OpenShift Virtualization with Isovalent Enterprise Platform

Implementation best practices

version: 1.0

May 2026

Legal Notice

Copyright © 2025 Red Hat, Inc.
The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at http://creativecommons.org/licenses/by-sa/3.0/. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Red Hat

Table of Contents

Legal Notice 1
Table of Contents 2
About this Document 4
Purpose of this document 4
Target Use Cases 4
Replatforming from Legacy Virtualization Platforms 6
Technology Overview 7
Red Hat OpenShift Virtualization 7
Isovalent Networking for Kubernetes 9
Considerations on design and architecture with Isovalent Enterprise Platform 10
Deploying and configuring OpenShift features to support virtual machines and applications 10
Cluster deployment network configuration 10
Bonded NICs for management and SDN 10
Software-Defined Networking (SDN) 11
Externally Connectivity to Workloads 11
Security 12
Isovalent Networking for Kubernetes Implementation 14
New OpenShift cluster deployment 14
Prerequisites 14
OpenShift installation manifests 15
OpenShift cluster deployment with Cilium 16
Network connectivity tests and troubleshooting 18
Existing OpenShift cluster deployment 20
Support considerations 20
Migration Approach 21
Migration to Cilium 21
Post-Migration Considerations 23
Advanced configuration 23
Permissions 23
BGP control plane in Isovalent Networking for Kubernetes 24
Enabling the BGP Control Plane 25
BGP Control Plane Resources 25
Configuring BGP Peering 26
Example: BGP Cluster Configuration 26
BGP Peer Configuration 27

Red Hat

Example: BGP Peer Configuration....27
BGP Advertisements....28
Example: Advertising Pod CIDRs.... 28
Example: Advertising Service Virtual IPs.... 29
Customizing BGP for Specific Nodes.... 30
Example: Setting Router ID for a Specific Node.... 31
Verifying and Troubleshooting BGP.... 32
Check BGP Peering State....32
Check Advertised Routes....33
Further Considerations: Traffic Engineering with Cilium.... 33
Multi-Pool IPAM.... 34
Enabling Multi-Pool IPAM in Cilium....34
Defining Pod Networks.... 36
Example: Creating a Secondary Network.... 36
Attaching Pods and VMs to Specific Pools.... 36
Example: Assigning a VM to different Pod IP Pools.... 37
How CiliumPodIPPool is Assigned to the virt-launcher Pod in KubeVirt....38
IP Allocation in Different KubeVirt Networking Modes.... 38
Validating Multi-Pool IPAM.... 38
Check Allocated IP Pools....38
Check Node IP Allocations....39
Check Running Pods and Assigned IPs.... 39
Routing and Connectivity.... 40
Observing Multi IPAM Pool Traffic with Hubble....41
Cilium Network Policies and Observability for Virtual Machines.... 42
Concepts....42
Layer 3....43
Layer 4....44
Layer 7....45
Troubleshooting.... 46
Authors....50
Appendix .. . Change log....50

Red Hat

About this Document

Purpose of this document

This document provides implementation best practices for deploying Red Hat OpenShift, the leading enterprise Kubernetes offering, as a platform also designed for hosting virtualization workloads using OpenShift Virtualization, with the integration of Isovalent Networking for Kubernetes, the enterprise offering of the popular Kubernetes Container Network Interface (CNI), Cilium.

This document is a supplement to product documentation providing opinionated suggestions and guidance for configuring OpenShift for hosting virtual machines, focusing on the specific integration of Isovalent Networking for Kubernetes (Enterprise-grade Cilium CNI). For more details about OpenShift Virtualization, please refer to the official documentation, and to the document 'OpenShift Virtualization Reference Implementation Guide'.

The best practices described in this document do not define a required configuration for support and are not exhaustive. Before implementing, consider the needs and requirements of your virtualized applications and choose the configuration that best meets those needs.

When Cilium is referenced in this document, it is referring to the hardened enterprise-grade Cilium distribution from Isovalent, known as Isovalent Networking for Kubernetes. Cilium's open source ecosystem is vibrant and diverse, with close to 1,000 contributors and over 20,000 GitHub Stars and members on Slack. But for organizations running production workloads, relying solely on best-effort community support isn't always enough. With Isovalent, you get enterprise-grade 24/7 support, backed by the engineers who build and maintain Cilium.

The Isovalent Enterprise Platform is validated and supported for use with Red Hat OpenShift Virtualization. Compatibility with supported Red Hat OpenShift versions can be confirmed via [Red Hat's official support matrix.](#)

Target Use Cases

OpenShift Virtualization helps customers optimize their data center footprint for today's and tomorrow's applications. It provides modern infrastructure for enterprise VMs that increases efficiency through automation, manageability, and scalability. OpenShift Virtualization delivers

the performance, scale, and stability of Kernel-based virtual machines (KVM), deployed and managed as part of a modern application platform.

OpenShift Virtualization enables businesses to accelerate infrastructure modernization by bringing modern virtualization infrastructure, via OpenShift, to support existing virtual machines.

Isovalent Networking for Kubernetes is a cloud native solution for providing, securing, and observing network connectivity between workloads. Isovalent created Cilium, a CNCF graduated project that has become the de-facto standard for cloud native networking, and eBPF, the revolutionary kernel-level technology that powers Cilium . The Isovalent Enterprise Platform consists of Enterprise-grade Cilium, Tetragon, Hubble, and provides functionality ranging from Kubernetes Networking, BGP Routing, Load Balancing, Service Mesh, Firewalling, Observability, and Runtime Security.

Customers deploying these joint solutions will see value in several ways:

Unified Platform for VMs and Containers: Customers can manage both virtual machines and containers under a single OpenShift platform. Cilium enhances this by providing consistent networking and security policies across both types of workloads, simplifying infrastructure management and reducing operational complexity.

Enhanced Network Security: With Cilium's advanced network security features, including transparent encryption, identity-aware microsegmentation, and network policies, customers can secure communication between workloads across containers and VMs. This is critical for enforcing Zero Trust principles and ensuring compliance in highly regulated environments.

Deep Observability Across Workloads: Cilium, through Hubble and Tetragon, delivers comprehensive observability into network and runtime activity, allowing customers to monitor and trace network traffic, interactions between VMs, containers, and services, as well as detect potential security threats. This deep visibility reduces troubleshooting time and improves the security posture.

Scalability and Performance: OpenShift Virtualization ensures enterprise-grade performance and scalability of VMs, while Cilium provides highly scalable and performant networking through its use of eBPF (the capability to run sandboxed programs in a privileged context such as the kernel). Together, they support businesses in scaling up their applications with minimal overhead.

Operational Efficiency through Automation: Both platforms emphasize automation and manageability. OpenShift's orchestration combined with Cilium's policy-driven approach to networking allows enterprises to automate network configurations, security policies, and scaling efforts, increasing efficiency and reducing manual overhead.

Future-Proof Infrastructure: As enterprises modernize, this combination of OpenShift Virtualization and Cilium allows for seamless transitions from legacy applications to cloud-native workloads. It supports a hybrid environment ready for future technology trends, including AI-driven workloads and distributed architectures.

Replatforming from Legacy Virtualization Platforms

OpenShift is a trusted, comprehensive, and consistent platform designed to scale with the needs of today's virtualization workloads. As customers move to OpenShift Virtualization from a legacy platform, they will see immediate benefits as the management of virtual machines shifts to a cloud-native paradigm via OpenShift's Kubernetes foundation.

Key Benefits of OpenShift Virtualization with Isovalent Enterprise Platform:

- **Modernize Today for Innovation Tomorrow :** OpenShift Virtualization allows organizations to modernize their infrastructure by running VMs and containers side by side. By leveraging Kubernetes-native tools and APIs, platform teams can integrate legacy workloads into modern CI/CD pipelines.

- **Consistency of Management:** Unified control through the OpenShift web console and/or the Kubernetes API enables teams to manage VMs, containers, and serverless applications from a single interface. This consistency enhances collaboration across platform, security, and infrastructure teams.

- **Increased Operational Efficiency:** Self-service deployment models and developer-friendly automation tooling empower teams to accelerate delivery. Integrating VMs with OpenShift pipelines and APIs simplifies deployment workflows and reduces time to production.

- **Streamlined Networking and Observability:** With Isovalent's enterprise Cilium offering, platform owners gain access to a unified networking, security, and observability stack. Advanced capabilities such as BGP, multi-networking, service mesh, and runtime

security are available out of the box. This aids in reducing tooling sprawl and improving time to value.

The Isovalent Enterprise Platform is validated and supported for use with Red Hat OpenShift Virtualization. Compatibility with supported Red Hat OpenShift versions can be confirmed via [Red Hat's official support matrix.]

Technology Overview

Red Hat OpenShift Virtualization

Red Hat OpenShift is an industry-leading enterprise Kubernetes platform that offers a complete, production-grade experience for managing containerized workloads at scale. With powerful developer tooling, automated operations, and a rich ecosystem of certified integrations, OpenShift is widely adopted by organizations modernizing their application infrastructure.

OpenShift Virtualization is built on KVM (Kernel-based Virtual Machine), the open-source hypervisor integrated directly into the Linux kernel. By using KVM, OpenShift treats virtual machines as native Kubernetes objects—enabling unified management, scheduling, and scaling alongside container workloads.

