OST
Eastern Switzerland
University of Applied Sciences

K8s L2 CNI for Containers and VMs

Semester Assignment

Department of Computer Science
OST – Eastern Switzerland University of Applied Sciences
Campus Rapperswil-Jona

Spring Term 2023

Authors

Michael Brändli & Leandro Ceriani

Supervision

Urs Baumann

Co-Supervision

Yannick Zwicker

05.06.2023

Abstract

This paper investigates how layer 2 networking inside Kubernetes can be realized. Layer 2 networking is vital for some applications and especially useful for simulating real-world networking scenarios in Kubernetes. The INS is providing networking laboratory exercises to the students of the Eastern Switzerland University of Applied Sciences. The underlying platform should be migrated from bare metal Docker to Kubernetes for performance and scalability improvements. This is the main reason why the Institute for Network and Security (INS) is looking into this technology. The INS needs layer 2 point-to-point connections. This means protocols like Link Layer Discovery Protocol (LLDP) and layer 3 multicast applications like Hot Standby Router Protocol (HSRP) should work inside Kubernetes without being filtered or blocked. This matter gets further complicated by the fact that not all workloads can be containerized. Some workloads need to be run as virtual machines. These virtual machines can be managed and run inside Kubernetes using KubeVirt.

Pods in Kubernetes use Container Network Interface (CNI) plugins to communicate with each other and external networks. Currently, no single CNI plugin can handle unfiltered layer 2 point-to-point connections between pods and KubeVirt virtual machines. Additionally, connections must be possible over multiple physical Kubernetes nodes.

This paper tests and analyzes possible methods of realizing layer 2 connectivity. There are two main challenges with establishing layer 2 connectivity. The first is getting layer 2 inter-node traffic inside Kubernetes working with a CNI. This involved testing multiple open-source CNIs that claim to provide layer 2 connectivity and breaking down how they work. The second challenge is connecting the Kubevirt VMs to the Kubernetes CNI without losing packages.

There are many usable approaches to getting pod-to-pod layer 2 connections working. The most successful way was to use a CNI called Kube-OVN. Kube-OVN leverages Open Virtual Network (OVN), which is based on an Open Virtual Switch (OVS), to allow for proper networking configuration inside Kubernetes. With many workarounds, getting all layer 2 traffic from a KubeVirt VM to a pod is possible. However, it is not flawless, as during proof of concept 1 each interface can only handle one static MAC Address. This limitation makes

using HSRP impossible. There are efforts from the community to make the virtual machines integrate better into an existing CNI. However, the macvtap binding mode, which the KubeVirt community is working on, did not succeed during testing.

It was shown that pod-to-pod connectivity could be realized on layer 2, but additional research is required to allow VM-to-pod connectivity in Kubernetes.

Keywords: K8s, Kubernetes, KubeVirt, MeshNet CNI, NetworkServiceMesh CNI, Megalos CNI, Kube-OVN CNI, macvtap-cni, veth pair.

Executive Summary

Initial situation

Containerization of applications has gained more and more growth in recent years and has become a standard in software development today. The container orchestration platform Kubernetes has therefore not without reason gained great popularity. It is therefore more understandable that companies and enterprises want to rely on this technology. The Institute for Network and Security (INS) offers students at the Eastern Switzerland University of Applied Sciences many laboratory exercises in network and security modules. The fundamentals of this infrastructure should now be migrated to Kubernetes. Individual architecture components only exist as virtual machines and not as containers as of today. The software KubeVirt enables the deployment of virtual machines in Kubernetes. Initial tests looked promising. However, it seems that while containers and virtual machines can work together, the communication between them can pose challenges when setting up a laboratory environment in the network field as required by the INS. For a standard enterprise operation, this works well. However, if a pure layer 2 is needed or even layer one connection between containers and virtual machines, it is not so easily implementable. Even more, not with the intent to have it in a multi-host setup running.

Approach

The goal was to show one or more proof of concepts of how the implementation of a layer 2 connection between virtual machines and containers could be possible, ideally, with the possibility of running this on multiple hosts. The analysis of various existing CNIs (container network interfaces) serves as the basis for the proof of concepts. These were checked for their function and it was tested how much the individual CNIs can contribute to an overall solution. Individual approaches and the technologies used by the CNIs were analyzed and documented.

Results

With the proof of concept 1, it was shown that using Kube-OVN and macvtap as CNI and veth pair as virtual network cables, a layer 2 connection between virtual machines and containers can be established even on a multi-host setup. This setup still had some technological flaws, which made it impossible to run certain networking scenarios. The proof of concept 1 is quite extensive in terms of the manual effort required to set up a lab environment. With proof of concept 2, the goal was to minimize this effort by eliminating a CNI and using veth pairs as virtual network cables. For this, a code improvement from the developer kvaps has been in the pipeline of KubeVirt for over a year but has yet to be integrated. Unfortunately, after dedicating a significant amount of time, the desired goal to implement this with the code improvement from the developer kvaps was not possible in the given time constraints of the semester assignment.

Further work

Since only some proof of concepts was worked out in the semester assignment, not all requirements by the INS will certainly be covered, and further improvements can be implemented. Furthermore, a solution can still be found at the proof of concept 2 in cooperation with the developer kvaps.

Overall, a dedicated CNI implementation for the INS promises to be the best approach. What this custom CNI would have to fulfill and which technologies should be used by the existing CNIs is described in the chapter "Future Steps" in the technical documentation.

Acknowledgment

We would like to thank the following people for helping with this semester assignment:

- **Urs Baumann** for the guidance and supervision during the course of this semester assignment.
- **Yannick Zwicker** for the guidance and supervision during the course of this semester assignment.
- **Andy Pfister** for proofreading the semester assignment.

Contents

Chapter 1

Introduction

The Institute for Network and Security (INS) supports and supervises many courses of the Eastern Switzerland University of Applied Sciences (OST). The students have the possibility to use the lab environment (called LTB) from the INS for the exercise and lab sessions. The exercises mainly teach configurations on standard network components, such as routers and switches, which are available in a virtual environment.

The software stack of LTB is currently running on docker containers and virtual machines hosted with KVM. To continue to explore and follow new technologies, INS would like to move the LTB software stack to a Kubernetes environment.

Initial attempts to move the lab environment to Kubernetes have shown that this is possible on a superficial level but still presents obstacles for individual use cases in the exercises. The current Kubernetes network implementations filter out some data packages, which prevent specific standard networking components, needed in the exercise sessions, from working correctly.

1.1 Assignment

The LTB software stack should be migrated to Kubernetes with the requirement that containers and virtual machines can communicate on layer 2.

