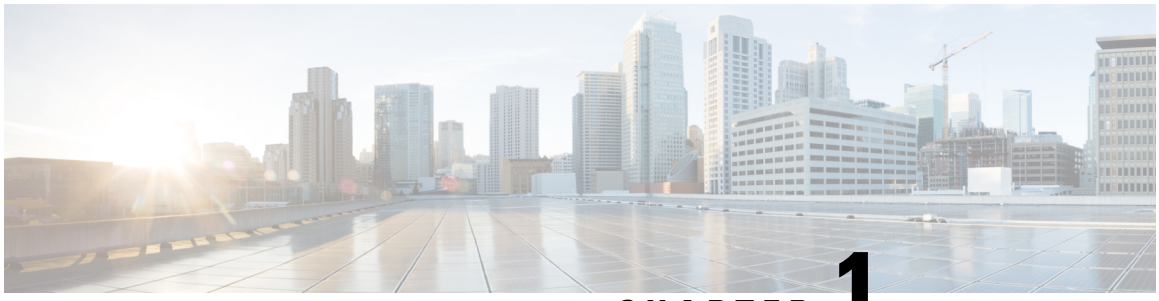
Unified Load Balancer Configuration and Administration Guide, Release 2026.02

First Published: 2026-04-23

Americas Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
http://www.cisco.com
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 527-0883

CONTENTS



CHAPTER 1

Introduction to Unified Load Balancer

This topic provides an overview of the Unified Load Balancer (ULB), highlighting its features, benefits, and limitations.

ULB overview

The Unified Load Balancer (ULB) is a Cisco proprietary node or network function (NF) that manages traffic distribution of incoming network traffic equally across all worker nodes and ensures high availability and reliability across network infrastructures.

- Developed using the Subscriber Microservices Infrastructure (SMI), a Cloud Native (CN) application.
- Delivered as a CN-based image and deployed as a CN application using the SMI Cluster Manager (SMI CM).
- Supports protocol applications utilizing UDP and TCP that require load balancing across multiple protocol pods.

ULB key features and capabilities

The ULB supports the following functions:

- Equal Traffic Distribution and Client IP Preservation:
  - Distributes incoming network traffic equitably across all worker nodes for Layer 4 protocols (UDP, TCP).
  - Maintains the original client IP address when distributing ingress traffic to backend pods.
- Direct Server Response (DSR) and Service IP Consistency:
  - Enables backend pods to reply directly to clients, bypassing the load balancer to reduce latency and load.

    - Ensures response packets have a consistent source IP address, matching the service IP address exposed for ingress.

- Dynamic Scaling and Real-Time Configuration Updates:
    - Seamlessly adds and removes worker nodes, automatically integrating new nodes into traffic distribution without downtime.
    - Dynamically incorporates new backend pods into the traffic routing scheme as they are deployed.
    - Triggers real-time updates to the load balancer's configuration with any changes to worker nodes or backend pods.
- Service IP Uniqueness and Consistency for Egress Traffic:
    - Maintains a unique service IP address for egress traffic, even when exiting through multiple worker nodes.
    - Uses the same service IP address for both ingress and egress requests, providing a consistent IP address to external systems.
- Response Routing and UDP Handling for Egress Traffic:
    - Accurately routes responses back to the backend pod that initiated the egress request, preserving the original client IP address.
    - Routes UDP protocol responses to backend pods, even if they did not originate the egress request.

For more information on configuring these features, see the ULB features and configuration, on page 15 section in this guide.

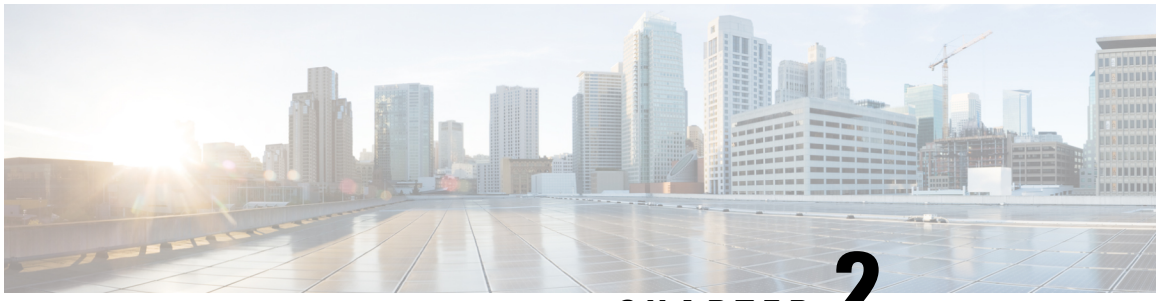
Benefits of using ULB

The ULB is crucial for the following reasons:

- Traffic management: ULB distributes incoming network traffic across multiple servers to ensure no single server is overwhelmed. This balanced traffic distribution prevents bottlenecks and enhances the overall performance and reliability of the network. This applies to Layer 4 (L4) protocols, such as UDP and TCP.
- High Availability: By distributing traffic among multiple servers, ULB ensures that the failure or unavailability of a single server does not impact the entire system. This redundancy is critical for maintaining continuous service availability to users.
- Scalability: ULB supports horizontal scaling, allowing the addition of more servers to handle increased traffic loads without disrupting existing services. This scalability is vital for accommodating growth and sudden spikes in traffic.
- Security: ULB can provide an additional layer of security by masking the internal server IP addresses and managing secure connections. This helps protect the backend infrastructure from direct exposure to potential threats.
- Performance optimization: By facilitating Direct Server Response (DSR), ULB reduces latency and the load on the load balancer node, thereby optimizing the performance of network communications.

Limitations

- EgressManagementPolicy for stateful applications without portRanges is only applicable for IPv4.
- EgressManagementPolicy (EMP) for stateful applications without portRanges configured currently works only for resources managed by StatefulSet workload.

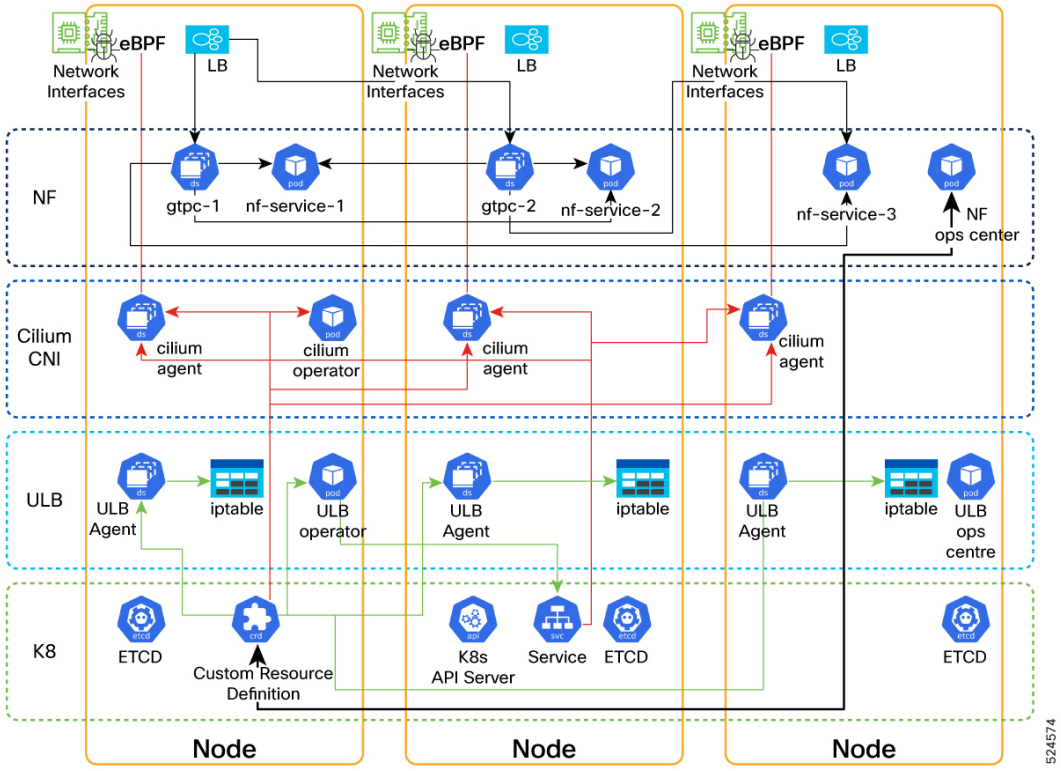


CHAPTER 2

ULB deployment architecture

The following diagram illustrates a typical deployment model of ULB.