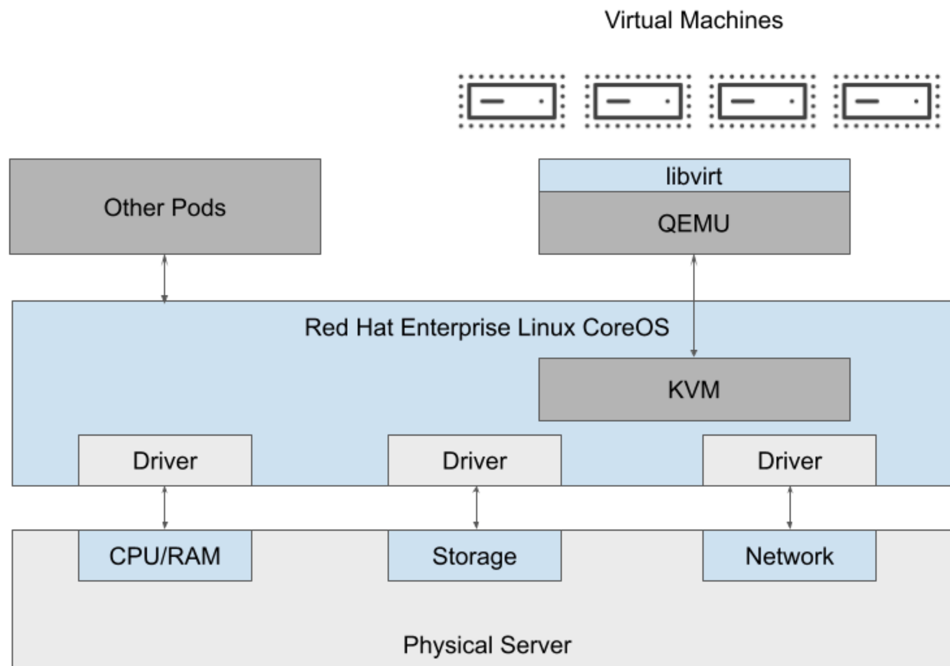
The underlying virtualization stack includes:

- QEMU (Quick Emulator): a generic and open-source machine emulator and virtualizer.
- Libvirt: a toolkit for managing virtualized platforms, used to interface with KVM and QEMU at runtime.
- Kubevirt: a highly successful open-source project with over 200 major contributors, enabling you to run, deploy, and manage virtual machines (VMs) on Kubernetes as the orchestration platform.

These same components have powered production-grade virtualization in platforms such as Red Hat Virtualization, Red Hat OpenStack Platform, and Red Hat Enterprise Linux for more than 15 years.

By embedding this proven technology stack into OpenShift, the platform delivers a robust, scalable, and enterprise-ready type-1 hypervisor that meets modern infrastructure needs, whether you're running traditional VMs, containerized apps, or both side-by-side.

This architecture is illustrated in the diagram below, showing how KVM, QEMU, and libvirt work together with Kubernetes-native control to deliver VM support in OpenShift.



Virtual machines are deployed and managed using the same scheduler as non-virtual machine workloads. Native features like host affinity, resource awareness, load balancing, and high availability are all inherited from OpenShift features for non-virtual machine workloads.

Virtual machines connected to the cluster's software-defined network (SDN) are treated as first-class citizens in the OpenShift environment. Because OpenShift is built on Kubernetes and extends it with enterprise-grade capabilities, VMs benefit from the same connectivity and policy constructs used for containers.

This means virtual machines can be:

- Exposed using standard Kubernetes Services
- Secured with Ingress/Egress controls and Network Policies
- Integrated into service meshes and cluster observability pipelines

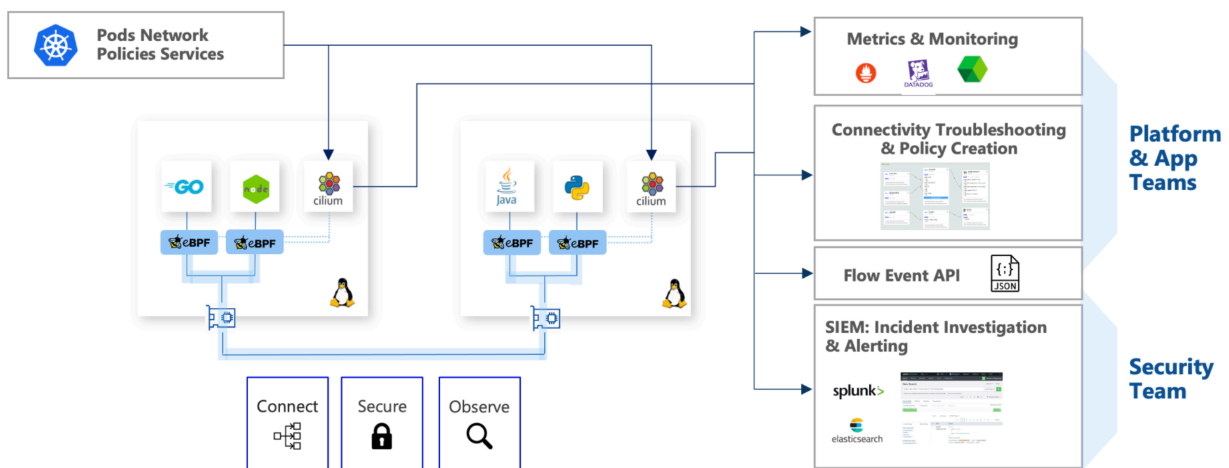
This approach provides consistent access and security management for VM-based workloads, both from within the cluster and from external networks, using the same Kubernetes-native tooling that developers and platform teams already know.

OpenShift Virtualization offers several deployment options, including self-managed bare metal hosts on-premises or in the cloud on AWS using either managed ROSA clusters or self-managed clusters (AWS deployment requires metal instances).

Isovalent Networking for Kubernetes

Isovalent Networking for Kubernetes helps platform and security teams to solve challenges and problems such as connectivity within cloud native platforms using different network topologies, implementation of zero trust security principles with network policies and transparent encryption, distributed and secure multicluster connectivity, and integrated sidecar-free service mesh capabilities. Cilium is the third most active project in the CNCF (behind only Kubernetes and OpenTelemetry), and became the first project to graduate in the cloud native networking category. Cilium is the de facto container networking interface (CNI), with the greatest adoption of all CNIs across the major cloud service providers' Kubernetes offerings.

Isovalent Networking for Kubernetes is the hardened, enterprise-grade, and 24×7-supported version of the eBPF-based cloud networking platform Cilium. In addition to all features available in the open-source version of Cilium, the enterprise edition includes advanced networking, security, and observability features popular with enterprises and telco providers.



Considerations on design and architecture with Isovalent Enterprise Platform

We recommend a modular architecture with a structured, standardized approach for designing and implementing Red Hat OpenShift Virtualization with third-party CNI provider Isovalent Networking for Kubernetes, a component of Isovalent Enterprise Platform. This document will focus on specific requirements tied to the enterprise Cilium integration; more details can be found in the document 'OpenShift Virtualization Reference Implementation Guide' for general information on virtualization implementation on OpenShift platform, notably including Physical Cluster Design.

Deploying and configuring OpenShift features to support virtual machines and applications

This section will describe the recommended OpenShift network configuration for hosting virtual machines. For recommended configuration on other components, please refer to the document 'OpenShift Virtualization Reference Implementation Guide (https://access.redhat.com/articles/7067871)

Cluster deployment network configuration

Bonded NICs for management and SDN

To enhance throughput and provide redundancy, it's a best practice in OpenShift environments to configure a bonded network interface on each host. This bond, typically comprising two physical adapters, should be assigned an IP address on the machine network during installation. This bonded interface can serve multiple purposes, including:

- Software-Defined Networking (SDN)
- Management traffic between the node and the control plane

- Administrator access
- Live migration traffic

During the OpenShift installation process, utilize the host network interface configuration options to set up the bond and assign the necessary IP address.

Cilium uses a runtime detection to select the active network adapter of the OpenShift node to use for network connectivity. If your OpenShift node has multiple active network adapters, to ensure that Cilium utilizes this bonded interface for all communications, specify the interface using the `devices` parameter in the CiliumConfig manifest cluster bootstrap, for example, `devices='{bond0}'.`

Software-Defined Networking (SDN)

Since the release of OpenShift 4.15 and later versions, the default Red Hat-provided SDN option is OVN-Kubernetes. OVN-Kubernetes, built on Open Virtual Network (OVN), provides an overlay-based networking implementation. Each node in a cluster running OVN-Kubernetes utilizes Open vSwitch (OVS), which is configured by OVN to implement the declared network configuration.

When deploying Isovalent Networking for Kubernetes on an existing or new OpenShift cluster, the default OVN-Kubernetes SDN is replaced by Isovalent's hardened Cilium distribution. Cilium offers a modern, eBPF-powered networking stack that simplifies operations, enhances security, and provides advanced observability for both containerized and virtual machine workloads. This replacement is part of a broader certification process that ensures Cilium's compatibility with OpenShift, supporting both networking and virtualization requirements.

Subsequent chapters in this document will delve deeper into the implementation details, including how Cilium integrates seamlessly into OpenShift to manage networking and how it provides consistent network policies across container and virtual machine environments. These chapters will also explore the benefits of Cilium's observability features and its ability to unify network and security management for hybrid workloads.

Externally Connectivity to Workloads

By default, OpenShift Virtualization facilitates most communication through the SDN. Leveraging Cilium in this architecture enables advanced networking features like BGP peering, Gateway API, and multi-pool IPAM capabilities, providing flexible options for external connectivity.

Cilium's Gateway API integration supports exposing workloads with a standardized and scalable approach, handling a wide range of protocols and ports beyond HTTP(S). This approach enhances flexibility for connecting VMs and container workloads to external networks. Additionally, Cilium's BGP support dynamically advertises services to external routers, simplifying service discovery and reducing manual configuration requirements.