This semester assignment consists of a large portion of the analysis of some CNI plugins as well as the KubeVirt networking. Listed, the work of the semester assignment contains:

- (Reverse)analysis KubeVirt networking
- (Reverse)analysis MeshNet CNI
- (Reverse)analysis NetworkServiceMesh CNI
- (Reverse)analysis Megalos-CNI
- Documentation of the (Reverse)analysis
- Creating flow graphs/diagrams of the network flows of the CNIs
- Proof of concept for layer 2 connectivity with KubeVirt

1.2 Motivation

Until today, the LTB software stack is running by INS in an environment based on bare metal Docker and some virtual machines hosted with KVM. This solution involves many manual steps to deploy the labs for students, which in turn affects scalability.

Kubernetes is no longer a technology trend and has proven itself over the past few years. Many software vendors are building applications that can only run as containers. The INS would like to update its lab exercises to Kubernetes to have a state-of-the-art environment.

This chance should also be used to clean up the code base around the topic LTB. With the current solution, if a server host encounters an error, all the labs running on it are broken since many manual steps must be taken to get them running again. Kubernetes solves this problem because each pod should have a particular desired state which is always tried to reach.

Far on the horizon is the goal to eventually release the LTB software stack as an open-source solution for lab environments.

1.3 Functional requirements

The proof of concept should demonstrate the tool's / CNI plugin's ability to achieve the following requirements:

- Peer-to-peer communication via multiple nodes
- Router & switch functions like STP, LLDP and other layer 2 frames
- There will be no MAC address learning done on the underlying network link between pods.
- There should be a possibility to do packet captures from the traffic between the pods / virtual machines.
- All the requirements above should be possible in the following compositions:
  - Container <-> Container
  - KubeVirt VM <-> KubeVirt VM
  - Container <-> KubeVirt VM

1.4 General Conditions

This work was conducted as part of a semester assignment (Studienarbeit). A time budget of 480 hours (2x 240h) is reserved for the work on this assignment. It will be rewarded with eight ECTS credits.

1.5 Preliminary work

This is a semester assignment that has no preliminary work from L. Ceriani and M. Brändli or other students. All work is done during the given amount of working hours described in the general conditions.

Some employees from the INS have made some first attempts to move the LTB stack to Kubernetes, but this has presented some obstacles for individual use cases.

1.6 Technical Documentation

Besides this documentation, which contains more project-related content, there is a technical documentation which is available here. The technical documentation is kept as MkDocs website because it is easier to share it with other universities and colleges afterward. The technical documentation contains more specific technical information, configuration details and debugs compared to this document.

The technical documentation is available at http://ins-stud.pages.gitlab.ost.ch/sa-ba/sa-fs23-k8s-l2-cni/sa-fs23-k8s-l2-cni-techdoc/. If you copy the repository, make sure to install all dependencies first:

```
// copy repository
git clone https://gitlab.ost.ch/ins-stud/sa-ba/sa-fs23-k8s-l2-cni/sa-fs23- ↵
k8s-l2-cni-techdoc.git

// install dependencies
pip install -r requirements.txt

// run MkDocs local
mkdocs serve
```

1.7 Structure of the Thesis

The semester assignment is organized as follows:

Chapter 2 – Problem Analysis The problem analysis chapter briefly describes the challenges with moving the current LTB stack to a Kubernetes cluster. It also describes a possible workaround that addresses two of the three difficulties but has a different limitation.

Chapter 3 – Research The research chapter contains the elaboration of different software that could be considered as a solution or partial solution to the problem in the semester assignment. In the beginning, the setup of the lab environment is briefly explained. Afterward, the findings of the different software are described.

Chapter 4 – Solution In the solution chapter, solutions or approaches are presented and described. The solutions consist of the software tested in the research chapter, which has been identified as suitable. The solutions can be either a complete software covering all aspects or a mixture of several small software fulfilling the specifications together.

Chapter 5 – Results In this chapter, the outcomes of the semester assignment will be examined, including both successes and shortcomings. To achieve this, a closer look will be taken at the use cases that were previously outlined, with a reflection on what was accomplished and what fell short.

Chapter 6 – Conclusion In the conclusion chapter, the entire semester assignment is reviewed, and it is concluded what went well and what did not. Open points are addressed, and in general, a conclusion is drawn.

Chapter 2

Problem Analysis

2.1 System context

Kubernetes is an open-source container orchestration engine for automating deployments, scaling, and management of containerized applications [1]. It is not designed out of the box to run networking equipment or labs. To run network components as pods, proper layer 2 networks (OSI model) between the pods are required. The two most significant challenges with layer 2 networking in Kubernetes are detailed below.

2.2 Current challenges

The obstacles from the initial attempt to move the LTB stack to Kubernetes are described in the following sub-sections.

Please be aware that further details of the challenges can be found in the technical documentation!

2.2.1 Linux bridge

The MAC address range "**01-80-C2-00-00-0x**" is defined as the "MAC Bridge Filtered MAC Group Addresses" by the IEEE standard 802.1D. This range is set aside by the IEEE for standard protocols that use link local multicast to communicate with a neighboring device. Frames using these MAC addresses are supposed to be explicitly link local.

MAC address	Protocol
01-80-C2-00-00-00	Spanning Tree (STP/RSPT/MSTP)
01-80-C2-00-00-01	Ethernet Flow Control (pause frames)
01-80-C2-00-00-02	Link Aggregation Control Protocol (LACP)
01-80-C2-00-00-03	802.1X Port-Based Network Access Control
01-80-C2-00-00-08	Provider Bridge protocols (STP)
01-80-C2-00-00-0D	Provider Bridge protocols (MVRP)
01-80-C2-00-00-0E	802.1AB Link Layer Discovery Protocol (LLDP)

Table 2.1: IEEE 802.1D MAC Bridge Filtered MAC Group Addresses [2]

Multicast filtering - Linux bridge challenge 1

The first challenge is that the Container Network Interface (CNI) plugins filter specific layer 2 traffic like Spanning Tree (STP) Packages, Link Layer Discovery Protocol (LLDP), and more because KubeVirt is using a Linux bridge to transfer the traffic. This uncertainty that all types of sent packages will arrive at the destination makes a Kubernetes deployment impractical for networking labs. Each pod does only need peer-to-peer connections to other pods. If packages like STP or LLDP get lost during transportation, the routers can not be fully used for a networking lab environment. The Kubernetes CNI handles the peer-to-peer connections. Some CNIs claim to allow layer 2 peer-to-peer connections in Kubernetes. Analyzing and verifying these functionalities will be covered in the next Section.

MAC address learning - Linux bridge challenge 2

The second challenge has arisen when using KubeVirt. KubeVirt allows the creation of Virtual machines in Kubernetes [3]. To allow this functionality, KubeVirt adds a Linux bridge (br1, as seen in the KubeVirt network stack diagram 2.1) into the network flow. A Linux bridge does MAC address learning, which is fine usually but can be challenging for specific network applications. For example, a Cisco router was deployed as a VM and the communication between two attached clients was observed. Port mirroring can be set up in a way that all traffic is sent out on a third interface. If the Linux bridge, which serves as underlay for all virtual interfaces, does MAC learning, the packets will be sent back to the original destinations.