Figure 1: Unified Load Balancer Deployment Architecture



ULB can be deployed both on-premises and cloud platform, providing a unified load-balancing abstraction that simplifies application usage and integration.

- On-Premise:
  - The service integrates with on-premise environments featuring leaf-spine network architecture, leveraging BGP and BFD protocols for robust route advertisement and rapid path failure detection.
- Cloud:

- For public cloud deployments, the service publishes routes with equal priority to facilitate faster convergence during failure events and maintains balanced traffic distribution across worker nodes.

Note Deployment of ULB on the cloud platform is not qualified in this release. Contact your Cisco account representative for more information.

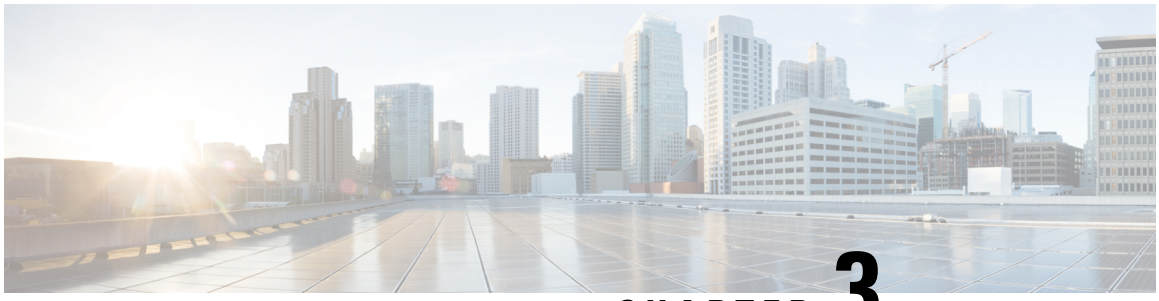
- Key components of ULB, on page 6

Key components of ULB

The ULB comprises the following components which run as pods in the Kubernetes cluster:

- Operator: This is a controller pod. It can run or get spawned on any of the K8s nodes. The Operator manages the necessary Kubernetes and Cilium resources associated with the Load Balancer and Egress Management Policy Custom Resources (CRs). The Operator automates deployment, scaling, and operational tasks for the resources it manages, ensuring that the state of the system matches the desired state expressed by the CRs.
- Agent: The ULB Agent is a DaemonSet that runs on each node in the K8s cluster. The ULB Agent manages the NAT table and RAW table entries on each node for the corresponding pods on that node that are subject to the Egress Management Policy. It ensures that traffic routing and policy enforcement are correctly applied at the node level.
- Ops Center: The ULB Operations (Ops) Center serves as the central point for configuration and upgrades of ULB components within the cluster. It helps to manage the lifecycle of ULB components, including deployment, monitoring, and maintenance activities.
- Load Balancer API: Implemented as a Kubernetes Custom Resource Definition (CRD), facilitating the creation, configuration, and management of load balancer instances within the Kubernetes cluster.
- Egress Management Policy API: Also implemented as a Kubernetes CRD, this API enables the definition and enforcement of egress policies for pods, ensuring that outbound traffic is controlled and adheres to specified rules.

ULB Operator and Agent contain Helm charts and Docker files (images), which allow conformance to industry standards for installing and managing products within Kubernetes. ULB Ops Center provides a stable CLI/API for operators to manage the product holistically. ULB Ops Center uses helm v3 to communicate directly to the Kubernetes API server to install, upgrade, query, and remove Kubernetes resources.



CHAPTER 3

Getting started

Requirements

To ensure optimal performance and compatibility, the following system requirements must be met for the ULB deployment.

- ULB should be installed on a container management platform based on Kubernetes.
- The Linux kernel version must be 5.4.x and above.
- Hosts with either AMD64 or AArch64 architecture.
- Packet forwarding should be enabled in the kernel, and reverse path filtering should be disabled (that is, rp_filter = 0) on each K8s node.
- Kubernetes version: v1.28.x and above.
- Cilium version: 1.17.9 and above.
- User-space utility program named ‘iptables’ should be installed to configure egress related source NAT IP packet filter rules on Ubuntu/Linux kernel Netfilter module.
- Ensure Kubernetes is installed and configured on the system.

Initial configurations for on-premise deployment

Make sure that the SMI Cluster Deployer is set up and SMI clusters are already deployed. The deployment of SMI clusters facilitates the installation of standalone Cilium CNI and supports various Cilium features and functionalities. For more information on deploying clusters, refer to the SMI documentation.

Install Cilium CNI

SMI supports Cilium CNI, which can be installed via the cluster deployer using the following CLI command:

configuration cni type cilium

Enable isovalent BGP and BFD CRDs

To enable BGP and BFD CRDs, configure the cluster deployer with the following CLI command:

configuration cilium enable-bgp true

When IsovalentBgpClusterConfig, IsovalentBgpPeerConfig, and IsovalentBgpAdvertisement CRs are created, Cilium deploys BGP routers, creates BGP sessions with configured peers, and advertises or unadvertises service IPs in response to service creation or deletion.

Enable isovalent egress-gateway CRD

To enable the egress-gateway CRD for stateful and stateless pods, use the following CLI command in the cluster deployer:

configuration cilium enable-egress-gateway true

Configure load balancer algorithm

To configure Cilium's load balancer algorithm, use the following CLI command:

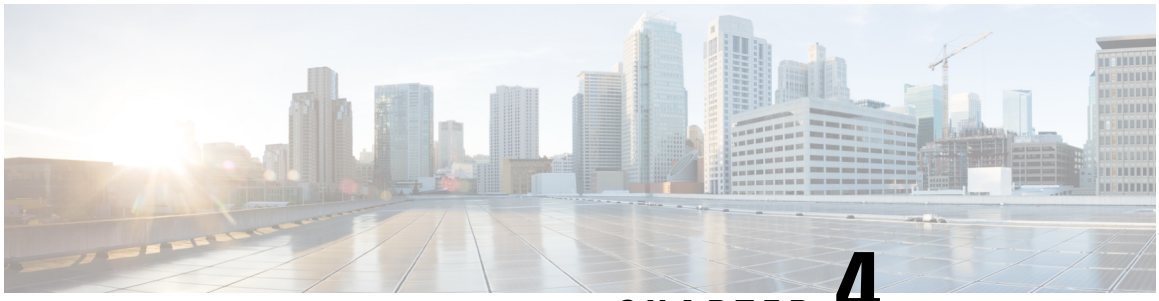
configuration cilium lb-algorithm { maglev | random }

Note Although 'maglev' is available as a load balancer algorithm, it is currently recommended to use the 'random' algorithm.

Enable legacy host routing

To enable Legacy Host Routing, which is disabled by default, configure the following command:

configuration cilium enable-legacy-host-routing true



CHAPTER 4

Deploy and configure ULB through Ops center

The installation process involves setting up the ULB on on-premise environment, configuring necessary components, and verifying the installation.

The ULB is deployed as CN application using the following image.