As a best practice, it is recommended to use the Gateway API for exposing workloads, BGP for dynamic service advertisement, and multi-pool IPAM configurations for enhanced flexibility.

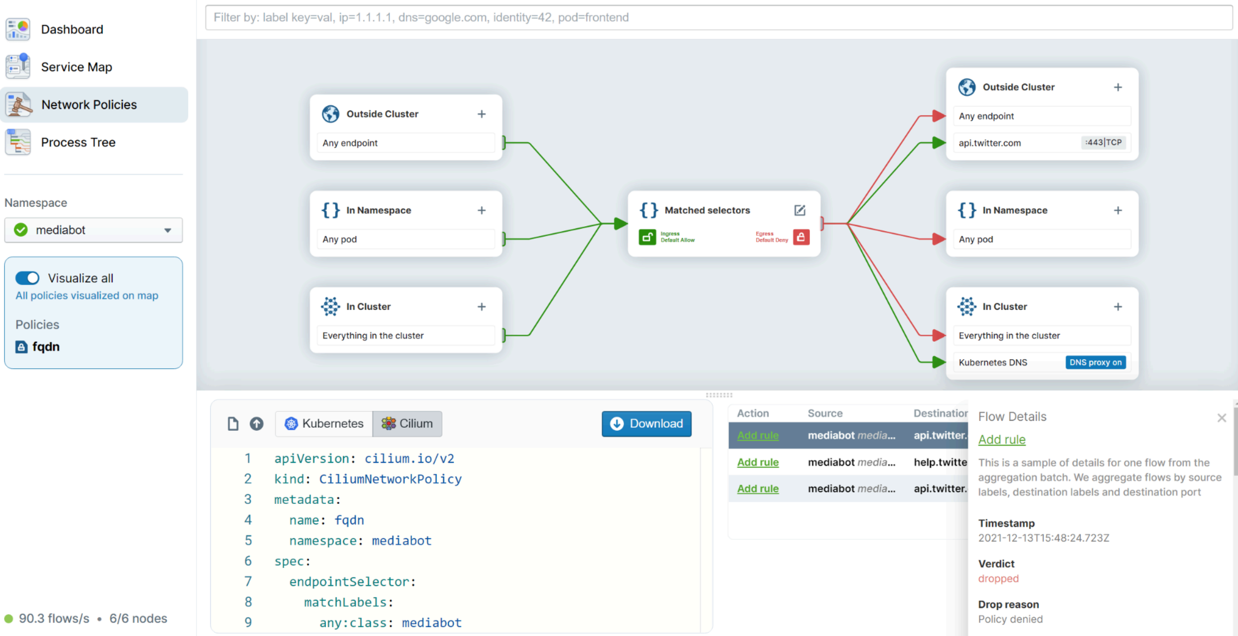
For more information about network configuration options in OpenShift, please see the documentation available here.

Security

Cilium provides robust security features to protect workloads in a cluster, with Cilium Network Policies (CNPs) at the core of its security model. These policies extend Kubernetes Network Policies by supporting Layer 7 (application layer) filtering and integration with DNS-based policies, known as Fully Qualified Domain Name (FQDN) policies.

FQDN policies allow you to define network rules based on domain names, enabling fine-grained control over egress traffic from your workloads. This is particularly useful for environments that require strict adherence to compliance or regulatory standards, as it ensures that workloads can only communicate with explicitly approved external domains.

Isovalent Networking for Kubernetes includes the Hubble Timescape UI, which allows for the real time monitoring of network flows, service map visualization and a network policy editor.

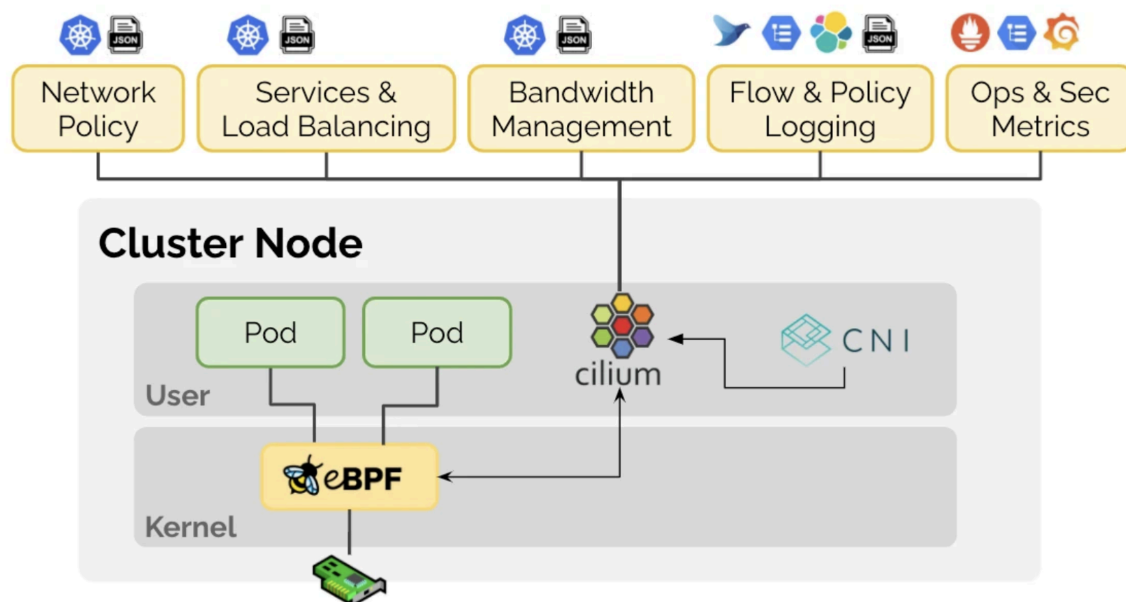


By combining these policies with Cilium's observability features, such as Hubble, administrators gain deep visibility into network traffic patterns, making it easier to enforce and monitor security requirements. This high-level approach ensures consistent and scalable security across both containerized and virtual machine workloads.

Isovalent Networking for Kubernetes Implementation

This section of the document will cover the base installation of a Red Hat OpenShift Cluster with Isovalent Networking for Kubernetes.

New OpenShift cluster deployment



Prerequisites

- Jump host to run the installation software from
  - We can download the `openshift-install` tool and `oc` tool from this [Red Hat repository.]
  - Alternatively we can go to [Red Hat Hybrid Cloud Console] > Click OpenShift > Clusters > Create a cluster > Select our platform. This will provide download links to the software and access to a pull secret, needed to complete the installation.
- A pull secret file/key from the [Red Hat Cloud Console] website
  - A trial activation period for 60 days can be requested using the same method.

- Access to the DNS server, which supports the infrastructure platform that we are deploying to.
- An SSH Key to be used for access to the deployed OpenShift nodes.

Extract the software tools and place them in your user location.

```
None
$ tar -zxvf openshift-client-linux-{version}.tar.gz
$ tar -zxvf openshift-install-linux-{version}.tar.gz

$ sudo cp openshift-install /usr/bin/local/openshift-install
$ sudo cp oc /usr/bin/local/oc
$ sudo cp kubectl /usr/bin/local/kubectl
```

Download the Isovalent Networking for Kubernetes Operator Lifecycle Manager (OLM) manifest files from the Isovalent website.

- Check the official Certified OpenShift CNI Plug-ins guide to see the compatibility matrix for Red Hat OpenShift and Cilium versions.

For the rest of this guide, it is assumed that you will complete the recommended installation steps for a Red Hat OpenShift Cluster as per the official documentation, and follow any additional steps necessary related to your particular chosen platform.

Once prerequisites are satisfied, stop at Creating the Kubernetes manifests and Ignition config files.

OpenShift installation manifests

In the process of creating your Red Hat OpenShift Cluster you should have stopped after completing the below command.

```
None
$ openshift-install create manifests
```

Now copy the Isovalent Networking for Kubernetes OLM Files into the newly created manifests folder.

```None
$ tar -xvf path/to/cilium/release
$ cp path/to/cilium/release/* ocp-manifests-dir/.
```

OpenShift cluster deployment with Cilium

OpenShift will by default create a `cluster-network-02-operator.yml` file. Within this file, the field `apiVersion` should be set to `operator.openshift.io/v1`, the networkType should be set to Cilium and all relevant clusterNetwork and serviceNetwork CIDRs should be defined, inline with that configured in the file `ciliumconfig.yaml` described after this file below.

```None
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  creationTimestamp: null
  name: cluster
spec:
# We need to explicitly disable KubeProxy
  deployKubeProxy: false
  clusterNetwork:
  - cidr: 10.253.0.0/16
    hostPrefix: 24
  externalIP:
    policy: {}
  networkType: Cilium
  serviceNetwork:
  - 172.31.0.0/16
status: {}
```

To configure Cilium, we can modify the `cluster-network-07-cilium-ciliumconfig.yml` file. The below example shows the configuration for Hubble Metrics and Prometheus Service Monitors enabled:

```None
apiVersion: cilium.io/v1alpha1
```

