatewayNode**. When an external system accesses a container, the next hop of the container CIDR must be set to the node specified by **gatewayNode** to avoid asymmetric routing restrictions.

2.4.7 IPv6 Subnet

Kube-OVN supports the coexistence of IPv4 and IPv6 subnets in a cluster. However, due to some restrictions of the Kubernetes control plane, if the pod network protocol is inconsistent with the protocol of the Kubernetes control plane, functions such as service discovery cannot work properly. Before using an IPv6 subnet, you are advised to configure the protocol of the Kubernetes control plane to IPv6 to prevent network problems.

The following is an example of creating an IPv6 subnet.

```
cat <<EOF | kubectl create -f -
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
  name: ipv6
spec:
  cidrBlock: 2001:4860::/32
  excludeIps:
  - 2001:4860::1
  gateway: 2001:4860::1
  gatewayType: distributed
```



```
  namespaces: [ls2]
  natOutgoing: false
  private: false
  protocol: "IPv6"
EOF
```

2.5 O&M Operations

2.5.1 Deleting a Node

The Kube-OVN-Controller running on a node periodically reconnects to the ovn-sb, but the chassis still registers again. As a result, some network configurations are residual, resources are wasted, and potential rule conflicts may occur. Therefore, when deleting a node from Kubernetes, perform the following steps to ensure that network information can be properly deleted.

Step 1 Drain a node.

```
kubectl drain <node-name> --ignore-daemonsets --force
```

Step 2 Log in to the node and stop kubelet and Docker to stop the corresponding DaemonSet.

```
systemctl stop kubelet && systemctl stop docker
```

Step 3 Delete the node from the master nodes.

```
kubectl delete node <node-name>
```

Step 4 Check whether the node is deleted from the ovn-sb.

```
kubectl ko sbctl show
```

Step 5 If any chassis corresponding to the hostname still exists, manually delete it.

```
kubectl ko sbctl chassis-del <chassis-uuid>
```

----End

2.5.2 Configuring QoS

In Kube-OVN, the ingress and egress bandwidths (in Mbit/s) of a pod can be specified by the **ovn.kubernetes.io/ingress_rate** and **ovn.kubernetes.io/egress_rate** fields respectively in the **annotations** section in the YAML configuration file on the pod. You can configure the QoS when creating a pod or dynamically adjust the QoS by modifying the two fields when the pod is running.

- Configuring the QoS when creating a pod

```
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: nginx:alpine
```

- Dynamically adjusting the QoS

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


2.5.3 Mirroring Traffic

By default, Kube-OVN creates a mirror0 virtual network interface card (vNIC) on each node to mirror the traffic of all containers on the node. You can run the **tcpdump -nni mirror0** command to view the traffic information or use other tools to export the traffic from the mirror0 vNIC for analysis.

2.6 Reference

2.6.1 YAML Files for Manual Installation

CAUTION

The Kube-OVN version and features are evolving rapidly. It is recommended that the Kube-OVN version obtained by the one-click installation script be the same as the image version. Otherwise, Kube-OVN may fail to be deployed and started in one-click mode. This section provides only the YAML deployment files based on Kube-OVN v1.2.1.

YAML File for CRD Deployment

```
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: ips.kubeovn.io
spec:
  group: kubeovn.io
  version: v1
  scope: Cluster
  names:
    plural: ips
    singular: ip
    kind: IP
    shortNames:
      - ip
  additionalPrinterColumns:
    - name: IP
      type: string
      JSONPath: .spec.ipAddress
    - name: Mac
      type: string
      JSONPath: .spec.macAddress
    - name: Node
      type: string
      JSONPath: .spec.nodeName
    - name: Subnet
      type: string
      JSONPath: .spec.subnet
---
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: subnets.kubeovn.io
spec:
  group: kubeovn.io
  version: v1
  scope: Cluster
  names:
```



```
    plural: subnets
    singular: subnet
    kind: Subnet
    shortNames:
      - subnet
  subresources:
    status: {}
  additionalPrinterColumns:
    - name: Provider
      type: string
      JSONPath: .spec.provider
    - name: Protocol
      type: string
      JSONPath: .spec.protocol
    - name: CIDR
      type: string
      JSONPath: .spec.cidrBlock
    - name: Private
      type: boolean
      JSONPath: .spec.private
    - name: NAT
      type: boolean
      JSONPath: .spec.natOutgoing
    - name: Default
      type: boolean
      JSONPath: .spec.default
    - name: GatewayType
      type: string
      JSONPath: .spec.gatewayType
    - name: Used
      type: number
      JSONPath: .status.usingIPs
    - name: Available
      type: number
      JSONPath: .status.availableIPs
  validation:
    openAPIV3Schema:
      properties:
        spec:
          required: ["cidrBlock"]
          properties:
            cidrBlock:
              type: "string"
            gateway:
              type: "string"
---
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: vlans.kubeovn.io
spec:
  group: kubeovn.io
  version: v1
  scope: Cluster
  names:
    plural: vlans
    singular: vlan
    kind: Vlan
    shortNames:
      - vlan
  additionalPrinterColumns:
    - name: VlanID
      type: string
      JSONPath: .spec.vlanId
    - name: ProviderInterfaceName
      type: string
      JSONPath: .spec.providerInterfaceName
    - name: Subnet
```



```
    type: string
    JSONPath: .spec.subnet
```

YAML File for OVN Deployment

```
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: ovn-config
  namespace: kube-system

---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: ovn
  namespace: kube-system

---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  annotations:
    rbac.authorization.k8s.io/system-only: "true"
  name: system:ovn
rules:
  - apiGroups:
      - "kubeovn.io"
    resources:
      - subnets
      - subnets/status
      - ips
      - vlans
    verbs:
      - "*"
  - apiGroups:
      - ""
    resources:
      - pods
      - namespaces
      - nodes
      - configmaps
    verbs:
      - create
      - get
      - list
      - watch
      - patch
      - update
  - apiGroups:
      - ""
      - networking.k8s.io
      - apps
      - extensions
    resources:
      - networkpolicies
      - services
      - endpoints
      - statefulsets
      - daemonsets
      - deployments
    verbs:
      - get
      - list
      - watch
  - apiGroups:
      - ""
    resources:
```



```
    - events
  verbs:
    - create
    - patch
    - update

---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: ovn
roleRef:
  name: system:ovn
  kind: ClusterRole
  apiGroup: rbac.authorization.k8s.io
subjects:
  - kind: ServiceAccount
    name: ovn
    namespace: kube-system

---
kind: Service
apiVersion: v1
metadata:
  name: ovn-nb
  namespace: kube-system
spec:
  ports:
    - name: ovn-nb
      protocol: TCP
      port: 6641
      targetPort: 6641
  type: ClusterIP
  selector:
    app: ovn-central
    ovn-nb-leader: "true"
  sessionAffinity: None

---
kind: Service
apiVersion: v1
metadata:
  name: ovn-sb
  namespace: kube-system
spec:
  ports:
    - name: ovn-sb
      protocol: TCP
      port: 6642
      targetPort: 6642
  type: ClusterIP
  selector:
    app: ovn-central
    ovn-sb-leader: "true"
  sessionAffinity: None

---
kind: Deployment
apiVersion: apps/v1
metadata:
  name: ovn-central
  namespace: kube-system
  annotations:
    kubernetes.io/description: |
      OVN components: northd, nb and sb.
spec:
  replicas: 1
  strategy:
    rollingUpdate:
```



```
    maxSurge: 0%
    maxUnavailable: 100%
  type: RollingUpdate
selector:
  matchLabels:
    app: ovn-central
template:
  metadata:
    labels:
      app: ovn-central
      component: network
      type: infra
  spec:
    tolerations:
    - operator: Exists
      effect: NoSchedule
    affinity:
      podAntiAffinity:
        requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchLabels:
                app: ovn-central
            topologyKey: kubernetes.io/hostname
    priorityClassName: system-cluster-critical
    serviceAccountName: ovn
    hostNetwork: true
    containers:
      - name: ovn-central
        image: "kubeovn/kube-ovn:v1.2.1-arm"
        imagePullPolicy: IfNotPresent
        command: ["/kube-ovn/start-db.sh"]
        securityContext:
          capabilities:
            add: ["SYS_NICE"]
        env:
          - name: POD_IP
            valueFrom:
              fieldRef:
                fieldPath: status.podIP
          - name: POD_NAME
            valueFrom:
              fieldRef:
                fieldPath: metadata.name
          - name: POD_NAMESPACE
            valueFrom:
              fieldRef:
                fieldPath: metadata.namespace
        resources:
          requests:
            cpu: 500m
            memory: 300Mi
        volumeMounts:
          - mountPath: /var/run/openvswitch
            name: host-run-ovs
          - mountPath: /var/run/ovn
            name: host-run-ovn
          - mountPath: /sys
            name: host-sys
            readOnly: true
          - mountPath: /etc/openvswitch
            name: host-config-openvswitch
          - mountPath: /etc/ovn
            name: host-config-ovn
          - mountPath: /var/log/openvswitch
            name: host-log-ovs
          - mountPath: /var/log/ovn
            name: host-log-ovn
        readinessProbe:
          exec:
```



```
            command:
              - sh
              - /kube-ovn/ovn-is-leader.sh
            periodSeconds: 3
          livenessProbe:
            exec:
              command:
              - sh
              - /kube-ovn/ovn-healthcheck.sh
            initialDelaySeconds: 30
            periodSeconds: 7
            failureThreshold: 5
      nodeSelector:
        kubernetes.io/os: "linux"
        kube-ovn/role: "master"
      volumes:
        - name: host-run-ovs
          hostPath:
            path: /run/openvswitch
        - name: host-run-ovn
          hostPath:
            path: /run/ovn
        - name: host-sys
          hostPath:
            path: /sys
        - name: host-config-openvswitch
          hostPath:
            path: /etc/origin/openvswitch
        - name: host-config-ovn
          hostPath:
            path: /etc/origin/ovn
        - name: host-log-ovs
          hostPath:
            path: /var/log/openvswitch
        - name: host-log-ovn
          hostPath:
            path: /var/log/ovn

---
kind: DaemonSet
apiVersion: apps/v1
metadata:
  name: ovs-ovn
  namespace: kube-system
  annotations:
    kubernetes.io/description: |
      This daemon set launches the openvswitch daemon.
spec:
  selector:
    matchLabels:
      app: ovs
  updateStrategy:
    type: OnDelete
  template:
    metadata:
      labels:
        app: ovs
        component: network
        type: infra
    spec:
      tolerations:
      - operator: Exists
        effect: NoSchedule
      priorityClassName: system-cluster-critical
      serviceAccountName: ovn
      hostNetwork: true
      hostPID: true
      containers:
        - name: openvswitch
```



```
      image: "kubeovn/kube-ovn:v1.2.1-arm"
      imagePullPolicy: IfNotPresent
      command: ["/kube-ovn/start-ovs.sh"]
      securityContext:
        runAsUser: 0
        privileged: true
      env:
        - name: POD_IP
          valueFrom:
            fieldRef:
              fieldPath: status.podIP
      volumeMounts:
        - mountPath: /lib/modules
          name: host-modules
          readOnly: true
        - mountPath: /var/run/openvswitch
          name: host-run-ovs
        - mountPath: /var/run/ovn
          name: host-run-ovn
        - mountPath: /sys
          name: host-sys
          readOnly: true
        - mountPath: /etc/openvswitch
          name: host-config-openvswitch
        - mountPath: /etc/ovn
          name: host-config-ovn
        - mountPath: /var/log/openvswitch
          name: host-log-ovs
        - mountPath: /var/log/ovn
          name: host-log-ovn
      readinessProbe:
        exec:
          command:
          - sh
          - /kube-ovn/ovs-healthcheck.sh
        periodSeconds: 5
      livenessProbe:
        exec:
          command:
          - sh
          - /kube-ovn/ovs-healthcheck.sh
        initialDelaySeconds: 10
        periodSeconds: 5
        failureThreshold: 5
      resources:
        requests:
          cpu: 200m
          memory: 300Mi
        limits:
          cpu: 1000m
          memory: 800Mi
  nodeSelector:
    kubernetes.io/os: "linux"
  volumes:
    - name: host-modules
      hostPath:
        path: /lib/modules
    - name: host-run-ovs
      hostPath:
        path: /run/openvswitch
    - name: host-run-ovn
      hostPath:
        path: /run/ovn
    - name: host-sys
      hostPath:
        path: /sys
    - name: host-config-openvswitch
      hostPath:
        path: /etc/origin/openvswitch
```



```
      - name: host-config-ovn
        hostPath:
          path: /etc/origin/ovn
      - name: host-log-ovs
        hostPath:
          path: /var/log/openvswitch
      - name: host-log-ovn
        hostPath:
          path: /var/log/ovn
```

YAML File for Kube-OVN Deployment

```
---
kind: Deployment
apiVersion: apps/v1
metadata:
  name: kube-ovn-controller
  namespace: kube-system
  annotations:
    kubernetes.io/description: |
      kube-ovn controller
spec:
  replicas: 2
  selector:
    matchLabels:
      app: kube-ovn-controller
  strategy:
    rollingUpdate:
      maxSurge: 0%
      maxUnavailable: 100%
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: kube-ovn-controller
        component: network
        type: infra
    spec:
      tolerations:
      - operator: Exists
        effect: NoSchedule
      affinity:
        podAntiAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            - labelSelector:
                matchLabels:
                  app: kube-ovn-controller
              topologyKey: kubernetes.io/hostname
      priorityClassName: system-cluster-critical
      serviceAccountName: ovn
      hostNetwork: true
      containers:
        - name: kube-ovn-controller
          image: "kubeovn/kube-ovn:v1.2.1-arm"
          imagePullPolicy: IfNotPresent
          command:
          - /kube-ovn/start-controller.sh
          args:
          - --default-cidr=10.16.0.0/16
          - --default-gateway=10.16.0.1
          - --node-switch-cidr=100.64.0.0/16
          env:
            - name: POD_NAME
              valueFrom:
                fieldRef:
                  fieldPath: metadata.name
            - name: KUBE_NAMESPACE
              valueFrom:
                fieldRef:
```



```
              fieldPath: metadata.namespace
        - name: KUBE_NODE_NAME
          valueFrom:
            fieldRef:
              fieldPath: spec.nodeName
        readinessProbe:
          exec:
            command:
              - sh
              - /kube-ovn/kube-ovn-controller-healthcheck.sh
          periodSeconds: 3
        livenessProbe:
          exec:
            command:
              - sh
              - /kube-ovn/kube-ovn-controller-healthcheck.sh
          initialDelaySeconds: 300
          periodSeconds: 7
          failureThreshold: 5
    nodeSelector:
      kubernetes.io/os: "linux"

---
kind: DaemonSet
apiVersion: apps/v1
metadata:
  name: kube-ovn-cni
  namespace: kube-system
  annotations:
    kubernetes.io/description: |
      This daemon set launches the kube-ovn cni daemon.
spec:
  selector:
    matchLabels:
      app: kube-ovn-cni
  updateStrategy:
    type: OnDelete
  template:
    metadata:
      labels:
        app: kube-ovn-cni
        component: network
        type: infra
    spec:
      tolerations:
      - operator: Exists
        effect: NoSchedule
      priorityClassName: system-cluster-critical
      serviceAccountName: ovn
      hostNetwork: true
      hostPID: true
      initContainers:
      - name: install-cni
        image: "kubeovn/kube-ovn:v1.2.1-arm"
        imagePullPolicy: IfNotPresent
        command: ["/kube-ovn/install-cni.sh"]
        securityContext:
          runAsUser: 0
          privileged: true
        volumeMounts:
          - mountPath: /etc/cni/net.d
            name: cni-conf
          - mountPath: /opt/cni/bin
            name: cni-bin
      containers:
      - name: cni-server
        image: "kubeovn/kube-ovn:v1.2.1-arm"
        imagePullPolicy: IfNotPresent
        command:
```



```
        - sh
        - /kube-ovn/start-cniserver.sh
      args:
        - --enable-mirror=true
      securityContext:
        runAsUser: 0
        privileged: true
      env:
        - name: POD_IP
          valueFrom:
            fieldRef:
              fieldPath: status.podIP
        - name: KUBE_NODE_NAME
          valueFrom:
            fieldRef:
              fieldPath: spec.nodeName
      volumeMounts:
        - mountPath: /run/openvswitch
          name: host-run-ovs
        - mountPath: /run/ovn
          name: host-run-ovn
        - mountPath: /var/run/netns
          name: host-ns
          mountPropagation: HostToContainer
      readinessProbe:
        exec:
          command:
            - nc
            - -z
            - -w3
            - 127.0.0.1
            - "10665"
        periodSeconds: 3
      livenessProbe:
        exec:
          command:
            - nc
            - -z
            - -w3
            - 127.0.0.1
            - "10665"
        initialDelaySeconds: 30
        periodSeconds: 7
        failureThreshold: 5
    nodeSelector:
      kubernetes.io/os: "linux"
    volumes:
      - name: host-run-ovs
        hostPath:
          path: /run/openvswitch
      - name: host-run-ovn
        hostPath:
          path: /run/ovn
      - name: cni-conf
        hostPath:
          path: /etc/cni/net.d
      - name: cni-bin
        hostPath:
          path: /opt/cni/bin
      - name: host-ns
        hostPath:
          path: /var/run/netns

---
kind: DaemonSet
apiVersion: apps/v1
metadata:
  name: kube-ovn-pinger
  namespace: kube-system
```



```
  annotations:
    kubernetes.io/description: |
      This daemon set launches the openvswitch daemon.
spec:
  selector:
    matchLabels:
      app: kube-ovn-pinger
  updateStrategy:
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: kube-ovn-pinger
        component: network
        type: infra
    spec:
      tolerations:
        - operator: Exists
          effect: NoSchedule
      serviceAccountName: ovn
      hostPID: true
      containers:
        - name: pinger
          image: "kubeovn/kube-ovn:v1.2.1-arm"
          command: ["/kube-ovn/kube-ovn-pinger", "--external-address=114.114.114.114"]
          imagePullPolicy: IfNotPresent
          securityContext:
            runAsUser: 0
            privileged: false
          env:
            - name: POD_IP
              valueFrom:
                fieldRef:
                  fieldPath: status.podIP
            - name: HOST_IP
              valueFrom:
                fieldRef:
                  fieldPath: status.hostIP
            - name: POD_NAME
              valueFrom:
                fieldRef:
                  fieldPath: metadata.name
            - name: NODE_NAME
              valueFrom:
                fieldRef:
                  fieldPath: spec.nodeName
          volumeMounts:
            - mountPath: /lib/modules
              name: host-modules
              readOnly: true
            - mountPath: /run/openvswitch
              name: host-run-ovs
            - mountPath: /var/run/openvswitch
              name: host-run-ovs
            - mountPath: /var/run/ovn
              name: host-run-ovn
            - mountPath: /sys
              name: host-sys
              readOnly: true
            - mountPath: /etc/openvswitch
              name: host-config-openvswitch
            - mountPath: /var/log/openvswitch
              name: host-log-ovs
            - mountPath: /var/log/ovn
              name: host-log-ovn
          resources:
            requests:
              cpu: 100m
              memory: 300Mi
```



```
          limits:
            cpu: 200m
            memory: 400Mi
      nodeSelector:
        kubernetes.io/os: "linux"
      volumes:
        - name: host-modules
          hostPath:
            path: /lib/modules
        - name: host-run-ovs
          hostPath:
            path: /run/openvswitch
        - name: host-run-ovn
          hostPath:
            path: /run/ovn
        - name: host-sys
          hostPath:
            path: /sys
        - name: host-config-openvswitch
          hostPath:
            path: /etc/origin/openvswitch
        - name: host-log-ovs
          hostPath:
            path: /var/log/openvswitch
        - name: host-log-ovn
          hostPath:
            path: /var/log/ovn
---
kind: Service
apiVersion: v1
metadata:
  name: kube-ovn-pinger
  namespace: kube-system
  labels:
    app: kube-ovn-pinger
spec:
  selector:
    app: kube-ovn-pinger
  ports:
    - port: 8080
      name: metrics
---
kind: Service
apiVersion: v1
metadata:
  name: kube-ovn-controller
  namespace: kube-system
  labels:
    app: kube-ovn-controller
spec:
  selector:
    app: kube-ovn-controller
  ports:
    - port: 10660
      name: metrics
---
kind: Service
apiVersion: v1
metadata:
  name: kube-ovn-cni
  namespace: kube-system
  labels:
    app: kube-ovn-cni
spec:
  selector:
    app: kube-ovn-cni
  ports:
    - port: 10665
      name: metrics
```


kubectl Plugin

```
#!/bin/bash
set -euo pipefail

KUBE_OVN_NS=kube-system
OVN_NB_POD=
OVN_SB_POD=

showHelp(){
  echo "kubectl ko {subcommand} [option...]"
  echo "Available Subcommands:"
  echo "  nbctl [ovn-nbctl options ...]    invoke ovn-nbctl"
  echo "  sbctl [ovn-sbctl options ...]    invoke ovn-sbctl"
  echo "  vsctl {nodeName} [ovs-vsctl options ...]   invoke ovs-vsctl on selected node"
  echo "  tcpdump {namespace/podname} [tcpdump options ...]     capture pod traffic"
  echo "  trace {namespace/podname} {target ip address} {icmp|tcp|udp} [target tcp or udp port]    trace
ovn microflow of specific packet"
  echo "  diagnose {all|node} [nodename]    diagnose connectivity of all nodes or a specific node"
}

tcpdump(){
  namespacedPod="$1"; shift
  namespace=$(echo "$namespacedPod" | cut -d "/" -f1)
  podName=$(echo "$namespacedPod" | cut -d "/" -f2)
  if [ "$podName" = "$namespacedPod" ]; then
    nodeName=$(kubectl get pod "$podName" -o jsonpath={.spec.nodeName})
    mac=$(kubectl get pod "$podName" -o jsonpath={.metadata.annotations.ovn\\.kubernetes\\.io/
mac_address})
    hostNetwork=$(kubectl get pod "$podName" -o jsonpath={.spec.hostNetwork})
  else
    nodeName=$(kubectl get pod "$podName" -n "$namespace" -o jsonpath={.spec.nodeName})
    mac=$(kubectl get pod "$podName" -n "$namespace" -o jsonpath={.metadata.annotations.ovn\
\.kubernetes\\.io/mac_address})
    hostNetwork=$(kubectl get pod "$podName" -n "$namespace" -o jsonpath={.spec.hostNetwork})
  fi

  if [ -z "$nodeName" ]; then
    echo "Pod $namespacedPod not exists on any node"
    exit 1
  fi

  if [ -z "$mac" ] && [ "$hostNetwork" != "true" ]; then
     echo "pod mac address not ready"
     exit 1
  fi
  mac=$(echo "$mac" | tr '[:upper:]' '[:lower:]')

  ovnCni=$(kubectl get pod -n $KUBE_OVN_NS -o wide| grep kube-ovn-cni| grep " $nodeName " | awk
'{print $1}')
  if [ -z "$ovnCni" ]; then
    echo "kube-ovn-cni not exist on node $nodeName"
    exit 1
  fi

  if [ "$hostNetwork" = "true" ]; then
    set -x
    kubectl exec -it "$ovnCni" -n $KUBE_OVN_NS -- tcpdump -nn "$@"
  else
    nicName=$(kubectl exec -it "$ovnCni" -n $KUBE_OVN_NS -- ovs-vsctl --data=bare --no-heading --
columns=name find interface mac_in_use="${mac//:/\\:}" | tr -d '\r')
    if [ -z "$nicName" ]; then
      echo "nic doesn't exist on node $nodeName"
      exit 1
    fi
    set -x
    kubectl exec -it "$ovnCni" -n $KUBE_OVN_NS -- tcpdump -nn -i "$nicName" "$@"
  fi
}
```



```
trace(){
  namespacedPod="$1"
  namespace=$(echo "$1" | cut -d "/" -f1)
  podName=$(echo "$1" | cut -d "/" -f2)
  if [ "$podName" = "$1" ]; then
    echo "namespace is required"
    exit 1
  fi

  podIP=$(kubectl get pod "$podName" -n "$namespace" -o jsonpath={.metadata.annotations.ovn\
\.kubernetes\\.io/ip_address})
  mac=$(kubectl get pod "$podName" -n "$namespace" -o jsonpath={.metadata.annotations.ovn\
\.kubernetes\\.io/mac_address})
  ls=$(kubectl get pod "$podName" -n "$namespace" -o jsonpath={.metadata.annotations.ovn\\.kubernetes\
\.io/logical_switch})
  hostNetwork=$(kubectl get pod "$podName" -n "$namespace" -o jsonpath={.spec.hostNetwork})

  if [ "$hostNetwork" = "true" ]; then
    echo "Can not trace host network pod"
    exit 1
  fi

  if [ -z "$ls" ]; then
    echo "pod address not ready"
    exit 1
  fi

  gwMac=$(kubectl exec -it $OVN_NB_POD -n $KUBE_OVN_NS -- ovn-nbctl --data=bare --no-heading --
columns=mac find logical_router_port name=ovn-cluster-"$ls" | tr -d '\r')

  if [ -z "$gwMac" ]; then
    echo "get gw mac failed"
    exit 1
  fi

  dst="$2"
  if [ -z "$dst" ]; then
    echo "need a target ip address"
    exit 1
  fi

  type="$3"

  case $type in
    icmp)
      set -x
      kubectl exec "$OVN_SB_POD" -n $KUBE_OVN_NS -- ovn-trace --ct=new "$ls" "inport == \"$podName.
$namespace\" && ip.ttl == 64 && icmp && eth.src == $mac && ip4.src == $podIP && eth.dst == $gwMac &&
ip4.dst == $dst"
      ;;
    tcp|udp)
      set -x
      kubectl exec "$OVN_SB_POD" -n $KUBE_OVN_NS -- ovn-trace --ct=new "$ls" "inport == \"$podName.
$namespace\" && ip.ttl == 64 && eth.src == $mac && ip4.src == $podIP && eth.dst == $gwMac && ip4.dst
== $dst && $type.src == 10000 && $type.dst == $4"
      ;;
    *)
      echo "type $type not supported"
      echo "kubectl ko trace {namespace/podname} {target ip address} {icmp|tcp|udp} [target tcp or udp
port]"
      ;;
  esac
}

vsctl(){
  nodeName="$1"; shift
  kubectl get no "$nodeName" > /dev/null
  ovsPod=$(kubectl get pod -n $KUBE_OVN_NS -o wide | grep " $nodeName " | grep ovs-ovn | awk '{print
$1}')
```



```
  if [ -z "$ovsPod" ]; then
     echo "ovs pod  doesn't exist on node $nodeName"
     exit 1
   fi
  kubectl exec "$ovsPod" -n $KUBE_OVN_NS -- ovs-vsctl "$@"
}

diagnose(){
  kubectl get crd subnets.kubeovn.io
  kubectl get crd ips.kubeovn.io
  kubectl get svc kube-dns -n kube-system
  kubectl get svc kubernetes -n default

  checkDaemonSet kube-proxy
  checkDeployment ovn-central
  checkDeployment kube-ovn-controller
  checkDaemonSet kube-ovn-cni
  checkDaemonSet ovs-ovn
  checkDeployment coredns
  type="$1"
  case $type in
    all)
      echo "### kube-ovn-controller recent log"
      set +e
      kubectl logs -n $KUBE_OVN_NS -l app=kube-ovn-controller --tail=100 | grep E$(date +%m%d)
      set -e
      echo ""
      pingers=$(kubectl get pod -n $KUBE_OVN_NS | grep kube-ovn-pinger | awk '{print $1}')
      for pinger in $pingers
      do
        nodeName=$(kubectl get pod "$pinger" -n "$KUBE_OVN_NS" -o jsonpath={.spec.nodeName})
        echo "### start to diagnose node $nodeName"
        echo "#### ovn-controller log:"
        kubectl exec -n $KUBE_OVN_NS -it "$pinger" -- tail /var/log/ovn/ovn-controller.log
        echo ""
        kubectl exec -n $KUBE_OVN_NS -it "$pinger" -- /kube-ovn/kube-ovn-pinger --mode=job
        echo "### finish diagnose node $nodeName"
        echo ""
      done
      ;;
    node)
      nodeName="$2"
      kubectl get no "$nodeName" > /dev/null
      pinger=$(kubectl get pod -n $KUBE_OVN_NS -o wide | grep kube-ovn-pinger | grep " $nodeName " |
awk '{print $1}')
      echo "### start to diagnose node nodeName"
      echo "#### ovn-controller log:"
      kubectl exec -n $KUBE_OVN_NS -it "$pinger" -- tail /var/log/ovn/ovn-controller.log
      echo ""
      kubectl exec -n $KUBE_OVN_NS -it "$pinger" -- /kube-ovn/kube-ovn-pinger --mode=job
      echo "### finish diagnose node nodeName"
      echo ""
      ;;
    *)
      echo "type $type not supported"
      echo "kubectl ko diagnose {all|node} [nodename]"
      ;;
    esac
}

getOvnCentralPod(){
    NB_POD=$(kubectl get pod -n $KUBE_OVN_NS -l ovn-nb-leader=true | grep ovn-central | head -n 1 |
awk '{print $1}')
    if [ -z "$NB_POD" ]; then
      echo "nb leader not exists"
      exit 1
    fi
    OVN_NB_POD=$NB_POD
    SB_POD=$(kubectl get pod -n $KUBE_OVN_NS -l ovn-sb-leader=true | grep ovn-central | head -n 1 | awk
```



```
'{print $1}')
    if [ -z "$SB_POD" ]; then
      echo "nb leader not exists"
      exit 1
    fi
    OVN_SB_POD=$SB_POD
}

checkDaemonSet(){
  name="$1"
  currentScheduled=$(kubectl get ds -n $KUBE_OVN_NS "$name" -o
jsonpath={.status.currentNumberScheduled})
  desiredScheduled=$(kubectl get ds -n $KUBE_OVN_NS "$name" -o
jsonpath={.status.desiredNumberScheduled})
  available=$(kubectl get ds -n $KUBE_OVN_NS "$name" -o jsonpath={.status.numberAvailable})
  ready=$(kubectl get ds -n $KUBE_OVN_NS "$name" -o jsonpath={.status.numberReady})
  if [ "$currentScheduled" = "$desiredScheduled" ] && [ "$desiredScheduled" = "$available" ] &&
[ "$available" = "$ready" ]; then
    echo "ds $name ready"
  else
    echo "Error ds $name not ready"
    exit 1
  fi
}

checkDeployment(){
  name="$1"
  ready=$(kubectl get deployment -n $KUBE_OVN_NS "$name" -o jsonpath={.status.readyReplicas})
  updated=$(kubectl get deployment -n $KUBE_OVN_NS "$name" -o jsonpath={.status.updatedReplicas})
  desire=$(kubectl get deployment -n $KUBE_OVN_NS "$name" -o jsonpath={.status.replicas})
  available=$(kubectl get deployment -n $KUBE_OVN_NS "$name" -o jsonpath={.status.availableReplicas})
  if [ "$ready" = "$updated" ] && [ "$updated" = "$desire" ] && [ "$desire" = "$available" ]; then
    echo "deployment $name ready"
  else
    echo "Error deployment $name not ready"
    exit 1
  fi
}

if [ $# -lt 1 ]; then
  showHelp
  exit 0
else
  subcommand="$1"; shift
fi

getOvnCentralPod

case $subcommand in
  nbctl)
    kubectl exec "$OVN_NB_POD" -n $KUBE_OVN_NS -- ovn-nbctl "$@"
    ;;
  sbctl)
    kubectl exec "$OVN_SB_POD" -n $KUBE_OVN_NS -- ovn-sbctl "$@"
    ;;
  vsctl)
    vsctl "$@"
    ;;
  tcpdump)
    tcpdump "$@"
    ;;
  trace)
    trace "$@"
    ;;
  diagnose)
    diagnose "$@"
    ;;
  *)
    showHelp
```



```
    ;;
esac
```


