ISOVALENT
now part of cisco

Kubernetes Traffic Engineering for Network Engineers: Cilium Best Practices

Version 2024-09-12-01

Contents

Contents

Preface

Traffic engineering can feel like a daunting puzzle.

Even for experienced network engineers, optimizing traffic flow, reducing congestion, and ensuring network reliability in cloud-native environments is challenging. This complexity only grows as infrastructures evolve, decentralizing into increasingly smaller pieces, with Kubernetes becoming the go-to for container orchestration.

As Kubernetes becomes more integral to modern infrastructure, network engineers are tasked with understanding traffic engineering focused around increasingly microservices-based architectures. Our goal is to provide a practical guide that speaks directly to network teams looking to understand traffic routing with Cilium. Presented solutions are compatible with any Kubernetes release supported by Cilium.

This whitepaper uses Cilium, the first and only CNCF-graduated networking plugin, to implement traffic engineering in Kubernetes environments in various deployment flows. As the creators of eBPF and Cilium, customers look to Isovalent and our expertise to guide them along their cloud native journey and into advanced production deployments for their mission-critical applications.

Isovalent offers the leading platform for cloud native networking & security, building open-source software and enterprise solutions powering modern cloud native infrastructure.

Traffic engineering in Kubernetes is essential for analyzing, predicting, and managing data flows. By optimizing network traffic, platform owners and network engineers can improve the experience for both end users and internal developers. The goal is to use network resources efficiently, minimize latency, enhance security, and cut costs, all while ensuring a smooth user experience. This whitepaper shows how to use Cilium to achieve these goals in Kubernetes, offering practical advice and a step-by-step blueprint for building a strong reference architecture.

Intended Audience

This guide is written for network engineers but accessible to anyone interested in traffic engineering concepts in Kubernetes or with Cilium. We assume readers are familiar with basic networking concepts. If you're comfortable with these topics, the content builds on itself as we progress from the prerequisites, introduce an example architecture diagram, move into the traffic engineering matrix, and bring it all together by managing traffic with Cilium.

For newer friends to the Kubernetes networking space, do not be alarmed. This guide references many diagrams (building in complexity) that can be used to help steer your first look at building an enterprise-grade reference architecture or implementing principles in lower environments.

Additionally, Kubernetes networking is difficult, even for experienced network architects. In a new eBook offered by Isovalent, you can learn about Kubernetes Networking and Cilium. Explained in terms and references traditional network engineers will understand, the eBook is accessible to anyone keen to learn about Kubernetes networking and the de facto cloud-native networking platform: Cilium.

About Isovalent

Isovalent is the company founded by the creators and maintainers of Cilium and eBPF. We build open source software and enterprise solutions solving networking, security, and observability needs for modern cloud native infrastructure. Google (GKE, Anthos), Amazon (EKS-A), and Microsoft (AKS) have all adopted Cilium to provide networking and security for Kubernetes. Cilium is used by platform engineering teams such as Adobe, Bell Canada, ByteDance, Capital One, Datadog, IKEA, Schuberg Philis, and Sky.

Authors

Piotr Jabłoński

Jeremy Colvin

Reviewers

Dean Lewis

Michael Kashin

Philip Schmid

Scott Lowe

Introduction to Traffic Engineering

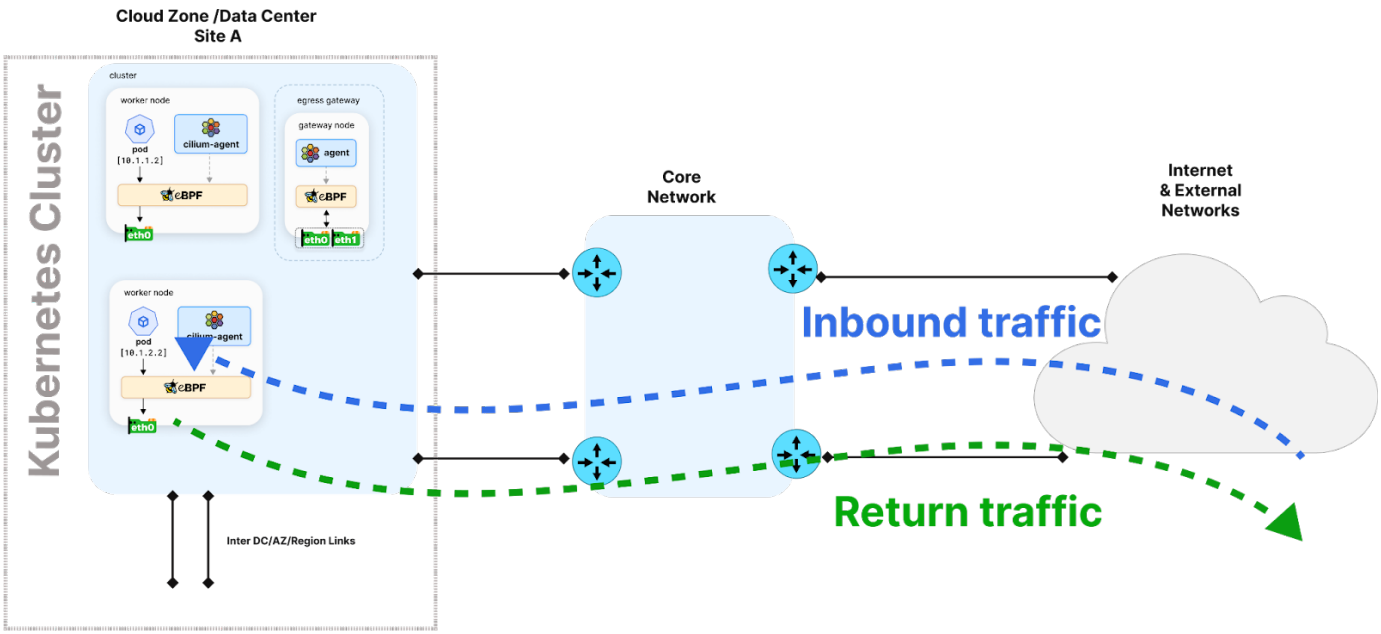
Traffic Engineering aims to direct traffic in a way that maximizes throughput, minimizes latency, enhances security, and optimizes costs, ultimately delivering a seamless and efficient user experience.

There are several use cases of traffic engineering with Cilium deployed on Kubernetes clusters:

- **Failover scenarios** — where one or more subsystems or devices become unavailable.
- **Security and identity** — to know who sends traffic where.
- **Application and network performance** — to provide a more optimal path from a resource perspective, i.e., throughput, latency, etc.
- **Cost optimization** — where selecting specific paths and services lowers usage.

These concepts can be applied in a cloud zone or a data center. It can also be configured at a larger scale, including multiple sites and other internal or external networks. Let's consider the following topology: a single site with the Kubernetes cluster, the core/edge network, and the Internet.