Red Hat

```
kind: CiliumConfig
metadata:
  labels:
    app.kubernetes.io/name: clife
  name: ciliumconfig
spec:
  cluster:
    name: default
  securityContext:
    privileged: true
  ipam:
    mode: "cluster-pool"
    operator:
      clusterPoolIPv4PodCIDRList:
      - "10.253.0.0/16"
      clusterPoolIPv4MaskSize: 24
  cni:
    binPath: "/var/lib/cni/bin"
    confPath: "/var/run/multus/cni/net.d"
    # portmap is needed for supporting HostPort without KPR
    chainingMode: portmap
    exclusive: false
  prometheus:
    enabled: true
    serviceMonitor:
      enabled: true
  hubble:
    enabled: true
   metrics:
        enabled:
        - dns:labelsContext=source_namespace,destination_namespace
        - drop:labelsContext=source_namespace,destination_namespace
        - tcp:labelsContext=source_namespace,destination_namespace
        - port-distribution:labelsContext=source_namespace,destination_namespace
        -
icmp:labelsContext=source_namespace,destination_namespace;sourceContext=workload-n
ame|reserved-identity;destinationContext=workload-name|reserved-identity
        -
flow:sourceContext=workload-name|reserved-identity;destinationContext=workload-nam
e|reserved-identity;labelsContext=source_namespace,destination_namespace
        -
"httpV2:exemplars=true;labelsContext=source_ip,source_namespace,source_workload,de
stination_ip,destination_namespace,destination_workload,traffic_direction;sourceCo
```

```
ntext=workload-name|reserved-identity;destinationContext=workload-name|reserved-id
entity"
        -
"policy:sourceContext=app|workload-name|pod|reserved-identity;destinationContext=a
pp|workload-name|pod|dns|reserved-identity;labelsContext=source_namespace,destinat
ion_namespace"
        - flow_export
  operator:
    prometheus:
      enabled: true
      serviceMonitor:
        enabled: true
  sessionAffinity: true
  kubeProxyReplacement: false
  # avoid conflicts
  clusterHealthPort: 9940
  # OVN Kubernetes default port, firewalls may already be configured for it
  tunnelPort: 4789
  enterprise:
    featureGate:
      approved:
      - CNIChainingMode
```

Note: In particular the configuration `socketlb.hostnamespaceonly` is necessary when using OpenShift Virtualization with Cilium. The other configurations apply to a standard OpenShift deployment aligned to the particular platform the cluster is deployed upon.

It is also possible to update the configuration values once the cluster is running by changing the `CiliumConfig` object, e.g., with `oc edit cilium config—n cilium cilium`. However, certain options may require restarting the Cilium agent pods.

You are now ready to proceed with the creation of your Red Hat OpenShift cluster, as per the installation instructions particular to your chosen platform.

We recommend waiting for the majority of `clusteroperators` to be in the Available state before provisioning the worker nodes

Network connectivity tests and troubleshooting

Once the bootstrap and creation of your Red Hat OpenShift Cluster is complete, we can now run network connectivity tests.

For Cilium connectivity test pods to run on OpenShift, Cilium requires a custom `SecurityContextConstraints` object. The object sets only the `hostPort` and `hostNetwork` values that some connectivity test pods rely on, without any other privileges.

```
None
oc apply -f - <<EOF
apiVersion: security.openshift.io/v1
kind: SecurityContextConstraints
metadata:
  name: cilium-test
allowHostPorts: true
allowHostNetwork: true
users:
  - system:serviceaccount:cilium-test:default
priority: null
readOnlyRootFilesystem: false
runAsUser:
  type: MustRunAsRange
seLinuxContext:
  type: MustRunAs
volumes: null
allowHostDirVolumePlugin: false
allowHostIPC: false
allowHostPID: false
allowPrivilegeEscalation: false
allowPrivilegedContainer: false
allowedCapabilities: null
defaultAddCapabilities: null
requiredDropCapabilities: null
groups: null
EOF
```

Now we can proceed to configure the tests to run, first create a namespace for the test to run:

```
None
oc create project cilium-test
```

Deploy the checks with the command:

Red Hat

```None
oc apply -n cilium-test -f
https://raw.githubusercontent.com/cilium/cilium/1.16.2/examples/kubernetes/connect
ivity-check/connectivity-check.yaml
```

This will configure a series of deployments using various connectivity paths to connect to each other. Connectivity paths include with and without service load-balancing and various network policy combinations. The pod name indicates the connectivity variant and the readiness and liveness gate indicates the success or failure of the test:

```None
oc get pods -n cilium-test
NAME                                                    READY   STATUS    RESTARTS
AGE
echo-a-568cb98744-tlvwv                                 1/1     Running   0
67s
echo-b-64db4dfd5d-q8kpj                                 1/1     Running   0
67s
echo-b-host-6b7bb88666-qnhz4                            1/1     Running   0
67s
host-to-b-multi-node-clusterip-6cfc94d779-5v2x7         1/1     Running   0
66s
host-to-b-multi-node-headless-5458c6bff-2v7m7           1/1     Running   0
66s
pod-to-a-allowed-cnp-55cb67b5c5-ltclc                   1/1     Running   0
66s
pod-to-a-c9b8bf6f7-z4k2h                                1/1     Running   0
66s
pod-to-a-denied-cnp-85fb9df657-ndg2n                    1/1     Running   0
66s
pod-to-b-intra-node-nodeport-55784cc5c9-t42kj           1/1     Running   0
66s
pod-to-b-multi-node-clusterip-5c46dd6677-jgzvf          1/1     Running   0
66s
pod-to-b-multi-node-headless-748dfc6fd7-2ggvq           1/1     Running   0
66s
pod-to-b-multi-node-nodeport-f6464499f-84t92            1/1     Running   0
66s
pod-to-external-1111-96c489555-srcvb                    1/1     Running   0
66s
pod-to-external-fqdn-allow-google-cnp-5f747dfc7-8jsxw   1/1     Running   0
66s
```

Existing OpenShift cluster deployment

As organizations seek to enhance networking capabilities within their OpenShift clusters, transitioning from default Container Network Interface (CNI) plugins like OpenShiftSDN (pre-4.12) or OVN-Kubernetes (default from 4.12) to Cilium has become a consideration.

Support considerations

While Cilium has achieved Red Hat OpenShift CNI certification, it's important to note that Red Hat's official support for CNI migration is limited. Organizations are advised to consult with both Red Hat and Isovalent support and thoroughly assess the implications before proceeding with such migrations.

Migration Approach

A typical migration process involves:

1. **Preparation**: Assessing the current cluster state, backing up configurations, and understanding the implications of the migration.
2. **Disabling the Existing CNI**: Disabling the current CNI plugin to prevent conflicts.
3. **Deploying Cilium**: Installing Cilium using the appropriate Operator and manifests, ensuring configurations align with cluster requirements.
4. **Validation**: Testing network functionality, verifying pod communication, and ensuring that services operate as expected.

It's crucial to recognize that this process may involve downtime and requires thorough testing in a non-production environment before any production deployment.

Migration to Cilium

1. Preparation

    - Backup the etcd database and current network configurations.

- Review the cluster's current CNI plugin and document all custom network policies and configurations.
- Determine whether Kube-Proxy is deployed in the environment and identify the settings required to use it in combination with Cilium.
- Configure the manifest `cluster-network-07-cilium-ciliumconfig.yaml` with your necessary files, as part of the Cilium manifest folder that will be applied to the cluster.

2. Disabling the Network Operator and Existing CNI

- Scale down the Cluster Network Operator (CNO) responsible for managing the current CNI to disable it.

```Shell
oc scale deployment -n openshift-network-operator network-operator --replicas=0
```

- Remove the `applied-configuration` from the Network Operator, once you have verified there are no associated pods for the Network Operator running.

```Shell
oc delete configmap applied-cluster -n openshift-network-operator
```

- Where necessary the below command stops the `openshift-machine-operator` from rebooting nodes when there is a material configuration change. This allows you to schedule node reboots when most appropriate during the migration. This configuration will be re-enabled at the end of the migration.

```Shell
oc patch --type=merge --patch='{"spec":{"paused":true}}' mcp/master
oc patch --type=merge --patch='{"spec":{"paused":true}}' mcp/worker
```

3. Update the Cluster to use Cilium as the CNI
   - Update the `network.config` and `network.operator` objects to match the same pod CIDR as defined in the Cilium manifests, set Cilium as the CNI.
   - In the `network.operator` object, set `deployKubeProxy` to false as necessary

  - Note: If Kube-Proxy is disabled, Cilium's Kube Proxy Replacement feature must be set to `strict`
4. Install Cilium
   - Apply the Cilium manifests to the cluster
   - Wait for the Cilium pods to deploy in the `cilium` namespace
   - Scale the `network-operator` to re-enable its functionality and clear the cluster version operator, so that it returns to a managed state

```
None
oc patch clusterversions version --type=merge --patch
'{"spec":{"overrides":null}}'
```

5. Remove the paused state on the `machineconfigpool` objects that manage the nodes in your cluster, and reboot the OpenShift nodes as per their platform specific steps.

Post-Migration Considerations

Testing and Validation:
- Run the `cilium connectivity tests -n cilium` command to deploy the Cilium network connectivity workloads. (Additional SCC permissions may be required for these tests to run)
- Check all Cluster Operators are in a healthy state.

Policy Translation:
- Translate existing network policies to Cilium's format, ensuring that security postures are maintained.

Monitoring and Observability:
- Leverage Cilium's Hubble for real-time visibility into network traffic and policy enforcement.

Performance Tuning:
- Adjust Cilium's configurations to optimize performance based on workload requirements.

Red Hat

Advanced configuration

BGP control plane in Isovalent Networking for Kubernetes

Isovalent's BGP Control Plane allows clusters to advertise PodCIDR, Service IPs, and LoadBalancer IPs to external routers. This enables external access to workloads and integration with existing network infrastructure.

Note: Enterprise BGP control plane implements all features which are available in open-source BGP Control Plane. It also supports additional features like egress gateway IP advertisement and SRv6 L3VPN control plane integration.