3 XPF User Guide

3.1 Introduction

XPF Overview

The Data Plane Development Kit (DPDK) is an open-source, high-performance, and user-mode software project developed by Intel. Open vSwitch (OVS) is an open-source implementation of a distributed virtual multilayer switch and is widely used in cloud computing. OVS+DPDK provides flexible network management and high-performance forwarding capabilities. For more information about DPDK and OVS, visit the official websites **https://www.dpdk.org/** and **https://www.openvswitch.org/**.

Based on OVS+DPDK, the Extensible Packet Framework (XPF) proposes flow table normalization to further accelerate data packet forwarding performance in cloud computing. The typical scenario is VXLAN+CT networking.

Programming language: C

Brief description: high-performance implementation of a virtual switch

Open-source protocols: BSD 3-Clause License, LGPL V2.1, GPL V2.0, Apache License V2.0

Recommended Version

Table 3-1 Software versions

Software	Version
DPDK	19.11
OVS	2.12.0
XPF	1.0.0

Security Hardening

Two common vulnerabilities and exposures (CVE) vulnerabilities are found in open-source DPDK 19.11. You are advised to integrate the following patches into DPDK 19.11 before using it:

https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-10726

https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-10725

3.2 Environment Requirements

Hardware Requirements

Table 3-2 lists the hardware requirements.

Table 3-2 Hardware requirements

Item	Description
CPU	Kunpeng processor
NIC	The NIC must support DPDK.

OS Requirements

Table 3-3 lists the OS requirements.

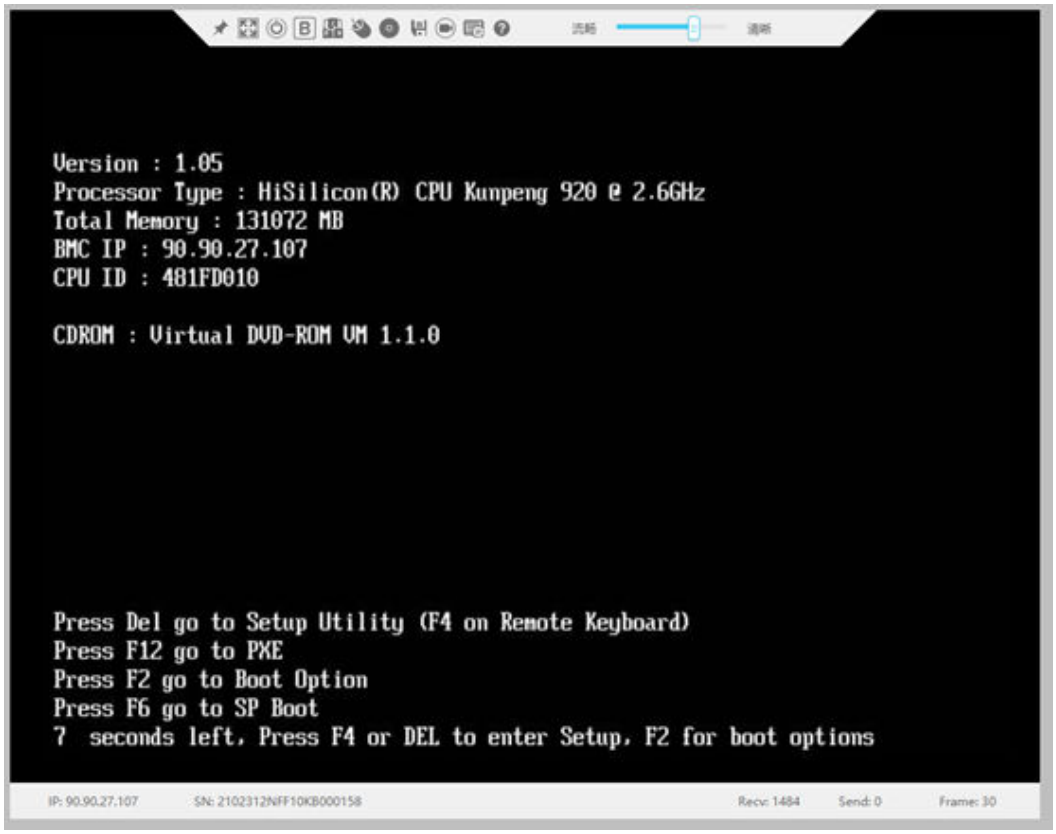
Table 3-3 OS requirements

Item	Version	How to Obtain
CentOS	7.6	**https://mirrors.huaweicloud.com/centos-vault/altarch/7.6.1810/isos/aarch64**

Item	Version	How to Obtain
Kernel	4.14.0	Included in the OS image.

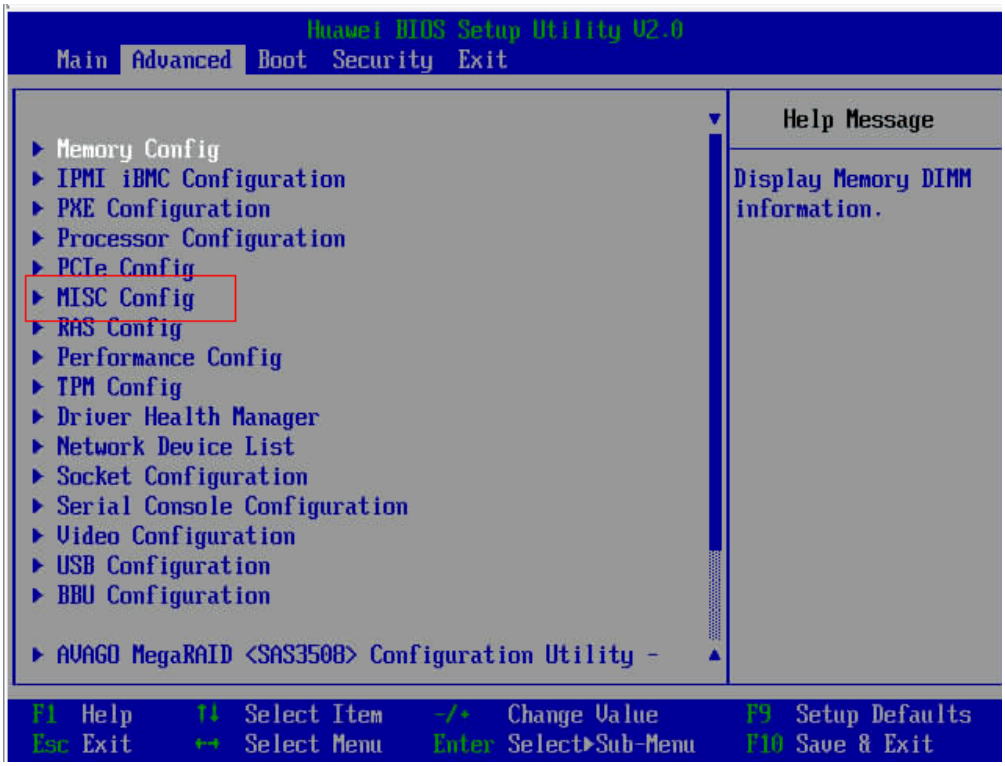
3.3 BIOS Settings

Step 1 Log in to the baseboard management controller (BMC) system of the server, restart the server, and press **DEL** to enter the BIOS.

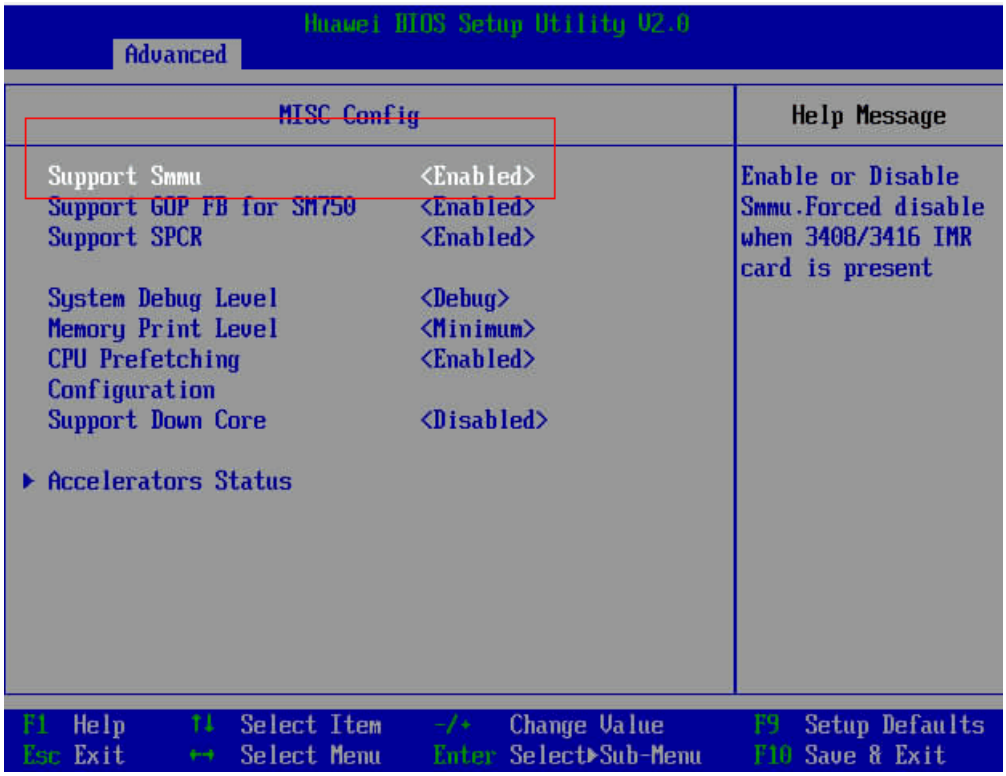


Step 2 Enable the input/output memory management unit (IOMMU).

1. Choose **Advanced** > **MISC Config** and press **Enter**.



2. Set **Support Smmu** to **Enabled**.



3. Press F10 to save the settings and exit.

----End

3.4 Configuring the Compilation Environment

Step 1 Install compilation dependencies.

```
sudo yum install -y automake cmake patch numactl numactl-devel kernel-devel libevent glib2 glib2-devel libtool openssl-devel selinux-policy-devel autoconf python-sphinx unbound-devel logrotate
```

Step 2 Install VM dependencies.

```
sudo yum install centos-release-qemu-ev
sudo yum install -y libvirt AAVMF virt-install qemu-guest-agent qemu-kvm-common-ev qemu-img-ev qemu-kvm-tools-ev qemu-kvm-ev
```

Step 3 Upgrade the GCC.

```
sudo yum install -y centos-release-scl
sudo yum install -y devtoolset-7-gcc devtoolset-7-gcc-c++
scl enable devtoolset-7 bash
```

Step 4 (Optional) Configure GCC environment variables.

1. The default values of GCC environment variables will be used upon Bash shutdown or re-login. In this case, run the following command again to retain GCC environment variables:
   ```
   scl enable devtoolset-7 bash
   ```
2. Retain GCC environment variables of Bash.

   Open the **~/.bash_profile** file.

   ```
   vim ~/.bash_profile
   ```

   Add the following content to the last line:

   ```
   scl enable devtoolset-7 bash
   ```

----End

CAUTION

- Some of the preceding software packages need to be downloaded from the Internet. Ensure that the server is connected to the Internet and the corresponding sources are configured.
- The QEMU software package with the file name extension **ev** must be installed. Otherwise, the dpdkvhostuser and dpdkvhostuserclient ports cannot be configured for VMs.
- By default, DPDK 19.11 does not support compilation with GCC 4.8.5. You need to upgrade GCC or modify compilation parameters. Modifying compilation parameters will affect the performance. Therefore, you are advised to upgrade GCC.

3.5 Compiling and Installing XPF

Obtaining the Source Code

Step 1 Create directories.


```
mkdir -p /home/source_code
mkdir -p /home/patch_code/ovs_patch
mkdir -p /home/rpm_packet
```

Step 2 Obtain the DPDK source code.

1. Download the source code.
   - Method 1 (recommended):

     Download the DPDK 19.11 source code from the Kunpeng community.

     https://github.com/kunpengcompute/ovs/releases/download/v2.12.0/dpdk-19.11.tar.gz
   - Method 2:

     Download the DPDK 19.11 source code from the official website.

     https://fast.dpdk.org/rel/dpdk-19.11.tar.xz
2. Copy the source code to the **/home/source_code** directory on the server.

Step 3 Obtain the OVS source code.

1. Download the source code.
   - Method 1 (recommended):

     Download the source code of the ovs_xpf_v1.2.10 branch for which the open-source **xpf.patch** has been installed.

     https://github.com/kunpengcompute/ovs/archive/ovs_xpf_v2.12.0.zip
   - Method 2:

     Download the source code of v2.12.0 released in the open-source community.

     https://www.openvswitch.org/releases/openvswitch-2.12.0.tar.gz
2. Copy the source code to the **/home/source_code** directory on the server.

Step 4 Obtain the patch code.

1. Download the patch code.

   https://github.com/kunpengcompute/ovs/releases/download/v2.12.0/xpf.patch
2. Copy the patch code to the **/home/patch_code/ovs_patch** directory on the server.

Step 5 Obtain the XPF library.

1. Download the binary RPM packages and digital signature files.
   - Method 1: Huawei Enterprise website

     xpf-1.0.0-1.aarch64.rpm

     xpf-devel-1.0.0-1.aarch64.rpm
   - Method 2: Huawei Carrier website

     xpf-1.0.0-1.aarch64.rpm

     xpf-devel-1.0.0-1.aarch64.rpm
2. Obtain the software verification tool.
   - Method 1: Huawei Enterprise website

     https://support.huawei.com/enterprise/en/tool/pgp-verify-TL1000000054

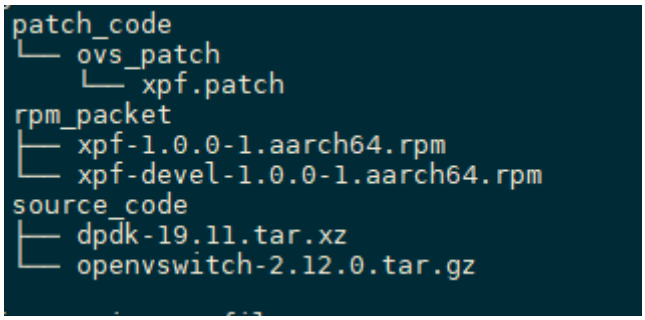
- Method 2: Huawei Carrier website

  https://support.huawei.com/carrier/digitalSignatureAction

3. Verify the software package integrity by following the procedure described in the *OpenPGP Signature Verification Guide* obtained in **Step 5.2**.
4. Copy the compressed RPM package to the **/home/rpm_packet** directory on the server.
5. Decompress the package.

   ```
   tar -xzf /home/rpm_packet/OVSOE_ALL.tar.gz -C /home/rpm_packet && rm -rf /home/rpm_packet/OVSOE_ALL.tar.gz
   ```

   The directory structure is as follows:

----End

Decompressing Source Code Packages

Step 1 Go to the **/home/source_code** directory.

```
cd /home/source_code
```

Step 2 Decompress the DPDK source code package.

```
tar -xf dpdk-19.11.tar.xz && rm -f dpdk-19.11.tar.xz
```

Step 3 Decompress the OVS source code package.

```
tar -xzf openvswitch-2.12.0.tar.gz && rm -f openvswitch-2.12.0.tar.gz
```

Step 4 Install the patch for the corresponding source code.

```
patch -d openvswitch-2.12.0 -p1 < ../patch_code/ovs_patch/xpf.patch
```

NOTE

Skip this step if the downloaded code is the source code of the ovx_xpf_v2.12.0 branch.

Step 5 Forcibly install the XPF binary library.

```
rpm -ivh --nodeps /home/rpm_packet/xpf-1.0.0-1.aarch64.rpm /home/rpm_packet/xpf-devel-1.0.0-1.aarch64.rpm
```

NOTE

Replace **/home/rpm_packet/** with the actual path.

----End

Compiling and Installing DPDK

Step 1 Go to the **/home/source_code/dpdk-19.11** directory.

```
cd /home/source_code/dpdk-19.11
```


Step 2 Compile and install DPDK.

```
make O=arm64-armv8a-linuxapp-gcc T=arm64-armv8a-linuxapp-gcc config
sed -ri 's,(RTE_APP_TEST=).*,\1n,' arm64-armv8a-linuxapp-gcc/.config
sed -ri 's,(RTE_BUILD_SHARED_LIB=).*,\1y,' arm64-armv8a-linuxapp-gcc/.config
make O=arm64-armv8a-linuxapp-gcc -j 96
make install O=arm64-armv8a-linuxapp-gcc prefix=/usr libdir=/lib64
```

If the **/usr/lib64** directory contains the **librte*xxx_xxx*.so** dynamic library file, the DPDK compilation and installation are complete.

----End

Compiling and Installing OVS

Step 1 Go to the root directory of source code.

- Run the following command if the source code of the ovs_xpf_v1.2.10 branch is downloaded from the Kunpeng community:

  ```
  cd /home/source_code/ovs-ovs_xpf_v2.12.0
  ```

- Run the following command if the code is downloaded from the official OVS website:

  ```
  cd /home/source_code/openvswitch-2.12.0
  ```

Step 2 Compile and install OVS.

```
./boot.sh
./configure CFLAGS="-g -O2 -march=armv8-a+crc" --prefix=/usr --sysconfdir=/etc --localstatedir=/var --libdir=/lib64 --enable-ssl --enable-shared --with-dpdk=yes --enable-Werror
make -j 96 && make install
```

Step 3 Copy necessary header files.

```
cp config.h /usr/include/openvswitch
mkdir /usr/include/openvswitch/lib
cp lib/*.h /usr/include/openvswitch/lib/
```

Step 4 Recompile and reinstall OVS.

```
make clean
./configure CFLAGS="-g -O2 -march=armv8-a+crc -ftree-vectorize -I/usr/include/xpf-1.0.0/xpf_include" --prefix=/usr --sysconfdir=/etc --localstatedir=/var --libdir=/lib64 --enable-ssl --enable-shared --with-dpdk=yes --enable-Werror --enable-xpf
make -j 96 && make install
```

Step 5 (Optional) Configure OVS to start in service mode.

NOTE

You need to run multiple commands to manually start, stop, and restart the OVS service. Running OVS in service mode can greatly simplify related operations.

1. Switch to the **rhel** directory in the OVS source code directory, copy the **etc_init.d_openvswitch** file to the **/etc/init.d** directory, rename the file as **openvswitch**, and change the file execution permission to 755.

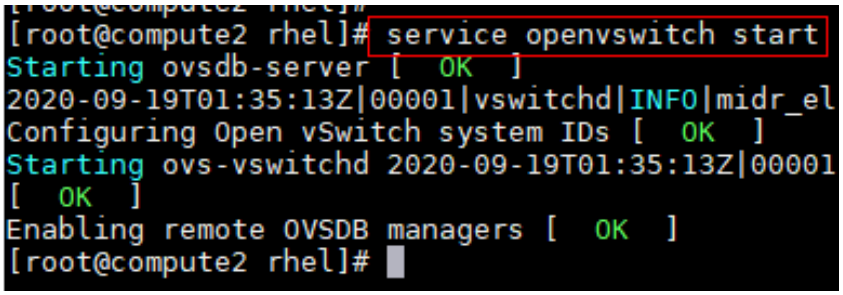
   ```
   cd rhel/
   cp etc_init_.d_openvswitch /etc/inin.d/openvswitch
   chmod 755 /etc/init.d/openvswitch
   ```

   ```
   [root@compute2 openvswitch-2.12.0]#
   [root@compute2 openvswitch-2.12.0]# cd rhel/
   [root@compute2 rhel]# cp etc_init.d_openvswitch /etc/init.d/openvswitch
   [root@compute2 rhel]# chmod 755 /etc/init.d/openvswitch
   ```

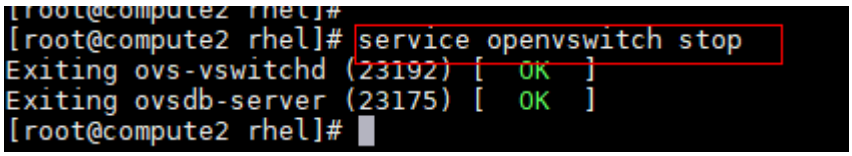
2. Run OVS in service mode.

   After the OVS startup configuration is complete (see **3.7 Running and Verifying XPF**), you can start, stop, and restart OVS in service mode.

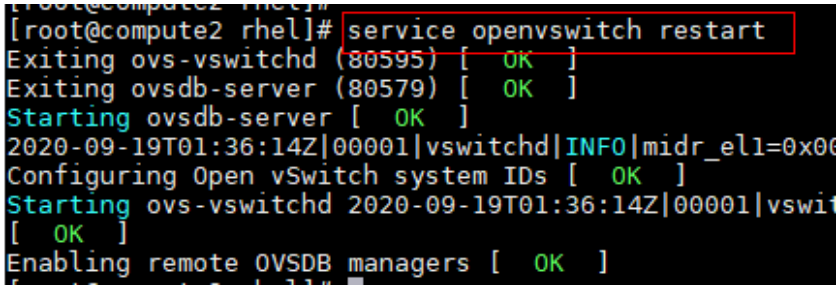
- Start the OVS service.
  service openvswitch start



- Stop the OVS service.
  service openvswitch stop



- Restart the OVS service.
  service openvswitch restart



----End

NOTE

- The XPF library depends on the open-source OVS, and the secondary developed OVS depends on the XPF library. Therefore, you need to compile OVS twice. The first step is to compile the open-source OVS and copy necessary header files. The second step is to compile the secondary developed OVS.
- The **boot.sh** script is used to generate the modified **configure** file by using Libtool, add the corresponding macro definition, and enable XPF code.

3.6 Configuring Logs

DPDK logs are output to dmesg by default. To facilitate fault locating, modify configuration files to output DPDK logs to OVS log files and configure log control methods. Perform the following configuration based on the site requirements.

Step 1 Create the DPDK log directory.

mkdir -p /var/log/dpdk

Step 2 Configure the Rsyslog configuration file of DPDK.

1. Create the **dpdk.conf** configuration file in the **/etc/rsyslog.d** directory.
   vim /etc/rsyslog.d/dpdk.conf

2. Add the following content to the file:

```
#DPDK_LOG_TAG "LibLogTag_DPDK"
template(name="template-dpdk" type="string" string="%TIMESTAMP:::date-rfc3339%|
%syslogseverity-text%|%programname%[%PROCID%]|%$!msg_pre%%$!msg_after%\n")
$outchannel dpdk,/var/log/dpdk/dpdk.log,2097152,/opt/esyslog/esyslog_log_rsyslog_dump.sh /var/log/
dpdk/dpdk.log dpdk

if ($msg contains "LibLogTag_DPDK" and $syslogseverity <= 7 ) then {
set $!msg_pre = field($msg,"LibLogTag_DPDK|",1);
set $!msg_after = field($msg,"LibLogTag_DPDK|",2);
:omfile:$dpdk;template-dpdk
stop
}


if ($msg contains "LibLogTag_DPDK" and $syslogseverity > 7 ) then {
/dev/null
stop
}
```

Step 3 Configure the DPDK log rollback file.

1. Create the **/etc/logrotate.d/dpdk** file.

```
vim /etc/logrotate.d/dpdk
```

2. Add the following content to the file:

```
/var/log/dpdk/dpdk.log {
hourly
compress
missingok
notifempty
maxsize 2048k
rotate 50
copytruncate
}
```

Step 4 Configure the OVS log rollback file.

1. Create the **/etc/logrotate.d/openvswitch** file.

```
vim /etc/logrotate.d/openvswitch
```

2. Add the following content to the file:

```
# Copyright (C) 2009, 2010, 2011, 2012 Nicira, Inc.
#
# Copying and distribution of this file, with or without modification,
# are permitted in any medium without royalty provided the copyright
# notice and this notice are preserved.  This file is offered as-is,
# without warranty of any kind.

/var/log/openvswitch/*.log {
    su root root
    hourly
    compress
    missingok
    maxsize 2048k
    rotate 50
    sharedscripts
    postrotate
        # Tell Open vSwitch daemons to reopen their log files
        if [ -d /var/run/openvswitch ]; then
            for ctl in /var/run/openvswitch/*.ctl; do
                ovs-appctl -t "$ctl" vlog/reopen 2>/dev/null || :
            done
        fi36
    endscript
}
```

Step 5 Configure the scheduled log processing file.