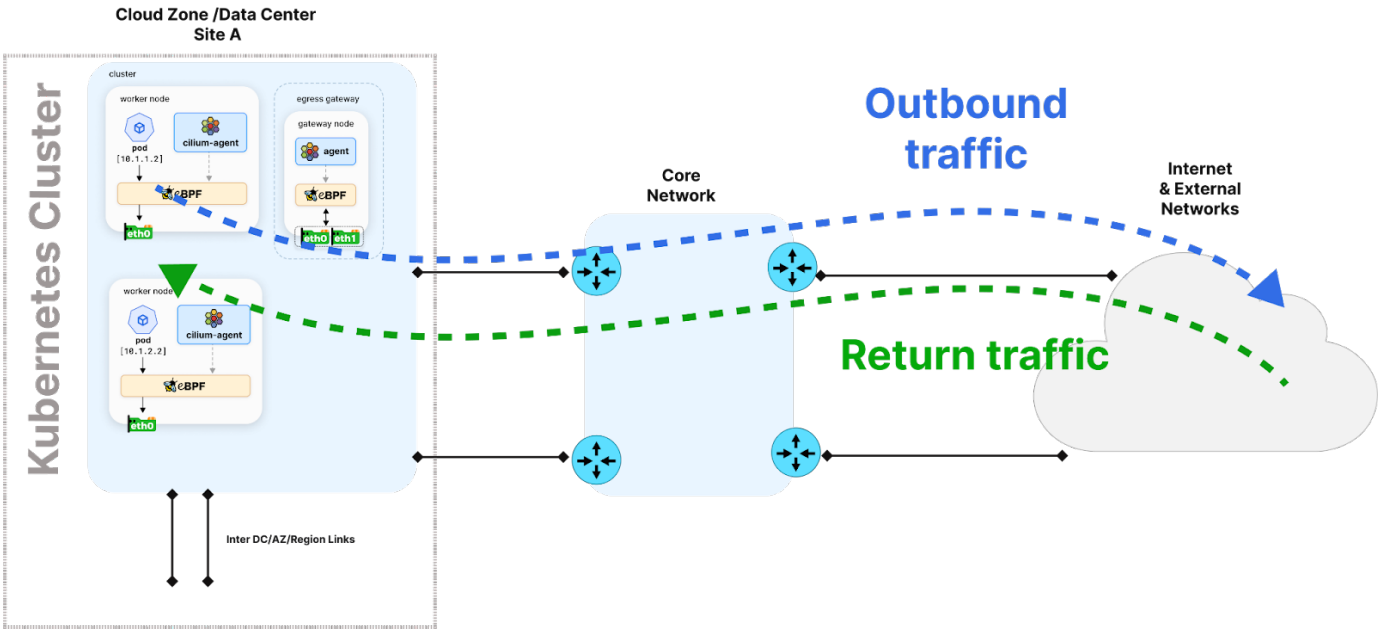
Traffic engineering techniques help steer traffic streams. They encompass the inbound and outbound directions, as in the diagram below.



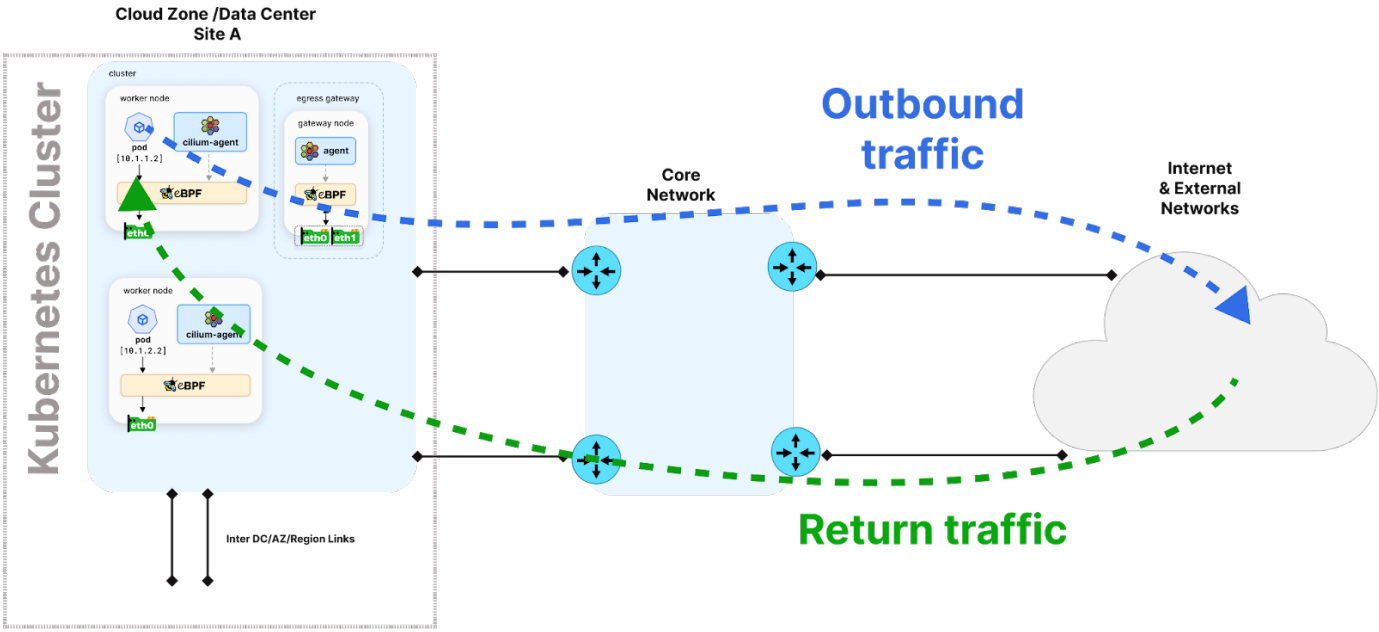
Inbound traffic refers to any traffic that originates outside the cluster, such as from the public internet or another cluster/region within your infrastructure.

This may also be known as ingress traffic, when using service mesh features such as Ingress Controller, Gateway API etc etc. In this document we will exclusively use the term, inbound traffic, as to avoid confusion.

Outbound traffic refers to traffic that originates inside the Kubernetes cluster and is routed to external destinations. In the outbound direction, traffic is initialized from inside the Kubernetes cluster and traverses outside the cluster. The return traffic comes from the external network to the target Kubernetes cluster.



You can independently steer the traffic in both directions. For example, as depicted in the following diagram, you can configure the network so that outbound traffic takes a different path than return traffic.



This document primarily focuses on traffic engineering with Cilium, related to inter-communication between Kubernetes clusters and external networks. In this context, Cilium should be considered one of the many components used to control traffic flows. Additional techniques, such as routing protocols, tunneling, and virtual private networks (VPNs), may also be employed, though they are outside the scope of this whitepaper.

Prerequisites

As we approach traffic engineering with Cilium, every environment is different, and this document aims to provide a clear best practices guide for users to build their own reference architecture plans.

This document contains features from the full Isovalent platform, including core enhancements (ex: Egress Gateway High Availability) from Isovalent Enterprise for Cilium. Isovalent Enterprise for Cilium is the hardened, enterprise-grade, and 24×7-supported version of the eBPF-based cloud networking platform Cilium.

The architecture is designed to work with any LAN, WAN, or core network that supports the IPv4 or IPv4/IPv6 dual-stack.

High-level features for controlling Kubernetes traffic with Cilium:

- Services include at least one of the following: ClusterIP, Gateway API, Ingress or Load Balancer. Enable the service if applications are exposed externally for traffic initialized from outside a K8s cluster.
- Egress Gateway: Enable if the applications initialize connections from inside to external entities.
- BGP: Enable if prefixes will be advertised dynamically.