Image	Description
ulb.<release>.SPA.tgz	The ULB software image that is used to on-board the software in your deployment environment.

The ULB deployment and configuration procedure involves the following stages.

Procedure

Step 1 Deploy the ULB through the SMI Cluster Deployer.

Step 2 Configure the settings or customizations through the ULB Ops Center.

The Ops Center is based on the ConfD CLI. The ULB configuration includes the feature-specific configurations to provide highly efficient load balancing capabilities.

ULB Ops center

The Ops Center is a system-level infrastructure that provides the following functionality:

- A user interface to trigger a deployment of microservices with the flexibility of providing variable helm chart parameters to control the scale and properties of Kubernetes objects (deployment, pod, services, and so on) associated with the deployment.
- A user interface to push application-specific configuration to one or more microservices through Kubernetes configuration maps.

- A user interface to issue application-specific execution commands (such as show and clear commands). These commands:
  - Invoke some APIs in application-specific pods
  - Display the information returned on the user interface application

The ULB Ops Center serves as the central point for configuration and deployment of ULB components (ULB operator and agent) within the cluster. ULB Operator and Agent contain Helm charts and Docker files (images), which allows us to conform to industry standards around how to install and manage ULB within Kubernetes. ULB Ops Center provides a stable CLI/API for operators to manage the ULB in a holistic way. ULB OpsCenter uses helm v3 to directly communicate with Kubernetes API server to install, upgrade, query, and remove Kubernetes resources.

The ULB Ops Center allows you to configure the features such as Neighbor Discovery, Resilience and Traffic Convergence, and so on.

Prerequisites for ULB deployment

Before deploying ULB on the SMI layer:

- Ensure that all the virtual network functions (VNFs) are deployed.
- Run the SMI synchronization operation for the ULB Ops Center and Cloud Native Common Execution Environment (CN-CEE).
- Ensure that the node labels are configured as per the recommended pod deployment layout.
- Configure the external VIPs as per the ULB deployment requirement.
- Enable Istio for pod-to-pod traffic load balancing.

Deploy the ULB

Use this procedure to deploy the ULB in an on-premise environment, including preparation, deployment, and configuration steps.

This procedure covers the end-to-end deployment of the ULB, including image handling, Helm chart configuration, and log management for both operator and agent components.

Before you begin

Before you begin the deployment of ULB in On-Premise environment, make sure the following prerequisites are met.

- Ensure the Kubernetes cluster is installed and configured.
- Linux Kernel version should be 5.4.x and above.
- Enable packet forwarding in the kernel.
- Disable Reverse Path Filtering (that is, ‘rp_filter = 0’).
- iptables: User-space utility program should be installed.

- Cilium is installed using SMI/CNDP.

Procedure

Step 1 Download the ULB image tar file from the designated source (https://software.cisco.com/).

Extract the tar file to unpack the Docker images.

Step 2 Onboard and deploy the ULB image.

a) Load the Docker images and tag them as required.
b) Login to the Docker Hub and push the images.
c) Ensure you have the Helm charts for the ULB operator and agent. Update the 'values.yaml' file in the Helm charts to use your Docker images.
d) Deploy the ULB (ULB operator and agents) using ULB Ops-center from cluster deployer.
e) Configure the ULB according to the requirements.
f) Ensure packet forwarding is enabled and reverse path filtering is disabled.
g) Update log tags configuration for ULB operator and agent.

Use the following configuration commands for the operator:

```
config
logging name lbs_operator.app.app level { debug | error | info | off
 | trace | warn }
logging name lbs_operator.app.empcrd level { debug | error | info |
 off | trace | warn }
logging name lbs_operator.app.lbcrd level { debug | error | info |
off | trace | warn }
logging name lbs_operator.app.service level { debug | error | info
| off | trace | warn }
end
```

Use the following configuration commands for the agent:

```
config
logging name lbs-agent.egress-mgr.app level { debug | error | info
| off | trace | warn }
logging name lbs-agent.egress-mgr.iptables level { debug | error |
info | off | trace | warn }
logging name lbs-agent.egress-mgr.iptables-oper level { debug | error
 | info | off | trace | warn }
logging name lbs-agent.egress-mgr.lbs-iptables level { debug | error
 | info | off | trace | warn }
logging name lbs-agent.egress-mgr.egressmgr level { debug | error |
 info | off | trace | warn }
logging name lbs-agent.egress-mgr.listener level { debug | error |
info | off | trace | warn }
```

```
end
```

This log tag configuration allows precise control over the logging output, enabling effective monitoring, troubleshooting, performance analysis, security auditing, and operational insights. By specifying different log levels for various components, administrators can tailor the logging to meet specific needs and ensure the smooth operation of the load balancing service.

Step 3 Define and apply LoadBalancer and Egress Management Policy CRs as needed.

For a sample YAML file that includes Kubernetes Custom Resources, refer to the Appendix Sample custom resource definitions for load balancer and egress management policy, on page 37.

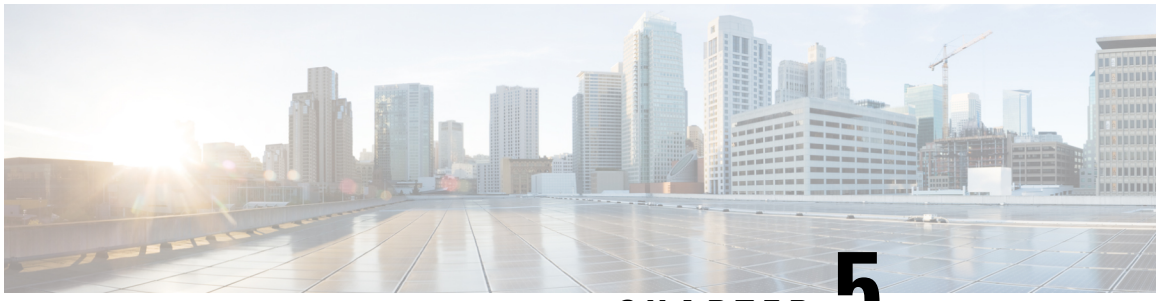
Access the ULB Ops center

You can connect to the ULB Ops Center through SSH using the following command:

ssh admin*@ *ops_center_pod_ip* **-p 2024**

Use the same user name and password as configured through the SMI Ops Center. For more information on the user management for access control, refer to the *CEE Configuration and Administration Guide* .

By default, the Day 0 configuration is loaded into the ULB.



CHAPTER 5

Unified Load Balancer traffic distribution

The Unified Load Balancer (ULB) is a Cisco proprietary network function that distributes incoming network traffic equally across all worker nodes. It ensures high availability and reliability across network infrastructures by maintaining consistent service IP addresses for ingress and egress traffic.

Key benefits

- Traffic management: Prevents bottlenecks by balancing traffic across multiple servers.
- Scalability: Supports horizontal scaling to handle increased traffic loads.
- Performance: Reduces latency through Direct Server Response (DSR).

- How the ULB works, on page 13

How the ULB works

The ULB distributes incoming requests to backend servers using the random load balancer algorithm. This approach is designed to minimize disruption when the backend server pool changes.

The algorithm ensures efficient request distribution and minimal impact during server pool changes.

Summary

Key components involved in this process include the ULB, backend servers, and the random load balancer algorithm.

- ULB: Receives client requests and manages distribution to backend servers.
- Backend servers: Handle the actual processing of client requests.
- Random load balancer algorithm: Maps incoming requests to backend servers using a hash-based lookup table.