Linux Bridge potential work around

This behavior of the IEEE 802.1D complainant bridge can be worked around by setting a bitmask in **/sys/class/net/br0/bridge/group_fwd_mask**.

- for a patched kernel all bits can be set: 1111 1111 1111 1111 = 65535

- for an unpatched kernel the first three bits can not be set: 1111 1111 1111 1000 = 65528

If a kernel is not patched, the first (least significant) three protocols will be filtered out. To patch this, the kernel needs to be recompiled.

- 01-80-C2-00-00-**00**: Spanning Tree (STP/RSPT/MSTP)
- 01-80-C2-00-00-**01**: Ethernet Flow Control (pause frames)
- 01-80-C2-00-00-**02**: Link Aggregation Control Protocol (LACP)

In other words: to enable out-of-the-box Linux bridges to forward all IEEE 802.1D MAC bridge filtered MAC group addresses except the restricted three types, this command needs to be executed:

```
1 echo 65528 > /sys/class/net/br0/bridge/group_fwd_mask
```

Note: It is not recommended to run this in a production environment. [2]

2.2.2 VM NIC injection - KubeVirt challenge

To start a KubeVirt compute container, the interface to be used must be present at the time of deployment. If it is not, the compute container will not start. Most CNIs inject the additional network interface after the start and therefore, the compute containers do not launch.

2.3 Known potential workaround for Linux bridge challenges

The INS tested a hacky workaround for the problems with the Linux bridge described above. It is illustrated in the diagram 2.1.

- The Linux bridge can be replaced manually with an xdp-xconnect to cross-connect linux interfaces. This prevents MAC learning.
- The Open vSwitch (OVS) CNI is installed as a second CNI using Multus. In OVS, it is possible to manually create direct unfiltered peer-to-peer connections.

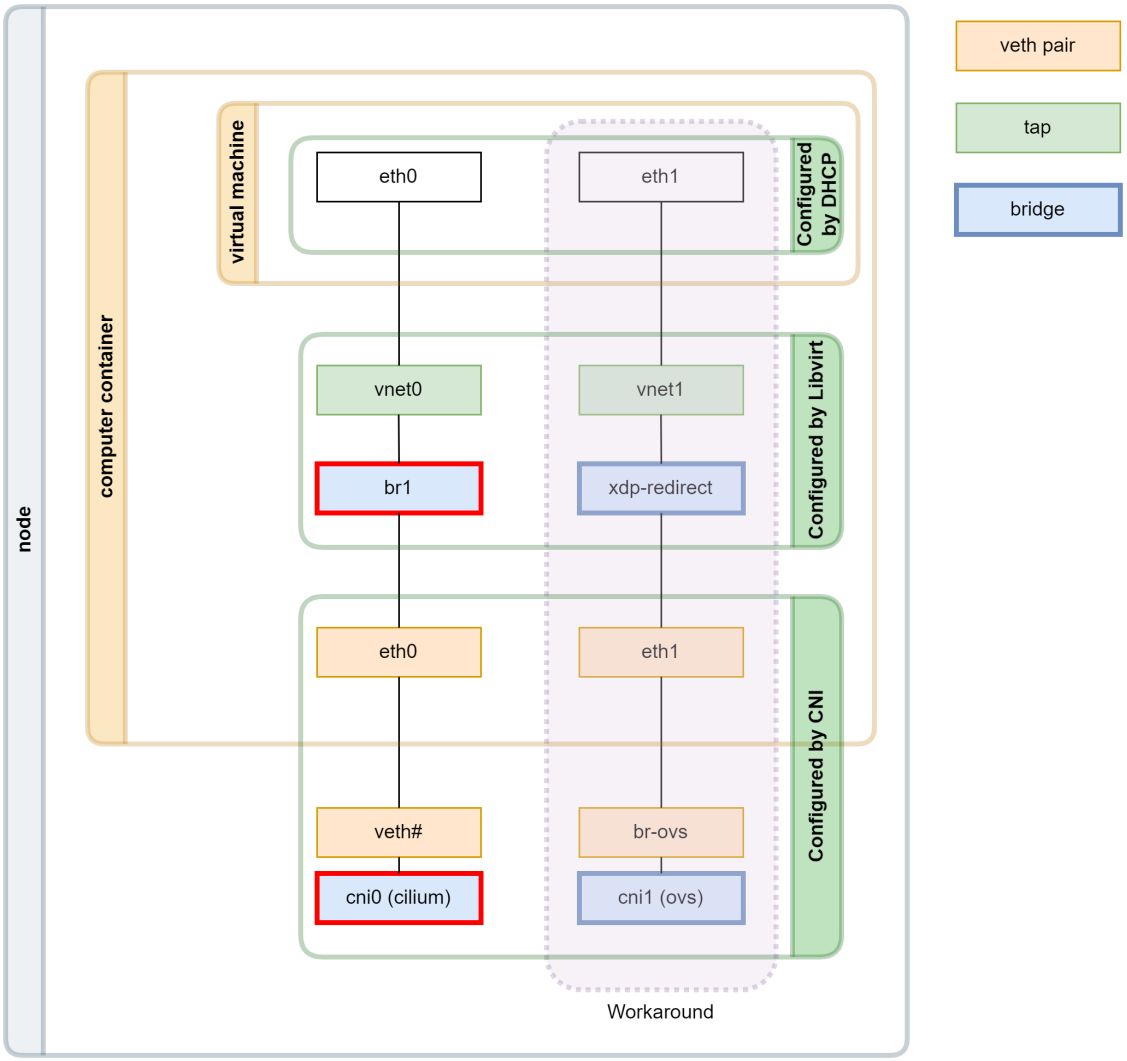

Figure 2.1: KubeVirt Networking setup with workaround [4]

This workaround seems promising. The problem is that the OVN CNI is not multi-node capable and thus, labs could only run on a single host.

2.4 Use cases

The following table lists the use cases which should be fulfilled in this semester assignment from a vision perspective. These use cases will be revisited later to see how they have or have not been achieved and why not.

Nr.	Title	Description
1	Frame filtering	The connection between endpoints should act like a virtual lan cable. This means the "MAC Bridge Filtered MAC Group Addresses" described in 2.2.1 or any other frames should not be filtered.
2	Endpoints	An endpoint can be a VM or a pod. It is possible to create a 1-to-1 connection between a pod to a pod, a VM to a VM, or a pod to a VM. The resulting connection should behave as if two endpoints would be connected with a genuine physical LAN cable.
3	Multinode	An Endpoint can be connected to any other endpoint running on the same or a different Kubernetes node.
4	MAC Learning	No MAC address learning is done on the connection between endpoints. The purpose is to allow port mirroring to work correctly on a virtual switch or router deployed as an endpoint.
5	Subinterfaces	Sending VLAN-tagged traffic to another endpoint is possible.
6	Multiple MAC addresses	It should be possible for any interface to have multiple MAC addresses. This is needed for protocols like HSRP that use a virtual IP with a virtual MAC address.
7	Automation	The solution can be deployed entirely as Kubernetes manifests.