Enabling the BGP Control Plane

To enable BGP, update the `CiliumConfig` resource:

None
```
oc edit ciliumconfig cilium-enterprise -n cilium
```

Add the following to the `spec.cilium` section:

None
```
enterprise:
  featureGate:
    approved:
    - EnterpriseBGPControlPlane
  bgpControlPlane:
    enabled: true
```

Note: see the above section if you want to use secrets for BGP in a different namespace.
After applying the change, restart the Cilium Operator and Cilium Agent:

None
```
oc rollout restart deployment/cilium-operator -n cilium
oc rollout restart daemonset/cilium -n cilium
```

Verify BGP settings:

Red Hat

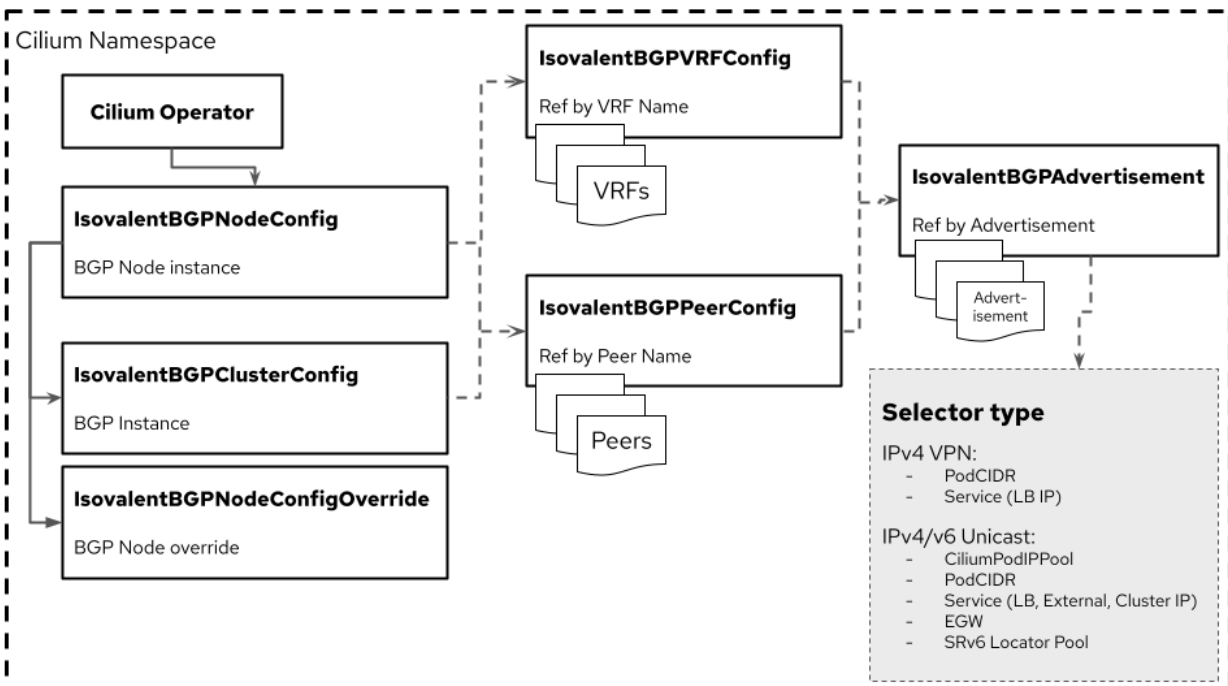
```
None
oc logs -n cilium deploy/cilium-operator | grep "bgp"
oc logs -n cilium ds/cilium | grep "bgp-control-plane"
```

BGP Control Plane Resources

Cilium manages BGP using custom resources that define BGP clusters, peers, advertisements, and overrides.

Resource	Description
`IsovalentBGPClusterConfig`	Defines BGP instances and peers for nodes.
`IsovalentBGPPeerConfig`	Defines peer configurations (ASN, passwords, timers).
`IsovalentBGPAdvertisement`	Defines peer configurations (ASN, passwords, timers).
`IsovalentBGPNodeConfigOverride`	Allows per-node BGP customizations.

The following diagram shows the relationship between the resources listed above:



Configuring BGP Peering

Define a BGP Cluster Configuration (IsovalentBGPClusterConfig) to specify autonomous system numbers (ASN) and peer information.

Example: BGP Cluster Configuration

The following example of an IsovalentBGPCluster configuration will be applied to the nodes with the label rack=rack0. The configuration has one BGP instance with the name instance-65001.

```
apiVersion: isovalent.com/v1alpha1
kind: IsovalentBGPClusterConfig
metadata:
  name: cilium-bgp
spec:
  nodeSelector:
    matchLabels:
      rack: rack0
  bgpInstances:
  - name: "instance-65001"
    localASN: 65001
    peers:
    - name: "peer-65001-tor1"
      peerASN: 65001
      peerAddress: fd00:10:0:0::1
      peerConfigRef:
        name: "cilium-peer"
```

- localASN: The ASN assigned to the Cilium node.
- peerASN: The ASN of the BGP router.
- peerAddress: The IP address of the BGP router.

BGP Peer Configuration

The IsovalentBGPPeerConfig defines peer-specific settings such as timers, authentication, and route advertisements.

Example: BGP Peer Configuration

```
apiVersion: isovalent.com/v1alpha1
kind: IsovalentBGPPeerConfig
metadata:
  name: cilium-peer
spec:
  bfdProfileRef: tor-bfd-profile
  timers:
   connectRetryTimeSeconds: 12
    holdTimeSeconds: 9
    keepAliveTimeSeconds: 3
  authSecretRef: bgp-auth-secret
```

Red Hat

```
families:
  - afi: ipv4
    safi: unicast
    advertisements:
      matchLabels:
        advertise: "bgp"
```

- bdfProfileRef: Enable BFD for BGP peers by referencing a BFD profile name
- holdTimeSeconds: Time before a peer is considered down.
- authSecretRef: used to configure an RFC-2385 TCP MD5 password on the session with the BGP peer, which references this configuration.
- Families: used to list AFI (Address Family Identifier), SAFI (Subsequent Address Family Identifier) pairs, and advertisement selector.

BGP Advertisements

Cilium supports advertising Pod CIDRs, Service IPs, and LoadBalancer IPs. The `advertisements` label selector defined in the `families` field of a peer configuration may match with one or more of the `IsovalentBGPAdvertisement` resources.

Advertisement Type	Description
`PodCIDR`	Advertises Kubernetes pod IP ranges.
`CiliumPodIPPool`	Advertises specific IP pools for Cilium-managed pods.
`Service`	Advertises ClusterIP, ExternalIP, and LoadBalancerIP.

Example: Advertising Pod CIDRs

The BGP Control Plane in Cilium can advertise the Pod CIDR prefixes of the nodes, enabling direct pod-to-pod connectivity across nodes and clusters without relying on overlay networks. This removes the need for encapsulation mechanisms like VXLAN or Geneve, improving performance, simplifying operations, and providing more predictable network paths.

Red Hat

```
None
apiVersion: isovalent.com/v1
kind: IsovalentBGPAdvertisement
metadata:
  name: tor-advertisements
  labels:
    advertise: tor
spec:
  advertisements:
    - advertisementType: "PodCIDR"
```

Example: Advertising Service Virtual IPs

In Kubernetes, a Service can have multiple virtual IP addresses, such as `.spec.clusterIP`, `.spec.clusterIPs`, `.status.loadBalancer.ingress[*].ip` or `.spec.externalIPs`.

The BGP control plane can advertise the virtual IP address of the Service to BGP peers. This allows you to directly access the Service from outside the cluster.

To advertise the service virtual IPs, specify the `.advertisementType` field to `Service` and the `.service.addresses` field to `LoadBalancerIP`, `ClusterIP` or `ExternalIP`.

By default, the Cilium BGP Control Plane advertises exact routes for the service VIPs (/32 or /128 prefixes). You can modify advertised prefix length with `.service.aggregationLength` field. This is especially useful in environments with a large number of services, where advertising many exact routes can consume valuable TCAM (Ternary Content-Addressable Memory) space on hardware switches and routers. Aggregating routes helps reduce the routing table size and optimizes hardware resource usage.

The `.selector` field is a label selector that selects Services matching the specified `.matchLabels` or `.matchExpressions`.

```
None
apiVersion: isovalent.com/v1alpha1
kind: IsovalentBGPAdvertisement
metadata:
```

```
  name: bgp-advertisements
  labels:
    advertise: bgp
spec:
  advertisements:
    - advertisementType: "Service"
      service:
        aggregationLength: 24
        addresses:
          - ClusterIP
          - ExternalIP
          - LoadBalancerIP
      selector:
        matchExpressions:
          - { key: bgp, operator: In, values: [ blue ] }
```

You can configure BGP path attributes for the prefixes advertised by Cilium BGP control plane using `attributes` field in `advertisements[*]`. There are two types of Path Attributes that can be advertised: `Communities` and `LocalPreference`.

Here is an example configuration of the `IsovalentBGPAdvertisement` resource that advertises pod prefixes with the community value of "65000:99" and local preference of 99.

```
apiVersion: isovalent.com/v1alpha1
kind: IsovalentBGPAdvertisement
metadata:
  name: bgp-advertisements
  labels:
    advertise: bgp
spec:
  advertisements:
    - advertisementType: "PodCIDR"
      attributes:
        communities:
          standard: [ "65000:99" ]
        localPreference: 99
```

Customizing BGP for Specific Nodes

The `IsovalentBGPNodeConfigOverride` can be used to override some of the auto-generated configuration on a per-node basis.