In the following sections we will guide you from a starting point of an architecture diagram with three different application types deployed on it. This will be our starting point to guide the use cases around the different traffic engineering outcomes we aim to accomplish for our apps. This builds into three core sections of this white paper:

Architecture Diagram:

- Requirements behind each application use case
- Introduction to concepts: Ingress services, Egress gateway, Connections, Role of BGP

Traffic engineering matrix:

- Inbound patterns
- Outbound patterns
- Not feasible patterns

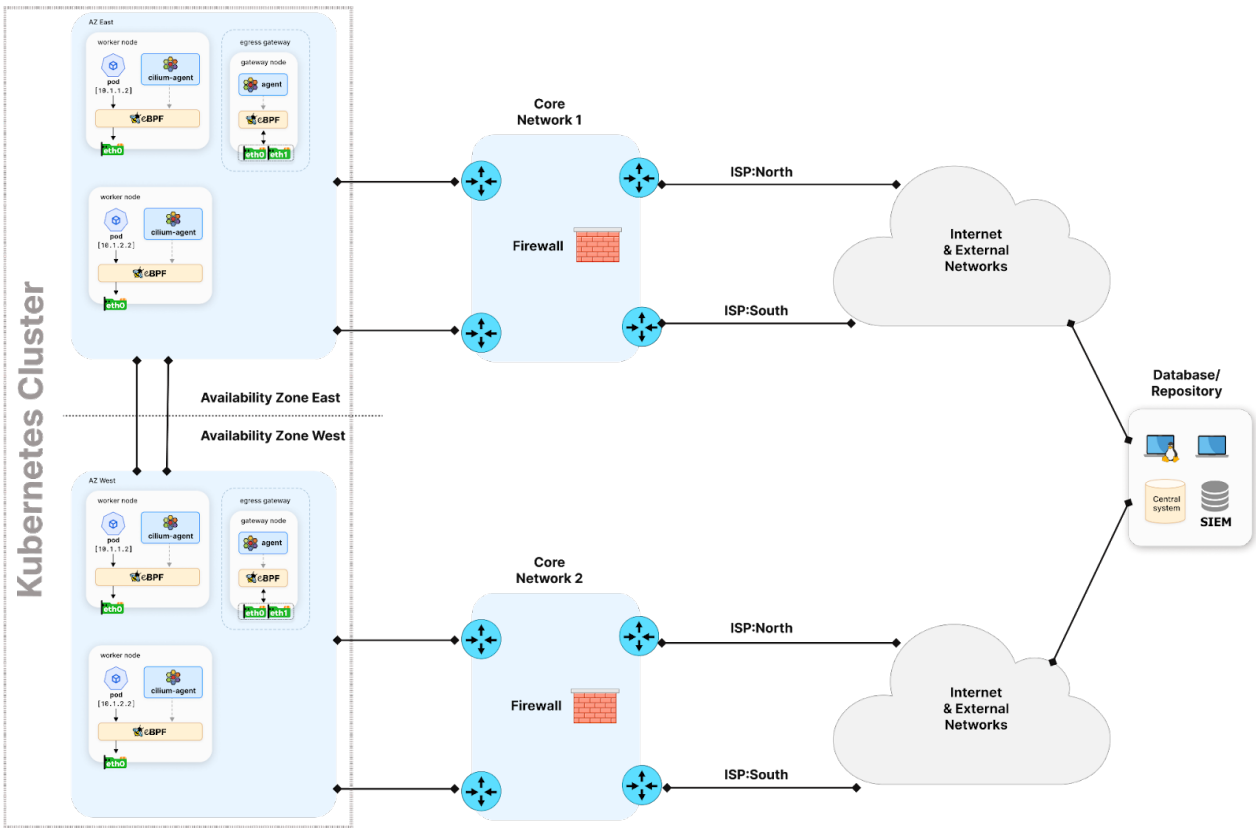
Traffic engineering with Cilium:

- General best practices with Cilium
- Application specific design

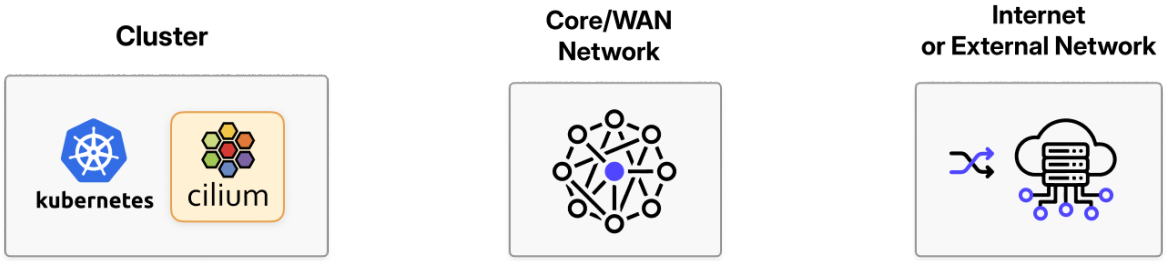
High Level Topology

The following architectural diagram depicts a high-level topology of a wide area network including a Kubernetes platform deployed with Cilium. This is composed of:

- The Kubernetes cluster deployed with Cilium stretched across two AWS Availability Zones (AZ): East and West. AZs are connected to each other using internal AWS VPC links.
- Customer-managed core networks connect the public cloud with the external networks.
- The uplinks for the core networks are provided by ISP:North and ISP:South.
- The firewalls inspect the traffic going through the core networks.
- The external site (i.e. externally hosted SaaS service) with a database, repository and other centralized systems.



The "Traffic Engineering Matrix" chapter describes overviews of traffic steering techniques with relation to a direction of the traffic (Inbound, Outbound) and the following high-level parts of the network ('Cluster', 'Core/WAN Network', and 'Internet or External Network').

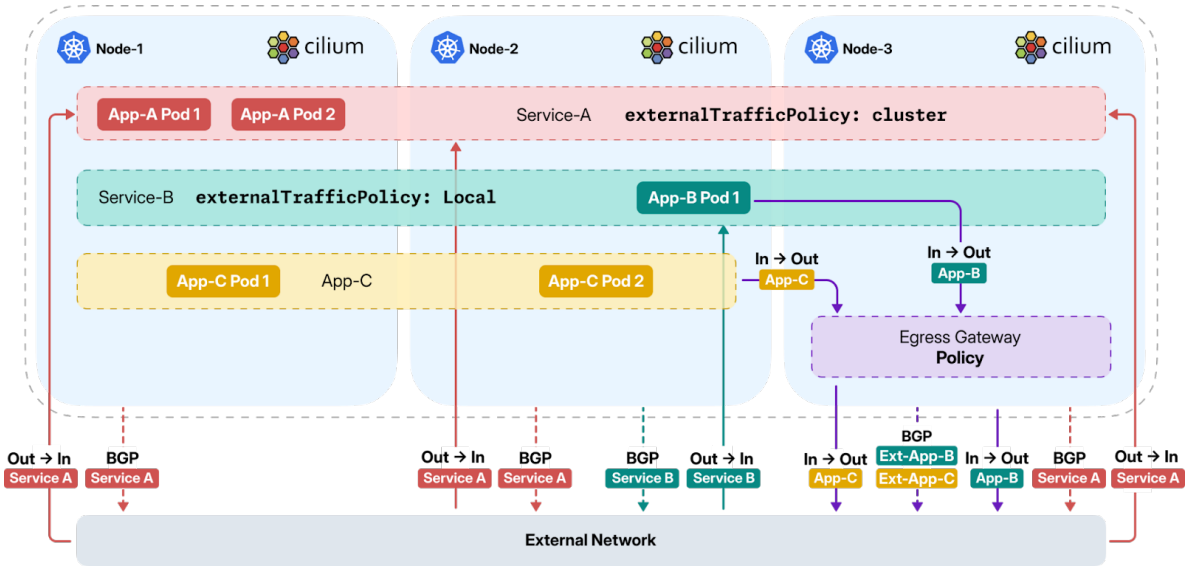


Cilium Architecture Diagram

The following architecture diagram related to Cilium shows the communication pattern and general components of traffic engineering in a Kubernetes deployment.

This diagram has many different components, over the course of this white paper we will continually break down sub-pieces of this diagram as the reference point for different traffic engineering techniques.

Let's first break down this initial diagram into the following components: applications (A, B, C), services, connections and the BGP role.