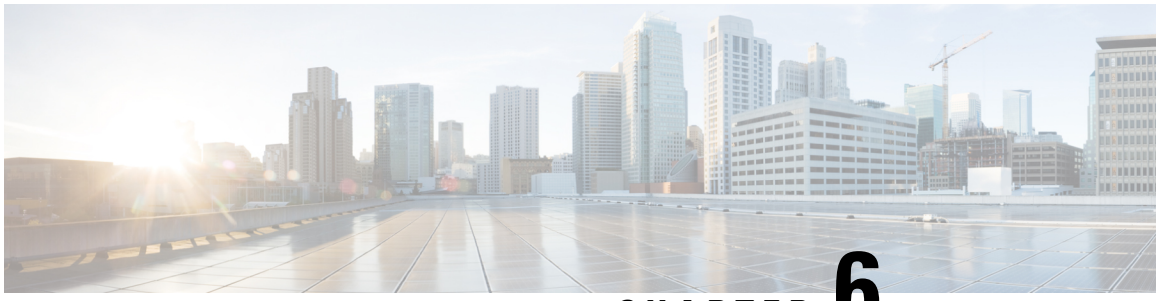
The process ensures that incoming requests are efficiently mapped to backend servers, and that changes in the server pool result in minimal disruption to existing request mappings.

Workflow

These stages describe how the ULB distributes requests using the random load balancer algorithm.

1. When a client request arrives, the ULB uses the random algorithm to determine the appropriate backend server.
   - The client's IP address (or another identifier) is used to generate a hash value.
   - The hash value is mapped to a backend server using a precomputed lookup table.
2.

When the server pool...	Then...	And...
changes (addition/removal)	the algorithm recalculates the lookup table	only a small fraction of mappings are affected.
remains stable	the lookup table remains constant	traffic distribution continues without interruption.



CHAPTER 6

ULB features and configuration

The ULB is designed to enhance network resilience, ensure rapid traffic convergence, and manage network resources effectively.

Note These features are currently applicable only for on-premises deployments.

Resilience and traffic convergence via BGP and BFD

The ULB employs BGP and BFD protocols to maintain network stability and enable quick traffic rerouting during node failures.

How the resilience and traffic convergence works

BGP integration

The ULB service integrates with routing servers using BGP, facilitating real-time route advertisement and traffic redistribution. This is implemented with Cilium support, requiring version 1.17 and above. Upon creating specific CRs, Cilium deploys BGP routers and manages BGP sessions.

BFD for rapid failure detection

BFD is used to detect path failures between routers swiftly, ensuring quick failover and minimal traffic disruption.

Limitations with BGP implementation

- Updating configurations, like timers, causes BGP sessions to restart.

- Due to Cilium's limitations, applications cannot segregate traffic and instead use BGP interfaces for all traffic.
- Even though BGP server mode is configured, Cilium's issues prevent it from working as a server.

Configure the BGP

Enable BGP functionality to support Resilience and Traffic Convergence in your network.

Procedure

Step 1 Define BFD Peer Profile
Step 2 Set up BGP Peer Profile
Step 3 Configure BGP Router

Define the BFD peer profile

BFD peer profile provides configuration for BFD functionalities such as timers for control packets, echo packets, and maximum retry for BFD messages.

The following steps describe how to define the BFD peer profile and the associated parameters on the ULB.

Procedure

Step 1 Configure the BFD peer profile in Global Configuration mode.

Example:

```
lbs(config)# bfd-peer-profile bfd-prof1
```

Step 2 Specify the timer in seconds for transmitting and receiving the BFD control packets.

The valid range of rx-timer for BFD control packets is 10 to 60000 milliseconds. Default value is 10 milliseconds.

The valid range of tx-timer for BFD control packets is 10 to 60000 milliseconds. Default value is 250 milliseconds.

Example:

```
lbs(config-bfd-peer-profile-bfd-prof1)# control-msg rx-timer 10 tx-timer 250
```

Step 3 Specify the timer in seconds for transmitting and receiving the BFD echo packets.

The valid range of rx-timer and tx-timer for echo packets is 10 to 60000 milliseconds. Default value is 90 milliseconds.

Example:

```
lbs(config-bfd-peer-profile-bfd-prof1)# echo-msg rx-timer 90 tx-timer 90
```

Step 4 Specify the max retry of BFD messages to mark the BFD session down.

The valid range of max retry count is 1 to 255. Default value is 3.

Example:

```
lbs(config-bfd-peer-profile-bfd-prof1)# max-retry 3
```

Step 5 Save and commit the configuration.

Example:

```
lbs(config-bfd-peer-profile-bfd-prof1)# end
```

This command lets you to either commit or ignore the configurations. Entering **yes** allows you to save or modify the configurations.

Set up the BGP peer profile

BGP Peer profile provides configuration options for BGP session related functionalities, such as connection retry timer, hold-timer, and keepalive-timer. This profile supports graceful restart, EBGP multihop, and advertise label features used for BGP advertisement resources. The profile configuration also includes support for enabling BFD functionality.

Use this procedure to define the BGP peer profile and the associated parameters on the ULB.

Procedure

Step 1 Configure the BGP peer profile in Global Configuration mode.

Example:

```
lbs(config)# bgp-peer-profile bgp-prof1
```

Step 2 Configure the BFD peer profile.

Example:

```
lbs(config)# bfd-peer-profile bfd-prof1
```

Step 3 Define the timers that are related to BGP sessions as needed.

Table 1: BGP Session Timer Parameters

CLI Command	Description	Range	Default Value
connection-retry-timer	Timer for connection to be retried on detecting peer down or restart of pod	1 to 300	12
graceful-restart restart-timer	Enables graceful restart functionality for BGP peers	1 to 4095	Disabled
hold-timer	Timer negotiated with BGP peer to hold the routes advertised on detecting peer down	3 to 300	3
keepalive-timer	Timer duration for sending keep alive messages to BGP peers	1 to 300	1

Note

Timers cannot be disabled; they can only be adjusted according to deployment needs. By default, graceful restart is disabled.

Step 4 Define the match label key/value configuration for BGP peer profiles. This label is used for BGP advertisement resources.

Example:

```
lbs(config-bgp-peer-profile-bgp-prof1)# advertise-label key advertise value bgp
```

Step 5 Configure EBGP multi hop count for BGP messages.

Example:

```
lbs(config-bgp-peer-profile-bgp-prof1)# ebgp-multihop 5
```

Step 6 Save and commit the configuration.

Configure the BGP router

The router supports configuration for virtual routers, BGP peering, and service IP advertisement. It also allows nodes to be configured as BGP routers.

Use this procedure to configure a router as a BGP router, including BGP peer profiles, node selectors, and service IPs.

Procedure

Step 1 Enter the global configuration mode.

Example:

```
[cluster1/data] lbs# config
```

Step 2 Configure the BGP router by specifying the ASN (Autonomous System Number) of BGP neighbor.

Example:

```
lbs(config)# router bgp 65000
```

Step 3 Configure the BGP peer profile by specifying the profile name.

Example:

```
lbs(config-router-bgp-65000)# bgp-peer-profile bgp-prof1
```

Step 4 Specify the node selector using key-value pairs.

Example:

```
lbs(config-bgp-peer-profile-bgp-prof1)# node-selector key feature value bgp
```

Step 5 Configure the service name and IP address.

Example:

```
lbs(config-bgp-peer-profile-bgp-prof1)# service name web-service ip 192.168.1.1
```

Step 6 Specify the peer name, AS number, and peer IP.

Example:

```
lbs(config-bgp-peer-profile-bgp-prof1)# peer-name peer_name
lbs(config-bgp-peer-profile-bgp-prof1)# peer-as AS peer-ip IP_address
```

Step 7 Save and commit the configuration.