Example: Setting Router ID for a Specific Node

The example below demonstrates a `IsovalentBGPNodeConfigOverride` resource that sets the `routerID` and `localAddress` used in each peer for the node `bgpv2-cplane-dev-multi-homing-worker`.

```
apiVersion: isovalent.com/v1alpha1
kind: IsovalentBGPNodeConfigOverride
metadata:
  name: bgpv2-cplane-dev-multi-homing-worker
spec:
  bgpInstances:
    - name: "instance-65000"
      routerID: "192.168.10.1"
      localPort: 1790
      peers:
        - name: "peer-65000-tor1"
          localAddress: fd00:10:0:2::2
        - name: "peer-65000-tor2"
          localAddress: fd00:11:0:2::2
```

The name of `IsovalentBGPNodeConfigOverride` resource must match the name of the node for which the configuration is intended. Similarly, the names of the BGP instance and peers must match with what is defined under `IsovalentBGPClusterConfig`.

This is a per-node configuration. Cilium also supports BGP unnumbered peering, which enables auto-discovery of directly connected neighbors via link-local IPv6, removing the need to statically configure peer addresses in dynamic or large-scale environments. To use this feature, configure the `interface` field instead of `peerAddress` in your peer definition.

Below is an example of using the interface field for a BGP unnumbered configuration:

```None
apiVersion: isovalent.com/v1alpha1
kind: IsovalentBGPClusterConfig
metadata:
  name: cilium-bgp
spec:
  nodeSelector:
    matchLabels:
      rack: rack0
  bgpInstances:
  - name: "instance-65001"
    localASN: 65001
    peers:
    - name: "peer-65001-tor1"
      interface: eth0
      peerConfigRef:
        name: "cilium-peer"
```

Verifying and Troubleshooting BGP

The running state of the BGP Control Plane can be observed using the Cilium CLI, a command-line tool for interacting with the Cilium agent and its components. The CLI provides commands for inspecting policy enforcement, service routing, BGP sessions, and more.

If not already installed, the Cilium CLI can be downloaded from the official Isovalent Enterprise documentation and is available for Linux, macOS, and Windows.

Once installed, use the CLI to inspect the status of BGP peers, routing tables, and control plane configurations directly from within your cluster environment.

Check BGP Peering State

`cilium bgp peers` command displays current peering states from all nodes in the kubernetes cluster.

In the following example, peering status is displayed for two nodes in the cluster.

Red Hat

```
None
$ cilium bgp peers
Node                                  Local AS   Peer AS   Peer Address
Session State   Uptime   Family          Received   Advertised
bgpv2-cplane-dev-service-control-plane   65001      65000     fd00:10::1
established     33m26s   ipv4/unicast   2          2

ipv6/unicast   2          2
bgpv2-cplane-dev-service-worker          65001      65000     fd00:10::1
established     33m25s   ipv4/unicast   2          2
```

Using this command, you can validate that the BGP session state is `established` and an expected number of routes are being advertised to the peers.

Check Advertised Routes

`cilium bgp routes` command displays detailed information about local BGP routing tables and per peer advertised routing information.

In the following example, the local BGP routing table for IPv4/Unicast address family is shown for two nodes in the cluster.

```
None
$ cilium bgp routes available ipv4 unicast
Node                                  VRouter   Prefix        NextHop   Age
Attrs
bgpv2-cplane-dev-service-control-plane   65001     10.1.0.0/24   0.0.0.0   46m45s
[{Origin: i} {Nexthop: 0.0.0.0}]
bgpv2-cplane-dev-service-worker          65001     10.1.1.0/24   0.0.0.0   46m45s
[{Origin: i} {Nexthop: 0.0.0.0}]
```

Similarly, you can inspect per peer advertisements using the following command.

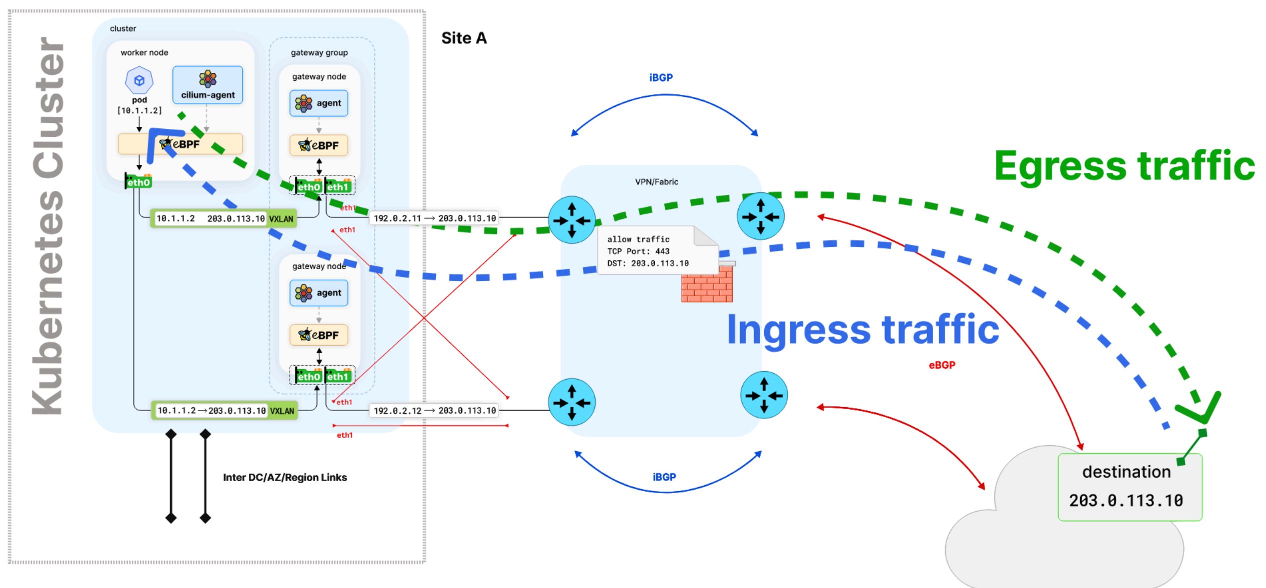
```
None
$ cilium bgp routes advertised ipv4 unicast
Node                                  VRouter   Peer         Prefix
NextHop          Age      Attrs
bgpv2-cplane-dev-service-control-plane   65001     fd00:10::1   10.1.0.0/24
fd00:10:0:1::2   47m0s    [{Origin: i} {AsPath: 65001} {Communities: 65000:99}
{MpReach(ipv4-unicast): {Nexthop: fd00:10:0:1::2, NLRIs: [10.1.0.0/24]}}]
```

```
bgpv2-cplane-dev-service-worker          65001     fd00:10::1   10.1.1.0/24
fd00:10:0:2::2   47m0s    [{Origin: i} {AsPath: 65001} {Communities: 65000:99}
{MpReach(ipv4-unicast): {Nexthop: fd00:10:0:2::2, NLRIs: [10.1.1.0/24]}}]
```

Further Considerations: Traffic Engineering with Cilium

The BGP Control Plane can be implemented as part of a wider Traffic Engineering (TE) strategy with Cilium deployed on Kubernetes clusters for several use cases:

- **Failover scenarios**: Managing situations where one or more subsystems or devices become unavailable.
- **Security and identity**: Understanding who sends traffic to whom.
- **Application and network performance**: Optimizing the path based on resource considerations such as throughput and latency.
- **Cost optimization**: Selecting specific paths and services to reduce costs.



See this deep-dive technical whitepaper for further information.

Multi-Pool IPAM

The Multi-Pool IPAM mode supports allocating PodCIDRs from multiple different IPAM pools, depending on properties of the workload defined by the user, e.g. annotations.

Enabling Multi-Pool IPAM in Cilium

To enable Cilium-native Multi-Pool IPAM, it must be configured in the CiliumConfig resource, `cluster-network-07-cilium-ciliumconfig.yml`, during cluster creation. It is not recommended to change the IPAM mode of a cluster after it is created:

```
enterprise:
  featureGate:
    approved:
    - IPAMMultiPool
routingMode: native
endpointRoutes:
  enabled: true
ipam:
  mode: multi-pool
  operator:
    autoCreateCiliumPodIPPools:
      default:
        ipv4:
          cidrs: ["10.128.0.0/46"]
          maskSize: 24
ipv4NativeRoutingCIDR: "10.0.0.0/8"
bpf:
  masquerade: true
kubeProxyReplacement: "true"
```

Configuration Breakdown:

Setting	Description
featureGate.approved: IPAMMultiPool	Enables multi-pool IPAM support in Cilium (Beta feature).
autoCreateDefaultPodNetwork: true	Creates a default pod network automatically.
autoDirectNodeRoutes: true	Enables multi-pool ipam-aware direct routing between nodes.
routingMode: native	Uses native routing mode, required for multi-pool ipam.
endpointRoutes.enabled: true	Ensures endpoint-specific routes are created per network.

Red Hat

Setting	Description
ipv4NativeRoutingCIDR: "10.0.0.0/8"	Excludes this CIDR from IP masquerading. This typically will be a superset of your various IPAM Pod Pools subnets.
ipam.mode: multi-pool	Uses multi-pool IP address management for separate networks.
ipam.mode.operator.autoCreateCiliumPodIPPools	Create IPAM pools automatically; this example shows the default pool being created. Additional arrays can be added to create further Pod IP Pools automatically.

Once the changes are applied, restart the Cilium DaemonSet:

```
None
kubectl rollout restart daemonset/cilium -n cilium
```

Note: if you are using IP CIDRs across different classes for the IPAM Pod Pools, then use the eBPF Masq Agent configuration.

Defining Pod Networks

By default, Cilium will create a default pod network, which corresponds to the standard Kubernetes network, as specified in the `ipam.operator.autoCreateCiliumPodIPPools.default.*` values.

You can create additional pod networks using the `CiliumPodIPPool` CRD.