Table 2.2: Use case description

Chapter 3

Research

Please note that more technical details and results from the research phase are available in the technical documentation.

This chapter delves into the existing CNI plugins available. Some CNIs which should be covered were given by the supervisors. However, there is the possibility that CNIs can be considered and analyzed, which is not given in the assignment.

This chapter begins with a description of two Kubernetes providers. A Kubernetes provider is the basic framework for a Kubernetes environment. A Kubernetes environment can be equipped with different CNIs accordingly. Afterward, an overview of different CNIs is given, and a small comparison between the CNIs is made in terms of the size of the CNI based on the number of lines of code and the number of contributors.

3.1 Research compute setup

For this semester assignment, a laboratory environment is needed. The INS provided two physical Fujitsu servers. The technical details of these servers can be found in the technical documentation.

During the first steps of the research phase, it quickly became clear that working efficiently with hardware servers was impossible. If a server gets misconfigured, it takes a relatively long time to reinstall. Therefore a Proxmox VE server was added and it was started to work with virtual machines. This makes it easy to work with snapshots. Proxmox nowadays supports nested virtualization. So virtual machines can be deployed with KubeVirt within the virtual machine itself.

3.2 Research Kubernetes setup

For the research phase, a Kubernetes environment is needed. Nowadays, Kubernetes is available at all major cloud providers like EKS from Amazon AWS or GKE from Google Cloud. Since the current LTB stack solution is also hosted locally by the INS, the Kubernetes environment should also be self-hosted.

One can choose to either install all the necessary components for Kubernetes independently or utilize a provider who simplifies the process and offers the required resources. The following section outlines two Kubernetes providers suitable for a self-hosted Kubernetes environment.

3.2.1 Kubernetes environments

RKE2

RKE2 is a fully conformant Kubernetes distribution that focuses on security and compliance within the U.S. Federal Government sector. It uses Containerd as its container runtime and is built on the Cluster API framework, which affects the way cluster configurations are managed, and changes are made.

For the research phase, the INS provided a Kubernetes cluster consisting of a master and a worker node. This environment is based on RKE2 since the INS already used RKE2 for courses in which Kubernetes is taught.

During the evaluation of the Containernetworking CNIs it stands out that RKE2 uses an old version of the mentioned CNI. Therefore newer plugins which are described in the documentation of the developers were missing. A new solution had to be found, and Kubespray appeared. [5]

Kubespray

Kubespray is a powerful open-source tool that combines Kubernetes and Ansible to deploy and manage Kubernetes clusters. It provides a balance of implementation flexibility and ease of use, making it ideal for managing highly available clusters across multiple platforms. Kubespray can be used to install Kubernetes using Ansible and deploy clusters on cloud computing services like EC2 (AWS) or self-hosted.

Kubespray uses the latest version of the Containernetworking CNIs and provides the ability to use Kube-OVN as a native CNI, which is not documented by RKE2 and seems to be a plausible solution to the assignment. [6]

3.3 Container Network Interfaces (CNIs)

Container network interface plugins are the basis of the Kubernetes networking part. Generally, a standard CNI is deployed in a Kubernetes environment. It is possible to add more CNIs to the environment using Multus, which can add more interfaces to a pod or virtual machine. This use case will come into play as a virtual router or switch in the LTB labs need more than one interface.

3.3.1 Containernetworking CNIs

The Containernetworking CNIs is a collection of different network plugins used by known CNI plugins like Cilium, Calico, Flannel, etc. The following plugins are available in the current version (1.1):

- tap plugin
- vlan plugin
- bridge plugin
- host-device
- ipvlan plugin
- macvlan plugin
- ptp plugin
- win-bridge plugin
- win-overlay plugin

[7]

This collection of plugins is used natively by Kubernetes and is included with all installations. Particularly the macvlan plugin was interesting, as it allows for a pod to be seen on the physical network. This is ideal when running one lab locally. However, this had two main problems in a Kubernetes cluster with many labs. Firstly, all the pods on the same Kubernetes host would see each other. Secondly, it doesn't allow for multi-node communication, as all the traffic just gets dumped on the local network interface and is not forwarded to a particular node and pod.

The general intention would be that there is a "management CNI" like Cilium and a complete additional plugin that establishes the layer 2 connection for the additional interfaces of a pod/virtual machine.

3.3.2 MeshNet CNI

MeshNet CNI is a Kubernetes CNI plugin that creates arbitrary network point-to-point links from topologies. This plugin interconnects pods via direct point-to-point links according to a predefined topology definition. The plugin uses three types of links:

- veth
- vxlan
- macvlan

MeshNet CNI consists of three main components. A datastore stores the topology information and runtime pod metadata (e.g. pod IP addresses and network namespaces). A MeshNet CNI binary is responsible for the network configuration of a pod, and the MeshNet daemon (meshnetd) is responsible for communicating with the Kubernetes cluster for VXLAN or gRPC link configuration updates.

MeshNet CNI can be configured in two modes. VXLAN or gRPC. In VXLAN mode, MeshNet CNI uses a Netlink library to create the VXLAN interface. For the gRPC mode, MeshNet CNI introduces a virtual wire called gRPCwire. A gRPCwire is like a virtual patch cable. A wire map tracks the wires that should and already have been created. To transport the traffic "within the wire", MeshNet CNI captures the traffic on one endpoint, sends it as a gopacket within a gRPC call to the other endpoint, and there each layer within the OSI model will be decoded using a layer feature from gRPC.

MeshNet CNI works great to add more interfaces to a pod, also with a multi-node setup. Unfortunately, it has no support for KubeVirt, and additional interfaces can’t be added to a virtual machine due to the fact that MeshNet CNI works with its own topology configuration files, and a virtual machine definition for KubeVirt needs the network configuration on startup.

3.3.3 Network Service Mesh CNI

The Network Service Mesh CNI offers the possibility of an IP service mesh via multi or hybrid cloud provider. For example, it allows connecting a Kubernetes environment at Google in the GKE with the Kubernetes environment at Amazon EKS. This is a very nice solution if you need it. However, this does not correspond to the solutions requirements, which are set in the semester assignment.

The documentation of Network Service Mesh CNI has been studied to some extent so far. However, NSM CNI will not be worked on further for the semester assignment.