Example:

```
lbs(config-bgp-peer-profile-bgp-prof1)# end
```

Use the following sample configuration to configure BGP and BFD settings.

```
config
        bfd-peer-profile bfd-prof1
        control-msg rx-timer 10 tx-timer 250
        echo-msg rx-timer 90 tx-timer 90
        max-retry 3
        exit
        bgp-peer-profile bgp-prof1
        connection-retry-timer 12 hold-timer 3 keepalive-timer 1 ebgp-multihop 255
                                                                                QAA
        bfd-peer-profile bfd-prof1
        exit
        router bgp 65000
        bgp-peer-profile bgp-prof1
        node-selector key feature value bgp
        service name pfcp ip 7.7.7.7
        service name s11 ip 8.8.8.8
        peer-name leaf1
        peer-as 63141 peer-ip 120.20.20.120
        exit
        peer-name leaf1-v6
        peer-as 63141 peer-ip 2006::98
        exit
        peer-name leaf2
        peer-as 63141 peer-ip 130.30.30.130
        exit
        peer-name leaf2-v6
        peer-as 63141 peer-ip 3006::98
        exit
        exit
```

Neighbor discovery

Neighbor discovery ensures stable network connectivity by addressing issues with ARP entries for neighbor IPs being removed intermittently. This feature is supported to prevent traffic drops by maintaining these entries.

How the neighbor discovery works

Neighbor discovery enables administrators to configure neighbor IPs and interfaces using the CLI. The process can be tailored to specific nodes by using labels.

- Administrators can configure multiple interfaces, each with their own IP addresses.
- Discovery is limited to nodes matching specified labels, or all nodes if no label is set.

Summary

Neighbor discovery involves the following key actors and components:

- Administrator: Configures neighbor IPs and interfaces using the CLI.
- CLI: Provides commands to set up neighbor discovery parameters.
- Nodes: Participate in neighbor discovery based on configured labels.

Administrators use the CLI to configure neighbor discovery, which is then dynamically applied to nodes matching the specified labels. If no label is configured, all nodes are included in the discovery process.

Workflow

These stages describe how neighbor discovery is configured and applied across nodes.

1. The administrator uses the CLI to configure neighbor IPs and interfaces, specifying labels if needed.
   - Access the CLI and enter neighbor discovery configuration mode.
   - Specify interfaces and assign IP addresses for each neighbor.
   - Optionally, configure labels to restrict discovery to specific nodes.

   Neighbor discovery is limited to nodes matching the specified labels. If no label is configured, discovery is performed on all nodes.

   Configuration changes are dynamically applied across nodes, updating neighbor entries as needed.

Configure the refresh interval, interfaces, and node labels

The following steps describe how to configure the refresh intervals and multiple interfaces with associated IPs.

Procedure

Step 1 Enter the global configuration mode.

Example:

```
[cluster1/data] lbs# config
```

Step 2 Configure the neighbor discovery details.

Example:

```
lbs(config)# neighbor-discovery
```

Step 3 Configure the refresh interval of neighbors in seconds.

The refresh interval is an integer from 1 to 3600 seconds. Default value is 30 seconds.

Example:

```
lbs(config)# neighbor-discovery refresh-interval 45
```

Step 4 Configure the interface through which the BGP neighbors should be reachable. Then, specify the associated IPv4 and IPv6 next-hop addresses.

Note
You can set up multiple interfaces, each with several IP addresses.

Example:

```
lbs(config)# neighbor-discovery interface v500
            lbs(config*)# neighbor-discovery interface v500 nexthop 7.7.7.3
            lbs(config*)# neighbor-discovery interface v500 nexthop 7:7:7::3
```

Step 5 Specify the node selector using key-value pairs.

Example:

```
lbs(config)# neighbor-discovery node-selector key feature value bgp
```

Neighbor discovery will be performed on each node matching any of the configured node labels. If no label is configured, neighbor discovery is performed on all nodes.

Step 6 Save and commit the configuration.

Example:

```
lbs(config)# end
```

Use the following sample configuration to configure neighbor discovery service.

```
config
        neighbor-discovery
        refresh-interval 45
        interface v500
        nexthop 7.7.7.3
        nexthop 7:7:7::3
        exit
        interface v501
        nexthop 6.6.6.3
        nexthop 6:6:6::3
        exit
        interface v502
        nexthop 5.5.5.3
        nexthop 5:5:5::3
        exit
        node-selector
        key frontend value protocol
        key test1 value value1
        exit
        exit
```

Monitor neighbor entries

Verify the entries using the following show command:

```
lbs# show running-config neighbor-discovery
neighbor-discovery
refresh-interval 10
node-selector
key key1 value value1
key key2 value value2
exit
```

```
interface v500
nexthop 7.7.7.3
nexthop 7:7:7::3
exit
interface v501
nexthop 6.6.6.3
nexthop 6:6:6::3
exit
exit
```

Core dump generation

ULB allows you to generate the core dump files for the ULB Agent and Operator processes in case of unexpected crashes or restarts. These core dumps capture the memory state of the process at the time of the failure, providing valuable information for debugging.

The core dump file is stored in the /var/lib/systemd/coredump/ directory on the node where the process was running.

Network administrators can manually collect the core dump file from each K8 node and analyze it using debugging tools.

Service outbound route (SOR)

The Service Outbound Route (SOR) feature provides a policy-based mechanism to isolate outbound traffic for applications within a Kubernetes cluster. It eliminates the need for manual configuration of static routes by allowing administrators to define custom routing policies using a Custom Resource (CR). The SOR CR is referenced by Load Balancer (LB) CRs for Direct Server Return (DSR) and Egress Management Policy (EMP) CRs for egress traffic.

Benefits of implementing SOR

- Simplified network configuration: Automates the configuration of outbound routing, reducing manual effort and potential errors.
- Enhanced security: Isolates outbound traffic, enabling more granular control and security policies.
- Improved compliance: Facilitates adherence to compliance requirements by ensuring outbound traffic follows defined paths.
- Resource optimization: Reduces the operational overhead associated with managing static routes.

Limitations and restrictions with SOR

- Backend application pods must use separate target ports for different applications or interfaces in the respective LB or EMP CRs.
- This feature is applicable only for LB/EMP CRs that have a target port defined.
- Currently, SOR is only applicable for UDP traffic.

How the SOR feature works

The SOR feature in the ULB isolates outbound traffic using routing tables and rules. For applications that require outbound traffic isolation, the network administrator creates an SOR CR that specifies the next-hop (gateway IP address) and device (network interface). The load balancer and endpoint custom resources reference this SOR CR to manage routing.

- Network administrator creates an SOR CR specifying next-hop and device.
- ULB assigns routing table ID and configures routes.
- Load balancer and endpoint custom resources reference the SOR CR.

Summary

The main actors and components involved in this process are:

- Network administrator: Creates and manages SOR CRs for outbound traffic isolation.
- ULB (Unified Load Balancer): Assigns routing table IDs and configures routes based on SOR CRs.
- Load balancer and endpoint custom resources: Reference SOR CRs to manage routing for associated pods.

This process ensures that outbound traffic from pods is routed through the specified gateway, enabling controlled and efficient routing for outbound requests.

Workflow

These stages describe how the SOR feature manages outbound traffic routing using SOR CRs and routing tables.

1. The network administrator creates an SOR CR specifying the next-hop (gateway IP address) and device (network interface).
2. The ULB assigns a unique routing table ID and configures routes using the specified next-hop and device.
3. Load balancer and endpoint custom resources reference the SOR CR to manage routing for associated pods.
4. IP rules are set so that pod traffic uses the corresponding routing table. Updates or deletions of pods trigger changes or removal of these IP rules. Iptables are adjusted for load balancer events.
5. When a pod generates outbound traffic, the routing table linked to the SOR CR ensures the traffic is sent through the configured gateway.

Support for data path metrics based on eBPF

ULB leverages Extended Berkeley Packet Filter (eBPF) technology to provide detailed visibility into the data path of the ULB service. It collects metrics related to ingress and egress traffic, enabling administrators to monitor performance and troubleshoot issues.

Note This feature is supported only on Ubuntu Kernel versions which support CGROUP (Control Group) v2.