Overview

The Kubernetes cluster hosts several applications. Three of them: A, B, and C are listed as examples. They can be distributed across the whole cluster on all nodes regardless of how services, including Egress Gateway, are configured. Each application has a set of requirements translated to general and specific design considerations that might be relevant use cases for your organization.

Application A requirements

1. Application A (red color) is an example e-commerce app for a retail or SaaS organization exposed externally via Ingress Services.
2. The application requires fast horizontal scalability and equal distribution of the traffic load across pods, including traffic spikes. The application's connections initialized from outside to inside traverse from the external network to any nodes. Then, depending on a load-balancing configuration, they are distributed in a Kubernetes cluster.
3. Advertisement of IP pools belonging to pods or Kubernetes services should be automatic.
4. No traffic initialized by internal pods to external networks is allowed.

Application B requirements

1. Application B (green color) is an example IoT and metrics data processing application. IoT telemetry devices and probes collect the data and send it to the processing app located on a Kubernetes cluster. Service IP addresses advertised automatically with a prefix filtering.
2. This application has to scale out horizontally as well, but the traffic rate is predictable without spikes. Data processing utilizes specialized TPU cards which are hosted on specific nodes. That's why the traffic distribution is expected per node.
3. App B should store data in a controlled way in an external database. This means the traffic initialized by App B should have a deterministic source IP address used as an identification on an inline firewall and listed nodes used as exit points to external networks.
4. Avoid a performance degradation and packet drop by managing a traffic asymmetry.
5. The traffic to Egress Gateway IPs should go via ISP:South links.
6. The traffic to Ingress Services IPs should go via ISP:North links.

Application C requirements

1. Application C (yellow color) - is a Cron Job sending applications' configurations, backups and other data to a repository. It is not exposed externally. Entities cannot initialize connections from outside to this application as there is no well-known TCP port listening for connections.
2. App C initializes the connections from inside to outside, and traffic goes through the Egress Gateway nodes, performing SNAT on dedicated IP addresses. The return traffic related to an established connection can be sent from outside to inside.
3. No dynamic advertisement of prefixes, but a bidirectional communication must be possible for the traffic initialized internally.
4. Bulk data sent to the repository should use the AWS AZ (Availability Zone) East links, instead of AWS AZ-West links to avoid unnecessary charges.

Across this deployment, Ingress and Egress Gateway services are playing different roles and are independent to each other. They can be configured separately for each application.

Ingress Services

“Ingress services” is used in this document to describe the different Kubernetes and Cilium mechanisms resources (i.e. Gateway API, Ingress, Envoy, Services) available to expose the application externally or within a selected area.

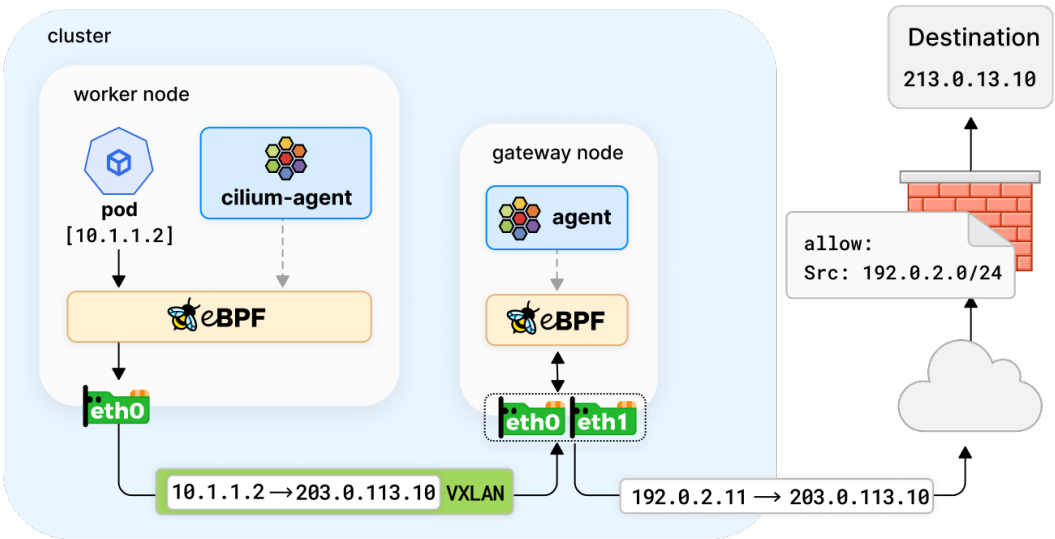
Users or other services can then connect to the exposed applications. A connection can be enriched with URL/URI, paths, headers, and other OSI Layer 7 information to influence the traffic management decision or provide a selected service. Additionally, security services such as SSL/TLS termination, authentication and authorization, certificate management, and traffic encryption can be provided.

Egress Gateway (EGW)

Egress gateways provide a way to route selected or all outbound traffic from certain pods using a deterministic IP address. This IP is allocated on purpose to the workload and can be useful to distinguish applications from each other and apply different security policies at the edge or on the firewall.

This is useful for scenarios where traffic destinations require a known source IP. Without an egress gateway, outbound traffic from pods typically use random node IP's, making it harder to enforce consistent security policies or manage predictable traffic flows.

In the following example, the pod with IP 10.1.1.2 exits the Kubernetes cluster with IP 192.0.2.11. It is then inspected by a firewall, translated by SNAT and routed further towards the destination IP 213.0.13.10.



Other pods selected by labels can share the same IP or have a different IP allocated from the IP address pools.

Egress Gateway requires the following features:

- BPF-based IPv4 masquerading (NAT)
- Kube Proxy Replacement (KPR)

> Egress Gateway is not compatible with kvstore, use the CRD mode.

Connections

Connections are TCP bidirectional traffic flows. When the initiator sends TCP SYN, the responder sends TCP ACK.

- "In -> Out" - means the connection initiated from inside the cluster and traverses to an entity at the external network.
- "Out -> In" - means the connection initiated from an entity at the external network and traverses to inside the cluster.

The return traffic from the responder to the initiator does not have to be symmetric. For example, if a connection is initialized from Out to In to App B (green), the return traffic goes back through the same services. It does not go through the Egress Gateway App B uses for self-initialized connections.

BGP role

BGP is not explicitly mandatory, but highly recommended. In the architecture diagram App-A and App-B use the BGP protocol. App-C uses a static routing.

BGP protocol advertises prefixes that describe either pods/services or Egress Gateway SNAT IP addresses. The dynamic nature of Kubernetes and networking topology can benefit from the dynamic advertisement capabilities of BGP. Any changes to pod CIDR or service LB IP allocations and services IP pools are automatically reflected in updates sent to BGP neighbors.

Traffic Engineering Matrix

Typically, traffic flow is bidirectional (inbound <> outbound). Each direction can be managed separately for inbound and outbound initialized traffic. That's why there are four directions:

- **Initial Inbound**: Traffic is initialized from an external network, traversing to inside the Kubernetes cluster.
- **Outbound Return**: Return traffic from inside to outside, exiting the Kubernetes cluster.
- **Initial Outbound**: Traffic is initialized from inside to outside, exiting the Kubernetes cluster.
- **Inbound Return**: Return traffic from an external network, traversing to inside the Kubernetes cluster.

Traffic steering in Kubernetes often requires configuring controls in the opposite direction of the traffic flow: So for the traffic:

- **Outbound**: configure steering methods in the Kubernetes cluster towards pods/services or through BGP/routing towards the Kubernetes cluster.
- **Inbound**: configure steering methods through BGP/routing towards external networks.

Traffic engineering with Cilium can be considered in the following variants depending on the service used.