1. Create the **ovslogrotat** file in **/etc/cron.hourly**.
   vim /etc/cron.hourly/ovslogrotate
2. Add the following content to the file:

```
#!/bin/sh

/usr/sbin/logrotate -s /var/lib/logrotate/logrotate.status /etc/logrotate.d/dpdk
EXITVALUE=$?
if [ $EXITVALUE != 0 ]; then
    /usr/bin/logger -t logrotate "ALERT exited abnormally with [$EXITVALUE]"
fi

/usr/sbin/logrotate -s /var/lib/logrotate/logrotate.status /etc/logrotate.d/openvswitch
EXITVALUE=$?
if [ $EXITVALUE != 0 ]; then
    /usr/bin/logger -t logrotate "ALERT exited abnormally with [$EXITVALUE]"
fi

exit 0
```

3. Add the read and execute permissions to the file.
   chmod 0644 /etc/cron.hourly/ovslogrotate
   chmod +x /etc/cron.hourly/ovslogrotate

----End

3.7 Running and Verifying XPF

Step 1 Set the huge page memory and edit the startup items.

vim /etc/grub2-efi.cfg

Add **default_hugepagesz=512M hugepagesz=512M hugepages=64** to the file.

```
88 menuentry 'CentOS Linux (4.14.0-115.el7a.0.1.aarch64) 7 (AltArch)' --class centos --class gnu-linux --class gnu --class o
   s --unrestricted $menuentry_id_option 'gnulinux-4.14.0-115.el7a.0.1.aarch64-advanced-c11e89ec-12f0-42ae-8373-090f6fffb0b0
   ' {
89         load_video
90         set gfxpayload=keep
91         insmod gzio
92         insmod part_gpt
93         insmod xfs
94         set root='hd0,gpt2'
95         if [ x$feature_platform_search_hint = xy ]; then
96           search --no-floppy --fs-uuid --set=root --hint-ieee1275='ieee1275//sas/disk@2500,gpt2' --hint-bios=hd0,gpt2 --h
   int-efi=hd0,gpt2 --hint-baremetal=ahci0,gpt2  b797efe0-86db-40b4-9ae2-9360345c4d85
97         else
98           search --no-floppy --fs-uuid --set=root b797efe0-86db-40b4-9ae2-9360345c4d85
99         fi
100        linux /vmlinuz-4.14.0-115.el7a.0.1.aarch64 root=/dev/mapper/centos-root ro crashkernel=auto rd.lvm.lv=centos/root
    rd.lvm.lv=centos/swap LANG=en_US.UTF-8 default_hugepagesz=512M hugepagesz=512M hugepages=64
101        initrd /initramfs-4.14.0-115.el7a.0.1.aarch64.img
102 }
```

Step 2 Enable IOMMU and CPU isolation.

1. Open the **/etc/grub2-efi.cfg** file.
   vim /etc/grub2-efi.cfg
2. Add **isolcpus=0-5 iommu.passthrough=1** to the file.


```
menuentry 'CentOS Linux (4.14.0-115.el7a.0.1.aarch64) 7 (AltArch)' --class centos --class gnu-linux --class gnu --class os --unrestricted $menuentry_id_option 'gnulinux-4.14.0-115.el7a.0.1.aarch64-advanced-c11e89ec-12f0-42ae-8373-090f6fffb0b0' {
        load_video
        set gfxpayload=keep
        insmod gzio
        insmod part_gpt
        insmod xfs
        set root='hd0,gpt2'
        if [ x$feature_platform_search_hint = xy ]; then
          search --no-floppy --fs-uuid --set=root --hint-ieee1275='ieee1275//sas/disk@2500,gpt2' --hint-bios=hd0,gpt2 --hint-efi=hd0,gpt2 --hint-baremetal=ahci0,gpt2  b797efe0-86db-40b4-9ae2-9360345c4d85
        else
          search --no-floppy --fs-uuid --set=root b797efe0-86db-40b4-9ae2-9360345c4d85
        fi
        linux /vmlinuz-4.14.0-115.el7a.0.1.aarch64 root=/dev/mapper/centos-root ro crashkernel=auto rd.lvm.lv=centos/root rd.lvm.lv=centos/swap LANG=en_US.UTF-8 default_hugepagesz=512M hugepagesz=512M hugepages=64 isolcpus=0-5 iommu.passthrough=1
        initrd /initramfs-4.14.0-115.el7a.0.1.aarch64.img
}
```

Step 3 Restart the host for the configuration to take effect.

NOTE

To enable IOMMU, you also need to configure the BIOS in addition to configuring the startup items. For details, see **3.3 BIOS Settings**.

Step 4 Start OVS.

1. Create the OVS working directory.
   mkdir -p /var/run/openvswitch
   mkdir -p /var/log/openvswitch
2. Create the OVS database file.
   ovsdb-tool create /etc/openvswitch/conf.db
3. Start the ovsdb-server program.
   ovsdb-server --remote=punix:/var/run/openvswitch/db.sock --remote=db:Open_vSwitch,Open_vSwitch,manager_options --private-key=db:Open_vSwitch,SSL,private_key --certificate=db:Open_vSwitch,SSL,certificate --bootstrap-ca-cert=db:Open_vSwitch,SSL,ca_cert --pidfile --detach --log-file
4. Set OVS startup parameters.
   ovs-vsctl --no-wait set Open_vSwitch . other_config:dpdk-init=true other_config:dpdk-socket-mem="4096" other_config:dpdk-lcore-mask="0x1F" other_config:pmd-cpu-mask="0x1E"
5. Start OVS.
   ovs-vswitchd --pidfile --detach --log-file


```
[root@localhost ovs+dpdk]# ovs-vswitchd --pidfile --detach --log-file
2020-09-02T03:14:30Z|00001|vswitchd|INFO|midr_el1=0x00000000481fd010, part_id=0xd01, vendor_id=0x48
2020-09-02T03:14:30Z|00002|vlog|INFO|opened log file /var/log/openvswitch/ovs-vswitchd.log
2020-09-02T03:14:30Z|00003|ovs_numa|INFO|Discovered 32 CPU cores on NUMA node 2
2020-09-02T03:14:30Z|00004|ovs_numa|INFO|Discovered 32 CPU cores on NUMA node 1
2020-09-02T03:14:30Z|00005|ovs_numa|INFO|Discovered 32 CPU cores on NUMA node 3
2020-09-02T03:14:30Z|00006|ovs_numa|INFO|Discovered 32 CPU cores on NUMA node 0
2020-09-02T03:14:30Z|00007|ovs_numa|INFO|Discovered 4 NUMA nodes and 128 CPU cores
2020-09-02T03:14:30Z|00008|reconnect|INFO|unix:/var/run/openvswitch/db.sock: connecting...
2020-09-02T03:14:30Z|00009|reconnect|INFO|unix:/var/run/openvswitch/db.sock: connected
2020-09-02T03:14:30Z|00010|dpdk|INFO|Using DPDK 19.11.0
2020-09-02T03:14:30Z|00011|dpdk|INFO|DPDK Enabled - initializing...
2020-09-02T03:14:30Z|00012|dpdk|INFO|No vhost-sock-dir provided - defaulting to /var/run/openvswitch
2020-09-02T03:14:30Z|00013|dpdk|INFO|IOMMU support for vhost-user-client disabled.
2020-09-02T03:14:30Z|00014|dpdk|INFO|POSTCOPY support for vhost-user-client disabled.
2020-09-02T03:14:30Z|00015|dpdk|INFO|Per port memory for DPDK devices disabled.
2020-09-02T03:14:30Z|00016|dpdk|INFO|EAL ARGS: ovs-vswitchd -c 0x1F -d /lib64/librte_pmd_hinic.so --soc
ket-mem 4096 --socket-limit 4096.
2020-09-02T03:14:30Z|00017|dpdk|INFO|EAL: Detected 128 lcore(s)
2020-09-02T03:14:30Z|00018|dpdk|INFO|EAL: Detected 4 NUMA nodes
2020-09-02T03:14:30Z|00019|dpdk|INFO|EAL: Multi-process socket /var/run/dpdk/rte/mp_socket
2020-09-02T03:14:30Z|00020|dpdk|INFO|EAL: Selected IOVA mode 'VA'
2020-09-02T03:14:30Z|00021|dpdk|WARN|EAL: No available hugepages reported in hugepages-2048kB
2020-09-02T03:14:30Z|00022|dpdk|INFO|EAL: Probing VFIO support...
2020-09-02T03:14:30Z|00023|dpdk|INFO|EAL: PCI device 0000:03:00.0 on NUMA socket 0
2020-09-02T03:14:30Z|00024|dpdk|INFO|EAL:   probe driver: 19e5:1822 net_hinic
2020-09-02T03:14:30Z|00025|dpdk|INFO|EAL: PCI device 0000:04:00.0 on NUMA socket 0
2020-09-02T03:14:30Z|00026|dpdk|INFO|EAL:   probe driver: 19e5:1822 net_hinic
2020-09-02T03:14:30Z|00027|dpdk|INFO|EAL: PCI device 0000:05:00.0 on NUMA socket 0
2020-09-02T03:14:30Z|00028|dpdk|INFO|EAL:   probe driver: 19e5:1822 net_hinic
2020-09-02T03:14:30Z|00029|dpdk|INFO|EAL: PCI device 0000:06:00.0 on NUMA socket 0
2020-09-02T03:14:30Z|00030|dpdk|INFO|EAL:   probe driver: 19e5:1822 net_hinic
2020-09-02T03:14:30Z|00031|dpdk|INFO|EAL: PCI device 0000:7d:00.0 on NUMA socket 0
2020-09-02T03:14:30Z|00032|dpdk|INFO|EAL:   probe driver: 19e5:a222 net_hns3
2020-09-02T03:14:30Z|00033|dpdk|INFO|EAL: PCI device 0000:7d:00.1 on NUMA socket 0
2020-09-02T03:14:30Z|00034|dpdk|INFO|EAL:   probe driver: 19e5:a221 net_hns3
2020-09-02T03:14:30Z|00035|dpdk|INFO|EAL: PCI device 0000:7d:00.2 on NUMA socket 0
2020-09-02T03:14:30Z|00036|dpdk|INFO|EAL:   probe driver: 19e5:a222 net_hns3
2020-09-02T03:14:30Z|00037|dpdk|INFO|EAL: PCI device 0000:7d:00.3 on NUMA socket 0
```

Step 5 Bind the NIC to the user-mode DPDK.

1. Load the igb_uio driver.
   modprobe igb_uio

   NOTE

   If this command is executed for the first time, run the **depmod** command to enable the system to process driver dependencies. The driver is provided by DPDK and is installed in **/lib/modules/4.14.0-115.el7a.0.1.aarch64/extra/dpdk/igb_uio.ko** by default. If the system restarts, you need to load the driver again.

2. View network port information.
   dpdk-devbind -s


```
[root@localhost home]# dpdk-devbind -s

Network devices using kernel driver
===================================
0000:03:00.0 'Hi1822 Family (4*25GE) 1822' if=enp3s0 drv=hinic unused=igb_uio
0000:04:00.0 'Hi1822 Family (4*25GE) 1822' if=enp4s0 drv=hinic unused=igb_uio
0000:05:00.0 'Hi1822 Family (4*25GE) 1822' if=enp5s0 drv=hinic unused=igb_uio
0000:06:00.0 'Hi1822 Family (4*25GE) 1822' if=enp6s0 drv=hinic unused=igb_uio
0000:7d:00.0 'HNS GE/10GE/25GE RDMA Network Controller a222' if=enp125s0f0 drv=hns3 unused=hns_roce_hw_
v2,igb_uio *Active*
0000:7d:00.1 'HNS GE/10GE/25GE Network Controller a221' if=enp125s0f1 drv=hns3 unused=igb_uio
0000:7d:00.2 'HNS GE/10GE/25GE RDMA Network Controller a222' if=enp125s0f2 drv=hns3 unused=hns_roce_hw_
v2,igb_uio
0000:7d:00.3 'HNS GE/10GE/25GE Network Controller a221' if=enp125s0f3 drv=hns3 unused=igb_uio

No 'Baseband' devices detected
==============================

Other Crypto devices
====================
0000:75:00.0 'HiSilicon SEC Engine a255' unused=igb_uio
0000:b5:00.0 'HiSilicon SEC Engine a255' unused=igb_uio

No 'Eventdev' devices detected
==============================

No 'Mempool' devices detected
=============================

No 'Compress' devices detected
==============================

No 'Misc (rawdev)' devices detected
===================================
[root@localhost home]#
```

Find the PCI address of the network port to be bound.

NOTE

The network port to be bound must be in the down state. Otherwise, the binding fails.

3. Bind the NIC to the user-mode DPDK.

```
dpdk-devbind --bind=igb_uio 0000:05:00.0
dpdk-devbind --bind=igb_uio 0000:06:00.0
```

NOTE

To roll back the operation, run the following command:

```
dpdk-devbind -u 0000:05:00.0
```

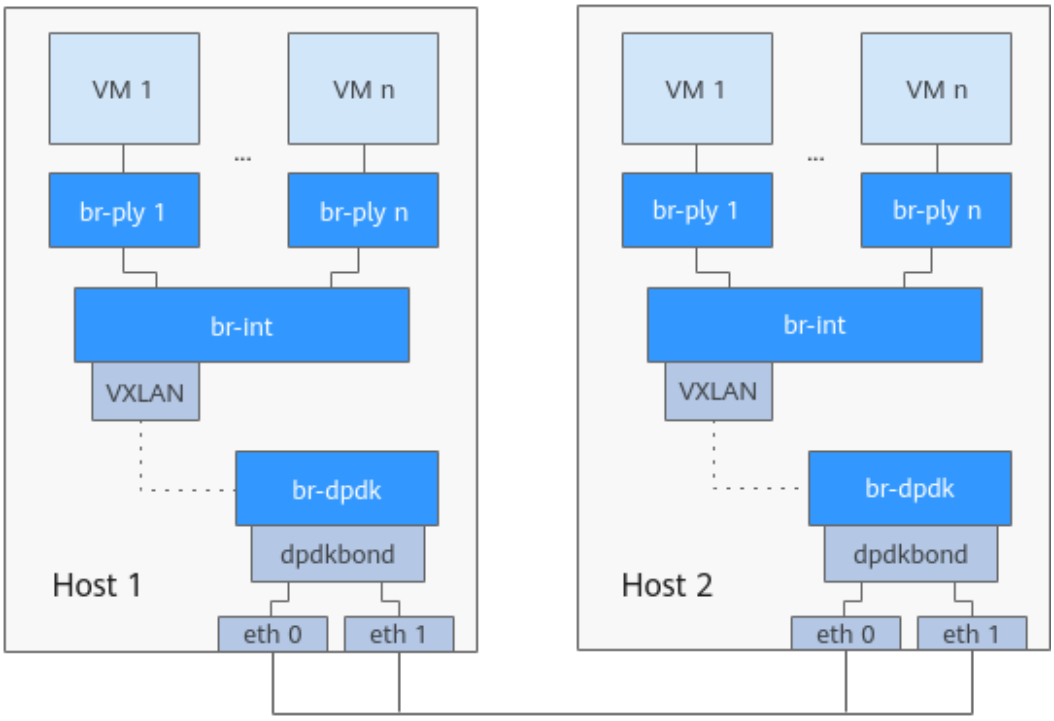
4. Check whether the binding is successful.


```
[root@localhost home]# dpdk-devbind -s

Network devices using DPDK-compatible driver
============================================
0000:05:00.0 'Hi1822 Family (4*25GE) 1822' drv=igb_uio unused=hinic
0000:06:00.0 'Hi1822 Family (4*25GE) 1822' drv=igb_uio unused=hinic

Network devices using kernel driver
===================================
0000:03:00.0 'Hi1822 Family (4*25GE) 1822' if=enp3s0 drv=hinic unused=igb_uio
0000:04:00.0 'Hi1822 Family (4*25GE) 1822' if=enp4s0 drv=hinic unused=igb_uio
0000:7d:00.0 'HNS GE/10GE/25GE RDMA Network Controller a222' if=enp125s0f0 drv=hns3 unused=hns_roce_hw_
v2,igb_uio *Active*
0000:7d:00.1 'HNS GE/10GE/25GE Network Controller a221' if=enp125s0f1 drv=hns3 unused=igb_uio
0000:7d:00.2 'HNS GE/10GE/25GE RDMA Network Controller a222' if=enp125s0f2 drv=hns3 unused=hns_roce_hw_
v2,igb_uio
0000:7d:00.3 'HNS GE/10GE/25GE Network Controller a221' if=enp125s0f3 drv=hns3 unused=igb_uio

No 'Baseband' devices detected
==============================

Other Crypto devices
====================
0000:75:00.0 'HiSilicon SEC Engine a255' unused=igb_uio
0000:b5:00.0 'HiSilicon SEC Engine a255' unused=igb_uio

No 'Eventdev' devices detected
==============================

No 'Mempool' devices detected
=============================

No 'Compress' devices detected
==============================

No 'Misc (rawdev)' devices detected
===================================
[root@localhost home]#
```

Step 6 Create a network.

The network to be verified is a typical OVS network, as shown in **Figure 3-1**.

Figure 3-1 OVS networking



NOTE

The following lists the commands run on Host 1. The commands run on Host 2 are similar, except that the IP address of br-dpdk on Host 2 is different.

1. Add and configure the br-dpdk bridge.

```
ovs-vsctl add-br br-dpdk -- set bridge br-dpdk datapath_type=netdev
ovs-vsctl add-bond br-dpdk dpdk-bond p0 p1 -- set Interface p0 type=dpdk options:dpdk-devargs=0000:05:00.0 -- set Interface p1 type=dpdk options:dpdk-devargs=0000:06:00.0
ovs-vsctl set port dpdk-bond bond_mode=balance-tcp
ovs-vsctl set port dpdk-bond lacp=active
ifconfig br-dpdk 192.168.2.1/24 up
```

NOTE

The **ifconfig br-dpdk 192.168.2.1/24 up** command is used to configure a virtual extensible LAN (VXLAN) tunnel (**192.168.2.1** indicates the IP address of the br-dpdk bridge). Run the **ifconfig br-dpdk 192.168.2.2/24 up** command on Host 2. The network segment of the tunnel is different from that of the VM.

2. Add and configure the br-int bridge.

```
ovs-vsctl add-br br-int -- set bridge br-int datapath_type=netdev
ovs-vsctl add-port br-int vxlan0 --  set Interface vxlan0 type=vxlan options:local_ip=192.168.2.1 options:remote_ip=192.168.2.2
```

NOTE

In this networking, br-int has a VXLAN port, and the VXLAN header is added to all outgoing traffic of the host. **local_ip** of the VXLAN port is set to the IP address of the local br-dpdk, and **remote_ip** is set to the IP address of the peer br-dpdk.

3. Add and configure the br-plyn bridge.

```
ovs-vsctl add-br br-ply1 -- set bridge br-ply1 datapath_type=netdev
ovs-vsctl add-port br-ply1 tap1 -- set Interface tap1 type=dpdkvhostuserclient options:vhost-server-path=/var/run/openvswitch/tap1
ovs-vsctl add-port br-ply1 p-tap1-int --  set Interface p-tap1-int type=patch options:peer=p-tap1
ovs-vsctl add-port br-int p-tap1 --  set Interface p-tap1 type=patch options:peer=p-tap1-int
```

NOTE

In this networking, a br-ply bridge is added each time a VM is added. The bridge has a dpdkvhostuser port for the VM, and the patch port is connected to the br-int bridge.

4. Verify the networking.

```
ovs-vsctl show
```



```
[root@localhost home]# ovs-vsctl show
2a6fd512-48eb-40e1-a511-f08ee6f81d72
    Bridge "br-ply1"
        datapath_type: netdev
        Port "br-ply1"
            Interface "br-ply1"
                type: internal
        Port "tap1"
            Interface "tap1"
                type: dpdkvhostuserclient
                options: {vhost-server-path="/var/run/openvswitch/tap1"}
        Port "p-tap1-int"
            Interface "p-tap1-int"
                type: patch
                options: {peer="p-tap1"}
    Bridge br-int
        datapath_type: netdev
        Port "vxlan0"
            Interface "vxlan0"
                type: vxlan
                options: {local_ip="192.168.2.1", remote_ip="192.168.2.2"}
        Port "p-tap1"
            Interface "p-tap1"
                type: patch
                options: {peer="p-tap1-int"}
        Port br-int
            Interface br-int
                type: internal
    Bridge br-dpdk
        datapath_type: netdev
        Port dpdk-bond
            Interface "p0"
                type: dpdk
                options: {dpdk-devargs="0000:05:00.0"}
            Interface "p1"
                type: dpdk
                options: {dpdk-devargs="0000:06:00.0"}
        Port br-dpdk
            Interface br-dpdk
                type: internal
    ovs_version: "2.12.0"
[root@localhost home]#
```

5. Check whether the br-dpdk bridge at the local end is connected to the br-dpdk bridge at the peer end.

ping 192.168.2.2

```
[root@localhost home]# ping 192.168.2.2
PING 192.168.2.2 (192.168.2.2) 56(84) bytes of data.
64 bytes from 192.168.2.2: icmp_seq=1 ttl=64 time=8.76 ms
64 bytes from 192.168.2.2: icmp_seq=2 ttl=64 time=14.7 ms
64 bytes from 192.168.2.2: icmp_seq=3 ttl=64 time=2.19 ms
64 bytes from 192.168.2.2: icmp_seq=4 ttl=64 time=2.18 ms
```

Step 7 Start the VM.

When configuring a VM, pay attention to the huge page memory and network port configuration. The following VM configuration file is for reference:

```
<domain type='kvm'>
  <name>VM1</name>
  <uuid>fb8eb9ff-21a7-42ad-b233-2a6e0470e0b5</uuid>
  <memory unit='KiB'>2097152</memory>
  <currentMemory unit='KiB'>2097152</currentMemory>
  <memoryBacking>
```



```
  <hugepages>
    <page size='524288' unit='KiB' nodeset='0'/>
  </hugepages>
  <locked/>
</memoryBacking>
<vcpu placement='static'>4</vcpu>
<cputune>
  <vcpupin vcpu='0' cpuset='6'/>
  <vcpupin vcpu='1' cpuset='7'/>
  <vcpupin vcpu='2' cpuset='8'/>
  <vcpupin vcpu='3' cpuset='9'/>
  <emulatorpin cpuset='0-3'/>
</cputune>
<numatune>
  <memory mode='strict' nodeset='0'/>
</numatune>
<os>
  <type arch='aarch64' machine='virt-rhel7.6.0'>hvm</type>
  <loader readonly='yes' type='pflash'>/usr/share/AAVMF/AAVMF_CODE.fd</loader>
  <nvram>/var/lib/libvirt/qemu/nvram/VM1_VARS.fd</nvram>
  <boot dev='hd'/>
</os>
<features>
  <acpi/>
  <gic version='3'/>
</features>
<cpu mode='host-passthrough' check='none'>
  <topology sockets='1' cores='4' threads='1'/>
  <numa>
    <cell id='0' cpus='0-3' memory='2097152' unit='KiB' memAccess='shared'/>
  </numa>
</cpu>
<clock offset='utc'/>
<on_poweroff>destroy</on_poweroff>
<on_reboot>restart</on_reboot>
<on_crash>destroy</on_crash>
<devices>
  <emulator>/usr/libexec/qemu-kvm</emulator>
  <disk type='file' device='disk'>
    <driver name='qemu' type='qcow2'/>
    <source file='/home/kvm/images/1.img'/>
    <target dev='vda' bus='virtio'/>
    <address type='pci' domain='0x0000' bus='0x04' slot='0x00' function='0x0'/>
  </disk>
  <disk type='file' device='cdrom'>
    <driver name='qemu' type='raw'/>
    <target dev='sda' bus='scsi'/>
    <readonly/>
    <address type='drive' controller='0' bus='0' target='0' unit='0'/>
  </disk>
  <controller type='usb' index='0' model='qemu-xhci' ports='8'>
    <address type='pci' domain='0x0000' bus='0x02' slot='0x00' function='0x0'/>
  </controller>
  <controller type='scsi' index='0' model='virtio-scsi'>
    <address type='pci' domain='0x0000' bus='0x03' slot='0x00' function='0x0'/>
  </controller>
  <controller type='pci' index='0' model='pcie-root'/>
  <controller type='pci' index='1' model='pcie-root-port'>
    <model name='pcie-root-port'/>
    <target chassis='1' port='0x8'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x01' function='0x0' multifunction='on'/>
  </controller>
  <controller type='pci' index='2' model='pcie-root-port'>
    <model name='pcie-root-port'/>
    <target chassis='2' port='0x9'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x01' function='0x1'/>
  </controller>
  <controller type='pci' index='3' model='pcie-root-port'>
    <model name='pcie-root-port'/>
```



```
    <target chassis='3' port='0xa'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x01' function='0x2'/>
   </controller>
   <controller type='pci' index='4' model='pcie-root-port'>
    <model name='pcie-root-port'/>
    <target chassis='4' port='0xb'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x01' function='0x3'/>
   </controller>
   <controller type='pci' index='5' model='pcie-root-port'>
    <model name='pcie-root-port'/>
    <target chassis='5' port='0xc'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x01' function='0x4'/>
   </controller>
   <interface type='vhostuser'>
    <source type='unix' path='/var/run/openvswitch/tap1' mode='server'/>
    <target dev='tap1'/>
    <model type='virtio'/>
    <driver name='vhost' queues='4' rx_queue_size='1024' tx_queue_size='1024'/>
   </interface>
   <serial type='pty'>
    <target type='system-serial' port='0'>
     <model name='pl011'/>
    </target>
   </serial>
   <console type='pty'>
    <target type='serial' port='0'/>
   </console>
  </devices>
</domain>
```

NOTE

- The **memoryBacking** section specifies the specifications of the huge page memory to be applied for. The huge page memory of 512 MB is configured for the host. Therefore, specify 512 MB in this section.
- The **numatune** section specifies the number of the NUMA node on which the memory is applied for. The NUMA node number of the VM must be the same as that of the NIC.
- The **numa** subsection specifies the VM memory mode. In this example, the virtual network port of vhostuser is configured. The VM must have a shared huge page memory.
- The **interface** section specifies the virtual network port of the VM.
  - In the **source** section, **path** specifies the location of the socket file for communication between the host and VM, and **mode** specifies the type of the VM socket. The OVS is configured with the dpdkvhostuserclient port, which is in the client mode. Therefore, **mode** is set to server in this example.
  - The **target** section specifies the name of socket used by the VM.
  - The **driver** section specifies the driver used by the VM and specifies the queue and queue depth. In this example, the vhostuser driver is used.

Step 8 Verify cross-host VM communication.

Check whether the VM on Host 1 can communicate with the VM on Host 2.

```
ping 192.168.1.21
```



```
[root@localhost ~]# ifconfig eth0 192.168.1.11/24 up
[root@localhost ~]# ping 192.168.1.21
PING 192.168.1.21 (192.168.1.21) 56(84) bytes of data.
64 bytes from 192.168.1.21: icmp_seq=1 ttl=64 time=1050 ms
64 bytes from 192.168.1.21: icmp_seq=2 ttl=64 time=0.764 ms
64 bytes from 192.168.1.21: icmp_seq=3 ttl=64 time=0.097 ms
64 bytes from 192.168.1.21: icmp_seq=4 ttl=64 time=0.091 ms
64 bytes from 192.168.1.21: icmp_seq=5 ttl=64 time=0.090 ms

--- 192.168.1.21 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4089ms
rtt min/avg/max/mdev = 0.090/210.271/1050.314/420.021 ms, pipe 2
[root@localhost ~]#
```

NOTE

The data length is added to the VXLAN header. The default maximum transmission unit (MTU) of the host is 1500 bytes. The MTU length of the VM needs to be reduced to ensure normal communication. It is recommended that the MTU length of the VM network port be 1400 bytes.