Benefits of using the data path metrics

ULB provides the following benefits when using the data path metrics.

- Enhanced debugging: Provides granular data path metrics, facilitating faster identification and resolution of network issues.
- Performance monitoring: Enables real-time monitoring of traffic throughput, allowing administrators to identify bottlenecks and optimize performance.
- Improved visibility: Offers insights into service traffic patterns, aiding in capacity planning and resource allocation.

Limitations in supporting data path metrics

eBPF monitoring can introduce some overhead, so it's important to monitor the system resource usage.

How the data path metric collection and reporting works

This feature helps in understanding network traffic flow within the Load Balancer Service.

Summary

The process involves several actors and components working together to collect and report network traffic metrics.

- eBPF: Passively observes traffic characteristics and counts packets and bytes for each service and pod.
- ULB agent: Periodically collects the counts from eBPF and makes them available as metrics.

These metrics can be used for debugging and performance analysis.

Workflow

These stages describe the collection and reporting of data path metrics within the Load Balancer Service.

1. Enable metric collection feature using this CLI command.

   metrics service-stats

   Once enabled, the monitoring program (eBPF) starts observing traffic characteristics.

   The feature is activated and ready for metric collection.

2. eBPF observes traffic and counts packets and bytes for each service and pod.

   eBPF operates in the background, collecting data without impacting performance.

   Traffic metrics are accumulated for each service and pod.

3. ULB agent collects the counts from eBPF and makes them available as metrics.

   Metrics are periodically gathered and made accessible for debugging and performance analysis.

Enable the data path metrics collection by eBPF

Procedure

Step 1 Enter the global configuration mode.

Step 2 Run this command to enable the data path metrics collection feature.

```
metrics service-stats
```

Note
This CLI is disabled by default.

Step 3 Commit the changes.

Miscellaneous configurations

Configure the Cilium BPF map

Use this procedure to edit the BPF map sizes for Cilium by entering the appropriate CLI commands.

You do not need to configure all map sizes; adjust them as needed for your setup. By default, this CLI does not set any values, so Cilium BPF map sizes will follow Cilium's standards. After committing changes in this CLI, the Cilium pod will restart. Ensure the values stay within the defined limits; otherwise, the commit will fail.

Procedure

Enter configuration mode and set the desired BPF map sizes using the following commands:

Example:

```
config
            [no] cilium bpf auth-map-max <value>
            [no] cilium bpf fragments-map-max <value>
            [no] cilium bpf lb-affinity-map-max <value>
            [no] cilium bpf lb-maglev-map-max <value>
            [no] cilium bpf lb-maglev-table-size <value>
            [no] cilium bpf lb-map-max <value>
            [no] cilium bpf lb-rev-nat-map-max <value>
            [no] cilium bpf lb-service-backend-map-max <value>
            [no] cilium bpf lb-service-map-max <value>
            [no] cilium bpf lb-source-range-map-max <value>
            [no] cilium bpf map-dynamic-size-ratio <value>
```

```
[no] cilium bpf neigh-global-max <value>
[no] cilium bpf node-map-max <value>
[no] cilium bpf policy-map-max <value>
exit
```

- You don't need to configure all map sizes; adjust them as needed for your setup.
- By default, this CLI doesn't set any values, so Cilium BPF map sizes will follow Cilium's standards.
- After committing changes in this CLI, the Cilium pod will restart.
- Ensure the values stay within the defined limits. Else, the commit will fail.

Use the following table as a reference for the maximum size associated with these commands.

Command	Description	Range
cilium bpf auth-map-max	Maximum size for the BPF auth map	65536 to 4194304
cilium bpf fragments-map-max	Maximum size for the BPF fragments map	4096 to 65536
cilium bpf lb-affinity-map-max	Maximum size for the BPF load balancer affinity map	65536 to 4194304
cilium bpf lb-maglev-map-max	Maximum size for the BPF load balancer Maglev map	65536 to 4194304
cilium bpf lb-maglev-table-size	Size of the BPF load balancer Maglev lookup table.	MAGLEV_TABLE_SIZE_251 MAGLEV_TABLE_SIZE_509 MAGLEV_TABLE_SIZE_1021 MAGLEV_TABLE_SIZE_2039 MAGLEV_TABLE_SIZE_4093 MAGLEV_TABLE_SIZE_8191 MAGLEV_TABLE_SIZE_16381 MAGLEV_TABLE_SIZE_32749 MAGLEV_TABLE_SIZE_65521 MAGLEV_TABLE_SIZE_131071
cilium bpf lb-map-max	Maximum size for the BPF load balancer map	65536 to 4194304
cilium bpf lb-rev-nat-map-max	Maximum size for the BPF load balancer reverse NAT map.	65536 to 4194304
cilium bpf lb-service-backend-map-max	Maximum size for the BPF load balancer service backend map.	65536 to 4194304
cilium bpf lb-service-map-max	Maximum size for the BPF load balancer service map.	65536 to 4194304

Command	Description	Range
cilium bpf lb-source-range-map-max	Maximum size for the BPF load balancer source range	65536 to 4194304
cilium bpf map-dynamic-size-ratio	Ratio used for dynamically sizing BPF maps relative to the amount of memory available	0.0 to 1.0
cilium bpf neigh-global-max	Global maximum size for the BPF neighbor map.	65536 to 4194304
cilium bpf node-map-max	Maximum size for the BPF node map	16384 to 4194304
cilium bpf policy-map-max	Maximum size for the BPF policy map.	8192 to 16384

Enable the Cilium Hubble Statistics

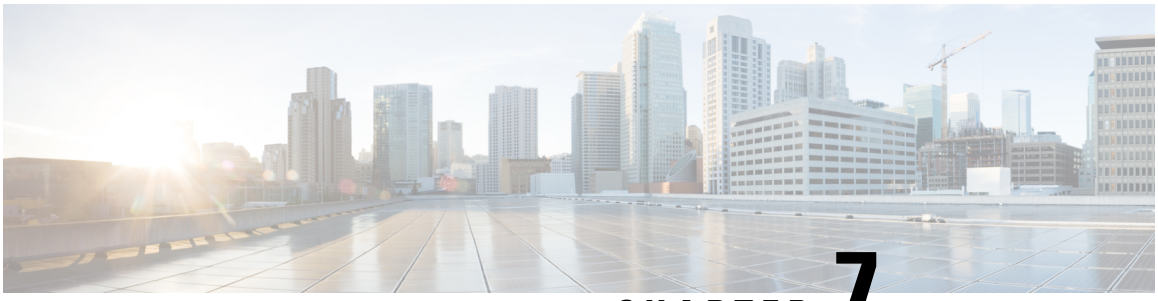
Procedure

Enter configuration mode and enable Hubble statistics for data collection:

Example:

```
config
metrics collection
exit
```

Note
This CLI is disabled by default. Enabling the Hubble statistics will restart the Cilium pod.



CHAPTER 7

Performance metrics and KPIs

This section outlines the key metrics that can be used to assess the performance of ULB.

ULB operator metrics

The following are the ULB operator metrics.

Name	Data Type	Metric Type	Description
loadbalancer_custom_resource_add_success_total Labels: [NameSpace, Kind, CrName, SvcName, PodName, Error]	INT64	Counter	Total number of successful loadbalancer custom resource add requests received.
loadbalancer_custom_resource_add_failed_total Labels: [NameSpace, Kind, CrName, SvcName, PodName, Error]	INT64	Counter	Total number of failed loadbalancer custom resource add requests received.
loadbalancer_custom_resource_modify_success_total Labels: [NameSpace, Kind, CrName, SvcName, PodName, Error]	INT64	Counter	Total number of successful loadbalancer custom resource modify requests received.
loadbalancer_custom_resource_modify_failed_total Labels: [NameSpace, Kind, CrName, SvcName, PodName, Error]	INT64	Counter	Total number of failed loadbalancer custom resource modify requests received.
loadbalancer_custom_resource_delete_success_total Labels: [NameSpace, Kind, CrName, SvcName, PodName, Error]	INT64	Counter	Total number of successful loadbalancer custom resource delete requests received.