Traffic originated from	Destination	Service	Steering direction	General steering method
External Network	Kubernetes	Ingress, API Gateway, or LB	Initial Inbound	BGP/routing externally, then load balancing in K8s
External Network	Kubernetes	Ingress, API Gateway, or LB	Outbound Return	Symmetric path to Inbound in K8s, then BGP/routing externally
Kubernetes	External network	Egress Gateway	Initial Outbound	Topology-aware EGW in K8s, then BGP/routing externally
Kubernetes	External network	Egress Gateway	Inbound Return	BGP/routing externally, then a direct VXLAN without steering

Inbound Traffic Engineering

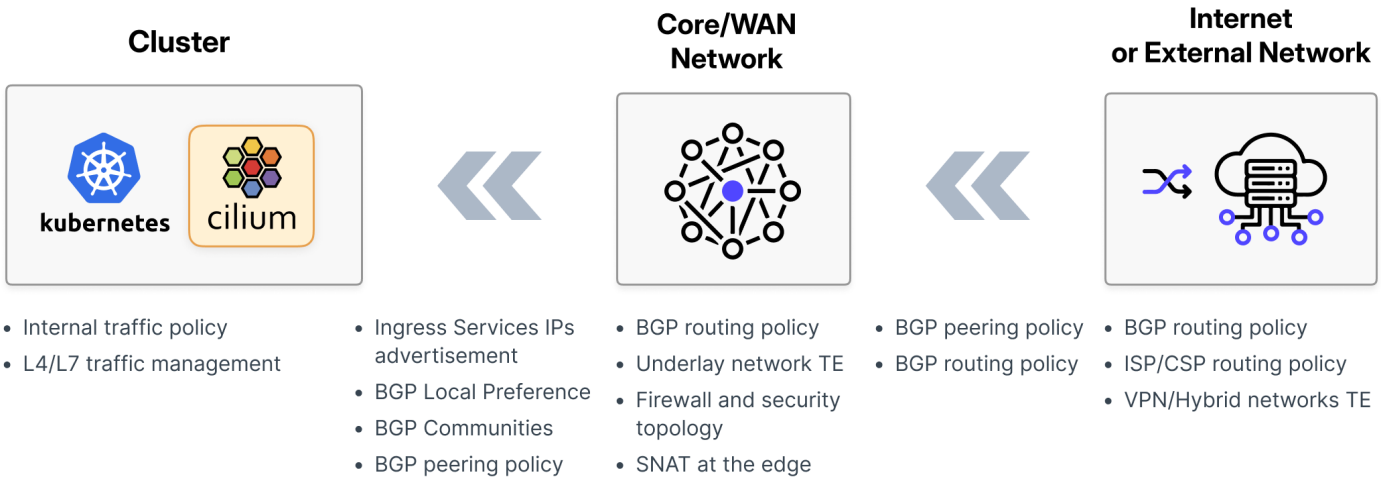
The following points and diagrams describe feasible steering methods for inbound traffic engineering (abbreviated as TE in following diagrams).

Initial Inbound

Traffic is initialized from an external network source.

- Packets are sent to IP addresses exposed by ingress services (not the pod IPs).
- Traffic is steered first by BGP or routing protocols externally, then by ingress services within the Kubernetes cluster.

Initial inbound traffic from an external network to a K8s cluster

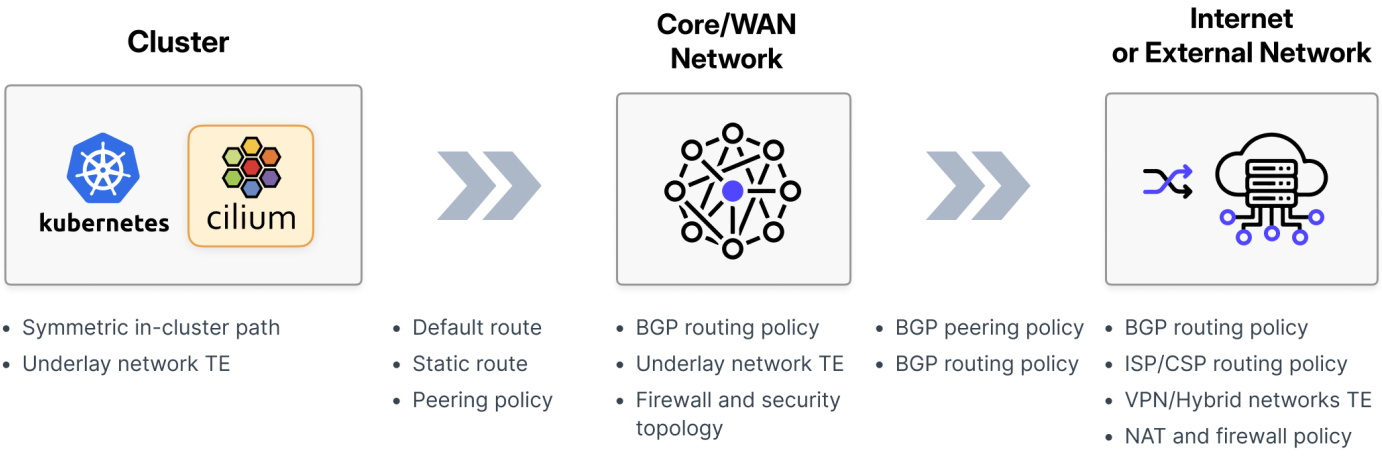


Outbound Return

This is the return traffic for the previous inbound case.

- Packets use a symmetric path through ingress services inside the Kubernetes cluster.
- Traffic is then steered by BGP and other routing mechanisms externally.

Return outbound traffic from a K8s cluster to an external destination



Outbound Traffic Engineering

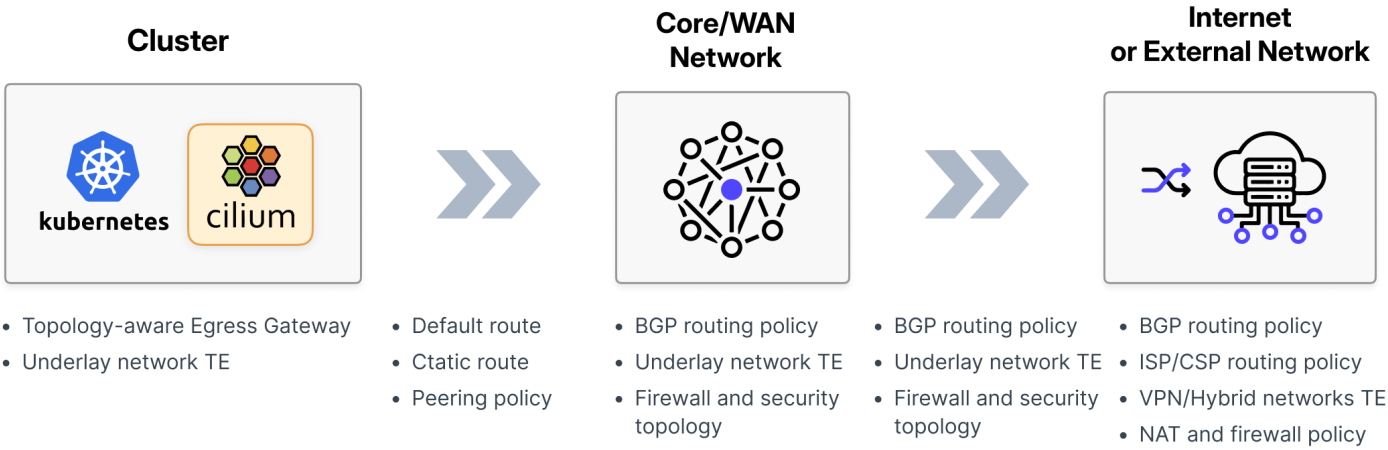
The following points and diagrams describe feasible steering methods for outbound traffic engineering.