----End

3.8 Troubleshooting

Problem 1: Insufficient Permission Upon VM Startup

Symptom

The VM fails to be started, and the following information is displayed:

```
[root@localhost home]# virsh start VM1
error: Failed to start domain VM1
error: internal error: qemu unexpectedly closed the monitor: 2020-08-31T04:11:16.783295Z qemu-kvm: -cha
rdev socket,id=charnet0,path=/var/run/openvswitch/tap1,server: Failed to bind socket to /var/run/openvs
witch/tap1: Permission denied
```

Possible Cause

- The SELinux of the system prevents the VM from creating the **/var/run/openvswitch/tap1** file.
- The QEMU process does not have the permission to access the file.

Procedure

1. In the QEMU configuration file **/etc/libvirt/qemu.conf**, ensure that the QEMU process user has the read and execute permissions on the **/var/run/openvswitch** directory. The following shows the default line numbers for configuration:


```
436 # Some examples of valid values are:
437 #
438 #       user = "qemu"   # A user named "qemu"
439 #       user = "+0"     # Super user (uid=0)
440 #       user = "100"    # A user named "100" or a user with uid=100
441 #
442 user = "root"
443
444 # The group for QEMU processes run by the system instance. It can be
445 # specified in a similar way to user.
446 group = "root"
447
```

Restart the libvirtd service.

```
systemctl restart libvirtd
```

2. Ensure that SELinux does not block processes.
   a. Disable SELinux.
      ```
      setenforce 0
      ```
   b. Start the VM.
      ```
      virsh start VM1
      ```
      The VM is started successfully, and the fault is rectified.

      ```
      [root@localhost home]# setenforce 0
      [root@localhost home]# virsh start VM1
      Domain VM1 started
      ```

3.9 OVS Command Description

Table 3-4 lists the OVS commands used for maintaining the status and functions of software offloading.

Table 3-4 OVS command description

Command	Function
ovs-appctl hwoff-flow-agent/add-protolist	Adds a protocol blacklist. Packets transmitted using the listed protocols are not offloaded.
ovs-appctl hwoff-flow-agent/add-rapid-protolist	Adds a protocol whitelist. Packets transmitted using the listed protocols are offloaded.
ovs-appctl hwoff-flow-agent/clear-error-stats	Clears the error statistics collected by hwoff-flow-agent.
ovs-appctl hwoff-flow-agent/disable	Disables the flow table offloading function (software offloading).
ovs-appctl hwoff-flow-agent/dump-ct	Queries the connection tracking (CT) information recorded by hwoff-flow-agent.
ovs-appctl hwoff-flow-agent/dump-policy	Queries parameters in the offloading policy of hwoff-flow-agent.

Command	Function
ovs-appctl hwoff-flow-agent/dump-protolist	Queries the protocol blacklist that has been added to hwoff-flow-agent.
ovs-appctl hwoff-flow-agent/dump-rapid-protolist	Queries the protocol whitelist that has been added to hwoff-flow-agent.
ovs-appctl hwoff-flow-agent/enable	Enables the flow table offloading function (software offloading). This function is enabled by default.
ovs-appctl hwoff-flow-agent/error-stats	Queries the error statistics collected by hwoff-flow-agent.
ovs-appctl hwoff-flow-agent/flush-flow	Clears information about hardware flow tables.
ovs-appctl hwoff-flow-agent/flush-protolist	Clears the protocol blacklist.
ovs-appctl hwoff-flow-agent/flush-rapid-protolist	Clears the protocol whitelist.
ovs-appctl hwoff-flow-agent/get-offloadrx-threshold	Obtains the threshold for automatically deleting cached flow tables.
ovs-appctl hwoff-flow-agent/live-time	Sets the aging time of flow tables offloaded to hwoff-flow-agent.
ovs-appctl hwoff-flow-agent/policy	Configures policy parameters for flow table offloading.
ovs-appctl hwoff-flow-agent/print-flow	Prints flow table details in OVS logs during flow table offloading.
ovs-appctl hwoff-flow-agent/set-protolist-mode	Specifies the protocol list type (blacklist or whitelist) recorded by hwoff-flow-agent.
ovs-appctl hwoff-flow-agent/stats	Lists non-error statistics collected by hwoff-flow-agent.
ovs-appctl hwoff-flow-agent/use-offloadrx-threshold	Sets the threshold for automatically deleting cached flow tables.
ovs-appctl hwoff/dump-hwoff-flows	Queries information about hardware flow tables.
ovs-appctl hwoff/shmap-dump-hw	Queries the unique flow IDs (UFIDs) of all hardware flow tables in the software-hardware flow table mapping maintained by the host; queries the UFID of the software flow table associated with the UFID of a specified hardware flow table.

Command	Function
ovs-appctl hwoff/shmap-dump-sw	Queries the UFIDs of all software flow tables associated with hardware flow tables in the software-hardware flow table mapping maintained by the host; queries the UFID of the hardware flow table associated with the UFID and type of a specified software flow table.
ovs-appctl hwoff/shmap-error-stats	Queries the error statistics of the software-hardware flow table mapping.
ovs-appctl hwoff/shmap-flush	Clears the software-hardware flow table mapping.

The following describes the commands in detail.

ovs-appctl hwoff-flow-agent/add-protolist

Syntax

ovs-appctl hwoff-flow-agent/add-protolist [PROTOLIST]

Function

Adds a protocol blacklist. Packets transmitted using the listed protocols are not offloaded.

Parameter Description

Parameter	Mandatory	Description
PROTOLIST	Yes	Protocol number list. The protocol number ranges from 0 to 255 or from 1536 to 65535, for example, 23, 25, or 1560.

Example

```
ovs-appctl hwoff-flow-agent/add-protolist 23,25,1560
```

ovs-appctl hwoff-flow-agent/add-rapid-protolist

Syntax

hwoff-flow-agent/add-rapid-protolist [PROTOLIST]

Function

Adds a protocol whitelist. Packets transmitted using the listed protocols are offloaded.

Parameter Description

Parameter	Mandatory	Description
PROTOLIST	Yes	Protocol number table. The protocol number can be 0, 1, 2, or 3 (0 indicates UDP, 1 indicates TCP, 2 indicates ICMP, and 3 indicates ICMPv6).

Example

```
ovs-appctl hwoff-flow-agent/add-rapid-protolist 0,1,2,3
```

ovs-appctl hwoff-flow-agent/clear-error-stats

Syntax

ovs-appctl hwoff-flow-agent/clear-error-stats

Function

Clears the error statistics collected by hwoff-flow-agent.

Parameter Description

N/A

Example

```
ovs-appctl hwoff-flow-agent/clear-error-stats
```

ovs-appctl hwoff-flow-agent/disable

Syntax

ovs-appctl hwoff-flow-agent/disable

Function

Disables the flow table offloading function (software offloading).

Parameter Description

N/A

Example

```
ovs-appctl hwoff-flow-agent/disable
```

ovs-appctl hwoff-flow-agent/dump-ct

Syntax

ovs-appctl hwoff-flow-agent/dump-ct [-m] [1-1000]

Function

Queries the connection tracking information recorded by hwoff-flow-agent.

Parameter Description

Parameter	Mandatory	Description
-m	No	Views details of the connection tracking table.
[1-1000]	No	Specifies the displayed number of records in the connection tracking table.

Example

```
ovs-appctl hwoff-flow-agent/dump-ct -m 1
```

ovs-appctl hwoff-flow-agent/dump-policy

Syntax

ovs-appctl hwoff-flow-agent/dump-policy

Function

Queries parameters in the offloading policy of hwoff-flow-agent.

Parameter Description

N/A

Example

```
ovs-appctl hwoff-flow-agent/dump-policy
```

ovs-appctl hwoff-flow-agent/dump-protolist

Syntax

ovs-appctl hwoff-flow-agent/dump-protolist

Function

Queries the protocol blacklist that has been added to hwoff-flow-agent.

Parameter Description

N/A

Example

```
ovs-appctl hwoff-flow-agent/dump-protolist
```


ovs-appctl hwoff-flow-agent/dump-rapid-protolist

Syntax

ovs-appctl hwoff-flow-agent/dump-rapid-protolist

Function

Queries the protocol whitelist that has been added to hwoff-flow-agent.

Parameter Description

N/A

Example

```
ovs-appctl hwoff-flow-agent/dump-rapid-protolist
```

ovs-appctl hwoff-flow-agent/enable

Syntax

ovs-appctl hwoff-flow-agent/enable

Function

Enables the flow table offloading function (software offloading). This function is enabled by default.

Parameter Description

N/A

Example

```
ovs-appctl hwoff-flow-agent/enable
```

ovs-appctl hwoff-flow-agent/error-stats

Syntax

ovs-appctl hwoff-flow-agent/error-stats

Function

Queries the error statistics collected by hwoff-flow-agent.

Parameter Description

N/A

Example

```
ovs-appctl hwoff-flow-agent/error-stats
```

ovs-appctl hwoff-flow-agent/flush-flow

Syntax

ovs-appctl hwoff-flow-agent/flush-flow

Function

Clears information about hardware flow tables.

Parameter Description

N/A

Example

```
ovs-appctl hwoff-flow-agent/flush-flow
```

ovs-appctl hwoff-flow-agent/flush-protolist

Syntax

ovs-appctl hwoff-flow-agent/flush-protolist

Function

Clears the protocol blacklist.

Parameter Description

N/A

Example

```
ovs-appctl hwoff-flow-agent/flush-protolist
```

ovs-appctl hwoff-flow-agent/flush-rapid-protolist

Syntax

ovs-appctl hwoff-flow-agent/flush-rapid-protolist

Function

Clears the protocol whitelist.

Parameter Description

N/A

Example

```
ovs-appctl hwoff-flow-agent/flush-rapid-protolist
```

ovs-appctl hwoff-flow-agent/get-offloadrx-threshold

Syntax

ovs-appctl hwoff-flow-agent/get-offloadrx-threshold

Function

Obtains the threshold for automatically deleting cached flow tables.

Parameter Description

N/A

Example

```
ovs-appctl hwoff-flow-agent/get-offloadrx-threshold
```


ovs-appctl hwoff-flow-agent/live-time

Syntax

ovs-appctl hwoff-flow-agent/live-time TIME

Function

Sets the aging time of flow tables offloaded to hwoff-flow-agent.

Parameter Description

Parameter	Mandatory	Description
TIME	Yes	Aging time, in ms. The value ranges from 1 to 86400000.

Example

```
ovs-appctl hwoff-flow-agent/live-time 200000
```

ovs-appctl hwoff-flow-agent/policy

Syntax

ovs-appctl hwoff-flow-agent/policy enable_permission <0|1> | user_default_permissions <0-10000000> | permission_update_interval <100-10000000 ms> | permission_credit <0-10000> offload_packet_num <0-10000000> | offload_delay_mode <0|1> | duration_before_offload <0-10000000 ms> | offload_pps <0-10000000> | garbage_clean_interval <100-10000000 ms> | garbage_max_life <10-10000000 ms> | enable_clean_window <0|1> | clean_pps <0-10000000> | clean_window_update_interval <100-10000000 ms> | limit_flow_nums <2000000-8000000> | user_idle_time <1-10000000>

Function

Configures policy parameters for flow table offloading.

Parameter Description

Parameter	Mandatory	Description
enable_permission	No	Indicates whether to enable offloading. The values are as follows: **0**: disable **1**: enable
user_default_permissions	No	Maximum number of offloaded table flows. The value ranges from 0 to 10000000.

Parameter	Mandatory	Description
permission_update_interval	No	Interval (in ms) for updating offloaded table flows. The value ranges from 100 to 10000000.
permission_credit	No	Maximum number of established connections. The value ranges from 0 to 10000. The value **0** indicates no limit.
offload_packet_num	No	Maximum number of packets before offloading. The value ranges from 0 to 10000000.
offload_delay_mode	No	Indicates whether to enable offloading delay. The values are as follows: **0**: disable **1**: enable
duration_before_offload	No	Offloading delay time, in ms. The value ranges from 0 to 10000000.
offload_pps	No	Offloading delay PPS (PPS is short for packet per second). Offloading occurs when the traffic reaches this value. The value ranges from 0 to 10000000.
garbage_clean_interval	No	Interval (in ms) for deleting garbage table flows. The value ranges from 100 to 10000000.
garbage_max_life	No	Maximum matching time (in ms) of table flows. If matching fails after this time, the table flows are deleted as garbage table flows. The value ranges from 10 to 10000000.

Parameter	Mandatory	Description
enable_clean_window	No	Threshold for deleting connection tracking status from the cache. The value ranges from 0 to 10000000. The value **0** indicates that the function is disabled.
clean_pps	No	Flow table replacement PPS. The old flow tables are replaced by new flow tables when the traffic reaches this value. The value ranges from 0 to 10000000.
clean_window_update_interval	No	Interval (in ms) for deleting connection tracking status from the cache. The value ranges from 100 to 10000000.
limit_flow_nums	No	Maximum number of flow tables generated after offloading. The value ranges from 2000000 to 8000000.
user_idle_time	No	Sets the connection tracking aging time (in second) for the hwoff-flow-agent offloading policy. The value ranges from 1 to 10000000.

Example

```
ovs-appctl hwoff-flow-agent/policy enable_permission 1
```

ovs-appctl hwoff-flow-agent/print-flow

Syntax

ovs-appctl hwoff-flow-agent/print-flow num

Function

Prints flow table details in OVS logs during flow table offloading.

Parameter Description

Parameter	Mandatory	Description
num	Yes	Prints the details of the *Nth* offloading. The value ranges from 0 to 15. The default value is **0**, indicating that the details are not printed.

Example

```
ovs-appctl hwoff-flow-agent/print-flow 1
```

ovs-appctl hwoff-flow-agent/set-protolist-mode

Syntax

ovs-appctl hwoff-flow-agent/set-protolist-mode [black] | [white]

Function

Specifies the protocol list type (blacklist or whitelist) recorded by hwoff-flow-agent.

This parameter must be used together with **ovs-appctl hwoff-flow-agent/add-protolist**.

Parameter Description

Parameter	Mandatory	Description
black	No	Prohibits offloading of the packets transmitted using the listed protocols.
white	No	Allows offloading of the packets transmitted using the listed protocols.

Example

- Configuring the protocol list
  ```
  ovs-appctl  hwoff-flow-agent/add-protolist 23,25,1560
  ```
- Prohibiting offloading of the packets transmitted using the listed protocols
  ```
  ovs-appctl hwoff-flow-agent/set-protolist-mode black
  ```

ovs-appctl hwoff-flow-agent/stats

Syntax

ovs-appctl hwoff-flow-agent/stats

Function

Lists non-error statistics collected by hwoff-flow-agent.

Parameter Description

N/A

Example

```
ovs-appctl hwoff-flow-agent/stats
```

ovs-appctl hwoff-flow-agent/use-offloadrx-threshold

Syntax

ovs-appctl hwoff-flow-agent/use-offloadrx-threshold NUM

Function

Sets the threshold for automatically deleting cached flow tables.

Parameter Description

Parameter	Mandatory	Description
NUM	Yes	Threshold for automatically deleting cached flow tables. The value ranges from 1 to 512.

Example

```
ovs-appctl hwoff-flow-agent/use-offloadrx-threshold 256
```

ovs-appctl hwoff/dump-hwoff-flows

Syntax

hwoff/dump-hwoff-flows [-m]|[ufid ufid]|[check]|[stop]|[-f file]|[-n]|[-h]

Function

Queries information about hardware flow tables.

Parameter Description

Parameter	Mandatory	Description
ufid hw_ufid	No	Queries information about hardware flow tables based on UFIDs. (The UFID is in the hexadecimal format.)

Parameter	Mandatory	Description
-n	No	Displays the total number of hardware flow tables that have been offloaded.
-f file	No	(Recommended) Dumps flow tables to a specified file (the file path is an absolute path) when the number of flow tables exceeds 1000. Before using this parameter, use the **-n** parameter to determine the total number of current hardware flow tables.
check	No	Queries the progress of dumping hardware flow tables to a file.
stop	No	Forcibly stops the operation of dumping flow tables to a file.
-h	No	Displays help information.

Example

- Displaying the total number of hardware flow tables that have been offloaded
  ```
  ovs-appctl hwoff/dump-hwoff-flows -n
  ```
- Dumping information about hardware flow tables in command line mode (It is recommended that the number of flow tables be less than 1000.)
  ```
  ovs-appctl hwoff/dump-hwoff-flows
  ```

  NOTE

  - Before running this command, run the **ovs-appctl hwoff/dump-hwoff-flows -n** command to check the total number of flow tables.
  - When the total number of flow tables exceeds 1000, run the **ovs-appctl hwoff/dump-hwoff-flows -f *Absolute path of a file*** command to dump the flow table information to a file. Otherwise, the command execution will be suspended for a long time. If this command is executed repeatedly, flow table information is repeatedly added to the file, and the file size keeps increasing. In this case, you need to manually clear the file.
  - The dumped hardware flow table information (the same as the information displayed by running the open-source **dpctl/dump-flows** command) contains the source IP address, destination IP address, source MAC address, destination MAC address, and quintuple information of the outbound interface.

- Saving information about hardware flow tables to a specified file
  ovs-appctl hwoff/dump-hwoff-flows -f /root/test.log
- Querying information about hardware flow tables based on UFIDs
  ovs-appctl hwoff/dump-hwoff-flows ufid 12345678-12345678
- Querying the progress of dumping hardware flow tables to a file
  ovs-appctl hwoff/dump-hwoff-flows check
- Forcibly stopping the operation of dumping flow tables to a file
  ovs-appctl hwoff/dump-hwoff-flows stop

ovs-appctl hwoff/shmap-dump-hw

Syntax

ovs-appctl hwoff/shmap-dump-hw [all | hw_ufid]

Function

Queries the UFIDs of all hardware flow tables in the software-hardware flow table mapping maintained by the host; queries the UFID of the software flow table associated with the UFID of a specified hardware flow table.

Parameter Description

Parameter	Mandatory	Description
all	No	Queries the UFIDs of all hardware flow tables in the software-hardware flow table mapping maintained by the host.
hw_ufid	No	Queries the UFID of the software flow table associated with the UFID of a specified hardware flow table. The UFID of a hardware flow table is in the hexadecimal format.

Example

- Querying the UFIDs of all hardware flow tables in the software-hardware flow table mapping maintained by the host
  ovs-appctl hwoff/shmap-dump-hw all
- Querying the UFID of the software flow table associated with the UFID of a specified hardware flow table
  ovs-appctl hwoff/shmap-dump-hw 69e285a3-78f2-4029-9350-82d03e02564f

ovs-appctl hwoff/shmap-dump-sw

Syntax

ovs-appctl hwoff/shmap-dump-sw [all |sw_ufid type]

Function

Queries the UFIDs of all software flow tables associated with hardware flow tables in the software-hardware flow table mapping maintained by the host; queries the UFID of the hardware flow table associated with the UFID and type of a specified software flow table.

Parameter Description

Parameter	Mandatory	Description
all	No	Queries the UFIDs of all software flow tables associated with hardware flow tables in the software-hardware flow table mapping maintained by the host.
sw_ufid, type	No	Queries the UFID of the hardware flow table associated with the UFID and type of a specified software flow table. The UFID of a software flow table is in the hexadecimal format. The options of **type** are as follows: **0**: OpenFlow data plane **1**: source IP address-based transparent transmission

Example

- Querying the UFIDs of all software flow tables associated with hardware flow tables in the software-hardware flow table mapping maintained by the host
  ```
  ovs-appctl hwoff/shmap-dump-sw all
  ```
- Querying the UFID of the hardware flow table associated with the UFID and type of a specified software flow table
  ```
  ovs-appctl hwoff/shmap-dump-sw e8288d3b-5daa-4815-8b23-2be16662977a 0
  ```

ovs-appctl hwoff/shmap-error-stats

Syntax

ovs-appctl hwoff/shmap-error-stats

Function

Queries the error statistics of the software-hardware flow table mapping.

Parameter Description

N/A

Example

```
ovs-appctl hwoff/shmap-error-stats
```

ovs-appctl hwoff/shmap-flush

Syntax

ovs-appctl hwoff/shmap-flush

Function

Clears the software-hardware flow table mapping.

Parameter Description

N/A

Example

```
ovs-appctl hwoff/shmap-flush
```

3.10 Change History

Date	Description
2021-01-20	This issue is the first official release.

4 SR-IOV User Guide

4.1 Introduction

Introduction to SR-IOV

Single-Root I/O Virtualization (SR-IOV) comprises physical functions (PFs) and virtual functions (VFs). It allows a physical device to provide multiple virtual functions, reducing the hardware cost of each additional function.

SR-IOV enables a single functional unit (for example, an Ethernet port) to serve as multiple independent physical devices. That is, a physical device that supports SR-IOV can be configured as multiple functional units. It eliminates the need for CPU and virtual machine (VM) management, improving system performance. However, its dependence on hardware results in certain limitations in terms of commonality, compatibility, and scalability.

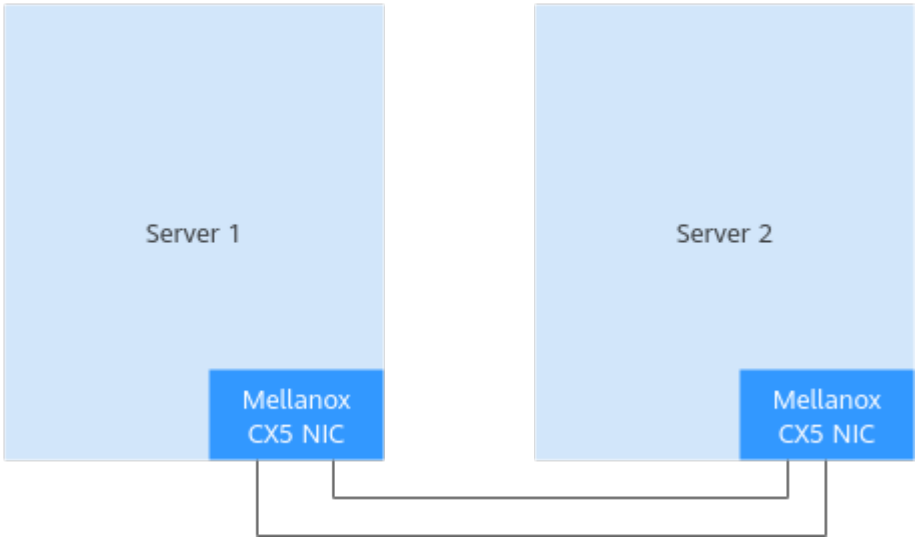
SR-IOV virtualizes NICs and enables them to be used by VMs, aiming to improve network traffic processing capabilities and switching performance.

4.2 Environment Requirements

Networking

As shown in **Figure 4-1**, SR-IOV requires two servers (server 1 and server 2). The two servers communicate with each other through direct network port connections.

Figure 4-1 Networking diagram



Server	IP Address	Mellanox CX5 Network Port
Server 1	90.90.48.60	• enp1s0f0 PCI port: 0000:01:00.0 • enp1s0f1 PCI port: 0000:01:00.1
Server 2	90.90.48.62	• enp1s0f0 PCI port: 0000:01:00.0 • enp1s0f1 PCI port: 0000:01:00.1

Hardware

Table 4-1 describes the hardware configuration.

Table 4-1 Hardware configuration

Item	Description
CPU	Kunpeng 920 processor
NIC	Mellanox CX5 public edition
Other	RAID controller cards must support pass-through.

Operating System

Table 4-2 lists the operating system (OS) requirements.

Table 4-2 OS requirements

Item	Version	Remarks
CentOS	CentOS Linux release 7.6.1810 (AltArch)	-
Kernel	4.14.0-115.el7a.0.1.aarch64	Contained in the OS image
NIC firmware	16.28.2006	Contained in the Mellanox driver package
NIC driver	OFED-5.1-2.3.7	-
QEMU	2.12.0	EV version, not contained in the OS image package
libvirt	4.5.0	Contained in the OS image
Python	2.7.5	Contained in the OS image
GCC	4.8.5	Contained in the OS image
VM OS	CentOS Linux release 7.6.1810 (AltArch)	-

Software Packages

Table 4-3 Software packages

Software Package	Description	How to Obtain
CentOS Linux release 7.6.1810 (AltArch)	CentOS 7.6 image file	**https://mirrors.huaweicloud.com/centos-vault/altarch/7.6.1810/isos/aarch64/**
MLNX_OFED_LINUX-5.1-2.3.7.1-rhel7.6alternate-aarch64.tgz	Mellanox NIC driver and software packages	**http://content.mellanox.com/ofed/MLNX_OFED-5.1-2.3.7.1/MLNX_OFED_LINUX-5.1-2.3.7.1-rhel7.6alternate-aarch64.tgz**

Software Package	Description	How to Obtain
qemu-kvm-tools-ev-2.12.0-33.1.el7.aarch64.rpm	qemu-2.12.0 centos-release-qemu-ev	http://mirrors.tools.huawei.com/centos-altarch/7/virt/aarch64/kvm-common/Packages/q/qemu-kvm-tools-ev-2.12.0-33.1.el7.aarch64.rpm
qume-img-ev-2.12.0-33.1.el7.aarch64.rpm		http://mirrors.tools.huawei.com/centos-altarch/7/virt/aarch64/kvm-common/Packages/q/qemu-img-ev-2.12.0-33.1.el7.aarch64.rpm
qume-kvm-common-ev-2.12.0-33.1.el7.aarch64.rpm		http://mirrors.tools.huawei.com/centos-altarch/7/virt/aarch64/kvm-common/Packages/q/qemu-kvm-common-ev-2.12.0-33.1.el7.aarch64.rpm
qume-kvm-ev-2.12.0-33.1.el7.aarch64.rpm		http://mirrors.tools.huawei.com/centos-altarch/7/virt/aarch64/kvm-common/Packages/q/qemu-kvm-ev-2.12.0-33.1.el7.aarch64.rpm
libvirt	-	Install through Yum.