3.3.4 Megalos CNI

The Megalos CNI allows for creating layer 2 LANs across different Kubernetes workers using VXLAN. Megalos CNI is part of the Kathara framework. The Kathara framework allows for easy deployment of container-based interactive network environments, ideal for testing or demo purposes. Megalos CNI can be installed in Kathara to deploy the containers in Kubernetes instead of Docker.

Megalos CNI uses VXLAN as an overlay technology. To handle broadcast traffic, VXLAN typically uses multicast. However, this can be limiting when deployed in a public cloud environment. Therefore, Megalos CNI has engineered a BGP EVPN solution to replace VXLAN MAC learning.

Megalos CNI is best used with Kathara, not the regular Kubernetes YAML files. This makes deploying simple labs quick but might limit future customisability. While testing Megalos CNI, it was impossible to reproduce the multi-node functionality, so it was decided not to investigate this CNI further in this semester assignment. [8] [9]

3.3.5 Kube-OVN CNI

Kube-OVN is a particularly interesting CNI as it uses OVN. OVN is an abstraction layer for OVS, which allows for a more straightforward configuration. Instead of flows, there are now logical routers and switches. This is reflected in the way this CNI can be configured. Creating actual network components as part of the CNI, like virtual private contexts (VPCs), subnets, and gateways, is possible. Kube-OVN is a big CNI with many functionalities beyond providing layer 2 connectivity, which leads to a significant codebase. Kube-OVN ships with some additional comfort features, like a Grafana dashboard for monitoring performance and availability. Some of the documentation is unfortunately only available in Chinese. [10]

Kube-OVN can fulfill many of the technical requirements. It is possible to create multi-node pod-to-pod connections on layer 2. It is also possible to create multiple different layer 2 connections. This truly allows for the creation of multiple labs in a Kubernetes cluster. Only the connection to KubeVirt VMs on layer 2 is, at the time of writing, not natively supported due to the kubevirt-launcher bridge. However, KubeVirt is mentioned in the Kube-OVN documentation regarding handling static IP addresses. [11]

As a proof of concept, manually injecting KubeVirt VM interface on Layer 2 into a Kube-OVN Subnet is possible. This makes KubeVirt-to-Pod or Kubevirt-to-Kubevirt Multinode layer 2 connections possible. The details of this workaround is described in the TechDoc at: /analysis/kube-ovn/#lets-try-to-add-a-veth-to-a-subnet

3.3.6 macvtap-CNI

The macvtap-CNI is part of KubeVirt and can expose a KubeVirt VM to the physical network as a layer 2 device. It is a very small CNI that is vital in making layer 2 connections to KubeVirt possible. [12]

In the beginning, the macvtap-CNI did not deliver the expected result. Ping was working, but LLDP was not, but it should have worked according to the documentation and the actual configuration of the network interfaces. To troubleshoot this behavior, a tcpdump was started to see all packages flowing through the network interfaces in the packet flow. As soon as that tcpdump was started, the LLDP discovery worked. This was very odd! After further investigation and a chat with the community of the macvtap-CNI, the missing setting was found. The interfaces needed to be set into Promiscuous mode. This mode makes the interface pass through all the traffic it receives [13]. The leading macvtap-CNI developer added a knob to enable this functionality immediately. [14]

HSRP problems with macvtap

Not all problems with the macvtap-CNI were solved. The packets were again not forwarded correctly when attempting to run routers using the hot standby readiness protocol (HSRP). This is due to the HSRP creating a virtual IP with a virtual MAC address, and this second virtual MAC address is not part of the interface configuration. Therefore the macvtap forwarding does not work as it is based on the MAC address.

3.3.7 Comparison of the CNIs

The following graphics should give an impression of the size and scope of the reviewed CNI plugins.