Initial Outbound

Traffic is initialized from a source pod to an external network.

- Encapsulated into VXLAN, then sent to the Egress Gateway node which performs Source NAT (SNAT).
- Packets leave the Kubernetes cluster, traversing to the destination through external networks where steering is done via BGP and other routing techniques.

Initial outbound traffic from a K8s cluster to an external destination

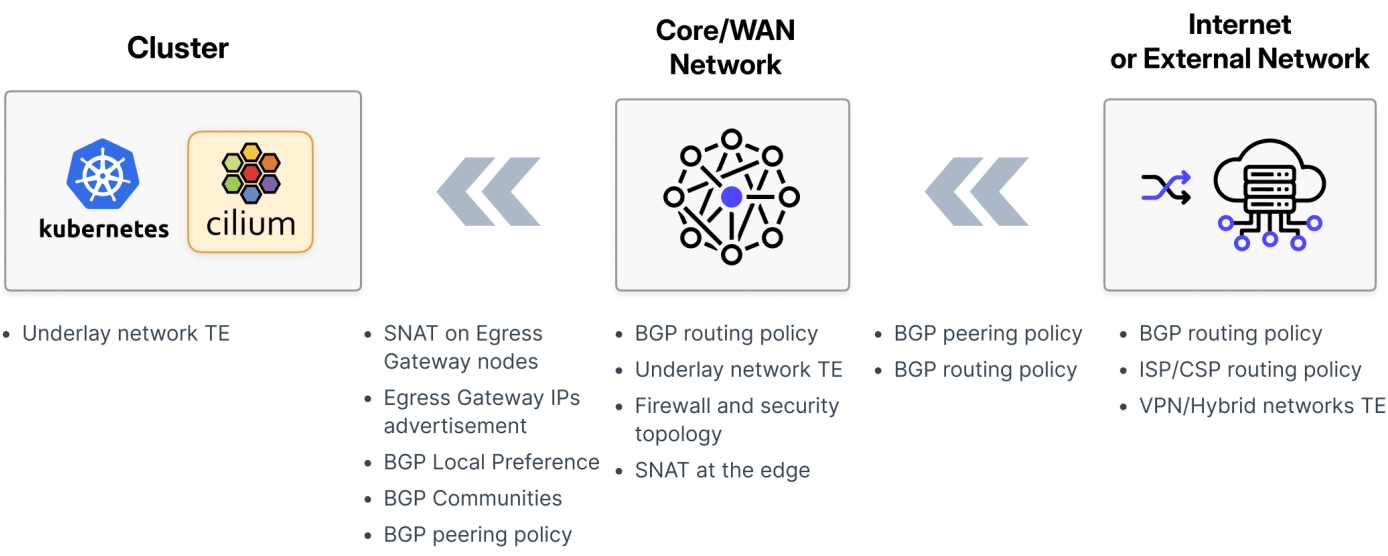


Inbound Return

Traffic is first steered by BGP or routing protocols to the Egress Gateway node.

- Destination NAT (DNAT) is performed based on an entry in the conntrack table revealing a pod's destination IP.
- Traffic is VXLAN encapsulated and sent directly to a destination worker where the pod runs.

Return inbound traffic from an external network to a K8s cluster



Non-feasible patterns

Certain traffic steering scenarios are non-feasible:

Traffic originated from	Destination	Service	Steering direction	General steering method
Kubernetes	External Network	Ingress/API Gateway/LB	Outbound	Outbound connections are using NAT or EGW
Kubernetes	External Network	Ingress/API Gateway/LB	Inbound Return	No connection established
External Network	Kubernetes	Egress Gateway	Inbound	An open egress port is dynamic, the inbound connections cannot be established
External Network	Kubernetes	Egress Gateway	Outbound Return	No connection established

Egress Gateway performs port-based SNAT, opening a dynamic port after establishing a TCP connection from inside to outbound. Before the TCP connection is established, traffic cannot be initiated from the external network. Once a dynamic source port is open on the egress node, traffic can only be sent and accepted if it matches the same port and TCP sequence numbers.

Traffic Engineering with Cilium

This section expands on technical best practices behind traffic steering and dives deeper into the three applications (A, B, C) from the architecture diagram above for design designs with Cilium.

First we start with an overview of best practices, before moving deeper into application specific design choices.

General Recommendations

This section describes general best practices and noteworthy compatibilities for traffic engineering with Cilium.

Unmanaged Pods

Traffic engineering techniques can be applied to pods with the CNI managed by Cilium. It is recommended that workers be manipulated with Kubernetes' taints to help prevent pods from starting before Cilium runs.

Placement of application pods

Technically, application pods can be located on any node, including Egress Gateway (EGW) nodes. From the perspective of resource allocation and separation of duties, dedicated EGW nodes without application pods are recommended, especially in larger clusters.

Ingress services

One or more Ingress Services must be configured for applications exposed to external networks. Services need to be labeled and selected for a BGP advertisement or the traffic to be routed statically. This is required for external networks to reach services.

This document focuses on external interactions. It does not cover in-detail in-cluster L4/L7 traffic management.

Kubernetes with Cilium offers four cluster-internal Layer 4/7 traffic management options, which can be applied in the inbound direction. These four are shown in the diagram below.

Ingress	Services	Gateway API	EnvoyConfig
Original L7 load-balancing standart in K8s	Use of K8s services with annotation	Originally labelled Ingress v2. Richer in features	Raw Envoy Config via CustomResource
L7 Simple	L4 `service.cilium.io/lb-l7=disabled` L7 `service.cilium.io/lb-l7=enabled` Simple	L7 Simple	L7 Advanced
Supported since Cilium 1.12	Supported since Cilium 1.13	Supported for v0.5.1 since Cilium 1.13	Supported since Cilium 1.12

LoadBalancer IP Address Management (LB IPAM) is recommended to better control IP pool allocations and advertisement through BGP.

Cilium provides proxy-based load balancing in addition to standard Kubernetes services, enabling the distribution of network traffic across multiple pods. This abstraction allows the pods to reach out to other pods by a single IP address, a virtual IP address, without knowing all the pods running that particular service.

VXLAN overhead

The Egress Gateway feature utilizes VXLAN tunneling to transport packets from a source worker to an Egress Gateway Node.

Due to the additional encapsulation, VXLAN tunneling introduces a 50-byte overhead to the original frames. Therefore, the underlay network must set IP MTU to at least 1550 bytes or higher. Otherwise, the packet drop may occur for IP packets larger than 1450 bytes.

> In data centers, it is recommended that 9000+ bytes of MTU be configured to increase performance and minimize VXLAN overhead.

BGP Protocol

BGP, though not mandatory, is recommended in medium- to large-scale environments. This provides better security control, simplifies management of prefixes, and adds additional traffic steering options.

As an alternative to BGP, static routes can be configured and redistributed into a dynamic routing protocol as needed. However, this introduces additional management overhead of the platform.

> Many routers have a limit on the number of ECMP paths they can hold in their routing table. You may exceed this limit when advertising the Service VIPs from many nodes.
>
> To avoid reaching this limit on ECMP paths on network devices, we recommend the following:
>
> - use the `Local` type of the `externalTrafficPolicy`
> - configure BGP sessions of selected groups of nodes that distribute the traffic inside a Kubernetes cluster

The Cilium agent advertises prefixes for both ingress and egress services but does not insert prefixes received from a neighbor. Thus, a default or a specific route must be present in the routing table on a host.

Ingress Services IPs Advertisement

Although BGP can advertise PodCIDR prefixes of the nodes, it is typically recommended to advertise virtual service IPs.