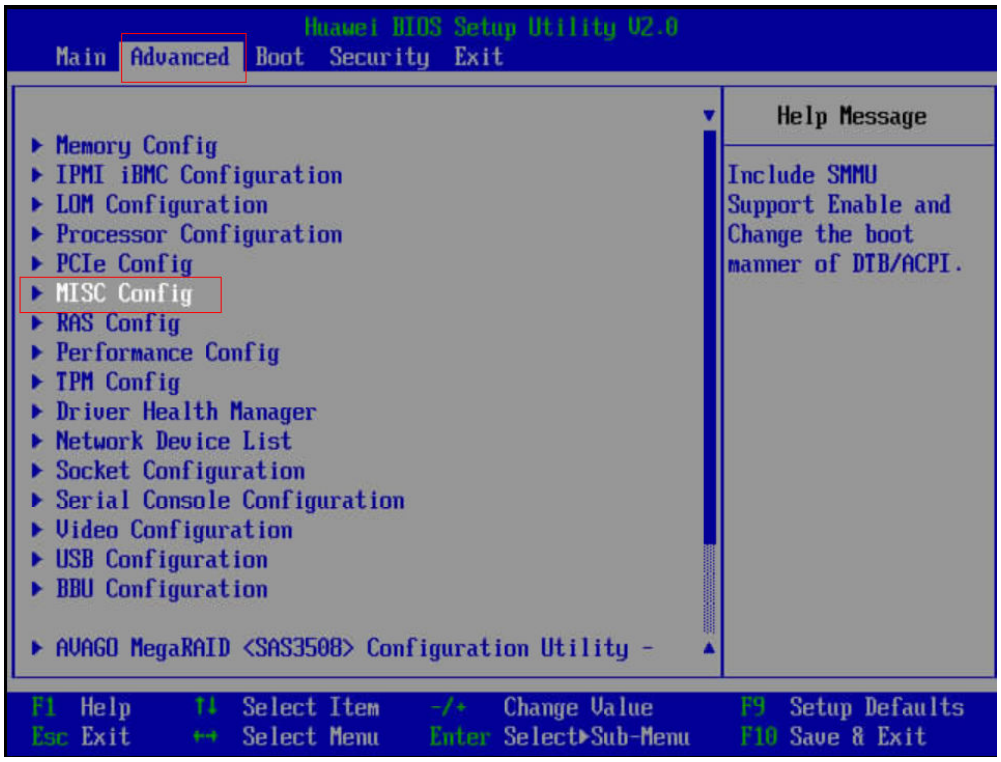
4.3 Configuring the Environment

NOTE

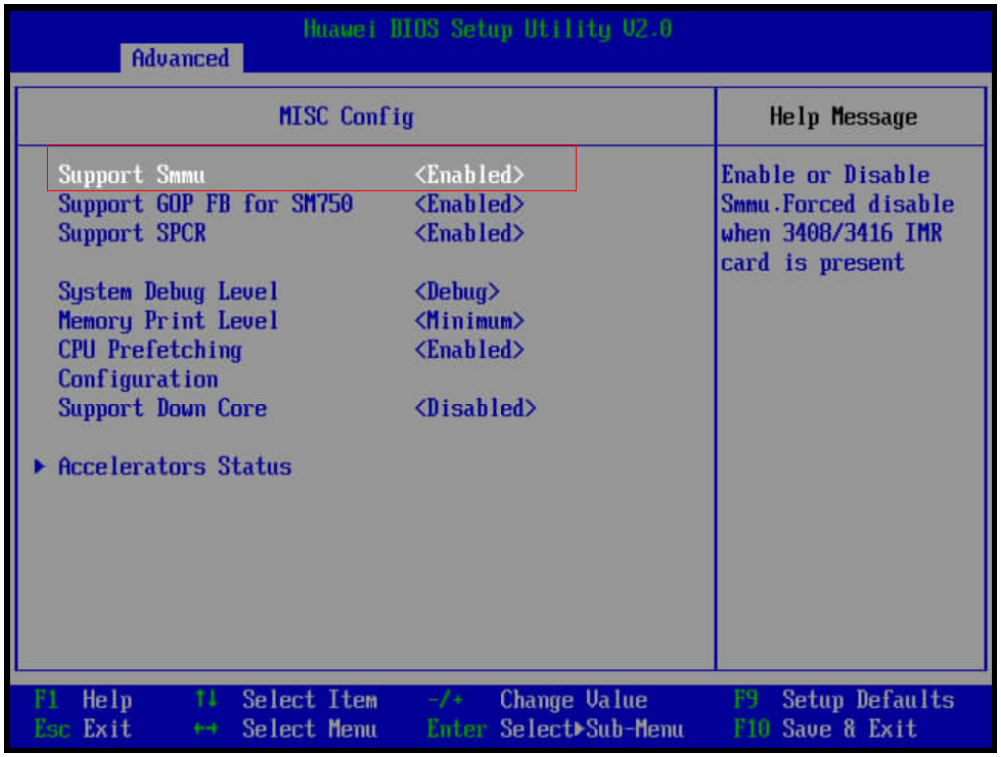
- Unless otherwise specified, all operations described in this document must be performed on both server 1 and server 2.
- In the following example, the Mellanox PF network port name is **enp1s0f0/1**, the VF port number is **enp1s0f0_$**, and the PCI port number is **0000:01:00/1.$**. Replace them with the actual values to match your configuration.

BIOS Settings

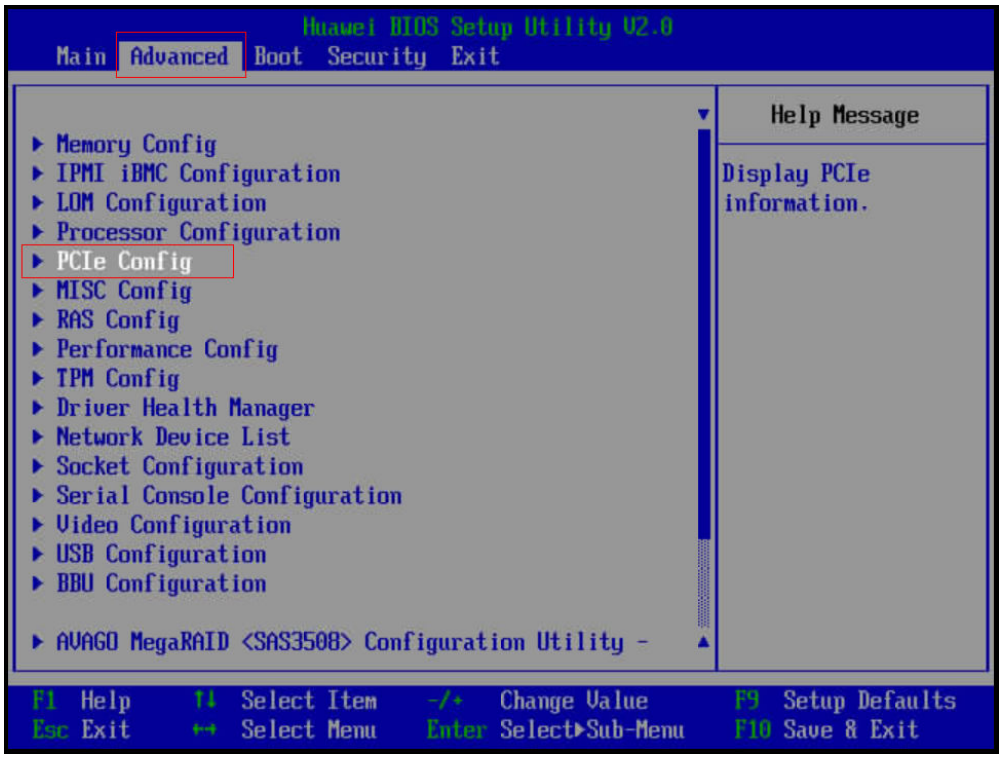
Step 1 Go to the BIOS and choose **Advanced** > **MISC Config**.



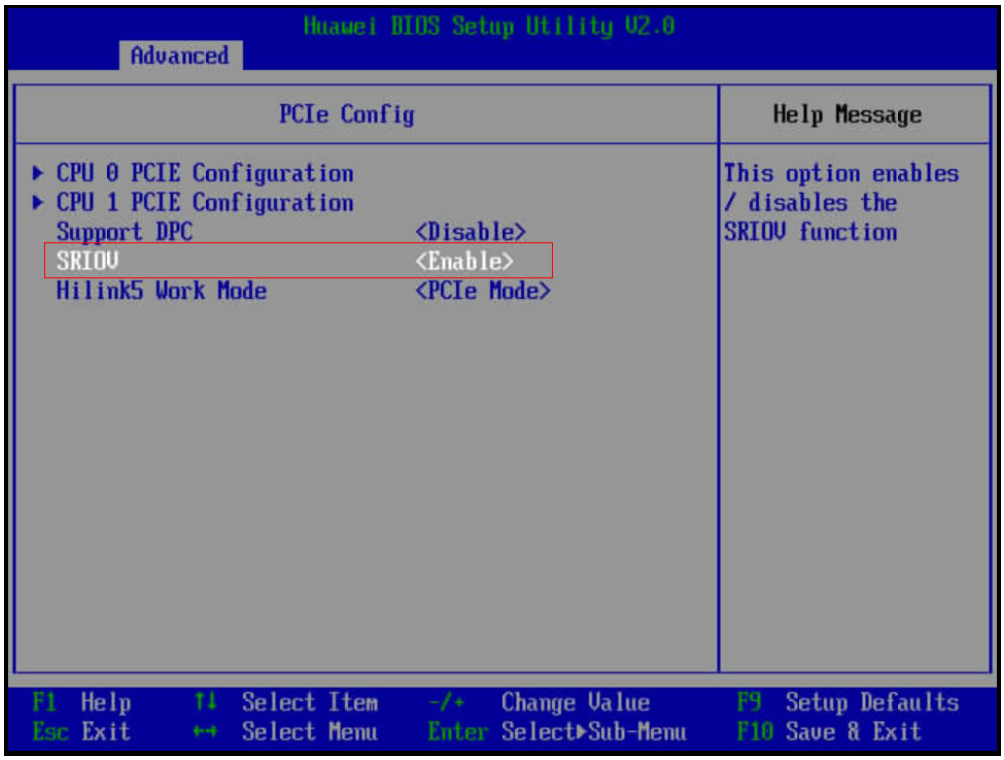
Step 2 Set **Support Smmu** to **Enabled**.



Step 3 Return to the upper-level directory and select **PCIe Config**.



Step 4 Set **SRIOV** to **Enable**.



----End

Configuring Memory Huge Pages

NOTE

You need to restart the server for this operation and subsequent operations including **Enabling IOMMU and CPU Isolation**, **Disabling NetworkManager**, and **Disabling SELinux**. However, you can restart the server once after all these operations are complete.

Step 1 Use an SSH remote login tool to log in to the server and switch to the root user account.

Step 2 Check whether memory huge pages have been configured.

```
cat /proc/meminfo |grep -i huge
```

```
[root@localhost ~]# cat /proc/meminfo |grep -i huge
AnonHugePages:         0 kB
ShmemHugePages:        0 kB
HugePages_Total:      64
HugePages_Free:       48
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:     524288 kB
[root@localhost ~]#
```

- If both **HugePages_Total** and **Hugepagesize** are greater than 0 and **Hugepagesize** is greater than 5 GB, no further action is required.
- If either **HugePages_Total** or **Hugepagesize** is 0, go to **Step 3**.

Step 3 Edit **/boot/efi/EFI/centos/grub.cfg** to modify the boot setting.

```
vim /boot/efi/EFI/centos/grub.cfg
```

1. Locate the boot item **menuentry**. Add the huge page option **default_hugepagesz=512M hugepagesz=512M hugepages=128** at the end of line 100. For example:

```
menuentry 'CentOS Linux (4.14.0-115.el7a.0.1.aarch64) 7 (AltArch)' --class centos --class gnu-linux --class gnu --clas
s os --unrestricted $menuentry_id_option 'gnulinux-4.14.0-115.el7a.0.1.aarch64-advanced-7df6249c-0ff7-4c5b-8ca0-f55775
4cb514' {
        load_video
        set gfxpayload=keep
        insmod gzio
        insmod part_gpt
        insmod xfs
        set root='hd0,gpt2'
        if [ x$feature_platform_search_hint = xy ]; then
          search --no-floppy --fs-uuid --set=root --hint-ieee1275='ieee1275//sas/disk@0,gpt2' --hint-bios=hd0,gpt2 --h
int-efi=hd0,gpt2 --hint-baremetal=ahci0,gpt2  1d2c4eaf-b91e-453f-b0de-0b343e0da3eb
        else
          search --no-floppy --fs-uuid --set=root 1d2c4eaf-b91e-453f-b0de-0b343e0da3eb
        fi
        linux /vmlinuz-4.14.0-115.el7a.0.1.aarch64 root=/dev/mapper/centos-root ro crashkernel=auto rd.lvm.lv=centos/r
oot rd.lvm.lv=centos/swap LANG=en_US.UTF-8 default_hugepagesz=512M hugepagesz=512M hugepages=128
        initrd /initramfs-4.14.0-115.el7a.0.1.aarch64.img
```

2. Save the configuration and exit.

```
:wq
```

Step 4 Configure the huge pages to be mounted upon system boot.

1. Edit the **/etc/fstab** file.

```
vim /etc/fstab
```

Add the following text:

```
nodev /mnt/huge hugetlbfs defaults 0 0
```



```
/dev/mapper/centos-root /                       xfs     defaults        0 0
UUID=efa56236-2137-43a3-b685-26ee0900e87a /boot                   xfs     defaults        0 0
UUID=D9A7-9293          /boot/efi               vfat    umask=0077,shortname=winnt 0 0
/dev/mapper/centos-home /home                   xfs     defaults        0 0
/dev/mapper/centos-swap swap                    swap    defaults        0 0
nodev /mnt/huge hugetlbfs defaults 0 0
```

2. Save the configuration and exit.
   :wq

Step 5 Create a **/mnt/huge** directory.

```
mkdir -p /mnt/huge
```

Step 6 Restart the server for the configuration to take effect.

```
reboot
```

Step 7 Perform **Step 2** again to check whether the configuration has taken effect.

----End

Enabling IOMMU and CPU Isolation

Step 1 Edit the **/etc/grub2-efi.cfg** file.

```
vim /etc/grub2-efi.cfg
```

Add the IOMMU setting next to the boot item (at the end of line 100).

```
isolcpus=0-5 iommu.passthrough=1
```

```
menuentry 'CentOS Linux (4.14.0-115.el7a.0.1.aarch64) 7 (AltArch)' --class centos --class gnu-linux --class gnu --class os --unrestricted $menuentry_id_option 'gnuli
nux-4.14.0-115.el7a.0.1.aarch64-advanced-9d2afe18-913a-45ea-8f4c-114cea9fc837' {
        load_video
        set gfxpayload=keep
        insmod gzio
        insmod part_gpt
        insmod xfs
        set root='hd0,gpt2'
        if [ x$feature_platform_search_hint = xy ]; then
          search --no-floppy --fs-uuid --set=root --hint-ieee1275='ieee1275//sas/disk@2b00,gpt2' --hint-bios=hd0,gpt2 --hint-efi=hd0,gpt2 --hint-baremetal=ahci0,gpt2
 efa56236-2137-43a3-b685-26ee0900e87a
        else
          search --no-floppy --fs-uuid --set=root efa56236-2137-43a3-b685-26ee0900e87a
        fi
        linux /vmlinuz-4.14.0-115.el7a.0.1.aarch64 root=/dev/mapper/centos-root ro crashkernel=auto rd.lvm.lv=centos/root rd.lvm.lv=centos/swap LANG=en_US.UTF-8 defa
ult_hugepagesz=512M hugepagesz=512M hugepages=128 isolcpus=0-5 iommu.passthrough=1
        initrd /initramfs-4.14.0-115.el7a.0.1.aarch64.img
}
```

Step 2 Restart the server for the configuration to take effect.

```
reboot
```

----End

Disabling NetworkManager

Run the following commands to disable NetworkManager:

```
systemctl stop NetworkManager
systemctl disable NetworkManager
```

Disabling SELinux

Step 1 Disable the firewall.

```
systemctl stop firewalld.service
systemctl disable firewalld.service
```

Step 2 Edit the **/etc/selinux/config** file.

```
vim /etc/selinux/config
```

Set **SELINUX** to **disabled**.


```
# This file controls the state of SELinux on the system.
# SELINUX= can take one of these three values:
#     enforcing - SELinux security policy is enforced.
#     permissive - SELinux prints warnings instead of enforcing.
#     disabled - No SELinux policy is loaded.
SELINUX=disabled
# SELINUXTYPE= can take one of three values:
#     targeted - Targeted processes are protected,
#     minimum - Modification of targeted policy. Only selected processes are protected.
#     mls - Multi Level Security protection.
SELINUXTYPE=targeted
```

NOTE

To disable SELinux temporarily, run the **setenforce 0** command.

Step 3 Restart the server for the configuration to take effect.

```
reboot
```

----End

Configuring the Local Mirror Source

NOTE

To match the kernel-devel, configure the local mirror source.

Step 1 Configure the local Yum source.

```
mkdir /mnt/repo
mount -o loop /home/iso/CentOS-7-aarch64-Everything-1810.iso /mnt/repo
cd /etc/yum.repos.d
mkdir backup
mv *.repo backup
```

Step 2 Edit the **local.repo** file.

```
vim local.repo
```

1. Add the following content to the file:

   ```
   [local]
   name=local
   baseurl=file:///mnt/repo
   enable=1
   gpgcheck=0
   gpgkey=file:///mnt/repo/RPM-GPG-KEY-CentOS-7
   ```

2. Save the configuration and exit.

   ```
   :wq
   ```

Step 3 Clear all the cached content.

```
yum clean all
```

Step 4 Build a cache.

```
yum makecache
```

----End

Installing the Mellanox NIC Driver

Step 1 Upload the Mellanox NIC driver package from the local PC to the server.

Step 2 Decompress the driver package and go to the decompressed folder.

```
tar -zxvf MLNX_OFED_LINUX-5.1-2.3.7.1-rhel7.6alternate-aarch64.tgz
cd MLNX_OFED_LINUX-5.1-2.3.7.1-rhel7.6alternate-aarch64
```

Step 3 Install the dependencies.

```
yum install unbound tcl gcc-gfortran fuse-libs tk createrepo kernel-devel python-devel redhat-rpm-config rpm-build gcc gcc-c++
```

Step 4 Install the driver.

```
./mlnxofedinstall --ovs-dpdk --upstream-libs --add-kernel-support
```

Step 5 Update initramfs.

```
dracut -f
```

Step 6 Load the driver.

```
/etc/init.d/openibd restart
```

NOTICE

If a **FAILED** error message is displayed, run the **rmmod hns_roce_hw_v2** command to retry.

```
[root@localhost MLNX_OFED_LINUX-5.1-2.3.7.1-rhel7.6alternate-aarch64]# /etc/init.d/openibd restart
Unloading hns_roce                                         [FAILED]
rmmod: ERROR: Module hns_roce is in use by: hns_roce_hw_v2
[root@localhost MLNX_OFED_LINUX-5.1-2.3.7.1-rhel7.6alternate-aarch64]#
```

NOTE

If a 1822 NIC exists in the environment, the server performance deteriorates due to a large number of software interrupts generated by the 1822 NIC. Therefore, disable all existing 1822 NICs for a higher server performance.

Run the following command to disable 1822 NICs:

```
rmmod hinic
```

Perform this operation each time after the server is restarted.

----End

4.4 Configuring SR-IOV

4.4.1 Checking Mellanox NIC Information

Step 1 Check for Mellanox NICs.

```
lspci -nn | grep Mellanox
```

```
[root@localhost ~]# lspci -nn | grep Mellanox
01:00.0 Ethernet controller [0200]: Mellanox Technologies MT27800 Family [ConnectX-5] [15b3:1017]
01:00.1 Ethernet controller [0200]: Mellanox Technologies MT27800 Family [ConnectX-5] [15b3:1017]
[root@localhost ~]#
```

Step 2 Query the port information about the Mellanox NIC.

```
ls -l /sys/class/net/ | grep 01:00
```

```
[root@localhost ~]# ls -l /sys/class/net/ | grep 01:00
lrwxrwxrwx 1 root root 0 Oct 13 09:58 enp1s0f0 -> ../../devices/pci0000:00/0000:00:00.0/0000:01:00.0/net/enp1s0f0
lrwxrwxrwx 1 root root 0 Oct 13 09:58 enp1s0f1 -> ../../devices/pci0000:00/0000:00:00.0/0000:01:00.1/net/enp1s0f1
```


Step 3 Check the hardware version. The installed firmware must match the version requirements specified in the Release Notes.

```
ethtool -i enp1s0f1 | head -5
```

```
[root@localhost ~]# ethtool -i enp1s0f0 | head -5
driver: mlx5_core
version: 5.1-2.3.7
firmware-version: 16.28.2006 (MT_0000000080)
expansion-rom-version:
bus-info: 0000:01:00.0
```

Step 4 Check the maximum number of VFs supported by the Mellanox network ports.

```
cat /sys/class/net/enp1s0f1/device/sriov_totalvfs
```

```
[root@localhost ~]# cat /sys/class/net/enp1s0f1/device/sriov_totalvfs
8
[root@localhost ~]#
```

----End

4.4.2 Configuring Kernel-Mode SR-IOV

Step 1 Add VFs to the PF network port.

1. Run the following command:
   ```
   echo 8 > /sys/class/net/enp1s0f1/device/sriov_numvfs
   ```
2. Check whether the adding operation is successful.
   ```
   cat /sys/class/net/enp1s0f1/device/sriov_numvfs
   ```
   ```
   [root@localhost ~]# echo 8 > /sys/class/net/enp1s0f1/device/sriov_numvfs
   [root@localhost ~]# cat /sys/class/net/enp1s0f1/device/sriov_numvfs
   8
   [root@localhost ~]#
   ```

Step 2 Configure the MAC address of the VF port.

1. Run the following command:
   ```
   ip link set enp1s0f1 vf 0 mac e4:11:22:33:44:50
   ip link set enp1s0f1 vf 1 mac e4:11:22:33:44:51
   ip link set enp1s0f1 vf 2 mac e4:11:22:33:44:52
   ip link set enp1s0f1 vf 3 mac e4:11:22:33:44:53
   ip link set enp1s0f1 vf 4 mac e4:11:22:33:44:54
   ip link set enp1s0f1 vf 5 mac e4:11:22:33:44:55
   ip link set enp1s0f1 vf 6 mac e4:11:22:33:44:56
   ip link set enp1s0f1 vf 7 mac e4:11:22:33:44:57
   ```
2. Verify the configuration.
   ```
   ip link show dev enp1s0f1
   ```
   ```
   [root@localhost ~]# ip link set enp1s0f1 vf 0 mac e4:11:22:33:44:50
   [root@localhost ~]# ip link set enp1s0f1 vf 1 mac e4:11:22:33:44:51
   [root@localhost ~]# ip link set enp1s0f1 vf 2 mac e4:11:22:33:44:52
   [root@localhost ~]# ip link set enp1s0f1 vf 3 mac e4:11:22:33:44:53
   [root@localhost ~]# ip link set enp1s0f1 vf 4 mac e4:11:22:33:44:54
   [root@localhost ~]# ip link set enp1s0f1 vf 5 mac e4:11:22:33:44:55
   [root@localhost ~]# ip link set enp1s0f1 vf 6 mac e4:11:22:33:44:56
   [root@localhost ~]# ip link set enp1s0f1 vf 7 mac e4:11:22:33:44:57
   [root@localhost ~]# ip link show dev enp1s0f1
   17: enp1s0f1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode DEFAULT group default qlen 1000
       link/ether 1c:43:63:ab:b3:ee brd ff:ff:ff:ff:ff:ff
       vf 0 MAC e4:11:22:33:44:50, spoof checking off, link-state auto, trust off, query_rss off
       vf 1 MAC e4:11:22:33:44:51, spoof checking off, link-state auto, trust off, query_rss off
       vf 2 MAC e4:11:22:33:44:52, spoof checking off, link-state auto, trust off, query_rss off
       vf 3 MAC e4:11:22:33:44:53, spoof checking off, link-state auto, trust off, query_rss off
       vf 4 MAC e4:11:22:33:44:54, spoof checking off, link-state auto, trust off, query_rss off
       vf 5 MAC e4:11:22:33:44:55, spoof checking off, link-state auto, trust off, query_rss off
       vf 6 MAC e4:11:22:33:44:56, spoof checking off, link-state auto, trust off, query_rss off
       vf 7 MAC e4:11:22:33:44:57, spoof checking off, link-state auto, trust off, query_rss off
   [root@localhost ~]#
   ```

NOTICE

Each MAC address must be unique on the local server, the peer server, and the switch.

3. Check the PCI port numbers of the eight virtual ports.
   ls -l /sys/class/net/

```
[root@localhost ~]# ls -l /sys/class/net/
total 0
lrwxrwxrwx 1 root root 0 Oct 13 10:14 enp125s0f0 -> ../../devices/pci0000:7c/0000:7c:00.0/0000:7d:00.0/net/enp125s0f0
lrwxrwxrwx 1 root root 0 Oct 13 10:14 enp125s0f1 -> ../../devices/pci0000:7c/0000:7c:00.0/0000:7d:00.1/net/enp125s0f1
lrwxrwxrwx 1 root root 0 Oct 13 10:14 enp125s0f2 -> ../../devices/pci0000:7c/0000:7c:00.0/0000:7d:00.2/net/enp125s0f2
lrwxrwxrwx 1 root root 0 Oct 13 10:14 enp125s0f3 -> ../../devices/pci0000:7c/0000:7c:00.0/0000:7d:00.3/net/enp125s0f3
lrwxrwxrwx 1 root root 0 Oct 13 10:14 enp131s0 -> ../../devices/pci0000:80/0000:80:00.0/0000:81:00.0/0000:82:00.0/0000:83:00.0/net/enp131s0
lrwxrwxrwx 1 root root 0 Oct 13 10:14 enp132s0 -> ../../devices/pci0000:80/0000:80:00.0/0000:81:00.0/0000:82:01.0/0000:84:00.0/net/enp132s0
lrwxrwxrwx 1 root root 0 Oct 13 10:14 enp133s0 -> ../../devices/pci0000:80/0000:80:00.0/0000:81:00.0/0000:82:02.0/0000:85:00.0/net/enp133s0
lrwxrwxrwx 1 root root 0 Oct 13 10:14 enp134s0 -> ../../devices/pci0000:80/0000:80:00.0/0000:81:00.0/0000:82:03.0/0000:86:00.0/net/enp134s0
lrwxrwxrwx 1 root root 0 Oct 13 10:14 enp1s0f0 -> ../../devices/pci0000:00/0000:00:00.0/0000:01:00.0/net/enp1s0f0
lrwxrwxrwx 1 root root 0 Oct 13 10:14 enp1s0f1 -> ../../devices/pci0000:00/0000:00:00.0/0000:01:00.1/net/enp1s0f1
lrwxrwxrwx 1 root root 0 Oct 13 10:33 enp1s1f2 -> ../../devices/pci0000:00/0000:00:00.0/0000:01:01.2/net/enp1s1f2
lrwxrwxrwx 1 root root 0 Oct 13 10:33 enp1s1f3 -> ../../devices/pci0000:00/0000:00:00.0/0000:01:01.3/net/enp1s1f3
lrwxrwxrwx 1 root root 0 Oct 13 10:33 enp1s1f4 -> ../../devices/pci0000:00/0000:00:00.0/0000:01:01.4/net/enp1s1f4
lrwxrwxrwx 1 root root 0 Oct 13 10:33 enp1s1f5 -> ../../devices/pci0000:00/0000:00:00.0/0000:01:01.5/net/enp1s1f5
lrwxrwxrwx 1 root root 0 Oct 13 10:33 enp1s1f6 -> ../../devices/pci0000:00/0000:00:00.0/0000:01:01.6/net/enp1s1f6
lrwxrwxrwx 1 root root 0 Oct 13 10:33 enp1s1f7 -> ../../devices/pci0000:00/0000:00:00.0/0000:01:01.7/net/enp1s1f7
lrwxrwxrwx 1 root root 0 Oct 13 10:33 enp1s2 -> ../../devices/pci0000:00/0000:00:00.0/0000:01:02.0/net/enp1s2
lrwxrwxrwx 1 root root 0 Oct 13 10:33 enp1s2f1 -> ../../devices/pci0000:00/0000:00:00.0/0000:01:02.1/net/enp1s2f1
lrwxrwxrwx 1 root root 0 Oct 13 10:14 enp5s0 -> ../../devices/pci0000:00/0000:00:0c.0/0000:03:00.0/0000:04:00.0/0000:05:00.0/net/enp5s0
lrwxrwxrwx 1 root root 0 Oct 13 10:14 enp6s0 -> ../../devices/pci0000:00/0000:00:0c.0/0000:03:00.0/0000:04:01.0/0000:06:00.0/net/enp6s0
lrwxrwxrwx 1 root root 0 Oct 13 10:14 enp7s0 -> ../../devices/pci0000:00/0000:00:0c.0/0000:03:00.0/0000:04:02.0/0000:07:00.0/net/enp7s0
lrwxrwxrwx 1 root root 0 Oct 13 10:14 enp8s0 -> ../../devices/pci0000:00/0000:00:0c.0/0000:03:00.0/0000:04:03.0/0000:08:00.0/net/enp8s0
lrwxrwxrwx 1 root root 0 Oct 13 10:14 lo -> ../../devices/virtual/net/lo
lrwxrwxrwx 1 root root 0 Oct 13 10:14 virbr0 -> ../../devices/virtual/net/virbr0
lrwxrwxrwx 1 root root 0 Oct 13 10:14 virbr0-nic -> ../../devices/virtual/net/virbr0-nic
```

Step 3 Change the network port mode.