Lines of code

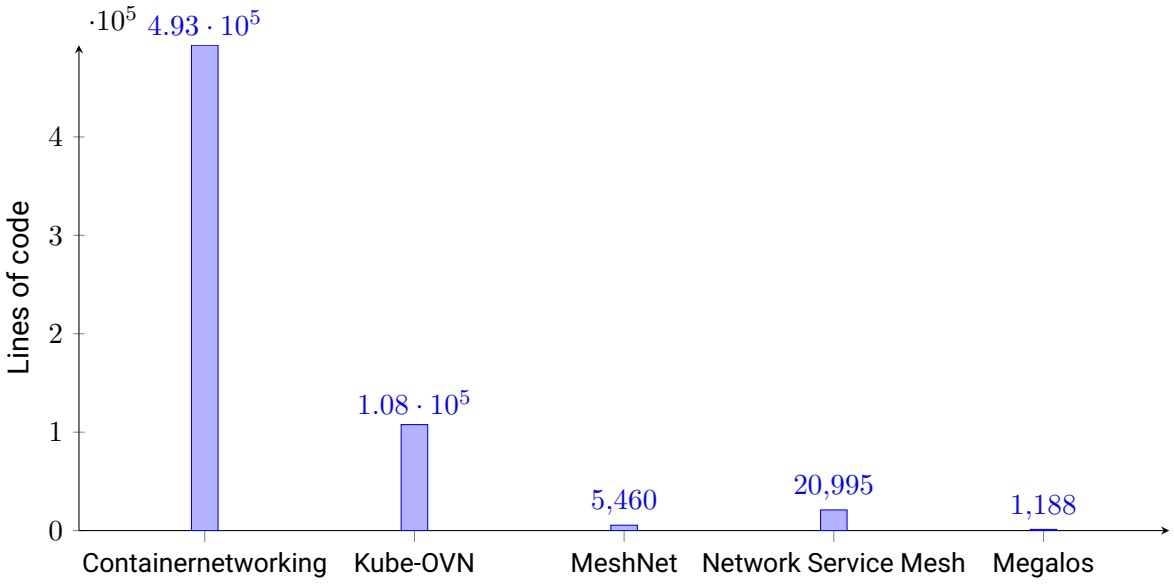

Figure 3.1: CNI comparison - Lines of code

Contributors

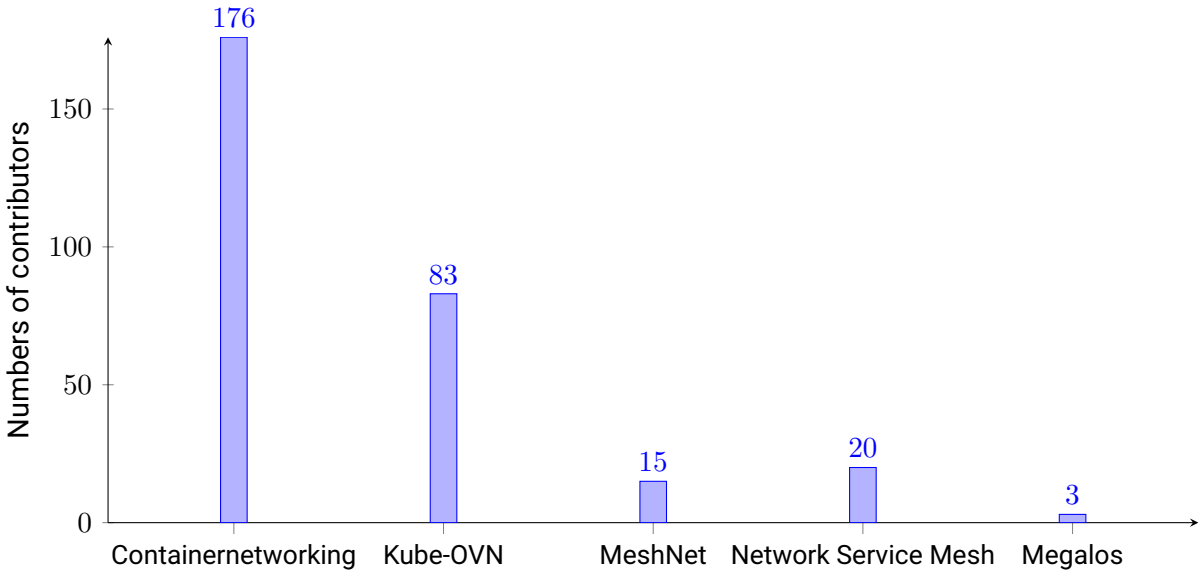

Figure 3.2: CNI comparison - Contributors

3.4 SUSE Harvester

Harvester is a cutting-edge Hyperconverged infrastructure (HCI) solution that runs on bare metal servers and uses high-quality open-source technologies such as Kubernetes, KubeVirt and Longhorn. It is a cloud-native HCI solution that offers flexibility and affordability for users who want to place VM workloads on the edge, near their IoT devices, and connect them to their cloud infrastructure.

It is based on Kubernetes, KubeVirt, Longhorn and Multus. These mentioned technologies are more or less exactly the ones needed for this semester assignment. Therefore, Harvester made a very good impression at first sight.

However, after initial testing, it has become apparent that while Harvester uses Kubernetes and KubeVirt in the background, administration of the Kubernetes cluster would technically be possible but is not recommended by Harvester. For this reason, Harvester is not a viable approach for the semester assignment. [15]

Chapter 4

Solution

4.1 Proof of concept 1 - Kube-OVN / macvtap-cni / veth pair

The first proof of concept serves as the initial validation that the requirements for the INS can be met. As it is a first concept, it involves a lot of workarounds and manual commands that need to be applied. The primary purpose is to show that it is possible to connect from a KubeVirt VM to a KubeVirt VM or a pod on layer 2, even if all resources are running on different physical nodes.

In summary, the following tools were used to establish the connectivity.

- KubeVirt to run the virtual machines
- macvtap-CNI 3.3.6 to connect to the VM on layer 2
- Kube-OVN 3.3.5 as a CNI that can transport packages on layer 2 over multiple nodes
- veth pair for a workaround to connect between macvtap-cni (the VM) and Kube-OVN

Thanks to the macvtap-cni 3.3.6, it is possible to connect the KubeVirt VM to a network interface in the default networking namespace. A veth pair was created. One side is used to connect to the KubeVirt VM via macvtap-CNI. The other side will be added to the Kube-OVN CNI. To connect one end of a veth pair into an existing Kube-OVN subnet, a couple of commands on the virtual switch had to be run. After this, the VMs were visible to each other on layer 2 with protocols like LLDP.

IP connectivity still needed to be established. Due to an issue with the ARP resolution, the Kube-OVN virtual switch was not aware of the VMs MAC address and did not learn it automatically. This can be overcome by telling Kube-OVN to send all packages destined to unknown MAC addresses to the veth ports. After manually configuring this option, ping was working.

Sending IEEE 802.1Q encapsulated traffic through the Kube-OVN network is possible. This functionally needs to be enabled manually as well in OVN.

A limitation of using macvtap is that only one MAC address per interface can be used. This means sub-interfaces for VLANs will still work, but protocols like HSRP with virtual MAC addresses will not.

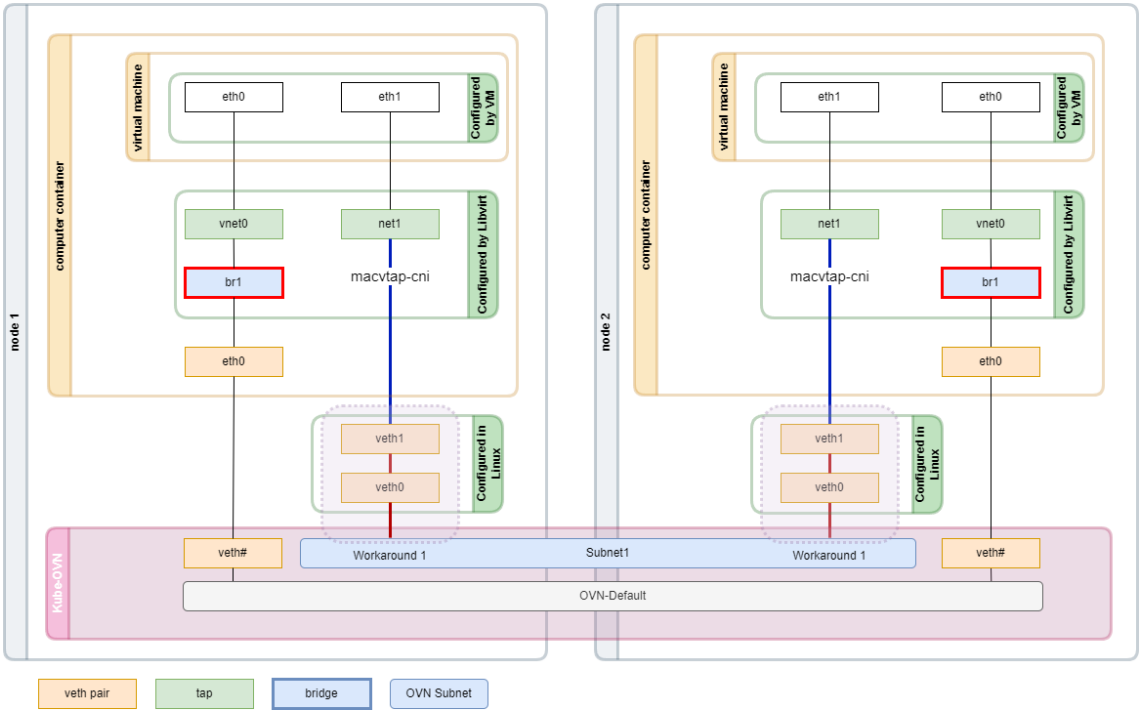

Figure 4.1: POC 1 overview

Details of the exact steps to implement this proof of concept can be found in the technical documentation. Further testing and evaluation are required to determine whether this is a long-term solution.