This way, external networks are separated from internal prefixes tied to pods. This approach's advantages include decoupling services from pod deployments, which bring higher flexibility, more stable prefix advertisements, better isolation of private address spaces, and increased security.

You can advertise IPs associated with the following K8s parameters: ClusterIP, ExternalIP, LoadBalancerIP.

Egress Gateway IPs Advertisement

It is recommended that the Egress Gateway IPs used on active gateway nodes through BGP are advertised using the Cilium BGP Control Plane.

BGP can automatically distribute prefixes describing the services and applications behind the egress IPs to other networks. This provides security control over what is advertised and allows for the steering of inbound traffic.

If BGP cannot be used, then static routes can be added between a Cilium and the rest of the network. The edge network devices can redistribute these static routes to a chosen dynamic routing protocol.

Egress Gateway interfaces

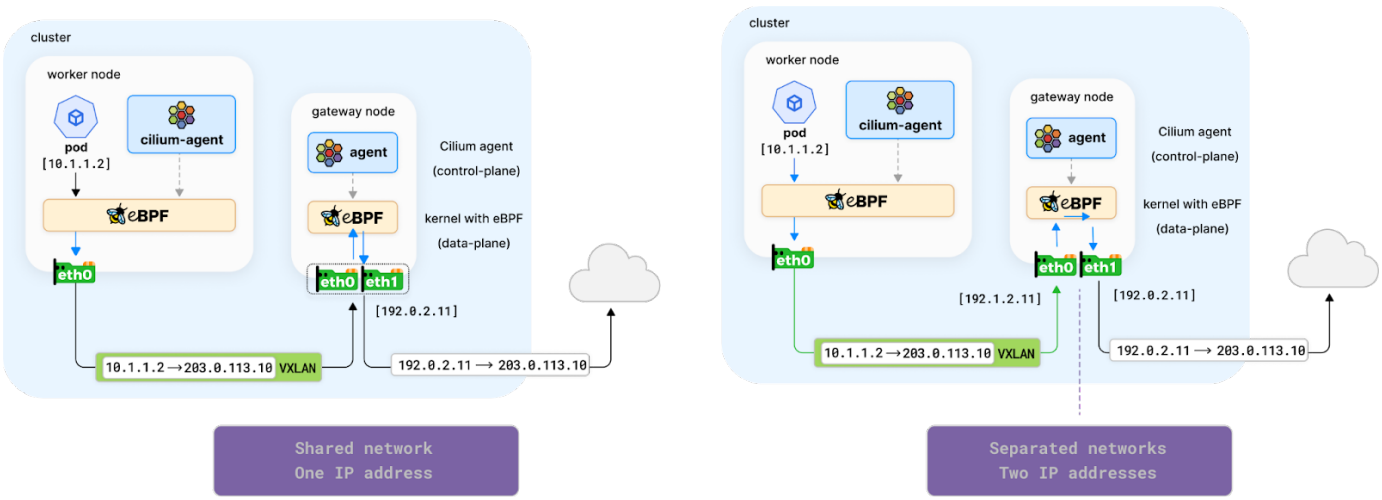
Egress Gateway nodes support the Egress Gateway feature with one or more network interfaces. With two or more interfaces, there are several options including, but not limited to:

- using interface teaming to create one logical interface (bundle interface)
- using separate interfaces for in-cluster and external traffic

The difference between these options is in how the traffic is split across (see the picture below):

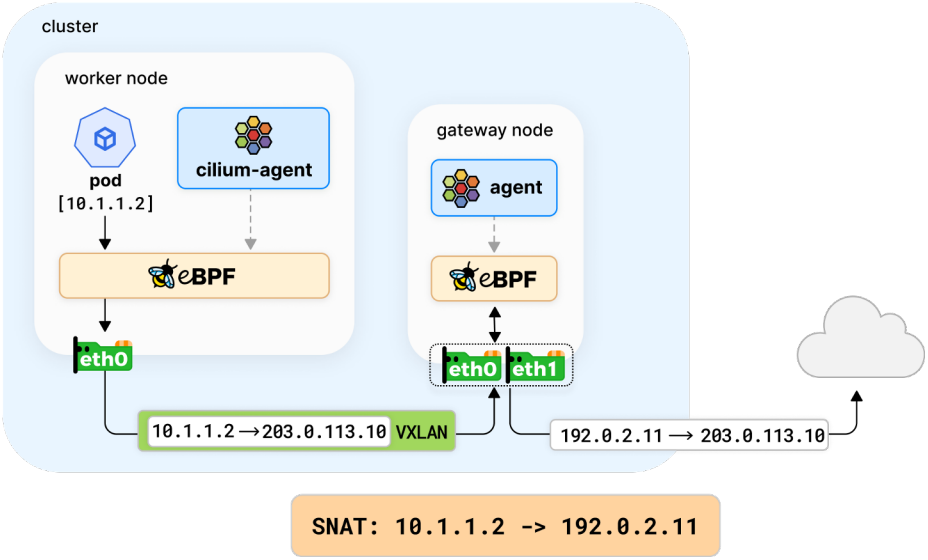
- upstream external devices and worker's interfaces
- worker's interfaces and the kernel

In a bundled option the traffic enters and leaves the worker using all physical interfaces.
In a separated option the traffic enters using one interface and leaves the other.



One or more interfaces can also connect other Kubernetes workers. A multi-interface setup can enforce network isolation or traffic management policies.

Let's look at the below diagram as an example. Inbound and outbound traffic are distinguished on egress nodes on the same or separate interfaces. Traffic is received internally as the VXLAN-encapsulated stream, then translated to the egress IP and sent externally as an IP-native stream. In the following example, the traffic from 10.1.1.2 is translated to the egress IP: 192.0.2.11.



Overlay coexistence

All traffic engineering scenarios with Cilium can coexist with other solutions deployed in the same network, data center, and cloud.

VXLAN encapsulation of Egress Gateway uses port UDP 8472 by default. Two or more overlays can coexist in the same network using the same UDP port if they use different VTEPs. Thus, networking controllers span the overlay VXLAN network across unique unicast IP addresses.

> Changing the VXLAN UDP port is recommended if a unique identification of the overlay service is required.

Application-specific design considerations

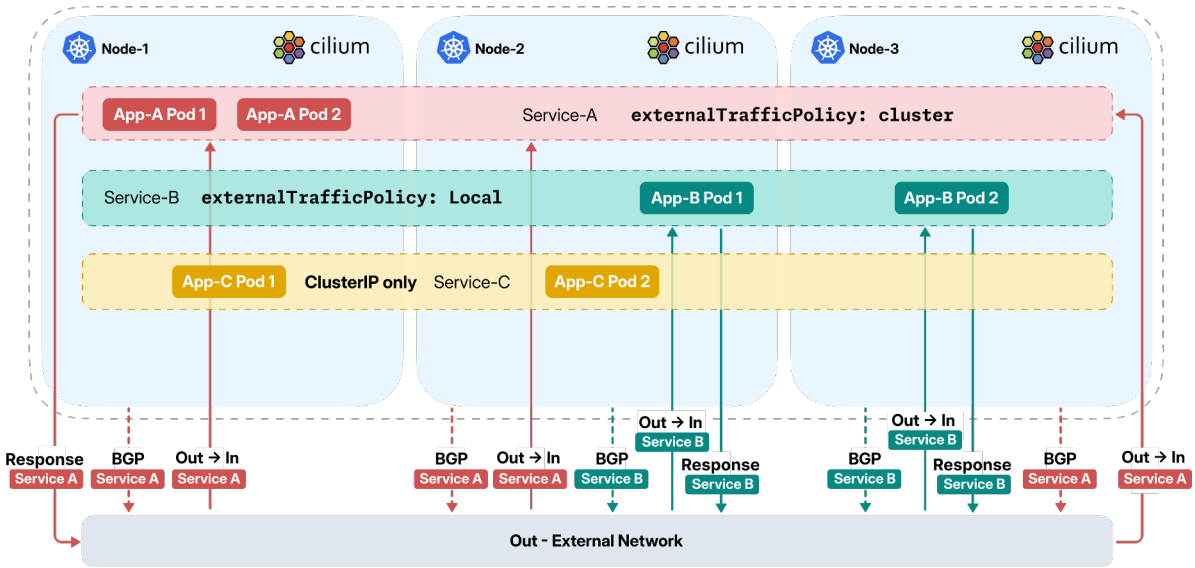
Using the same frame of reference from the Architecture Diagram section (Apps A, B, and C), the following section connects key traffic engineering concepts and design choices with Cilium, mapping them back to the architecture diagram.