1. Unbind the VFs.

```
echo 0000:01:01.2 > /sys/bus/pci/drivers/mlx5_core/unbind
echo 0000:01:01.3 > /sys/bus/pci/drivers/mlx5_core/unbind
echo 0000:01:01.4 > /sys/bus/pci/drivers/mlx5_core/unbind
echo 0000:01:01.5 > /sys/bus/pci/drivers/mlx5_core/unbind
echo 0000:01:01.6 > /sys/bus/pci/drivers/mlx5_core/unbind
echo 0000:01:01.7 > /sys/bus/pci/drivers/mlx5_core/unbind
echo 0000:01:02.0 > /sys/bus/pci/drivers/mlx5_core/unbind
echo 0000:01:02.1 > /sys/bus/pci/drivers/mlx5_core/unbind
```

```
[root@localhost ~]# echo 0000:01:01.2 > /sys/bus/pci/drivers/mlx5_core/unbind
[root@localhost ~]# echo 0000:01:01.3 > /sys/bus/pci/drivers/mlx5_core/unbind
[root@localhost ~]# echo 0000:01:01.4 > /sys/bus/pci/drivers/mlx5_core/unbind
[root@localhost ~]# echo 0000:01:01.5 > /sys/bus/pci/drivers/mlx5_core/unbind
[root@localhost ~]# echo 0000:01:01.6 > /sys/bus/pci/drivers/mlx5_core/unbind
[root@localhost ~]# echo 0000:01:01.7 > /sys/bus/pci/drivers/mlx5_core/unbind
[root@localhost ~]# echo 0000:01:02.0 > /sys/bus/pci/drivers/mlx5_core/unbind
[root@localhost ~]# echo 0000:01:02.1 > /sys/bus/pci/drivers/mlx5_core/unbind
[root@localhost ~]#
```

2. On the PF, change the **eSwitch** mode from **Legacy** to **SwitchDev**.

```
devlink dev eswitch set pci/0000:01:00.1 mode switchdev
echo switchdev > /sys/class/net/enp1s0f1/compat/devlink/mode
cat /sys/class/net/enp1s0f1/compat/devlink/mode
```

```
[root@localhost ~]# devlink dev eswitch set pci/0000:01:00.1 mode switchdev
[root@localhost ~]# echo switchdev > /sys/class/net/enp1s0f1/compat/devlink/mode
[root@localhost ~]# cat /sys/class/net/enp1s0f1/compat/devlink/mode
switchdev
```

3. Check whether the device name of the Representor has been changed.
   ls -l /sys/class/net/


```
[root@localhost ~]# ls -l /sys/class/net/
total 0
lrwxrwxrwx 1 root root 0 Oct 13 10:14 enp125s0f0 -> ../../devices/pci0000:7c/0000:7c:00.0/0000:7d:00.0/net/enp125s0f0
lrwxrwxrwx 1 root root 0 Oct 13 10:14 enp125s0f1 -> ../../devices/pci0000:7c/0000:7c:00.0/0000:7d:00.1/net/enp125s0f1
lrwxrwxrwx 1 root root 0 Oct 13 10:14 enp125s0f2 -> ../../devices/pci0000:7c/0000:7c:00.0/0000:7d:00.2/net/enp125s0f2
lrwxrwxrwx 1 root root 0 Oct 13 10:14 enp125s0f3 -> ../../devices/pci0000:7c/0000:7c:00.0/0000:7d:00.3/net/enp125s0f3
lrwxrwxrwx 1 root root 0 Oct 13 10:14 enp131s0 -> ../../devices/pci0000:80/0000:80:00.0/0000:81:00.0/0000:82:00.0/0000:83:00.0/net/enp131s0
lrwxrwxrwx 1 root root 0 Oct 13 10:14 enp132s0 -> ../../devices/pci0000:80/0000:80:00.0/0000:81:00.0/0000:82:01.0/0000:84:00.0/net/enp132s0
lrwxrwxrwx 1 root root 0 Oct 13 10:14 enp133s0 -> ../../devices/pci0000:80/0000:80:00.0/0000:81:00.0/0000:82:02.0/0000:85:00.0/net/enp133s0
lrwxrwxrwx 1 root root 0 Oct 13 10:14 enp134s0 -> ../../devices/pci0000:80/0000:80:00.0/0000:81:00.0/0000:82:03.0/0000:86:00.0/net/enp134s0
lrwxrwxrwx 1 root root 0 Oct 13 10:14 enp1s0f0 -> ../../devices/pci0000:00/0000:00:00.0/0000:01:00.0/net/enp1s0f0
lrwxrwxrwx 1 root root 0 Oct 13 10:14 enp1s0f1 -> ../../devices/pci0000:00/0000:00:00.0/0000:01:00.1/net/enp1s0f1
lrwxrwxrwx 1 root root 0 Oct 13 10:47 enp1s0f1_0 -> ../../devices/virtual/net/enp1s0f1_0
lrwxrwxrwx 1 root root 0 Oct 13 10:47 enp1s0f1_1 -> ../../devices/virtual/net/enp1s0f1_1
lrwxrwxrwx 1 root root 0 Oct 13 10:47 enp1s0f1_2 -> ../../devices/virtual/net/enp1s0f1_2
lrwxrwxrwx 1 root root 0 Oct 13 10:47 enp1s0f1_3 -> ../../devices/virtual/net/enp1s0f1_3
lrwxrwxrwx 1 root root 0 Oct 13 10:47 enp1s0f1_4 -> ../../devices/virtual/net/enp1s0f1_4
lrwxrwxrwx 1 root root 0 Oct 13 10:47 enp1s0f1_5 -> ../../devices/virtual/net/enp1s0f1_5
lrwxrwxrwx 1 root root 0 Oct 13 10:47 enp1s0f1_6 -> ../../devices/virtual/net/enp1s0f1_6
lrwxrwxrwx 1 root root 0 Oct 13 10:47 enp1s0f1_7 -> ../../devices/virtual/net/enp1s0f1_7
lrwxrwxrwx 1 root root 0 Oct 13 10:14 enp5s0 -> ../../devices/pci0000:00/0000:00:0c.0/0000:03:00.0/0000:04:00.0/0000:05:00.0/net/enp5s0
lrwxrwxrwx 1 root root 0 Oct 13 10:14 enp6s0 -> ../../devices/pci0000:00/0000:00:0c.0/0000:03:00.0/0000:04:01.0/0000:06:00.0/net/enp6s0
lrwxrwxrwx 1 root root 0 Oct 13 10:14 enp7s0 -> ../../devices/pci0000:00/0000:00:0c.0/0000:03:00.0/0000:04:02.0/0000:07:00.0/net/enp7s0
lrwxrwxrwx 1 root root 0 Oct 13 10:14 enp8s0 -> ../../devices/pci0000:00/0000:00:0c.0/0000:03:00.0/0000:04:03.0/0000:08:00.0/net/enp8s0
lrwxrwxrwx 1 root root 0 Oct 13 10:14 lo -> ../../devices/virtual/net/lo
lrwxrwxrwx 1 root root 0 Oct 13 10:14 virbr0 -> ../../devices/virtual/net/virbr0
lrwxrwxrwx 1 root root 0 Oct 13 10:14 virbr0-nic -> ../../devices/virtual/net/virbr0-nic
[root@localhost ~]#
```

The device name of the VFs has been changed from **enp1s0f$** to **enp1s0f1_$**.

Step 4 Bind the VFs.

```
echo 0000:01:01.2 > /sys/bus/pci/drivers/mlx5_core/bind
echo 0000:01:01.3 > /sys/bus/pci/drivers/mlx5_core/bind
echo 0000:01:01.4 > /sys/bus/pci/drivers/mlx5_core/bind
echo 0000:01:01.5 > /sys/bus/pci/drivers/mlx5_core/bind
echo 0000:01:01.6 > /sys/bus/pci/drivers/mlx5_core/bind
echo 0000:01:01.7 > /sys/bus/pci/drivers/mlx5_core/bind
echo 0000:01:02.0 > /sys/bus/pci/drivers/mlx5_core/bind
echo 0000:01:02.1 > /sys/bus/pci/drivers/mlx5_core/bind
```

----End

4.4.3 Configuring OVS Boot Parameters

Step 1 Start OVS.

```
systemctl start openvswitch
```

Step 2 Enable offloading.

```
ovs-vsctl set Open_vSwitch . other_config:hw-offload=true
ovs-vsctl set Open_vSwitch . other_config:tc-policy=verbose
```

Step 3 Restart the OVS.

```
systemctl restart openvswitch
```

```
[root@localhost ~]# systemctl start openvswitch
[root@localhost ~]# ovs-vsctl set Open_vSwitch . other_config:hw-offload=true
[root@localhost ~]# ovs-vsctl set Open_vSwitch . other_config:tc-policy=verbose
[root@localhost ~]# systemctl restart openvswitch
```

Step 4 View the OVS information.

```
ovs-vsctl list open_vswitch
```

```
[root@localhost ~]# ovs-vsctl list open_vswitch
_uuid               : 8280022b-f508-48dc-aae6-d57eed15402b
bridges             : []
cur_cfg             : 377
datapath_types      : [netdev, system]
datapaths           : {}
db_version          : "8.2.0"
dpdk_initialized    : false
dpdk_version        : "MLNX_DPDK 19.11.0.0.23"
external_ids        : {hostname=localhost, rundir="/var/run/openvswitch", system-id="2092072c-a582-484f-8f0f-ed1551b7ce8b"}
iface_types         : [erspan, geneve, gre, internal, ip6erspan, ip6gre, lisp, patch, stt, system, tap, vxlan]
manager_options     : []
next_cfg            : 377
other_config        : {hw-offload="true", tc-policy=verbose}
ovs_version         : "2.13.1"
ssl                 : []
statistics          : {}
system_type         : centos
system_version      : "7"
```

----End

4.4.4 Configuring Network Data

Step 1 Set up the normal network.

```
ovs-vsctl add-br ovs-sriov
ovs-vsctl add-port ovs-sriov enp1s0f1
ovs-vsctl add-port ovs-sriov enp1s0f1_0
ovs-vsctl add-port ovs-sriov enp1s0f1_1
ovs-vsctl add-port ovs-sriov enp1s0f1_2
ovs-vsctl add-port ovs-sriov enp1s0f1_3
ovs-vsctl add-port ovs-sriov enp1s0f1_4
ovs-vsctl add-port ovs-sriov enp1s0f1_5
ovs-vsctl add-port ovs-sriov enp1s0f1_6
ovs-vsctl add-port ovs-sriov enp1s0f1_7
ip link set dev enp1s0f1 up
ip link set dev enp1s0f1_0 up
ip link set dev enp1s0f1_1 up
ip link set dev enp1s0f1_2 up
ip link set dev enp1s0f1_3 up
ip link set dev enp1s0f1_4 up
ip link set dev enp1s0f1_5 up
ip link set dev enp1s0f1_6 up
ip link set dev enp1s0f1_7 up
```

```
[root@localhost openvswitch]# ovs-vsctl add-br ovs-sriov
[root@localhost openvswitch]# ovs-vsctl add-port ovs-sriov enp1s0f1
[root@localhost openvswitch]# ovs-vsctl add-port ovs-sriov enp1s0f1_0
[root@localhost openvswitch]# ovs-vsctl add-port ovs-sriov enp1s0f1_1
[root@localhost openvswitch]# ovs-vsctl add-port ovs-sriov enp1s0f1_2
[root@localhost openvswitch]# ovs-vsctl add-port ovs-sriov enp1s0f1_3
[root@localhost openvswitch]# ovs-vsctl add-port ovs-sriov enp1s0f1_4
[root@localhost openvswitch]# ovs-vsctl add-port ovs-sriov enp1s0f1_5
[root@localhost openvswitch]# ovs-vsctl add-port ovs-sriov enp1s0f1_6
[root@localhost openvswitch]# ovs-vsctl add-port ovs-sriov enp1s0f1_7
[root@localhost openvswitch]# ip link set dev enp1s0f1 up
[root@localhost openvswitch]# ip link set dev enp1s0f1_0 up
[root@localhost openvswitch]# ip link set dev enp1s0f1_1 up
[root@localhost openvswitch]# ip link set dev enp1s0f1_2 up
[root@localhost openvswitch]# ip link set dev enp1s0f1_3 up
[root@localhost openvswitch]# ip link set dev enp1s0f1_4 up
[root@localhost openvswitch]# ip link set dev enp1s0f1_5 up
[root@localhost openvswitch]# ip link set dev enp1s0f1_6 up
[root@localhost openvswitch]# ip link set dev enp1s0f1_7 up
[root@localhost openvswitch]#
```

Step 2 View the OVS bridge information.

```
ovs-vsctl show
```



```
[root@localhost openvswitch]# ovs-vsctl show
2238fe77-7808-4a0d-bbb4-4295d30b780e
    Bridge ovs-sriov
        Port enp1s0f1_3
            Interface enp1s0f1_3
        Port enp1s0f1_6
            Interface enp1s0f1_6
        Port ovs-sriov
            Interface ovs-sriov
                type: internal
        Port enp1s0f1
            Interface enp1s0f1
        Port enp1s0f1_4
            Interface enp1s0f1_4
        Port enp1s0f1_5
            Interface enp1s0f1_5
        Port enp1s0f1_1
            Interface enp1s0f1_1
        Port enp1s0f1_2
            Interface enp1s0f1_2
        Port enp1s0f1_7
            Interface enp1s0f1_7
        Port enp1s0f1_0
            Interface enp1s0f1_0
    ovs_version: "2.13.1"
[root@localhost openvswitch]#
```

----End

4.4.5 Creating a VM

Installing the VM Software Packages

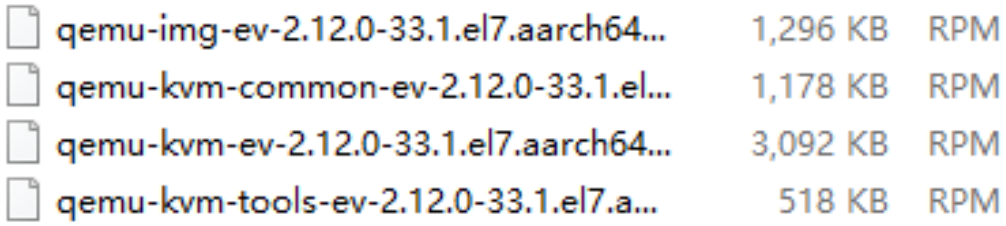
Step 1 Install VM dependencies.

```
yum install centos-release-qemu-ev
yum install -y libvirt AAVMF virt-install qemu-guest-agent
```

Step 2 Upload the qemu-2.12.0 software package obtained in **Software Packages** to the server and install it.

```
yum localinstall -y *.rpm
```

Software packages:



Step 3 Modify the QEMU configuration file.

1. Open the **qemu.conf** file.
   ```
   vim /etc/libvirt/qemu.conf
   ```
2. Set both **user** and **group** to **root**.
   ```
   user = "root"

   # The group for QEMU processes run by the system instance. It can be
   # specified in a similar way to user.
   ```



```
group = "root"

# Whether libvirt should dynamically change file ownership
```

----End

Creating a VM

Step 1 Start the libvertd service and set it to automatically start upon system boot.

```
systemctl start libvirtd
systemctl enable libvirtd
```

Step 2 Create a storage pool.

1. Create a storage pool directory and configure the directory permissions.
   ```
   mkdir -p /home/kvm/images
   chown root:root /home/kvm/images
   chmod 755 /home/kvm/images
   ```
2. Define a storage pool and bind it to the storage pool directory. Create a folder-based storage pool, activate it, and set it to start upon system boot.
   ```
   virsh pool-define-as StoragePool --type dir --target /home/kvm/images
   virsh pool-build StoragePool
   virsh pool-start StoragePool
   virsh pool-autostart StoragePool
   ```
3. View the storage pool information.
   ```
   virsh pool-info StoragePool
   virsh pool-list
   ```

Step 3 Create a drive space for the VM.

1. Create a volume.

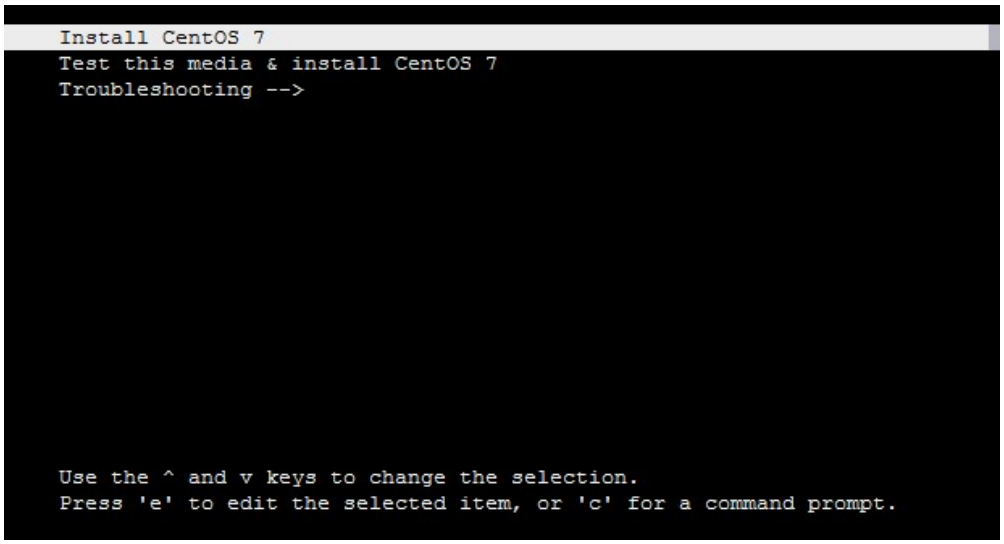
   For example, the volume is named **1.img**, the storage pool is **StoragePool**, the volume capacity is **50 GB**, the initially allocated capacity is **1 GB**, the file format is **qcow2**, and the drive file format is **qcow2**.

   ```
   virsh vol-create-as --pool StoragePool --name 1.img --capacity 50G --allocation 1G --format qcow2
   ```
2. View the volume information.
   ```
   virsh vol-info /home/kvm/images/1.img
   ```

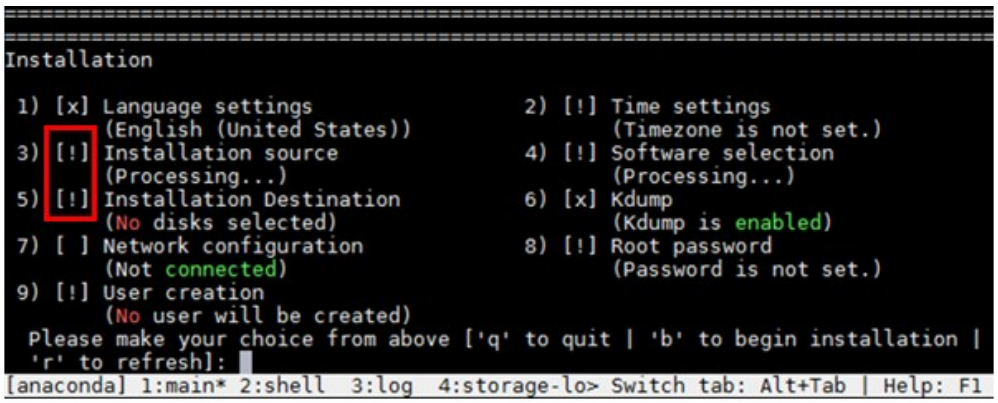
   ```
   [root@webserver ~]# virsh vol-info /home/kvm/images/1.img
   Name:           1.img
   Type:           file
   Capacity:       50.00 GiB
   Allocation:     13.15 GiB
   ```

Step 4 Create a VM.

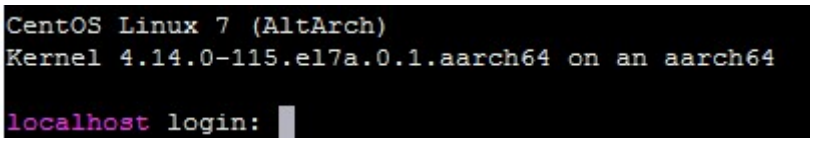
1. Create a VM **vm1**. Allocate four CPUs and 8 GB memory to it, and use **1.img** as the drive space. Copy the .iso file to **/home/iso/** and install CentOS 7.6.
   ```
   virt-install --name=vm1 --vcpus=4 --ram=8192 \
   --disk path=/home/kvm/images/1.img,format=qcow2,size=50,bus=virtio \
   --cdrom /home/iso/CentOS-7-aarch64-Everything-1810.iso
   ```
2. Install the VM OS.



3. Configure all items that contain an exclamation mark (!). Enter the serial number corresponding to the option, configure the parameter as prompted, and press **b** to start the installation.



4. After the installation is complete, the login prompt is displayed.



----End

Configuring the VM

Step 1 Modify the configuration of **vm1**.

```
virsh edit vm1
```

Delete **<interface type='network'>xxx</interface>**. Add the following content before **</devices>**:

```
<hostdev mode='subsystem' type='pci' managed='yes'>
<source>
<address domain='0x0000' bus='0x01' slot='0x01' function='0x2'/>
</source>
</hostdev>
```


NOTE

The values of **domain**, **bus**, **slot**, and **function** in **<address domain='0x0000' bus='0x01' slot='0x01' function='0x2'/>** correspond to the PCI port number **0000:01:01.2** of the VF.

The following figure shows the VM settings after modification:

```
    <serial type='pty'>
      <target type='system-serial' port='0'>
        <model name='pl011'/>
      </target>
    </serial>
    <console type='pty'>
      <target type='serial' port='0'/>
    </console>
    <hostdev mode='subsystem' type='pci' managed='yes'>
      <source>
        <address domain='0x0000' bus='0x01' slot='0x01' function='0x2'/>
      </source>
      <address type='pci' domain='0x0000' bus='0x01' slot='0x00' function='0x0'/>
    </hostdev>
  </devices>
</domain>
```

Step 2 Modify the same configuration for other VMs.

NOTE

When configuring the PCI port numbers for the other VMs, bind the PCI port numbers to VF ports that are not in use.

----End

Cloning the VM

NOTE

Subsequent verification operations require up to eight VMs for a single server. Clone a sufficient number of VMs.

Step 1 Stop **vm1**. Run the **virt-clone** command in the **virt-install** software package to clone **vm10**.

```
virt-clone -o vm1 -n vm10 -f /home/kvm/images/10.img
```

In the command, **-o** indicates the source VM, **-n** indicates the new VM, and **-f** indicates that the newly created VM uses the file on the host machine as the image file.

NOTE

After the command is executed, **vm10** is created. The CPU, memory, drive, and network resources allocated to **vm10** are the same as those allocated to **vm1**. The CPU and network resources allocated to **vm10** need to be configured separately.

Step 2 Check the status of the created VM.

```
virsh list --all
```

```
[root@webserver ~]# virsh list --all
 Id    Name                           State
----------------------------------------------------
 -     vm1                            shut off
 -     vm10                           shut off
```


Step 3 Change the host name of the VM.

The cloned VM has the same host name and IP address as **vm1**. Log in to **vm10** and run the following command to change the host name:

```
hostnamectl --static set-hostname vm10
```

Step 4 Modify the VM IP address.

```
vim /etc/sysconfig/network-scripts/ifcfg-eth0
mv etc/sysconfig/network-scripts/ifcfg-eth0 etc/sysconfig/network-scripts/ifcfg-enp1s0
```

NOTE

When pass-through is enabled, the default VF driver is installed in the OS. By default the configuration defines the virtual NIC name. The NIC name varies according to the OS. This document uses **enp1s0** as an example. Replace it with the actual name in your OS.

----End

4.4.6 Verifying Communication Between VMs

Step 1 Start the VMs.

```
virsh start vm1
virsh start vm2
```

Step 2 Log in to the VM.

Run the following command on server 1:

```
virsh console vm1
```

Step 3 Verify the pass-through.

Run the following command on VM 1 of server 1:

```
ping <Host1vm2_ip>
ping <Host2vm1_ip>
```

```
[root@vm1 ~]# ping 192.168.15.21
PING 192.168.15.21 (192.168.15.21) 56(84) bytes of data.
64 bytes from 192.168.15.21: icmp_seq=1 ttl=64 time=2.73 ms
64 bytes from 192.168.15.21: icmp_seq=2 ttl=64 time=0.165 ms
64 bytes from 192.168.15.21: icmp_seq=3 ttl=64 time=0.178 ms
64 bytes from 192.168.15.21: icmp_seq=4 ttl=64 time=0.140 ms
64 bytes from 192.168.15.21: icmp_seq=5 ttl=64 time=0.168 ms

--- 192.168.15.21 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4005ms
rtt min/avg/max/mdev = 0.140/0.676/2.731/1.027 ms
[root@vm1 ~]# ping 192.168.15.60
PING 192.168.15.60 (192.168.15.60) 56(84) bytes of data.
64 bytes from 192.168.15.60: icmp_seq=1 ttl=64 time=8.33 ms
64 bytes from 192.168.15.60: icmp_seq=2 ttl=64 time=0.167 ms
64 bytes from 192.168.15.60: icmp_seq=3 ttl=64 time=0.164 ms
64 bytes from 192.168.15.60: icmp_seq=4 ttl=64 time=0.138 ms
64 bytes from 192.168.15.60: icmp_seq=5 ttl=64 time=0.156 ms
64 bytes from 192.168.15.60: icmp_seq=6 ttl=64 time=0.135 ms

--- 192.168.15.60 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5007ms
rtt min/avg/max/mdev = 0.135/1.515/8.333/3.049 ms
```


VMs on the same physical machine and VMs on different physical machines can ping each other.

----End

4.5 Verifying SR-IOV

4.5.1 Restoring the Environment

Run the following command to restore the environment:

```
virsh shutdown vm$
ovs-vsctl del-br br-ovs
echo 0 > /sys/class/net/enp1s0f0/device/sriov_numvfs
echo 0 > /sys/class/net/enp1s0f1/device/sriov_numvfs
```

NOTE

The verification operations that follow are performed for each different function of SR-IOV. Before verifying each function, restore the environment.

4.5.2 Bonding

Step 1 Create VFs.

```
echo 4 > /sys/class/net/enp1s0f0/device/sriov_numvfs
ip link set enp1s0f0 vf 0 mac e4:11:22:33:61:11
ip link set enp1s0f0 vf 1 mac e4:11:22:33:61:22
ip link set enp1s0f0 vf 2 mac e4:11:22:33:61:33
ip link set enp1s0f0 vf 3 mac e4:11:22:33:61:44
echo 0000:01:00.2 > /sys/bus/pci/drivers/mlx5_core/unbind
echo 0000:01:00.3 > /sys/bus/pci/drivers/mlx5_core/unbind
echo 0000:01:00.4 > /sys/bus/pci/drivers/mlx5_core/unbind
echo 0000:01:00.5 > /sys/bus/pci/drivers/mlx5_core/unbind
devlink dev eswitch set pci/0000:01:00.0 mode switchdev
echo 4 > /sys/class/net/enp1s0f1/device/sriov_numvfs
ip link set enp1s0f1 vf 0 mac e4:11:22:33:62:11
ip link set enp1s0f1 vf 1 mac e4:11:22:33:62:22
ip link set enp1s0f1 vf 2 mac e4:11:22:33:62:33
ip link set enp1s0f1 vf 3 mac e4:11:22:33:62:44
echo 0000:01:01.2 > /sys/bus/pci/drivers/mlx5_core/unbind
echo 0000:01:01.3 > /sys/bus/pci/drivers/mlx5_core/unbind
echo 0000:01:01.4 > /sys/bus/pci/drivers/mlx5_core/unbind
echo 0000:01:01.5 > /sys/bus/pci/drivers/mlx5_core/unbind
devlink dev eswitch set pci/0000:01:00.1 mode switchdev
```

NOTICE

- For a Linux bond, after setting **pci mode** to **switchdev**, strictly follow the operation procedure. Do not rebind the VFs immediately. Otherwise, an error is reported.
- Though it is allowed to create a bond immediately after the system is started, but it is not advised to do so. You can set **onboot** to **no** in the **ifcfg-xxxxx** file.
- Each MAC address must be unique on the local server, the peer server, and the switch.

Step 2 Create a Linux bond.