4.2 Proof of concept 2 - Kube-OVN / macvtap

The second proof of concept aimed to improve the first proof of concept. The main downside to the first proof of concept was that it relied heavily on commands and interfaces that needed to be configured on all the Kubernetes nodes. There are many reasons why relying on the manually created veth pair is not a scaleable and reliable solution. The biggest downside is that it does not fit into the concept of Kubernetes to have manually provisioned resources. Equally important is that the veth is only used as a detour to get the traffic from the VMs into the CNI. There is no technical reason why the veths are needed. The veths are only in place because there are no other configuration options to get the VM traffic directly into the CNI.

Surprisingly, it was discovered that other people are also trying to establish a direct layer 2 connection from the VMs to the CNI. A person called "kvaps" has implemented a direct route by implementing macvtap directly into the virt-launcher container. This change was proposed to KubeVirt over 1 year ago but has never reached the main branch.

Nevertheless, it would be a good solution for the INS. Unfortunately, running kvaps version of Kubevirt proved more challenging than anticipated. There was no pre-compiled version of the kvaps:macvtap-pod-network branch, and a custom-build version was compiled. The custom builds did not run smoothly and had some inexplicable behaviors. In the end, the custom compiled version never delivered the functionality kvaps promised.

The closed solution allowed for VM traffic to flow to the CNI, but not the other way around. After stating this issue to kvaps, it was mentioned that it works fine in their environment running a Kubernetes flavor called Deckhouse. It was not possible to test Deckhouse as the images are not public. [16] [17]

Chapter 5

Results

This chapter references the use cases described in section 2.4. These are now undergoing a review. It is checked what works, what does not work, how much can be installed and set up as an automatic deployment without manual commands, and which things not yet.

Nr.	Title	fulfilled	Description	in YAML declarable	Next steps
1	Frame filtering	YES	Due to the implementation of multiple workarounds from the proof of concept 1, it has been shown that no frames are filtered.	NO	Consolidate workaround into single CNI (promisc mode, macvtap, Multicast)
2	Endpoints	YES	In general, a 1-to-1 connection between pods and virtual machines can be established with proof of concept 1. Not all labs could be deployed, but protocols like LLDP works successfully.	NO	The 1-to-1 connections are currently implemented as subnets, which are n-to-n connections. In the next step, proper 1-to-1 connection should be implemented.
3	Multinode	YES	Due to the usage of Kube-OVN in proof of concept 1, an endpoint can be connected to any other endpoint running on the same or on a different Kubernetes node.	YES	Use Kube-OVNs implementation as a template for a custom CNI.
4	MAC Learning	NO	MAC learning could not be verified due to an issue with the deployment of the port mirroring lab.	YES	Fix the deployment of the port mirroring lab with proof of concept 1.
5	Subinterfaces	YES	The implementation of proof of concept 1 showed that subinterfaces worked after manually enabling vlan-passthru="true" in the Kube-OVN	NO	Implement feature request to allow vlan-passthru="true"
6	Multiple MAC addresses	NO	Due to a limitation with using macvtap, each VM interface can only have one static MAC address	YES	Stop using a macvtap based solution to connect the VM to the CNI.
7	Automation	NO	Currently, there are about ten manual commands required for a connection between two endpoints	NO	Further work on proof of concept 2 which reduces manual steps.

Table 5.1: Use case evaluation

Chapter 6

Conclusion

This chapter reflects on the goals and outcomes of the semester assignment.

6.1 Conclusion

The main goal of this semester assignment is to establish a direct layer 2 connection between containers and virtual machines in Kubernetes. Several reverse analyses of different CNIs were performed and tested in the field. The findings were then used to develop one or more proof of concepts on how this can be done.

The individually analyzed CNIs are very similar in broad outline, but cover quite different aspects in detail. Furthermore, how the CNIs are used is very different. The Kube-OVN CNI in cooperation with the macvtap CNI, which was used in the proof of concept 1, have finally brought the most success. MeshNet CNI, Network Service Mesh CNI and Megalos CNI were less successful for the given task. A layer two connection between pods and virtual machines could be realized with the proof of concept 1 and even deployed as a multi-node setup. Protocols like LLDP worked without problems. Creating a subinterface with different VLANS worked as well. Unfortunately, the lab with HSRP did not work, and the traffic mirroring lab could not be tested successfully.

6.2 Custom CNI

In conclusion, the proof of concepts described in the solution chapter are not very intuitive. Many manual tasks are required, therefore they do not scale well. In addition, with the proof of concept 1, not all the labs are working properly. The idea of developing a custom CNI is therefore still on the list. Which specifications that own CNI must have is described in the chapter "future steps" in the technical documentation.

Worth mentioning is that there is one major pre-requisite that must be fulfilled before developing a custom CNI. The KubeVirt to CNI connection must be completely transparent, and this connection must pass through all the VM traffic to the CNI without any MAC address limitations or layer 2 filters. It has been proven by this semester assignment that there is currently no existing solution for this.

6.3 Open Issues & Feature Requests

At the end of the semester assignment, the following issues or pull requests are still open, which should be tracked and followed up in the future:

6.3.1 Macvtap binding mode for pod network

The implementation of the macvtap binding mode never really worked for the scenario of the semester assignment. The latest status is that the developer kvaps was asked to provide the YAML files with the operator and the cr images, and there has yet to be an answer. The whole pull request is still open to merge it to KubeVirt.

Direct link: https://github.com/kubevirt/kubevirt/pull/7648

6.3.2 Allow other_config: vlan-passthru="true"

In order to make subinterfaces work in the proof of concept 1, the flag vlan-passthru="true" in the other_config has to be set. A feature request is still open to implementing this as a tag in the subnet config.