Example: Creating a Secondary Network

The below example creates a new Pod IP Pool called mars, this will allocate /27-sized pod subnets from `10.20.0.0/16`.

```
cat <<EOF | oc apply -f -
apiVersion: cilium.io/v2alpha1
kind: CiliumPodIPPool
metadata:
  name: mars
```

```
spec:
 ipv4:
  cidrs:
  - 10.20.0.0/16
  maskSize: 27
EOF
```

Attaching Pods and VMs to Specific Pools

Pods can be assigned to a specific IP pool using annotations.

```
None
metadata:
  annotations:
       ipam.cilium.io/ip-pool: mars
```

- If no annotation is present, the default pool is used.
- If the namespace has an `ipam.cilium.io/ip-pool` annotation, all pods within that namespace will inherit the setting.

Example: Assigning a VM to different Pod IP Pools

VM Attached to Default Network

```
# Some fields are omitted for brevity
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: nginx-vm-default-net
spec:
  running: true
  template:
    metadata:
      labels:
        network.kubevirt.io/headlessService: headless
      annotations:
       ipam.cilium.io/ip-pool: default
    spec:
      domain:
        devices:
          autoattachPodInterface: false
          disks: []
          interfaces:
            - masquerade: {}
              name: default
      networks:
```

```
      - name: default
        pod: {}
    subdomain: headless
```

VM Attached to Secondary Pool (mars)

```
# Some fields are omitted for brevity
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: nginx-vm-default-net
spec:
  running: true
  template:
    metadata:
      labels:
        network.kubevirt.io/headlessService: headless
      annotations:
       ipam.cilium.io/ip-pool: mars
    spec:
      domain:
        devices:
          autoattachPodInterface: false
          disks: []
          interfaces:
            - masquerade: {}
              name: default
      networks:
        - name: default
          pod: {}
      subdomain: headless
```

How `CiliumPodIPPool` is Assigned to the `virt-launcher` Pod in KubeVirt

In KubeVirt, each virtual machine (VM) runs inside a Kubernetes pod called a `virt-launcher` pod. This pod acts as a wrapper for the VM, handling network interfaces and IP allocation. With Cilium Multi-Pool IPAM, the `virt-launcher` pod is treated like any other Kubernetes pod, and its IP is assigned from a `CiliumPodIPPool.`

IP Allocation in Different KubeVirt Networking Modes

Red Hat Openshift Virtalization supports both KubeVirt Masquerade network mode, for flexible IP allocation:

- Masquerade Mode (Default for VMs)
    - The `virt-launcher` pod gets an IP from CiliumPodIPPool.

- The VM uses a private IP (e.g., 192.168.122.0/24), and all outgoing traffic is NAT-ed using the pod's IP.

Using this mode and Isovalent Networking for Kubernetes as the CNI, the `CiliumIPPodPool` address range can be advertised using the BGP Control Plane. (See section further below.)

Validating Multi-Pool IPAM

Check Allocated IP Pools

Begin by verifying that the expected `CiliumPodIPPools` have been created and are active in the cluster. This ensures that the necessary IP address blocks are available for pod and VM assignment. Use the following command to list the defined pools and inspect their configuration, including CIDR ranges and availability.

```
None
$ oc get CiliumPodIPPool
NAME   AGE
default   6d
mars   107s
```

Check Node IP Allocations

Once the pools are confirmed, the next step is to validate that each node has been allocated IPs from the correct pool(s). This allocation enables the Cilium agent on each node to assign IPs from the pool to new workloads. Reviewing node-level allocations helps confirm that IPs are being correctly distributed in alignment with pool configuration and node selectors.

```
None
$ oc get ciliumnodes -o jsonpath='{.items[*].spec.ipam.pools.allocated}' | jq
[
  {
       "cidrs": [
       "10.128.4.0/24"
       ],
       "pool": "default"
  },
  {
       "cidrs": [
       "10.20.0.64/27"
```

Red Hat

```
      ],
      "pool": "mars"
  }
]
```

Check Running Pods and Assigned IPs

Validate that running pods, including `virt-launcher` pods for KubeVirt virtual machines, are receiving IP addresses from the correct pool. This step ensures the pool-to-workload mapping is functioning as expected. It's important to correlate pod labels, namespaces, and IP assignments to verify that workload-specific IPAM logic is applied correctly.

```
None
$ oc get pods -o wide
NAME                                       READY   STATUS              RESTARTS   AGE
IP     NODE                           NOMINATED NODE   READINESS GATES
virt-launcher-nginx-vm-mars-net-bphn9   1/1     Running             0
13s    10.20.0.84    ocp2-vs7tm-worker-0-5mb8r    <none>              1/1
```

Routing and Connectivity

When using Multi-Pool IPAM, the BGP Control Plane can be used to advertise these address ranges to the wider network.

Specify the `advertisementType` field to `CiliumPodIPPool`. The `selector` field is a label selector that selects `CiliumPodIPPool` matching the specified `.matchLabels` or `.matchExpressions`.

```
---
apiVersion: cilium.io/v2alpha1
kind: CiliumPodIPPool
metadata:
  name: default
  labels:
    pool: blue
---
apiVersion: cilium.io/v2alpha1
kind: CiliumBGPAdvertisement
metadata:
  name: pod-ip-pool-advert
```

```
  labels:
    advertise: bgp
spec:
  advertisements:
    - advertisementType: "CiliumPodIPPool"
      selector:
        matchLabels:
          pool: "blue"
```

- The CIDR Range must be allocated to a `CiliumNode` resource, this happens when a workload using the IPAM Pool is running and available.

If you wish to announce *all* `CiliumPodIPPool` CIDRs within the cluster, a `NotIn` match expression with a dummy key and value can be used like this:

```
apiVersion: cilium.io/v2alpha1
kind: CiliumBGPAdvertisement
metadata:
  name: pod-ip-pool-advert
  labels:
    advertise: bgp
spec:
  advertisements:
    - advertisementType: "CiliumPodIPPool"
      selector:
        matchExpressions:
        - {key: somekey, operator: NotIn, values: ['never-used-value']}
```

Observing Multi IPAM Pool Traffic with Hubble

If Hubble is enabled, you can observe network flows for both Pods and VMs.

The below example output shows ICMP traffic between a Fedora virtual machine deployed into the default namespace and the external IP address 8.8.8.8 (commonly known as Google DNS Server).

The below output shows the use of the Hubble CLI. The Hubble CLI is a command-line tool used to observe real-time network traffic, policy decisions, and service communication within a Cilium-powered Kubernetes cluster. It can be downloaded from the Isovalent Enterprise documentation pages.

```
None
$ hubble observe --protocol icmp
```

Red Hat

```
Apr 10 14:07:28.498: default/virt-launcher-fedora-h2njd (ID:36294) -> 8.8.8.8
(world) to-network FORWARDED (ICMPv4 EchoRequest)
Apr 10 14:07:28.506: default/virt-launcher-fedora-h2njd (ID:36294) <- 8.8.8.8
(world) to-endpoint FORWARDED (ICMPv4 EchoReply)
```

You will note that the name shown is that of the `virt-launcher` pod associated with the VM, this is because of the unique relationship that is created to establish connectivity for the VM in a KubeVirt environment and its deployed components.

Red Hat

Cilium Network Policies and Observability for Virtual Machines

Cilium enforces network policies for VMs running in KubeVirt by utilizing the same security identity model applied to Kubernetes pods. Each VM operates within a `virt-launcher` pod, which serves as the enforcement point for network policies. By integrating Cilium Network Policies, administrators can effectively control and secure network traffic to and from the VMs. This approach ensures that VMs benefit from the same network policy enforcement mechanisms as other Kubernetes workloads.

Complementing this, Hubble, Cilium's observability platform, provides deep visibility into the network traffic involving these VMs. By leveraging eBPF, Hubble captures detailed flow information, enabling administrators to monitor, troubleshoot, and audit network interactions seamlessly. This integration ensures that both policy enforcement and observability are consistently applied across all workloads, whether containerized or virtualized.

Concepts

Cilium assigns a Security Identity to workloads pod based on their labels. This identity is used in policy decisions across the cluster.

- Labels such as `kubevirt.io/domain`, `app.kubernetes.io/name`, or custom tenant identifiers can be used in policies.

- These labels are treated just like pod labels when calculating subject selectors and peer selectors.

For example:

```
None
endpointSelector:
  matchLabels:
      kubevirt.io/domain: vm-nginx
```

Red Hat

Layer 3

Cilium's Layer 3 (L3) policies define which endpoints are allowed to communicate with each other based on identity or addressing. These policies form the foundation of Zero Trust connectivity and can be applied using several mechanisms:

- Endpoints Based: Policies are defined using labels assigned to Cilium-managed workloads. This method avoids reliance on IP addresses and ensures that policy is fully decoupled from addressing, improving portability and security.
- Services based: Uses the orchestration system's Service abstraction (e.g., Kubernetes Services) to target backend endpoints dynamically. This allows communication rules to be written without hardcoding IP addresses, even when the destination is not Cilium-managed.
- Entities Based: Entities refer to logical groupings such as `cluster`, `world`, `host`, or `remote-node`. They allow policies to define communication with external or internal systems without explicit IP references, supporting policy decisions like "allow egress to the internet" or "block access to the host."
- Node based: An extension of the `remote-node` entity, this allows individual nodes to be assigned identities and used in policies. This can be useful for restricting VM access to specific infrastructure nodes or enforcing tighter east-west controls.
- IP/CIDR based: For external services not managed by Kubernetes or Cilium, policies can be written using static IP addresses or CIDR blocks. While powerful, this method should be used only when necessary, as it introduces coupling to fixed network assignments.
- DNS based: Enables policies that match traffic by FQDNs rather than IPs. DNS queries are intercepted via a proxy, and responses are used to resolve destination IPs dynamically. While more flexible than hardcoding IPs, DNS-based policies still share some of the same limitations as CIDR-based rules, including dependency on DNS TTLs.