Name	Data Type	Metric Type	Description
loadbalancer_custom_resource_delete_failed Labels: [NameSpace, Kind, CrName, SvcName, PodName, Error]	INT64	Counter	Total number of failed loadbalancer custom resource delete requests received.
egressmanagementpolicies_custom_resource_add_success Labels: [NameSpace, Kind, CrName, SvcName, PodName, Error]	INT64	Counter	Total number of successful egressmanagementpolicies custom resource add requests received.
egressmanagementpolicies_custom_resource_add_failed Labels: [NameSpace, Kind, CrName, SvcName, PodName, Error]	INT64	Counter	Total number of failed egressmanagementpolicies custom resource add requests received.
egressmanagementpolicies_custom_resource_modify_success Labels: [NameSpace, Kind, CrName, SvcName, PodName, Error]	INT64	Counter	Total number of successful egressmanagementpolicies custom resource modify requests received.
egressmanagementpolicies_custom_resource_modify_failed Labels: [NameSpace, Kind, CrName, SvcName, PodName, Error]	INT64	Counter	Total number of failed egressmanagementpolicies custom resource modify requests received.
egressmanagementpolicies_custom_resource_delete_success Labels: [NameSpace, Kind, CrName, SvcName, PodName, Error]	INT64	Counter	Total number of successful egressmanagementpolicies custom resource delete requests received.
egressmanagementpolicies_custom_resource_delete_failed Labels: [NameSpace, Kind, CrName, SvcName, PodName, Error]	INT64	Counter	Total number of failed egressmanagementpolicies custom resource delete requests received.
service_modify_failed_not_loadbalancer_owned Labels: [NameSpace, Kind, CrName, SvcName, PodName, Error]	INT64	Counter	Total number of service modify requests failed due to service not owned by load balancer

Name	Data Type	Metric Type	Description
service_delete_total_not_loadbalancer_owned Labels: [NameSpace, Kind, CrName, SvcName, PodName, Error]	INT64	Counter	Total number of service delete requests failed due to service not owned by load balancer
service_modify_total_egressmanagementpolicies_owned Labels: [NameSpace, Kind, CrName, SvcName, PodName, Error]	INT64	Counter	Total number of service modify requests failed due to service not owned by egressmanagementpolicies
service_delete_total_egressmanagementpolicies_owned Labels: [NameSpace, Kind, CrName, SvcName, PodName, Error]	INT64	Counter	Total number of service modify requests failed due to service not owned by egressmanagementpolicies.

The following table provides description of the labels associated with the operator metrics.

Label Name	Label Type	Description
NameSpace	STRING	Namespace in which the resource is created.
Kind	STRING	It will display the kind of resource created.
CrName	STRING	Name of the Custom Resource.
SvcName	STRING	Name of the service created.
PodName	STRING	Name of the pod.
Error	STRING	Error string in case of failure.

ULB agent metrics

The following are the ULB agent metrics.

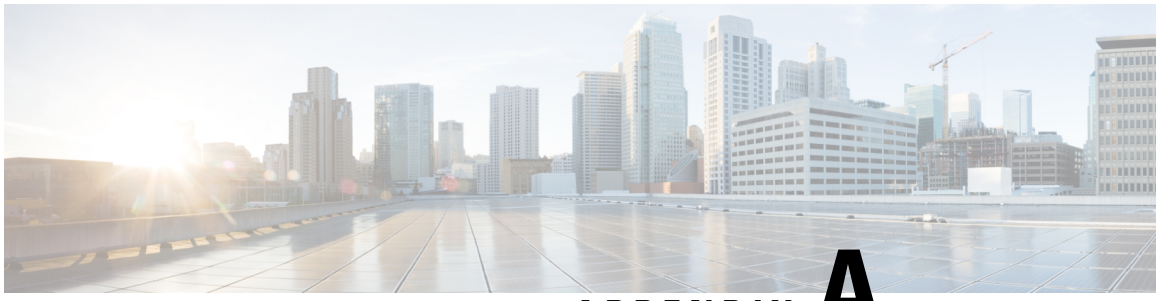
Name	Data Type	Metric Type	Description
egressmanagementpolicy_custom_resource_add_event_total	INT64	Counter	Total number of egressmanagementpolicies custom resource add events received.
egressmanagementpolicy_custom_resource_modify_event_total	INT64	Counter	Total number of egressmanagementpolicies custom resource modify events received.
egressmanagementpolicy_custom_resource_delete_event_total	INT64	Counter	Total number of egressmanagementpolicies custom resource delete events received.
pod_add_event_total	INT64	Counter	Total number of pod add events received.
pod_add_event_ignore_total	INT64	Counter	Total number of pod add events ignored.
pod_modify_event_total	INT64	Counter	Total number of pod modify events received.
pod_modify_event_ignore_total	INT64	Counter	Total number of pod modify events ignored.
pod_delete_event_total	INT64	Counter	Total number of pod delete events received.
node_add_event_total Label: component	INT64	Counter	Total number of node add events received.
node_add_event_ignored_total Label: component (value: neigbor)	INT64	Counter	Total number of node add events ignored.
node_modify_event_total Label: component (value: neigbor)	INT64	Counter	Total number of node modify events received.
node_modify_event_ignored_total Label: component (value: neigbor)	INT64	Counter	Total number of node modify events ignored.
node_delete_event_total Label: component (value: neigbor)	INT64	Counter	Total number of node delete events received.

Name	Data Type	Metric Type	Description
loadbalancer_customresource_add_event_total	INT64	Counter	Total number of load balancer custom resource add events received.
loadbalancer_customresource_delete_event_total	INT64	Counter	Total number of load balancer custom resource delete events received.
loadbalancer_customresource_modify_event_total	INT64	Counter	Total number of load balancer custom resource modify events received.
loadbalancer_pod_add_event_total	INT64	Counter	Total number of load balancer pod add events received.
loadbalancer_pod_modify_event_total	INT64	Counter	Total number of load balancer pod modify events received.
loadbalancer_pod_delete_event_total	INT64	Counter	Total number of load balancer pod delete events received.
loadbalancer_pod_add_event_ignored_total	INT64	Counter	Total number of load balancer pod add events ignored.
loadbalancer_pod_modify_event_ignored_total	INT64	Counter	Total number of load balancer pod modify events ignored.
loadbalancer_pod_delete_event_ignored_total	INT64	Counter	Total number of load balancer pod delete events ignored.

Name	Data Type	Metric Type	Description
svc_outbound_custom_resource_add_events_rcvd	INT64	Counter	Total number of service outbound route custom resource add events received.
svc_outbound_custom_resource_modify_events_rcvd	INT64	Counter	Total number of service outbound route custom resource modify events received.
svc_outbound_custom_resource_delete_events_rcvd	INT64	Counter	Total number of service outbound route custom resource delete events received.
svc_outbound_custom_resource_add_events_ignored	INT64	Counter	Total number of service outbound route custom resource add events ignored.
svc_outbound_custom_resource_modify_events_ignored	INT64	Counter	Total number of service outbound route custom resource modify events ignored.
svc_outbound_custom_resource_delete_events_ignored	INT64	Counter	Total number of service outbound route custom resource delete events ignored.
svc_packet_count Labels: [svc_ip, svc_name, pod_ip, pod_port, pod_name, protocol, direction]	INT64	Counter	Total number of packets per service IP.
svc_byte_count Labels: [svc_ip, svc_name, pod_ip, pod_port, pod_name, protocol, direction]	INT64	Counter	Total number of bytes per service IP.