1. Modify the configuration file of the physical NIC **enp1s0f0 enp1s0f1**.

```
vim /etc/sysconfig/network-scripts/ifcfg-enp1s0f0
vim /etc/sysconfig/network-scripts/ifcfg-enp1s0f1
```

Add the following two lines:

```
MASTER=bond0
SLAVE=yes
```

```
TYPE=Ethernet
PROXY_METHOD=none
BROWSER_ONLY=no
BOOTPROTO=none
IPV4_FAILURE_FATAL=no
NAME=enp3s0f0
UUID=65312721-3ff2-4fd6-bbfc-a5f97181b2c9
DEVICE=enp3s0f0
ONBOOT=yes
NM_CONTROLLED=no
MASTER=bond0
SLAVE=yes
```

2. Add a configuration file for the port bond.

```
vim /etc/sysconfig/network-scripts/ifcfg-bond0
```

Add the following content:

```
DEVICE=bond0
NAME='bond0'
TYPE=Ethernet
NM_CONTROLLED=no
ONBOOT=yes
BOOTPROTO=none
BONDING_OPTS='mode=4 miimon=100 xmit_hash_policy=layer3+4'
IPV6INIT=no
```

NOTE

- In the **BONDING_OPTS** configuration:
  - **mode** indicates the mode. The value **4** indicates LACP.
  - **miimon=100** indicates that the monitoring is performed every 100 ms.
  - **xmit_hash_policy=layer3+4** indicates the LACP balancing policy, which uses Layer 3 and Layer 4 balancing, IP addresses, and ports.
- The active/standby mode parameter is **mode=1 miimon=100**.

3. Load the bond kernel module.

```
modprobe bonding mode=4 miimon=100
```

NOTE

If needed, set **mode** to **1** for the active/standby mode, and to **4** for the LACP mode.

4. Start the bond.

```
ifup bond0
```

NOTICE

Do not restart the network.

5. Check the bond configuration.
   cat /proc/net/bonding/bond0

```
[root@localhost ~]# cat /proc/net/bonding/bond0
Ethernet Channel Bonding Driver: v3.7.1 (April 27, 2011)

Bonding Mode: IEEE 802.3ad Dynamic link aggregation
Transmit Hash Policy: layer3+4 (1)
MII Status: down
MII Polling Interval (ms): 100
Up Delay (ms): 0
Down Delay (ms): 0

802.3ad info
LACP rate: slow
Min links: 0
Aggregator selection policy (ad_select): stable
System priority: 0
System MAC address: 00:00:00:00:00:00
bond bond0 has no active aggregator
```

Step 3 Bind the VFs.

```
echo 0000:01:00.2 > /sys/bus/pci/drivers/mlx5_core/bind
echo 0000:01:00.3 > /sys/bus/pci/drivers/mlx5_core/bind
echo 0000:01:00.4 > /sys/bus/pci/drivers/mlx5_core/bind
echo 0000:01:00.5 > /sys/bus/pci/drivers/mlx5_core/bind
echo 0000:01:01.2 > /sys/bus/pci/drivers/mlx5_core/bind
echo 0000:01:01.3 > /sys/bus/pci/drivers/mlx5_core/bind
echo 0000:01:01.4 > /sys/bus/pci/drivers/mlx5_core/bind
echo 0000:01:01.5 > /sys/bus/pci/drivers/mlx5_core/bind
```

Step 4 Start OVS and configure the network.

```
systemctl start openvswitch
ovs-vsctl add-br br-ovs
ovs-vsctl add-port br-ovs bond0
ovs-vsctl add-port br-ovs enp1s0f0_0
ovs-vsctl add-port br-ovs enp1s0f0_1
ovs-vsctl add-port br-ovs enp1s0f0_2
ovs-vsctl add-port br-ovs enp1s0f0_3
ovs-vsctl add-port br-ovs enp1s0f1_3
ovs-vsctl add-port br-ovs enp1s0f1_2
ovs-vsctl add-port br-ovs enp1s0f1_1
ovs-vsctl add-port br-ovs enp1s0f1_0
ifconfig bond0 up
ip link set dev enp1s0f0_0 up
ip link set dev enp1s0f0_1 up
ip link set dev enp1s0f0_2 up
ip link set dev enp1s0f0_3 up
ip link set dev enp1s0f1_0 up
ip link set dev enp1s0f1_1 up
ip link set dev enp1s0f1_2 up
ip link set dev enp1s0f1_3 up
```

Step 5 Start the VM and log in to it.

```
virsh start vm1
virsh console vm1
```

Step 6 Use the two VMs to send traffic.

- Run the following command on VM 1 of server 2:
  ```
  iperf3 -s
  ```
- Run the following command on VM 1 of server 1:
  ```
  iperf3 -c <Host2vm1_ip> -t 0
  ```


Step 7 Press **Ctrl+]** to exit from the VMs. View the flow table offloading on a physical machine.

```
watch -n 1 -d ovs-appctl dpctl/dump-flows type=offloaded
```

```
Every 1.0s: ovs-appctl dpctl/dump-flows type=offloaded                    Wed Oct 28 15:12:32 2020

recirc_id(0),in_port(2),eth(src=3a:db:2d:34:18:fd,dst=7a:f9:c8:cf:d7:ec),eth_type(0x0800),ipv4(frag=no)
, packets:204859, bytes:13521585, used:1.750s, actions:10
recirc_id(0),in_port(10),eth(src=7a:f9:c8:cf:d7:ec,dst=3a:db:2d:34:18:fd),eth_type(0x0800),ipv4(frag=no
), packets:8063314, bytes:12207828941, used:1.750s, actions:2
```

Verify the bidirectional flow table offloading. In LACP mode, verify that the total bandwidth of the eight VMs is close to the total bandwidth of the two network ports. In active/standby mode, verify the network connectivity when one network port is disconnected.

Step 8 Delete the bond.

1. Stop the VM.
   ```
   virsh shutdown vm1
   ```
2. Delete the OVS bridge.
   ```
   ovs-vsctl del-br br-ovs
   ```
3. Unbind all related VFs (for details, see **Step 1**) or clear the VFs.
   ```
   echo 0 > /sys/class/net/enp1s0f0/device/sriov_numvfs
   echo 0 > /sys/class/net/enp1s0f1/device/sriov_numvfs
   ```
4. Delete the bond.
   ```
   ip link delete bond0
   rmmod bonding
   ```

----End

4.5.3 QoS

Step 1 Configure VFs.

For details, see **4.4.2 Configuring Kernel-Mode SR-IOV**.

Step 2 Start Open Virtual Switch (OVS) and configure the network.

```
systemctl start openvswitch
ovs-vsctl add-br br-ovs
ovs-vsctl add-port br-ovs enp1s0f0_0
ovs-vsctl add-port br-ovs enp1s0f0
ip link set dev enp1s0f0 up
ip link set dev enp1s0f0_0 up
```

Step 3 Start the VM.

```
virsh start vm1
```

Step 4 Configure QoS rate limiting in the ingress direction.

```
ovs-vsctl set Interface enp1s0f0_0 ingress_policing_rate=100000
```

Step 5 Log in to the VM.

```
virsh console vm1
```

Step 6 Generate traffic to verify the rate limiting.

- Run the following command on VM 1 of server 2:
  ```
  iperf3 -s
  ```
- Run the following command on VM 1 of server 1:
  ```
  iperf3 -c <Host2vm1_ip> -t 0
  ```

When the two VMs send traffic to each other, rate limiting does not work.


```
[root@localhost ~]# iperf3 -c 192.168.11.11
Connecting to host 192.168.11.11, port 5201
[  4] local 192.168.11.21 port 39182 connected to 192.168.11.11 port 5201
[ ID] Interval           Transfer     Bandwidth       Retr  Cwnd
[  4]   0.00-1.00   sec  1.09 GBytes  9.35 Gbits/sec    0    949 KBytes
[  4]   1.00-2.00   sec  1.09 GBytes  9.38 Gbits/sec    0   1.08 MBytes
[  4]   2.00-2.18   sec   202 MBytes  9.36 Gbits/sec    0   1.08 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth       Retr
[  4]   0.00-2.18   sec  2.38 GBytes  9.36 Gbits/sec    0             sender
[  4]   0.00-2.18   sec  0.00 Bytes  0.00 bits/sec                  receiver
iperf3: interrupt - the client has terminated
[root@localhost ~]# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
Accepted connection from 192.168.11.11, port 38754
[  5] local 192.168.11.21 port 5201 connected to 192.168.11.11 port 38756
[ ID] Interval           Transfer     Bandwidth
[  5]   0.00-1.00   sec  1.05 GBytes  9.03 Gbits/sec
[  5]   1.00-2.00   sec  1.09 GBytes  9.39 Gbits/sec
[  5]   1.00-2.00   sec  1.09 GBytes  9.39 Gbits/sec
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth
[  5]   0.00-2.00   sec  0.00 Bytes  0.00 bits/sec                  sender
[  5]   0.00-2.00   sec  2.29 GBytes  9.83 Gbits/sec                  receiver
iperf3: the client has terminated
```

NOTE

The kernel version must be 5.7 or later to support inbound port rate limiting.

Step 7 Press **Ctrl+]** to exit from the VM. Configure QoS rate limiting in the outbound direction on the physical machine.

```
ovs-vsctl set port enp1s0f0_0 qos=@newqos -- --id=@newqos create qos type=linux-htb other-config:max-rate=200000000 \
queues=123=@q1 -- --id=@q1 create queue other-config:max-rate=200000000
ovs-ofctl add-flow br-ovs "in_port=2,actions=set_queue:123,normal"
```

Step 8 Log in to the VM again and generate traffic to verify outbound rate limiting.

```
[root@localhost ~]# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
Accepted connection from 192.168.11.11, port 38764
[  5] local 192.168.11.21 port 5201 connected to 192.168.11.11 port 38766
[ ID] Interval           Transfer     Bandwidth
[  5]   0.00-1.00   sec  22.1 MBytes   186 Mbits/sec
[  5]   1.00-2.00   sec  22.8 MBytes   191 Mbits/sec
[  5]   2.00-3.00   sec  22.8 MBytes   191 Mbits/sec
[  5]   3.00-4.00   sec  22.8 MBytes   191 Mbits/sec
[  5]   4.00-5.00   sec  22.8 MBytes   191 Mbits/sec
[  5]   5.00-6.00   sec  22.8 MBytes   191 Mbits/sec
[  5]   5.00-6.00   sec  22.8 MBytes   191 Mbits/sec
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth
[  5]   0.00-6.00   sec  0.00 Bytes  0.00 bits/sec                  sender
[  5]   0.00-6.00   sec   156 MBytes   217 Mbits/sec                  receiver
iperf3: the client has terminated
```


Step 9 Press **Ctrl+]** to exit from the VM. View the flow table offloading on a physical machine.

```
watch -n 1 -d ovs-appctl dpctl/dump-flows
```

Flow table offloading:

```
[root@localhost scripts]# ovs-appctl dpctl/dump-flows
recirc_id(0),in_port(2),eth(src=16:d7:bf:3e:c3:b4,dst=9e:f3:0e:8a:89:6f),eth_type(0x0800),ipv4(frag=no)
, packets:9308, bytes:614340, used:0.240s, actions:3
recirc_id(0),skb_priority(0),in_port(3),eth(src=9e:f3:0e:8a:89:6f,dst=16:d7:bf:3e:c3:b4),eth_type(0x080
0),ipv4(frag=no), packets:10259, bytes:205368128, used:0.020s, flags:SP., actions:set(skb_priority(0x10
07c)),2
[root@localhost scripts]# ovs-appctl dpctl/dump-flows type=offloaded
recirc_id(0),in_port(2),eth(src=16:d7:bf:3e:c3:b4,dst=9e:f3:0e:8a:89:6f),eth_type(0x0800),ipv4(frag=no)
, packets:14638, bytes:966120, used:0.790s, actions:3
```

----End

Verification Result

Rate limiting works in the inbound direction but not in the outbound direction. The rate limiting flow table is not offloaded.

4.5.4 Port Mirroring

Step 1 Configure VFs.

For details, see **4.4.2 Configuring Kernel-Mode SR-IOV**.

Step 2 Start OVS and configure the network.

```
systemctl start openvswitch
ovs-vsctl add-br br-ovs
ovs-vsctl add-port br-ovs enp1s0f0_0
ovs-vsctl add-port br-ovs enp1s0f0_1
ovs-vsctl add-port br-ovs enp1s0f0_2
ovs-vsctl add-port br-ovs enp1s0f0_3
ovs-vsctl add-port br-ovs enp1s0f0_4
ovs-vsctl add-port br-ovs enp1s0f0_5
ovs-vsctl add-port br-ovs enp1s0f0_6
ovs-vsctl add-port br-ovs enp1s0f0_7
ovs-vsctl add-port br-ovs enp1s0f0
ip link set dev enp1s0f0 up
ip link set dev enp1s0f0_0 up
ip link set dev enp1s0f0_1 up
ip link set dev enp1s0f0_2 up
ip link set dev enp1s0f0_3 up
ip link set dev enp1s0f0_4 up
ip link set dev enp1s0f0_5 up
ip link set dev enp1s0f0_6 up
ip link set dev enp1s0f0_7 up
```

Step 3 Start the VMs.

- Run the following command on server 1:
  ```
  virsh start vm1
  ```
- Run the following command on server 2:
  ```
  virsh start vm1
  virsh start vm4
  ```

Step 4 Configure the OVS SPAN port mirroring.

```
ovs-vsctl -- --id=@p get port enp1s0f0_3 -- --id=@q get port enp1s0f0_0 -- --id=@m create mirror
name=m0 select_src_port=@q select_dst_port=@q output-port=@p -- set bridge br-ovs mirrors=@m
```


NOTE

In this command, configure port mirroring on **br-ovs** to mirror the incoming and outgoing traffic of port **enp1s0f0_0** to port **enp1s0f0_3**.

- **-- --id=@p get port enp1s0f0_3**: Create an alias of port **enp1s0f0_3**.
- **-- --id=@m create mirror name=m0**: Create a port mirror.
- **select_src_port=@q select_dst_port=@q output-port=@p**: Set a port mirroring rule. Among the parameters, **select_src_port** indicates that the traffic entering this port is mirrored, **select_dst_port** indicates that the traffic leaving this port is mirrored, and **output-port** indicates that the mirrored traffic is outputted to the designated port.
- **-- set bridge br-ovs mirrors=@m**: Specify the port mirroring rule of the bridge.

Step 5 Log in to the VM.

Run the following command on server 2:

```
virsh console vm4
```

Step 6 Capture packets on the mirrored port. Send packets from server 1 to VM 1 (**enp1s0f0_0**) of server 2, and capture packets on VM 4 (**enp1s0f0_3**) of server 2.

Run the following command on VM 4 of server 2:

```
tcpdump -i enp1s0
```

Step 7 Generate traffic on the VMs and check the captured packets.

- Run the following command on VM 1 of server 2:
  ```
  iperf3 -s
  ```
- Run the following command on VM 1 of server 1:
  ```
  iperf3 -c <Host2vm1_ip> -t 0
  ```

Captured packets:

```
[root@localhost ~]# tcpdump -i enp1s0
[ 1708.340259] device enp1s0 entered promiscuous mode
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on enp1s0, link-type EN10MB (Ethernet), capture size 262144 bytes
11:20:05.439715 IP 192.168.11.21 > 192.168.11.11: ICMP echo request, id 1329, seq 896, length 64
11:20:05.439765 IP 192.168.11.11 > 192.168.11.21: ICMP echo reply, id 1329, seq 896, length 64
11:20:06.479672 IP 192.168.11.21 > 192.168.11.11: ICMP echo request, id 1329, seq 897, length 64
11:20:06.479701 IP 192.168.11.11 > 192.168.11.21: ICMP echo reply, id 1329, seq 897, length 64

4 packets captured
4 packets received by filter
0 packets dropped by kernel
[ 1710.804540] device enp1s0 left promiscuous mode
[root@localhost ~]#
```

Step 8 Press **Ctrl+]** to exit from the VMs and view the flow table offloading.

```
watch -n 1 -d ovs-appctl dpctl/dump-flows type=offloaded
```

Offloading status:

```
Every 1.0s: ovs-appctl dpctl/dump-flows type=offloaded  localhost.localdomain: Tue Nov 17 11:21:17 2020

recirc_id(0),in_port(2),eth(src=9a:20:f6:b0:f2:db,dst=1a:aa:ff:41:f8:ae),eth_type(0x0806), packets:60,
bytes:3600, used:8.410s, actions:3,6
recirc_id(0),in_port(2),eth(src=9a:20:f6:b0:f2:db,dst=1a:aa:ff:41:f8:ae),eth_type(0x0800),ipv4(frag=no)
, packets:964, bytes:94472, used:0.090s, actions:3,6
recirc_id(0),in_port(3),eth(src=1a:aa:ff:41:f8:ae,dst=9a:20:f6:b0:f2:db),eth_type(0x0806), packets:59,
bytes:3540, used:8.410s, actions:6,2
recirc_id(0),in_port(3),eth(src=1a:aa:ff:41:f8:ae,dst=9a:20:f6:b0:f2:db),eth_type(0x0800),ipv4(frag=no)
, packets:964, bytes:94472, used:0.090s, actions:6,2
```

Step 9 Clear port mirrors.


```
ovs-vsctl clear bridge br-ovs mirrors
```

Step 10 Configure the OVS RSPAN port mirroring.

```
ovs-vsctl set bridge br-ovs flood_vlans=111
ovs-vsctl -- --id=@q get port enp1s0f0_0 -- --id=@m create mirror name=m0 select_src_port=@q
select_dst_port=@q output_vlan=111 -- set bridge br-ovs mirrors=@m
```

Step 11 Repeat **Step 5** to **Step 8** to check the packet capturing and offloading.

NOTE

Capture packets on VM 4 of server 2:

```
tcpdump -i enp1s0 -ne
```

Captured packets:

```
[root@localhost ~]# tcpdump -i enp1s0 -ne
[ 3662.420264] device enp1s0 entered promiscuous mode
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on enp1s0, link-type EN10MB (Ethernet), capture size 262144 bytes
11:52:39.324841 9a:20:f6:b0:f2:db > 1a:aa:ff:41:f8:ae, ethertype 802.1Q (0x8100), length 102: vlan 111, p 0, e
thertype IPv4, 192.168.11.21 > 192.168.11.11: ICMP echo request, id 1370, seq 8, length 64
11:52:39.330157 1a:aa:ff:41:f8:ae > 9a:20:f6:b0:f2:db, ethertype 802.1Q (0x8100), length 102: vlan 111, p 0, e
thertype IPv4, 192.168.11.11 > 192.168.11.21: ICMP echo reply, id 1370, seq 8, length 64
11:52:40.326050 9a:20:f6:b0:f2:db > 1a:aa:ff:41:f8:ae, ethertype 802.1Q (0x8100), length 102: vlan 111, p 0, e
thertype IPv4, 192.168.11.21 > 192.168.11.11: ICMP echo request, id 1370, seq 9, length 64
11:52:40.331387 1a:aa:ff:41:f8:ae > 9a:20:f6:b0:f2:db, ethertype 802.1Q (0x8100), length 102: vlan 111, p 0, e
thertype IPv4, 192.168.11.11 > 192.168.11.21: ICMP echo reply, id 1370, seq 9, length 64
11:52:41.327284 9a:20:f6:b0:f2:db > 1a:aa:ff:41:f8:ae, ethertype 802.1Q (0x8100), length 102: vlan 111, p 0, e
thertype IPv4, 192.168.11.21 > 192.168.11.11: ICMP echo request, id 1370, seq 10, length 64
11:52:41.332598 1a:aa:ff:41:f8:ae > 9a:20:f6:b0:f2:db, ethertype 802.1Q (0x8100), length 102: vlan 111, p 0, e
thertype IPv4, 192.168.11.11 > 192.168.11.21: ICMP echo reply, id 1370, seq 10, length 64

6 packets captured
6 packets received by filter
0 packets dropped by kernel
[ 3665.353231] device enp1s0 left promiscuous mode
[root@localhost ~]#
```

Offloading status:

```
[root@localhost kvm]# ovs-appctl dpctl/dump-flows type=offloaded
[root@localhost kvm]#
```

----End

Verification Result

Port mirroring in SPAN mode supports packet capturing and flow table offloading. Port mirroring in other modes supports only packet capturing.

4.5.5 GRO

Step 1 Check the generic receive offload (GRO) and large receive offload (LRO) status on the VM.

```
ethtool -k <dev> |grep generic-receive-offload
ethtool -k <dev> |grep large-receive-offload
```

GRO and LRO are enabled by default.

Step 2 Disable GRO and LRO on the receiver.

Run the following command on VM 1 of server 2:

```
ethtool -K <dev> lro off
ethtool -K <dev> gro off
```


Step 3 Receive packets on the receiver.

Run the following command on VM 1 of server 2:

```
iperf3 -s &
tcpdump tcp -i <dev>
```

Step 4 Send packets from the sender.

Run the following command on VM 1 of server 1:

```
iperf3 -c <dst_ip>
```

```
[root@VM1 ~]# iperf3 -c 192.168.1.11
Connecting to host 192.168.1.11, port 5201
[  5] local 192.168.1.21 port 40456 connected to 192.168.1.11 port 5201



[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[  5]   0.00-1.00   sec  1.01 GBytes  8.69 Gbits/sec  222    557 KBytes
[  5]   1.00-2.00   sec  1.29 GBytes  11.1 Gbits/sec  188   1.45 MBytes
[  5]   2.00-3.00   sec  1.28 GBytes  11.0 Gbits/sec   70    807 KBytes
[  5]   3.00-4.00   sec  1.28 GBytes  11.0 Gbits/sec    0    809 KBytes
[  5]   4.00-5.00   sec  1.28 GBytes  11.0 Gbits/sec    0    809 KBytes
[  5]   5.00-6.00   sec  1.28 GBytes  11.0 Gbits/sec   92    740 KBytes
[  5]   6.00-7.00   sec  1.22 GBytes  10.5 Gbits/sec  265   1.33 MBytes
[  5]   7.00-8.00   sec  1.15 GBytes  9.88 Gbits/sec  1071   1.06 MBytes
[  5]   8.00-9.00   sec  1.16 GBytes  10.0 Gbits/sec  1114   1.39 MBytes
[  5]   9.00-10.00  sec  1.16 GBytes  9.97 Gbits/sec  723   1.29 MBytes
```

The length on the receiver is 1448.

```
11:39:46.208837 IP 192.168.1.21.40460 > VM1.targus-getdata1: Flags [.], seq 3661083982:3661085430, ack 1, win 229, options [nop,nop,TS val 3504113174 ecr 2864200762], length 1448
11:39:46.211589 IP 192.168.1.21.40460 > VM1.targus-getdata1: Flags [.], seq 3664121886:3664123334, ack 1, win 229, options [nop,nop,TS val 3504113177 ecr 2864200764], length 1448
11:39:46.214197 IP 192.168.1.21.40460 > VM1.targus-getdata1: Flags [.], seq 3666984582:3666986030, ack 1, win 229, options [nop,nop,TS val 3504113179 ecr 2864200767], length 1448
11:39:46.216783 IP 192.168.1.21.40460 > VM1.targus-getdata1: Flags [.], seq 3669811078:3669812526, ack 1, win 229, options [nop,nop,TS val 3504113182 ecr 2864200770], length 1448
```

Step 5 Enable LRO at the receiver.

```
ethtool -K <dev> lro on
11:52:01.964338 IP 192.168.1.21.40472 > VM1.targus-getdata1: Flags [.], seq 4106154718:4106156166, ack 1, win 229, options [nop,nop,TS val 3504848929 ecr 2864936518], length 1448
11:52:01.966923 IP 192.168.1.21.40472 > VM1.targus-getdata1: Flags [.], seq 4112365190:4112430350, ack 1, win 229, options [nop,nop,TS val 3504848931 ecr 2864936521], length 65160
11:52:01.969460 IP 192.168.1.21.40472 > VM1.targus-getdata1: Flags [.], seq 4118834854:4118836302, ack 1, win 229, options [nop,nop,TS val 3504848934 ecr 2864936523], length 1448
11:52:01.972144 IP 192.168.1.21.40472 > VM1.targus-getdata1: Flags [.], seq 4125595566:4125660726, ack 1, win 229, options [nop,nop,TS val 3504848937 ecr 2864936526], length 65160
11:52:01.974685 IP 192.168.1.21.40472 > VM1.targus-getdata1: Flags [.], seq 4131612006:4131677166, ack 1, win 229, options [nop,nop,TS val 3504848939 ecr 2864936528], length 65160
11:52:01.977213 IP 192.168.1.21.40472 > VM1.targus-getdata1: Flags [.], seq 4137750078:4137751526, ack 1, win 229, options [nop,nop,TS val 3504848942 ecr 2864936531], length 1448
11:52:01.979802 IP 192.168.1.21.40472 > VM1.targus-getdata1: Flags [.], seq 4144198022:4144199470, ack 1, win 229, options [nop,nop,TS val 3504848944 ecr 2864936533], length 1448
```

Step 6 Enable GRO.

```
ethtool -K <dev> gro on
11:49:03.845248 IP 192.168.1.21.40468 > VM1.targus-getdata1: Flags [.], seq 1786770278:1786835438, ack 1, win 229, options [nop,nop,TS val 3504670810 ecr 2864758399], length 65160
11:49:03.849930 IP 192.168.1.21.40468 > VM1.targus-getdata1: Flags [.], seq 1800345278:1800410438, ack 1, win 229, options [nop,nop,TS val 3504670815 ecr 2864758404], length 65160
11:49:03.852684 IP 192.168.1.21.40468 > VM1.targus-getdata1: Flags [.], seq 1808393262:1808458422, ack 1, win 229, options [nop,nop,TS val 3504670817 ecr 2864758406], length 65160
11:49:03.855304 IP 192.168.1.21.40468 > VM1.targus-getdata1: Flags [.], seq 1815960510:1816025670, ack 1, win 229, options [nop,nop,TS val 3504670820 ecr 2864758409], length 65160
11:49:03.857815 IP 192.168.1.21.40468 > VM1.targus-getdata1: Flags [.], seq 1823200510:1823265670, ack 1, win 229, options [nop,nop,TS val 3504670823 ecr 2864758411], length 65160
11:49:03.860765 IP 192.168.1.21.40468 > VM1.targus-getdata1: Flags [.], seq 1830801062:1830866222, ack 1, win 229, options [nop,nop,TS val 3504670826 ecr 2864758414], length 65160
```



```
11:49:03.863560 IP 192.168.1.21.40468 > VM1.targus-getdata1: Flags [.], seq 1836160110:1836225270, ack
1, win 229, options [nop,nop,TS val 3504670828 ecr 2864758417], length 65160
11:49:03.866532 IP 192.168.1.21.40468 > VM1.targus-getdata1: Flags [.], seq 1840640222:1840705382, ack
1, win 229, options [nop,nop,TS val 3504670831 ecr 2864758420], length 65160
11:49:03.869366 IP 192.168.1.21.40468 > VM1.targus-getdata1: Flags [.], seq 1844448462:1844489006, ack
1, win 229, options [nop,nop,TS val 3504670834 ecr 2864758423], length 40544
```

----End

4.5.6 Traffic Sampling

Step 1 Configure sFlow on the sender.

Run the following command on server 1:

```
ovs-vsctl -- --id=@sflow create sflow agent=enp3s0 target=\"<host2_ip>:6343\" header=128 sampling=64
polling=10 -- set bridge br-ovs sflow=@sflow
```

NOTE

Set **target** to the collector IP address, **agent** to the sflow egress. Add the OVS bridge name next to **bridge**.

```
[root@master ~]# ovs-vsctl list sFlow
_uuid               : 46acb9e3-17fb-4689-b33b-562acd0c85bd
agent               : enp4s0f0
external_ids        : {}
header              : 128
polling             : 10
sampling            : 64
targets             : ["192.168.1.3:6343"]
```

Step 2 Install the sflowtool on the collector.

```
git config --global http.sslVerify false
git clone https://github.com/sflow/sflowtool
cd sflowtool
./boot.sh
./configure
make
make install
```

Step 3 Use the sFlow.