The following example illustrates how to use a simple egress rule to allow communication to endpoints with the label `app=backend` from endpoints with the label `kubevirt.io/domain=vm-nginx`.

This approach illustrates the label-based, identity-aware enforcement model in action—even for virtual machines—allowing seamless and secure policy management across mixed workload types.

```
None
apiVersion: "cilium.io/v2"
kind: CiliumNetworkPolicy
metadata:
  name: "l3-egress-rule"
spec:
  endpointSelector:
    matchLabels:
      kubevirt.io/domain: vm-nginx
  egress:
  - toEndpoints:
    - matchLabels:
        role: backend
```

Layer 4

Layer 4 policies in Cilium build on top of Layer 3 identity-based rules by enabling control over specific protocols and ports. This provides granular traffic filtering and is critical for enforcing least privilege principles in hybrid environments where both containers and virtual machines coexist.

When used with KubeVirt, L4 policies apply to the `virt-launcher` pod interfaces that encapsulate each virtual machine. This ensures that traffic to or from a VM is not just permitted based on identity, but also restricted by allowed transport-layer operations, such as TCP or UDP on specific ports.

This example permits ingress traffic on TCP port 443 (HTTPS) to a VM with the label `kubevirt.io/domain=vm-nginx`, but only from endpoints with the label `role=frontend`.

This rule ensures that only authenticated frontend workloads may initiate HTTPS traffic to the virtual machine, minimizing exposure and aligning with Zero Trust best practices.

```
None
apiVersion: "cilium.io/v2"
```

```
kind: CiliumNetworkPolicy
metadata:
  name: "allow-http-get-to-vm"
spec:
  endpointSelector:
    matchLabels:
      kubevirt.io/domain: vm-nginx
  ingress:
  - fromEndpoints:
    - matchLabels:
        Role: frontend
    toPorts:
    - ports:
      - port: '80'
        protocol: TCP
```

Layer 7

Layer 7 (L7) policies in Cilium enable deep packet inspection and application-level filtering for traffic based on protocols such as HTTP, DNS, Kafka, and TLS. These policies allow you to define what actions are permitted within a session, such as specific HTTP methods, URL paths, DNS queries, or message patterns.

When applied to virtual machines in KubeVirt, L7 enforcement occurs at the `virt-launcher` pod network interface, without requiring any software inside the VM. This means that even traditional workloads can benefit from modern, zero-trust security controls driven by protocol-level awareness.

The below example builds on the above from the Layer 4 section, but now includes HTTP Method (GET) and Path (/public).

All other requests – different methods, paths, or clients – will be dropped by default, reinforcing a least-privilege access model.

None

```
apiVersion: "cilium.io/v2"
kind: CiliumNetworkPolicy
metadata:
  name: "allow-http-get-to-vm"
spec:
  endpointSelector:
    matchLabels:
      kubevirt.io/domain: vm-nginx
  ingress:
  - fromEndpoints:
    - matchLabels:
        Role: frontend
    toPorts:
    - ports:
      - port: '80'
        protocol: TCP
      rules:
        http:
        - method: GET
          path: "/public"
```

Using Cilium with KubeVirt, organizations can apply the same identity-based security model to virtual machines as they do for Kubernetes pods. This unified approach enables:

- Layer 3 policies to control who can talk to whom based on labels and workload identities, not IP addresses.
- Layer 4 policies to enforce protocol- and port-specific rules, such as limiting VM exposure to only required TCP/UDP ports.
- Layer 7 policies to define application-aware controls, filtering traffic by HTTP methods, URL paths, DNS queries, and more.

Cilium attaches eBPF programs to the virt-launcher pod’s network interface, making the VM's traffic observable and enforceable—without requiring an agent inside the guest OS. This enables Zero Trust segmentation, granular access control, and deep observability for all workloads, container or VM alike.

Red Hat

Troubleshooting

Use **Hubble** to trace traffic and identify whether packets are being allowed or dropped due to a policy:

```
None
hubble observe --to-label vm.kubevirt.io/name=nginx-vm-mars-net  --from-label
app=netshoot-a
Apr 16 15:50:47.870: default/netshoot-a-56dc8f556d-qglg7:56974 (ID:53220) ->
default/virt-launcher-nginx-vm-mars-net-px25q:80 (ID:25655) policy-verdict:all
EGRESS ALLOWED (TCP Flags: SYN)
Apr 16 15:50:47.870: default/netshoot-a-56dc8f556d-qglg7:56974 (ID:53220) ->
default/virt-launcher-nginx-vm-mars-net-px25q:80 (ID:25655) policy-verdict:L3-L4
INGRESS ALLOWED (TCP Flags: SYN)
Apr 16 15:50:57.793: default/netshoot-a-56dc8f556d-qglg7:41950 (ID:53220) ->
default/virt-launcher-nginx-vm-mars-net-px25q:80 (ID:25655) http-request FORWARDED
(HTTP/1.1 GET http://nginx-vm.default.svc.cluster.local/)
```

Look for:

- **DROP** verdicts (policy denying traffic)
- **FORWARDED** or **ALLOWED** events
- Source and destination **labels and IPs**
- Port and protocol-level decisions

This is particularly useful when combining **L3/L4 and L7 policies**, to confirm whether traffic was denied due to path, method, or port mismatches.

By viewing the JSON output of Hubble flows, for Type: Policy Verdicts, if a Cilium Network Policy is used to allow or deny the traffic flow, it will be captured in the JSON output.

Consider the following example;

We have deployed the following Cilium Network Policy which is designed to allow HTTP GET requests from pods labeled `app=netshoot-a` to a virtual machine (VM) managed by KubeVirt with the label `vm.kubevirt.io/name=nginx-vm-mars-net`. Specifically, it permits traffic on TCP port 80 with the HTTP method `GET` and the root (empty) path.

```
apiVersion: cilium.io/v2
kind: CiliumNetworkPolicy
metadata:
  name: allow-http-from-netshoot-a-to-nginx-vm
spec:
```

```
endpointSelector:
  matchLabels:
    vm.kubevirt.io/name: nginx-vm-mars-net
ingress:
- fromEndpoints:
  - matchLabels:
      app: netshoot-a
  toPorts:
  - ports:
    - port: "80"
      protocol: TCP
    rules:
      http:
      - method: GET
```

To verify that this policy is correctly allowing traffic, use the following Hubble command:

```None
hubble observe -f \
  --type policy-verdict \
  --to-label vm.kubevirt.io/name=nginx-vm-mars-net \
  --from-label app=netshoot-a \
  -o json | jq '.flow.ingress_allowed_by'
```

Command Breakdown:

- `hubble observe`: Initiates the observation of network flows.
- `-f` or `--follow`: Continuously streams the flow logs as they occur, providing real-time monitoring.
- `--type policy-verdict`: Filters the observed flows to include only those that have a policy verdict, i.e., flows that have been evaluated against network policies.
  `--to-label vm.kubevirt.io/name=nginx-vm-mars-net`: Restricts the observation to flows where the destination has the specified label, targeting the VM named `nginx-vm-mars-net`.
- `--from-label app=netshoot-a`: Restricts the observation to flows originating from sources with the specified label, targeting pods labeled `app=netshoot-a`.
- `-o json`: Outputs the observed flows in JSON format, facilitating structured data processing.
- `| jq '.flow.ingress_allowed_by'`: Pipes the JSON output to `jq`, a command-line JSON processor, to extract the `ingress_allowed_by` field from each flow, revealing which policies permitted the ingress traffic.

Red Hat

This output indicates that the flow was allowed by the `CiliumNetworkPolicy` named `allow-http-from-netshoot-a-to-nginx-vm` in the `default` namespace. The `labels` provide metadata about the policy, and the `revision` denotes the version of the policy at the time of enforcement.


```
{
        "name": "allow-http-from-netshoot-a-to-nginx-vm",
        "namespace": "default",
        "labels": [
        "k8s:io.cilium.k8s.policy.derived-from=CiliumNetworkPolicy",
        "k8s:io.cilium.k8s.policy.name=allow-http-from-netshoot-a-to-nginx-vm",
        "k8s:io.cilium.k8s.policy.namespace=default",
        "k8s:io.cilium.k8s.policy.uid=e3342d85-571c-4c7e-885f-c3d95e55311a"
        ],
        "revision": "8"
}
```


By using this command, you can verify in real-time which policies are permitting traffic between specific sources and destinations. This is particularly useful for:

- Validating that your network policies are functioning as intended.
- Troubleshooting unexpected traffic behavior.
- Auditing policy enforcement for compliance and security purposes.

Additional recommended resources:

- Cilium Hubble Cheat Sheet
    - A handy command sheet for using Hubble CLI with examples
- Kubernetes Network Policies Done the Right Way by Isovalent
    - This practical eBook covers Zero Trust, effective network policy design, and tools like Cilium and Hubble to help you protect and optimize your infrastructure
- Cilium Network Policy Deep Dive by Isovalent
    - This eBook provides deep dive insights and understanding of Cilium's network policy engine. This guide offers comprehensive insights into crafting, implementing, and managing network policies using Cilium

Red Hat

Authors

- Dean Lewis - Technical Marketing, Isovalent at Cisco
- David Szegedi - Field CTO Organization, Red Hat

Appendix .. . Change log

April 2025 - version 1.0.0

- *Initial release.*