The following table provides description of the labels associated with some of the agent metrics.

Label Name	Label Type	Description
svc_ip	STRING	IP address of the service.
svc_name	STRING	Name of our EMP service
pod_ip	STRING	Ip address of the backend pod
pod_port	STRING	Port used by backend pod
pod_name	STRING	Name of the backend pod.
protocol	STRING	L4 protocol – TCP or UDP
direction	STRING	IN/OUT packet direction.



APPENDIX A

Sample custom resource definitions for load balancer and egress management policy

This section offers a sample YAML configuration file featuring the Load Balancer (LB) and Egress Management Policy (EMP) Custom Resources (CRs). Use this example to define and apply the LB and EMP CRs based on your deployment needs.

Load balancer CR (IPv4)

```
apiVersion: lbs.cisco.com/v1alpha1
kind: LoadBalancer
metadata:
  name: udp-lb
spec:
  protocol: UDP
  servicePort: 8765
  serviceIPs:
    - 2.2.2.2
  target:
    port: 8807
    selectors:
      matchLabels:
        release: udp-nf
```

Load balancer CR (IPv6)

```
apiVersion: lbs.cisco.com/v1alpha1
kind: LoadBalancer
```

```
metadata:
  name: udp-lb-v6
spec:
  protocol: UDP
  servicePort: 8766
  serviceIPs:
    - 2201::110B
  target:
    port: 8808
    selectors:
      matchLabels:
        release: udp-nf
```

Egress management policy CR for stateful pods (IPv4)

```
apiVersion: lbs.cisco.com/v1alpha1
kind: EgressManagementPolicy
metadata:
  name: udp-emp
spec:
  protocol: UDP
  egressPort:
    stateful:
      - podName: pod-1
        portRanges:
          - "15001…15003"
          - 16001
      - podName: pod-2
        portRanges:
          - "17001…17004"
  target:
    selectors:
      matchLabels:
        release: udp-nf
  egressIPs:
    - 2.2.2.2
```

Egress management policy CR for stateful pods (IPv6)

```
apiVersion: lbs.cisco.com/v1alpha1
kind: EgressManagementPolicy
metadata:
  name: udp-emp-v6
spec:
  protocol: UDP
  egressPort:
    stateful:
      - podName: pod-1
        portRanges:
          - "25001…25003"
      - podName: pod-2
        portRanges:
          - "27001…27003"
  target:
    selectors:
      matchLabels:
        release: udp-nf
  egressIPs:
    - 2201::110B
```

Egress management policy CR for stateless pods (IPv4)

```
apiVersion: lbs.cisco.com/v1alpha1
kind: EgressManagementPolicy
metadata:
  name: udp-emp-sl
spec:
  protocol: UDP
  egressPort:
    stateless:
        portRanges:
          - "35001…35003"
  target:
    selectors:
      matchLabels:
        release: udp-nf
  egressIPs:
    - 4.4.4.4
```

Egress management policy CR for stateless pods (IPv6)

The configuration of the LoadBalancer and EgressManagementPolicy is similar to that described for IPv4 in the previous section; the only difference is that the 'serviceIPs' field in the LoadBalancer and the 'egressIPs' field in the EgressManagementPolicy will use IPv6 addresses.

Egress management policy CR for stateful pods without port ranges (IPv4)

```
apiVersion: lbs.cisco.com/v1alpha1
kind: EgressManagementPolicy
metadata:
  name: udp-emp-st-wo-pr
spec:
  protocol: UDP
  egressPort:
    stateful:
      - podName: pod-1
      - podName: pod-2
  target:
    selectors:
      matchLabels:
        release: udp-nf
  node:
    selectors:
      matchLabels:
        smi.cisco.com/egress: enable
  egressIPs:
    - 7.7.7.7
```

Service outbound route (SOR) IPv4

```
Name:         sor4
Namespace:    default
Labels:       <none>
Annotations:  <none>
API Version:  lbs.cisco.com/v1alpha1
Kind:         ServiceOutboundRoute
Metadata:
  Creation Timestamp:  2025-02-25T05:23:57Z
  Generation:          1
  Resource Version:    465878
  UID:                 8d3a629c-b9cc-4195-9bb3-a8cdae948613
Spec:
  Device:    ens224.502
  Next Hop:  209.165.201.1
Status:
  Node Info:
    Default Route:  {Ifindex: 64 Dst: 0.0.0.0/0 Src: <nil> Gw: 209.165.201.2 Flags: []
Table: 1002 Realm: 0}
    Node Name:      rid7656453-user-1-worker2
    Table Id:       1002
    Default Route:  {Ifindex: 94 Dst: 0.0.0.0/0 Src: <nil> Gw: 209.165.201.2 Flags: []
Table: 1002 Realm: 0}
    Node Name:      rid7656453-user-1-master2
    Table Id:       1002
    Default Route:  {Ifindex: 78 Dst: 0.0.0.0/0 Src: <nil> Gw: 209.165.201.2 Flags: []
Table: 1002 Realm: 0}
    Node Name:      rid7656453-user-1-master3
    Table Id:       1002
    Default Route:  {Ifindex: 92 Dst: 0.0.0.0/0 Src: <nil> Gw: 209.165.201.2 Flags: []
Table: 1002 Realm: 0}
    Node Name:      rid7656453-user-1-master1
    Table Id:       1002
    Default Route:  {Ifindex: 72 Dst: 0.0.0.0/0 Src: <nil> Gw: 209.165.201.2 Flags: []
Table: 1002 Realm: 0}
    Node Name:      rid7656453-user-1-worker1
    Table Id:       1002
```

Output:

```
sudo ip route show table all | grep default
local default dev lo table 2004 proto kernel scope host
default via 209.165.201.2 dev ens224.502 table 1002
```

Service outbound route (SOR) IPv6

```
Name:         sor4-v6
Namespace:    default
Labels:       <none>
Annotations:  <none>
API Version:  lbs.cisco.com/v1alpha1
Kind:         ServiceOutboundRoute
Metadata:
  Creation Timestamp:  2025-02-25T07:35:33Z
  Generation:          1
  Resource Version:    489696
  UID:                 8d43362a-8662-435e-a1dd-32600cc0d0d3
Spec:
  Device:    ens224.502
```

Sample custom resource definitions for load balancer and egress management policy

```
  Next Hop:  4444::100
Status:
  Node Info:
    Default Route:  {Ifindex: 78 Dst: ::/0 Src: <nil> Gw: 4444::100 Flags: [] Table: 1003
Realm: 0}
    Node Name:      rid7656453-user-1-master3
    Table Id:       1003
    Default Route:  {Ifindex: 72 Dst: ::/0 Src: <nil> Gw: 4444::100 Flags: [] Table: 1003
Realm: 0}
    Node Name:      rid7656453-user-1-worker1
    Table Id:       1003
    Default Route:  {Ifindex: 64 Dst: ::/0 Src: <nil> Gw: 4444::100 Flags: [] Table: 1003
Realm: 0}
    Node Name:      rid7656453-user-1-worker2
    Table Id:       1003
    Default Route:  {Ifindex: 94 Dst: ::/0 Src: <nil> Gw: 4444::100 Flags: [] Table: 1003
Realm: 0}
    Node Name:      rid7656453-user-1-master2
    Table Id:       1003
    Default Route:  {Ifindex: 92 Dst: ::/0 Src: <nil> Gw: 4444::100 Flags: [] Table: 1003
Realm: 0}
    Node Name:      rid7656453-user-1-master1
    Table Id:       1003
Events:             <none>
```

Output:

```
sudo ip route show table all | grep default
default via 4444::100 dev ens224.502 table 1003 metric 1024 pref medium
```

Sample custom resource definitions for load balancer and egress management policy