Run the following command on server 2:

```
./src/sflowtool -p 6343
```

Step 4 Start two VMs on the sender to send packets. On the physical machine used to sample traffic, check the sampling result.

The following information is displayed:

```
headerBytes E4-11-22-33-44-51-E4-11-22-33-44-50-08-00-45-00-00-54-D4-4E-00-00-40-01-22-DF-C0-
A8-01-15-C0-A8-01-16-00-00-B9-20-2B-93-00-1A-D6-87-9A-5F-00-00-00-00-
E7-77-04-00-00-00-00-00-10-11-12-13-14-15-16-17-18-19-1A-1B-1C-1D-1E-1F-20-21-22-23-24-25-26-27-2
8-29-2A-2B-2C-2D-2E-2F-30-31-32-33-34-35-36-37
dstMAC e41122334451
srcMAC e41122334450
IPSize 84
ip.tot_len 84
srcIP 192.168.1.21
dstIP 192.168.1.22
```

Step 5 Clear the network configuration.

```
ovs-vsctl del-br br-ovs
```


Step 6 Configure netflow on the sender.

Run the following command on server 1:

```
ovs-vsctl -- set Bridge br-ovs netflow=@nf -- --id=@nf create NetFlow target=\"<host2_ip>:2055\" active-timeout=60
```

Step 7 Use the NetFlow.

Run the following command on server 2:

```
./src/sflowtool -p 2055
```

Step 8 Repeat **Step 4** and check the sampling result.

The following information is displayed:

```
startDatagram =================================
datagramSourceIP 90.90.48.60
datagramSize 120
unixSecondsUTC 1607393677
localtime 2020-12-08T10:14:37+0800
datagramVersion 327682
unexpected datagram version number
 (source IP = 90.90.48.60)
00-05-00-02-<*>-00-00-d7-15-5f-ce-e1-ac-2a-e4-37-46
00-00-00-01-5b-5b-00-00-c0-a8-63-0b-c0-a8-63-15
00-00-00-00-00-09-00-01-00-00-00-08-00-00-02-a6
00-00-5f-a7-00-00-60-1a-c9-26-14-51-00-1a-06-00
00-00-00-00-00-00-00-00-c0-a8-63-15-c0-a8-63-0b
00-00-00-00-00-01-00-09-00-00-00-07-00-00-01-da
00-00-5f-a9-00-00-60-1a-14-51-c9-26-00-1a-06-00
00-00-00-00-00-00-00-00
caught exception: 2
endDatagram   =================================
```

----End

4.5.7 Protocol Offloading

VLAN Offloading

Step 1 Configure VFs.

For details, see **4.4.2 Configuring Kernel-Mode SR-IOV**.

Step 2 Configure the network.

```
systemctl start openvswitch
ovs-vsctl add-br br-ovs
ovs-vsctl add-port br-ovs enp1s0f0_0
ovs-vsctl add-port br-ovs enp1s0f0_1
ovs-vsctl add-port br-ovs enp1s0f0_2
ovs-vsctl add-port br-ovs enp1s0f0_3
ovs-vsctl add-port br-ovs enp1s0f0
ip link set dev enp1s0f0 up
ip link set dev enp1s0f0_0 up
ip link set dev enp1s0f0_1 up
ip link set dev enp1s0f0_2 up
ip link set dev enp1s0f0_3 up
ovs-vsctl set Port enp1s0f0_0 tag=100
```

Step 3 Start the VM and log in to it.

```
virsh start vm1
virsh console vm1
```

Step 4 Verify the network connectivity.

Run the following command on VM 1 of server 1:


```
ping <Host2 vm1_IP>
```

```
[root@localhost ~]# ping 192.168.11.21
PING 192.168.11.21 (192.168.11.21) 56(84) bytes of data.
64 bytes from 192.168.11.21: icmp_seq=1 ttl=64 time=3.08 ms
64 bytes from 192.168.11.21: icmp_seq=2 ttl=64 time=0.159 ms

--- 192.168.11.21 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0.159/1.622/3.086/1.464 ms
[root@localhost ~]# ping 192.168.11.22
PING 192.168.11.22 (192.168.11.22) 56(84) bytes of data.

--- 192.168.11.22 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1044ms

[root@localhost ~]# ping 192.168.11.23
PING 192.168.11.23 (192.168.11.23) 56(84) bytes of data.

--- 192.168.11.23 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1033ms
```

Step 5 Verify flow table offloading and VLAN tagging.

Use the two VMs to send traffic and check whether the flow table is offloaded.

- Run the following command on VM 1 of server 2:
  ```
  iperf3 -s
  ```
- Run the following command on VM 1 of server 1:
  ```
  iperf3 -c <Host2vm1_ip> -t 0
  ```
- Run the following command on any of the physical machines:
  ```
  ovs-appctl dpctl/dump-flows type=offloaded
  ```

```
Every 1.0s: ovs-appctl dpctl/dump-flows type=offloaded                  Mon Nov  2 11:06:12 2020

recirc_id(0),in_port(3),eth(src=06:6a:aa:09:7d:2b,dst=86:fa:b7:67:a0:e3),eth_type(0x0800),ipv4(frag=no)
, packets:409287, bytes:28650654, used:0.080s, actions:push_vlan(vid=100,pcp=0),1
recirc_id(0),in_port(1),eth(src=86:fa:b7:67:a0:e3,dst=06:6a:aa:09:7d:2b),eth_type(0x8100),vlan(vid=100,
pcp=0),encap(eth_type(0x0800),ipv4(frag=no)), packets:17598886, bytes:2664468710, used:0.080s, actions
:pop_vlan,3
```

Step 6 Verify the CT configuration.

1. Configure the CT flow table on the physical machine.
   ```
   ovs-ofctl del-flows br-ovs
   ovs-ofctl add-flow br-ovs "arp, actions=normal"
   ovs-ofctl add-flow br-ovs "table=0, ip,ct_state=-trk, actions=ct(table=1)"
   ovs-ofctl add-flow br-ovs "table=1, ip,ct_state=+trk+new, actions=ct(commit),normal"
   ovs-ofctl add-flow br-ovs "table=1, ip,ct_state=+trk+est, actions=normal"
   ```
2. Verify the flow table offloading.
   ```
   ovs-ofctl dump-flows br-ovs
   ```



```
Every 1.0s: ovs-ofctl dump-flows br-ovs                                 Mon Nov  2 11:12:20 2020

NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=269.111s, table=0, n_packets=319628, n_bytes=483896412, idle_age=3, arp actions=
NORMAL
 cookie=0x0, duration=269.097s, table=0, n_packets=5, n_bytes=4710, idle_age=267, ct_state=-trk,ip acti
ons=ct(table=1)
 cookie=0x0, duration=269.083s, table=1, n_packets=1, n_bytes=1518, idle_age=268, ct_state=+new+trk,ip
actions=ct(commit),NORMAL
 cookie=0x0, duration=268.273s, table=1, n_packets=218547051, n_bytes=32334269184 9, idle_age=1, ct_stat
e=+est+trk,ip actions=NORMAL
```

NOTE

After the preceding commands are executed, the command output is displayed only once. You can run the **watch -n 1 -d 'commands'** command to view the command output again. Replace **commands** with the actual command.

ovs-appctl dpctl/dump-flows

```
Every 1.0s: ovs-appctl dpctl/dump-flows                                 Mon Nov  2 11:13:58 2020

recirc_id(0x6),in_port(3),eth(src=06:6a:aa:09:7d:2b,dst=86:fa:b7:67:a0:e3),eth_type(0x0800),ipv4(frag=n
o), packets:53149, bytes:3720610, used:0.770s, actions:push_vlan(vid=100,pcp=0),1
recirc_id(0x5),in_port(1),eth(src=86:fa:b7:67:a0:e3,dst=06:6a:aa:09:7d:2b),eth_type(0x8100),vlan(vid=10
0,pcp=0),encap(eth_type(0x0800),ipv4(frag=no)), packets:1596285, bytes:241667430 0, used:0.770s, actions
:pop_vlan,3
recirc_id(0),in_port(3),ct_state(-trk),eth(),eth_type(0x0800),ipv4(frag=no), packets:0, bytes:0, used:n
ever, actions:ct,recirc(0x6)
recirc_id(0),in_port(1),ct_state(-trk),eth(),eth_type(0x8100),vlan(vid=100/0x64),encap(eth_type(0x0800)
,ipv4(frag=no)), packets:1, bytes:70, used:2.860s, flags:., actions:ct,recirc(0x5)
recirc_id(0x5),in_port(1),ct_state(+new+trk),eth(src=86:fa:b7:67:a0:e3,dst=06:6a:aa:09:7d:2b),eth_type(
0x8100),vlan(vid=100,pcp=0),encap(eth_type(0x0800),ipv4(frag=no)), packets:0, bytes:0, used:never, acti
ons:ct(commit),pop_vlan,3
```

ovs-appctl dpctl/dump-flows type=offloaded

```
Every 1.0s: ovs-appctl dpctl/dump-flows type=offloaded                  Mon Nov  2 11:20:13 2020

recirc_id(0x8),in_port(3),eth(src=06:6a:aa:09:7d:2b,dst=86:fa:b7:67:a0:e3),eth_type(0x0800),ipv4(frag=n
o), packets:92280, bytes:6461388, used:0.690s, actions:push_vlan(vid=100,pcp=0),1
recirc_id(0x7),in_port(1),eth(src=86:fa:b7:67:a0:e3,dst=06:6a:aa:09:7d:2b),eth_type(0x8100),vlan(vid=10
0,pcp=0),encap(eth_type(0x0800),ipv4(frag=no)), packets:9900768, bytes:149897180 34, used:0.690s, action
s:pop_vlan,3
```

NOTE

CT stateful offloading requires the support of the complete TC module of kernel 5.7. Kernel 4.14 of CentOS 7.6 does not fully support this function.

----End

VXLAN Offloading

Step 1 Configure VFs.

For details, see **4.4.2 Configuring Kernel-Mode SR-IOV**.

Step 2 Configure the network.

```
systemctl start openvswitch
ovs-vsctl add-br br-ovs
ovs-vsctl add-port br-ovs enp1s0f0_0
ovs-vsctl add-port br-ovs enp1s0f0_1
ovs-vsctl add-port br-ovs enp1s0f0_2
ovs-vsctl add-port br-ovs enp1s0f0_3
ip link set dev enp1s0f0 up
ip link set dev enp1s0f0_0 up
ip link set dev enp1s0f0_1 up
ip link set dev enp1s0f0_2 up
ip link set dev enp1s0f0_3 up
ovs-vsctl add-port br-ovs vxlan0 -- set Interface vxlan0 type=vxlan options:local_ip=192.168.1.12
options:remote_ip=192.168.1.11 options:key=98
ifconfig enp1s0f0 192.168.1.12/24 up
```


> **NOTE**
>
> When configuring the VXLAN port on the peer server, change **local_ip** and **remote_ip** to each other.

Step 3 Start the VM and log in to it.

```
virsh start vm1
virsh console vm1
```

Step 4 Set the maximum transmission unit (MTU) of the two VMs to a value not greater than 1450.

```
ifconfig <dev> mtu 1450
```

Step 5 Use the two VMs to send traffic.

- Run the following command on VM 1 of server 2:
  ```
  iperf3 -s
  ```
- Run the following command on VM 1 of server 1:
  ```
  iperf3 -c <Host2vm1_ip> -t 0
  ```

Step 6 Verify the flow table offloading on the physical machine.

```
ovs-appctl dpctl/dump-flows
```

```
Every 1.0s: ovs-appctl dpctl/dump-flows                                  Mon Nov  2 11:37:35 2020

tunnel(tun_id=0x62,src=192.168.1.12,dst=192.168.1.11,tp_dst=4789,flags(+key)),recirc_id(0),in_port(7),e
th(src=86:fa:b7:67:a0:e3,dst=06:6a:aa:09:7d:2b),eth_type(0x0800),ipv4(frag=no), packets:47382536, bytes
:69367908948, used:0.450s, actions:3
tunnel(tun_id=0x62,src=192.168.1.12,dst=192.168.1.11,tp_dst=4789,flags(+key)),recirc_id(0),in_port(7),e
th(src=86:fa:b7:67:a0:e3,dst=06:6a:aa:09:7d:2b),eth_type(0x0806), packets:0, bytes:0, used:20.220s, act
ions:3
recirc_id(0),in_port(3),eth(src=06:6a:aa:09:7d:2b,dst=86:fa:b7:67:a0:e3),eth_type(0x0806), packets:0, b
ytes:0, used:20.220s, actions:set(tunnel(tun_id=0x62,src=192.168.1.11,dst=192.168.1.12,tp_dst=4789,flag
s(key))),7
recirc_id(0),in_port(3),eth(src=06:6a:aa:09:7d:2b,dst=86:fa:b7:67:a0:e3),eth_type(0x0800),ipv4(tos=0/0x
3,frag=no), packets:1629591, bytes:18903678, used:0.450s, actions:set(tunnel(tun_id=0x62,src=192.168.1
.11,dst=192.168.1.12,tp_dst=4789,flags(key))),7
```

```
Every 1.0s: ovs-appctl dpctl/dump-flows type=offloaded                   Mon Nov  2 11:38:03 2020

tunnel(tun_id=0x62,src=192.168.1.12,dst=192.168.1.11,tp_dst=4789,flags(+key)),recirc_id(0),in_port(7),e
th(src=86:fa:b7:67:a0:e3,dst=06:6a:aa:09:7d:2b),eth_type(0x0800),ipv4(frag=no), packets:70136931, bytes
:102680343228, used:0.360s, actions:3
tunnel(tun_id=0x62,src=192.168.1.12,dst=192.168.1.11,tp_dst=4789,flags(+key)),recirc_id(0),in_port(7),e
th(src=86:fa:b7:67:a0:e3,dst=06:6a:aa:09:7d:2b),eth_type(0x0806), packets:0, bytes:0, used:9.740s, acti
ons:3
recirc_id(0),in_port(3),eth(src=06:6a:aa:09:7d:2b,dst=86:fa:b7:67:a0:e3),eth_type(0x0806), packets:0, b
ytes:0, used:9.740s, actions:set(tunnel(tun_id=0x62,src=192.168.1.11,dst=192.168.1.12,tp_dst=4789,flags
(key))),7
recirc_id(0),in_port(3),eth(src=06:6a:aa:09:7d:2b,dst=86:fa:b7:67:a0:e3),eth_type(0x0800),ipv4(tos=0/0x
3,frag=no), packets:2413875, bytes:280011534, used:0.360s, actions:set(tunnel(tun_id=0x62,src=192.168.1
.11,dst=192.168.1.12,tp_dst=4789,flags(key))),7
```

Step 7 Verify the CT configuration.

1. Configure the CT flow table on the physical machine.
   ```
   ovs-ofctl del-flows br-ovs
   ovs-ofctl add-flow br-ovs "arp, actions=normal"
   ovs-ofctl add-flow br-ovs "table=0, ip,ct_state=-trk, actions=ct(table=1)"
   ovs-ofctl add-flow br-ovs "table=1, ip,ct_state=+trk+new, actions=ct(commit),normal"
   ovs-ofctl add-flow br-ovs "table=1, ip,ct_state=+trk+est, actions=normal"
   ```



```
Every 1.0s: ovs-appctl dpctl/dump-flows                                  Mon Nov  2 11:48:57 2020

tunnel(tun_id=0x62,src=192.168.1.12,dst=192.168.1.11,tp_dst=4789,flags(+key)),recirc_id(0xb),in_port(7)
,eth(src=86:fa:b7:67:a0:e3,dst=06:6a:aa:09:7d:2b),eth_type(0x0800),ipv4(frag=no), packets:1593089, byte
s:2332238161, used:0.461s, actions:3
recirc_id(0xc),in_port(3),eth(src=06:6a:aa:09:7d:2b,dst=86:fa:b7:67:a0:e3),eth_type(0x0800),ipv4(tos=0/
0x3,frag=no), packets:54795, bytes:6356388, used:0.460s, actions:set(tunnel(tun_id=0x62,src=192.168.1.1
1,dst=192.168.1.12,tp_dst=4789,flags(key))),7
recirc_id(0xb),tunnel(tun_id=0x62,src=192.168.1.12,dst=192.168.1.11,flags(-df-csum+key)),in_port(7),ct_
state(+new+trk),eth(src=86:fa:b7:67:a0:e3,dst=06:6a:aa:09:7d:2b),eth_type(0x0800),ipv4(frag=no), packet
s:0, bytes:0, used:never, actions:ct(commit),3
recirc_id(0),tunnel(tun_id=0x62,src=192.168.1.12,dst=192.168.1.11,flags(-df-csum+key)),in_port(7),ct_st
ate(-trk),eth(),eth_type(0x0800),ipv4(frag=no), packets:2, bytes:169, used:2.531s, flags:P., actions:ct
,recirc(0xb)
recirc_id(0),in_port(3),ct_state(-trk),eth(),eth_type(0x0800),ipv4(frag=no), packets:0, bytes:0, used:n
ever, actions:ct,recirc(0xc)
```

2. Verify the flow table offloading.

```
ovs-appctl dpctl/dump-flows type=offloaded
```

```
Every 1.0s: ovs-appctl dpctl/dump-flows type=offloaded                   Mon Nov  2 11:51:23 2020

tunnel(tun_id=0x62,src=192.168.1.12,dst=192.168.1.11,tp_dst=4789,flags(+key)),recirc_id(0xd),in_port(7)
,eth(src=86:fa:b7:67:a0:e3,dst=06:6a:aa:09:7d:2b),eth_type(0x0800),ipv4(frag=no), packets:5008062, byte
s:7331736393, used:0.390s, actions:3
recirc_id(0xe),in_port(3),eth(src=06:6a:aa:09:7d:2b,dst=86:fa:b7:67:a0:e3),eth_type(0x0800),ipv4(tos=0/
0x3,frag=no), packets:178704, bytes:20729920, used:0.390s, actions:set(tunnel(tun_id=0x62,src=192.168.1
.11,dst=192.168.1.12,tp_dst=4789,flags(key))),7
```

NOTE

CT stateful offloading requires the support of the complete TC module of kernel 5.7. Kernel 4.14 of CentOS 7.6 does not fully support this function.

----End

Geneve Offloading

Step 1 Configure VFs.

For details, see **4.4.2 Configuring Kernel-Mode SR-IOV**.

Step 2 Configure the network.

```
systemctl start openvswitch
ovs-vsctl add-br br-ovs
ovs-vsctl add-port br-ovs enp1s0f0_0
ovs-vsctl add-port br-ovs enp1s0f0_1
ovs-vsctl add-port br-ovs enp1s0f0_2
ovs-vsctl add-port br-ovs enp1s0f0_3
ip link set dev enp1s0f0 up
ip link set dev enp1s0f0_0 up
ip link set dev enp1s0f0_1 up
ip link set dev enp1s0f0_2 up
ip link set dev enp1s0f0_3 up
ovs-vsctl add-port br-ovs tun0 -- set Interface tun0 type=geneve options:local_ip=192.168.1.12
options:remote_ip=192.168.1.11
ifconfig enp1s0f0 192.168.1.12/24 up
```

NOTE

When configuring the VXLAN port on the peer server, change **local_ip** and **remote_ip** to each other.

Step 3 Start the VM and log in to it.

```
virsh start vm1
virsh console vm1
```

Step 4 Set the MTU of the two VMs to a value not greater than 1450.

```
ifconfig <dev> mtu 1450
```


Step 5 Use the two VMs to send traffic.

- Run the following command on VM 1 of server 2:
  ```
  iperf3 -s
  ```
- Run the following command on VM 1 of server 1:
  ```
  iperf3 -c <Host2vm1_ip> -t 0
  ```

Step 6 Verify the flow table offloading on the physical machine.

```
ovs-appctl dpctl/dump-flows
```

```
Every 1.0s: ovs-appctl dpctl/dump-flows                                  Mon Nov  2 14:30:30 2020

tunnel(tun_id=0x0,src=192.168.1.12,dst=192.168.1.11,tp_dst=6081,flags(+key)),recirc_id(0),in_port(1),et
h(src=86:fa:b7:67:a0:e3,dst=06:6a:aa:09:7d:2b),eth_type(0x0800),ipv4(frag=no), packets:3397201, bytes:4
755477850, used:0.000s, actions:3
recirc_id(0),in_port(3),eth(src=06:6a:aa:09:7d:2b,dst=86:fa:b7:67:a0:e3),eth_type(0x0800),ipv4(tos=0/0x
3,frag=no), packets:40799, bytes:2695014, used:0.000s, flags:SP., actions:set(tunnel(src=192.168.1.11,d
st=192.168.1.12,ttl=64,tp_dst=6081,flags(df))),1
```

```
Every 1.0s: ovs-appctl dpctl/dump-flows type=offloaded                   Mon Nov  2 14:31:22 2020

tunnel(tun_id=0x0,src=192.168.1.12,dst=192.168.1.11,tp_dst=6081,flags(+key)),recirc_id(0),in_port(1),et
h(src=86:fa:b7:67:a0:e3,dst=06:6a:aa:09:7d:2b),eth_type(0x0800),ipv4(frag=no), packets:21386546, bytes:
29937304652, used:0.000s, actions:3
tunnel(tun_id=0x0,src=192.168.1.12,dst=192.168.1.11,tp_dst=6081,flags(+key)),recirc_id(0),in_port(1),et
h(src=86:fa:b7:67:a0:e3,dst=06:6a:aa:09:7d:2b),eth_type(0x0806), packets:0, bytes:0, used:16.130s, acti
ons:3
```

NOTE

Geneve protocol offloading requires that the kernel version be 5.3 or later.

Step 7 Verify the CT configuration.

1. Configure the CT flow table on the physical machine.
   ```
   ovs-ofctl del-flows br-ovs
   ovs-ofctl add-flow br-ovs "arp, actions=normal"
   ovs-ofctl add-flow br-ovs "table=0, ip,ct_state=-trk, actions=ct(table=1)"
   ovs-ofctl add-flow br-ovs "table=1, ip,ct_state=+trk+new, actions=ct(commit),normal"
   ovs-ofctl add-flow br-ovs "table=1, ip,ct_state=+trk+est, actions=normal"
   ```

   ```
   Every 1.0s: ovs-appctl dpctl/dump-flows                                  Mon Nov  2 14:33:12 2020

   tunnel(tun_id=0x0,src=192.168.1.12,dst=192.168.1.11,tp_dst=6081,flags(+key)),recirc_id(0x10),in_port(1)
   ,eth(src=86:fa:b7:67:a0:e3,dst=06:6a:aa:09:7d:2b),eth_type(0x0800),ipv4(frag=no), packets:0, bytes:0, u
   sed:2.710s, actions:3
   recirc_id(0),in_port(3),ct_state(-trk),eth(),eth_type(0x0800),ipv4(frag=no), packets:5291, bytes:352246
   , used:0.000s, flags:SP., actions:ct,recirc(0x11)
   recirc_id(0),tunnel(tun_id=0x0,src=192.168.1.12,dst=192.168.1.11,flags(-df-csum+key)),in_port(1),ct_sta
   te(-trk),eth(),eth_type(0x0800),ipv4(frag=no), packets:2617, bytes:1437936, used:0.000s, flags:SP., ac
   tions:ct,recirc(0x10)
   recirc_id(0x10),tunnel(tun_id=0x0,src=192.168.1.12,dst=192.168.1.11,flags(-df-csum+key)),in_port(1),ct_
   state(+new+trk),eth(src=86:fa:b7:67:a0:e3,dst=06:6a:aa:09:7d:2b),eth_type(0x0800),ipv4(frag=no), packet
   s:1, bytes:74, used:2.660s, flags:S, actions:ct(commit),3
   recirc_id(0x11),in_port(3),ct_state(-new+est+trk),eth(src=06:6a:aa:09:7d:2b,dst=86:fa:b7:67:a0:e3),eth_
   type(0x0800),ipv4(tos=0/0x3,frag=no), packets:5291, bytes:352246, used:0.000s, flags:SP., actions:set(t
   unnel(src=192.168.1.11,dst=192.168.1.12,ttl=64,tp_dst=6081,flags(df))),1
   ```

2. Verify the flow table offloading.
   ```
   ovs-appctl dpctl/dump-flows type=offloaded
   ```

   ```
   Every 1.0s: ovs-appctl dpctl/dump-flows type=offloaded                   Mon Nov  2 14:33:56 2020

   tunnel(tun_id=0x0,src=192.168.1.12,dst=192.168.1.11,tp_dst=6081,flags(+key)),recirc_id(0),in_port(1),et
   h(src=86:fa:b7:67:a0:e3,dst=06:6a:aa:09:7d:2b),eth_type(0x0806), packets:0, bytes:0, used:12.920s, acti
   ons:3
   tunnel(tun_id=0x0,src=192.168.1.12,dst=192.168.1.11,tp_dst=6081,flags(+key)),recirc_id(0x10),in_port(1)
   ,eth(src=86:fa:b7:67:a0:e3,dst=06:6a:aa:09:7d:2b),eth_type(0x0800),ipv4(frag=no), packets:0, bytes:0, u
   sed:16.310s, actions:3
   ```


> **NOTE**
>
> CT stateful offloading requires the support of the complete TC module of kernel 5.7. Kernel 4.14 of CentOS 7.6 does not fully support this function. After flow table offloading on the hardware, the network connection becomes unidirectional.

----End

IPv6 Offloading

Step 1 Configure VFs.

For details, see **4.4.2 Configuring Kernel-Mode SR-IOV**.

Step 2 Configure the network.

```
systemctl start openvswitch
ovs-vsctl add-br br-ovs
ovs-vsctl add-port br-ovs enp1s0f0_0
ovs-vsctl add-port br-ovs enp1s0f0_1
ovs-vsctl add-port br-ovs enp1s0f0_2
ovs-vsctl add-port br-ovs enp1s0f0_3
ovs-vsctl add-port br-ovs enp1s0f0
ip link set dev enp1s0f0 up
ip link set dev enp1s0f0_0 up
ip link set dev enp1s0f0_1 up
ip link set dev enp1s0f0_2 up
ip link set dev enp1s0f0_3 up
```

Step 3 Start the VM and log in to it.

```
virsh start vm1
virsh console vm1
```

Step 4 Add IPv6 addresses to the VMs.

- Run the following command on VM 1 of server 1:
  ```
  ifconfig enp1s0 add 3000:1::11/64
  ```
- Run the following command on VM 1 of server 2:
  ```
  ifconfig enp1s0 add 3000:1::12/64
  ```

Step 5 Use the two VMs to send traffic.

- Run the following command on VM 1 of server 2 (receiver):
  ```
  iperf3 -6 -s
  ```
- Run the following command on VM 1 of server 1 (sender):
  ```
  iperf3 -6 -c 3000:1::12 -u -l 512 -t 999
  ```

Step 6 Check the flow table offloading on the physical machine.

```
ovs-appctl dpctl/dump-flows type=offloaded
```

```
Every 1.0s: ovs-appctl dpctl/dump-flows type=offloaded                  Mon Nov  2 15:18:34 2020

recirc_id(0),in_port(6),eth(src=86:fa:b7:67:a0:e3,dst=06:6a:aa:09:7d:2b),eth_type(0x86dd),ipv6(frag=no)
, packets:15948551, bytes:24146011216, used:0.810s, actions:2
recirc_id(0),in_port(2),eth(src=06:6a:aa:09:7d:2b,dst=86:fa:b7:67:a0:e3),eth_type(0x86dd),ipv6(frag=no)
, packets:581767, bytes:50032382, used:0.810s, actions:6
```

----End

A Change History

Date	Description
2021-03-23	This issue is the sixth official release. Changed the solution name from "Kunpeng virtualization solution" to "Kunpeng BoostKit for Virtualization".
2020-12-30	This issue is the fifth official release. Added the ***4 SR-IOV User Guide***.
2020-11-30	This issue is the fourth official release. Added the ***3 XPF User Guide***.
2020-09-28	This issue is the third official release. Added the ***2 Kube-OVN User Guide***.
2020-09-21	This issue is the second official release. Changed the solution name from "Kunpeng cloud platform solution" to "Kunpeng virtualization solution".
2020-06-24	This issue is the first official release.