Direct link: https://github.com/kubeovn/kube-ovn/issues/2737

Glossary

ARP Address Resolution Protocol (ARP) is a communication protocol used for discovering the link layer address, such as a MAC address.
https://en.wikipedia.org/wiki/Address_Resolution_Protocol 19

CNI CNI (Container Network Interface), a Cloud Native Computing Foundation project, consists of a specification and libraries for writing plugins to configure network interfaces in Linux containers, along with a number of supported plugins. CNI concerns itself only with network connectivity of containers and removing allocated resources when the container is deleted.
https://github.com/containernetworking/cni i, ii, iii, iv, 2, 3, 7, 8, 9, 11, 12, 13, 15, 16, 17, 19, 21, 23, 24, 25

ECTS European Credit Transfer and Accumulation System https://en.wikipedia.org/wiki/European_Credit_Transfer_and_Accumulation_System 4

HSRP Hot Standby Router Protocol (HSRP) is a Cisco proprietary redundancy protocol for establishing a fault-tolerant default gateway
https://en.wikipedia.org/wiki/Hot_Standby_Router_Protocol i, ii, 10, 16, 20, 24

IEEE 802.1Q IEEE 802.1Q (Dot1q), is the networking standard that supports virtual local area networking (VLAN) on an IEEE 802.3 Ethernet network. The standard defines a system of VLAN tagging for Ethernet frames.
https://en.wikipedia.org/wiki/IEEE_802.1Q 20

INS Institute for Network and Security: supports organizations in planning, implementing, operating and monitoring networks that are based on the latest technologies and offer greater security and intelligence - on-site or in the cloud. We build networks of the future that enable innovation and are ready for new cloud-native applications. We know not only how to secure infrastructure, but also

how to develop secure software.
https://www.ost.ch/de/forschung-und-dienstleistungen/informatik/ins-institut-fuer-netzwerke-und-sicherheit i, iii, iv, 1, 2, 4, 9, 11, 12, 19

Kubernetes Kubernetes is an open source container orchestration engine for automating deployment, scaling, and management of containerized applications. The open source project is hosted by the Cloud Native Computing Foundation (CNCF). [1] https://kubernetes.io/docs/home/ i, ii, iii, vi, 1, 2, 6, 7, 11, 12, 24

KubeVirt KubeVirt provides a unified development platform where developers can build, modify, and deploy applications residing in both Application Containers as well as Virtual Machines in a common, shared environment. [3] https://kubevirt.io/ i, 7, 19

KVM Kernel Virtual Machine: is a full virtualization solution for Linux on x86 hardware containing virtualization extensions (Intel VT or AMD-V). Using KVM, one can run multiple virtual machines running unmodified Linux or Windows images. Each virtual machine has private virtualized hardware: a network card, disk, graphics adapter, etc. https://www.linux-kvm.org/page/Main_Page 1, 2

LLDP The Link Layer Discovery Protocol (LLDP) is a vendor-neutral link layer protocol used by network devices for advertising their identity, capabilities, and neighbors on a local area network based on IEEE 802 technology, principally wired Ethernet. https://en.wikipedia.org/wiki/Link_Layer_Discovery_Protocol i, 3, 7, 16, 19, 24

LTB The Lab Topology Builder is a software stack made by the INS to deploy test environments for students. 1, 2, 4, 6, 12, 13

MkDocs MkDocs is a fast, simple and downright gorgeous static site generator that's geared towards building project documentation. https://www.mkdocs.org/ 4

Multus Multus CNI enables attaching multiple network interfaces to pods in Kubernetes. https://github.com/k8snetworkplumbingwg/multus-cni 9, 13

OSI model In the Open Systems Interconnection model (OSI model) the communications between a computing system are split into seven different abstraction layers: Physical, Data Link, Network, Transport, Session, Presentation, and Application. https://en.wikipedia.org/wiki/OSI_model 6

OST Eastern Switzerland University of Applied Sciences: In this document focused on the campus Rapperswil-Jona. 1

OVN OVN (Open Virtual Network) is a series of daemons for the Open vSwitch that translate virtual network configurations into OpenFlow. OVN provides a higher-layer of abstraction than Open vSwitch, working with logical routers and logical switches, rather than flows. OVN is intended to be used by cloud management software (CMS). https://www.ovn.org/en/ i, 15

OVS This plugin allows user to define Kubernetes networks on top of Open vSwitch bridges available on nodes. Note that ovs-cni does not configure bridges, it's up to a user to create them and connect them to L2, L3 or an overlay network. https://github.com/k8snetworkplumbingwg/ovs-cni i, 9, 15

Proxmox VE Proxmox VE is a Debian-based open-source virtualization platform for running virtual machines with a web interface for setting up and controlling x86 virtualizations. The environment is based on QEMU with the Kernel-based Virtual Machine. https://www.proxmox.com/en/ 11

RKE2 Rancher Kubernetes Engine Version 2. It is a Kubernetes provider for self-hosted Kubernetes environments. https://docs.rke2.io/ 12

STP The Spanning Tree Protocol (STP) is a network protocol that builds a loop-free logical topology for Ethernet networks. https://en.wikipedia.org/wiki/Spanning_Tree_Protocol 7

veth pair Virtual Ethernet Device (veth) is a linux networking component, which is always created in interconnected pairs. They can be used as standalone network devices or to interconnect network namespaces. https://man7.org/linux/man-pages/man4/veth.4.html 19

VLAN Virtual local area network (VLAN) is any broadcast domain that is partitioned and isolated in a computer network at the data link layer (OSI layer 2). https://en.wikipedia.org/wiki/VLAN 26

xdp-xconnect Cross-connect Linux interfaces with XDP redirect https://github.com/networkop/xdp-xconnect/ 9

List of Figures

List of Tables

Bibliography

[1] Kubernetes documentation. [Online]. Available: https://kubernetes.io/docs/home/

[2] An oddly specific post about group_fwd_mask. [Online]. Available: https://interestingtraffic.nl/2017/11/21/an-oddly-specific-post-about-group_fwd_mask

[3] Kubevirt documentation. [Online]. Available: https://kubevirt.io/

[4] Kubevirt network stack diagram. [Online]. Available: https://github.com/kubevirt/kubevirt.github.io/blob/main/assets/images/diagram.png

[5] Rke2 kubernetes provider. [Online]. Available: https://docs.rke2.io/

[6] Kubespray kubernetes provider. [Online]. Available: https://kubespray.io/#/

[7] Containernetworking cnis plugins. [Online]. Available: https://www.cni.dev/

[8] Megalos-cni. [Online]. Available: https://github.com/KatharaFramework/Megalos-CNI

[9] Kathara. [Online]. Available: https://www.kathara.org/man-pages/kathara.1.html

[10] Kube-ovn. [Online]. Available: https://kubeovn.github.io/docs/v1.11.x/en/

[11] kubevirt-vm-fixed-address. [Online]. Available: https://kubeovn.github.io/docs/v1.11.x/en/guide/static-ip-mac/#kubevirt-vm-fixed-address

[12] macvtap. [Online]. Available: https://kubevirt.io/user-guide/virtual_machines/interfaces_and_networks/#macvtap

[13] Promiscuous-mode. [Online]. Available: https://en.wikipedia.org/wiki/Promiscuous_mode

[14] add promiscous mode knob. [Online]. Available: https://github.com/kubevirt/macvtap-cni/pull/98

[15] Harvester by suse. [Online]. Available: https://harvesterhci.io/

[16] Macvtap binding mode for pod network #7648. [Online]. Available: https://github.com/kubevirt/kubevirt/pull/7648

[17] Harvester by suse. [Online]. Available: https://github.com/deckhouse/deckhouse/tree/main/modules/490-virtualization/templates