- Exposing applications externally
- Controlling and identifying outbound connections
- Distributing traffic with Ingress Services
- Configuring outbound exit preference
- Avoiding asymmetric traffic
- Inbound traffic engineering with BGP
- Applying static routes

Exposing applications externally - App A and App B

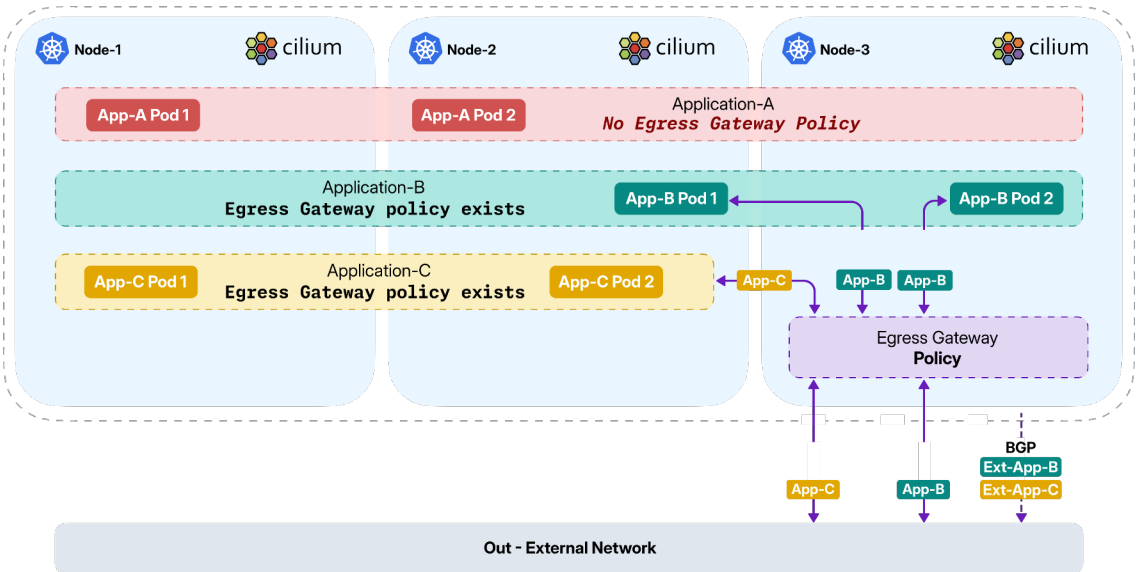
Ingress services expose applications to external entities. The following diagram depicts an example of three applications. Two of them are exposed externally through the Services: Service-A and Service-B (`requirement App A/1, App B/1`).

App C is not exposed externally (`requirement App C/1`). It is exposed internally through ClusterIP Service-C.



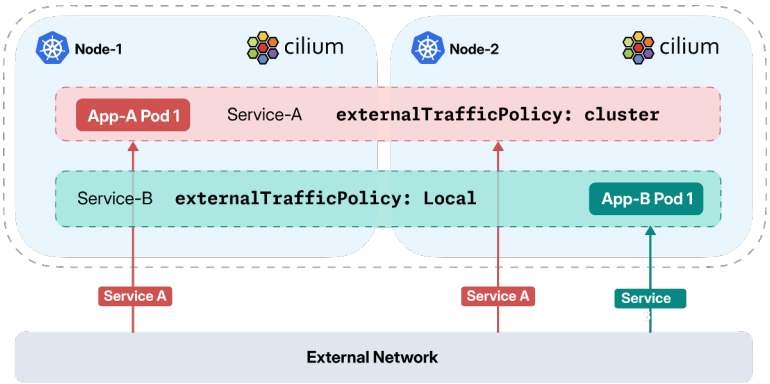
Controlling and identifying outbound connections - App B and App C

Egress Gateway (EGW) services allow cluster workloads to initialize traffic to external entities (In -> Out) through a dedicated IP address assigned to applications. A response comes through the same path in reverse direction (Out -> In) marked as Response in the following picture. App B and App C have EGW services configured (`requirement App B/3, App C/2`). App A has no such option (`requirement App A/4`). BGP advertises IP addresses used for SNAT on EGW Node-3.



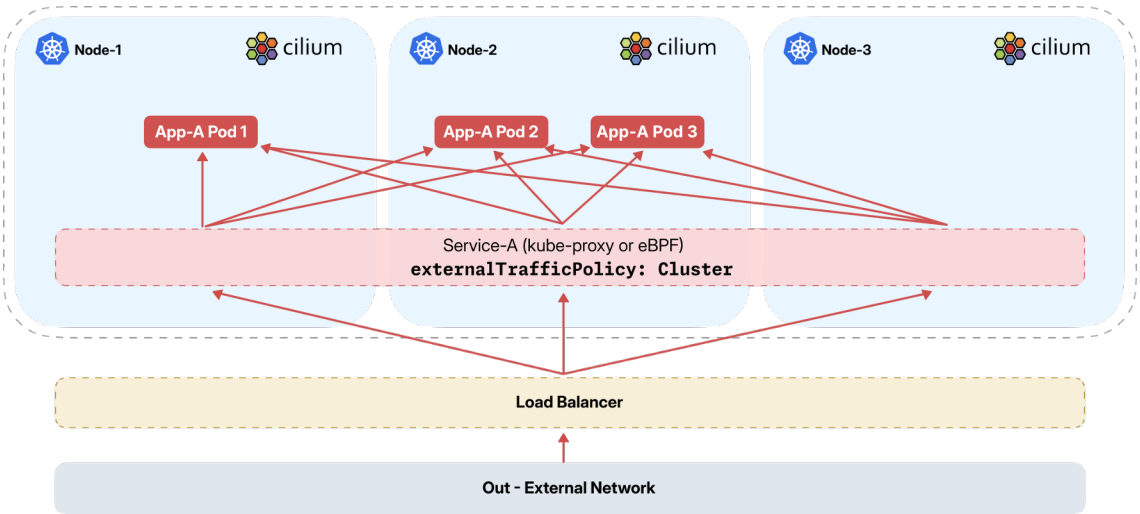
Distributing traffic with Ingress Services - App A and App B

In the diagram below, App A (red) is exposed via Service-A, App B (green) is exposed via Service-B, and their IP addresses are advertised through BGP (`requirement App A/3, App B/1`).



IP addresses of load balancing services are exposed externally through `externallTrafficPolicy`, which describes a traffic engineering strategy for each service. Service-A is advertised by all cluster nodes that are part of the cluster, whereas Service-B is advertised only through nodes where Green pods run. More about externallTrafficPolicy.

See the following diagram for this type of traffic flow, with App-A deployed on Node-1 and Node-2, receiving load-balanced traffic across the cluster.

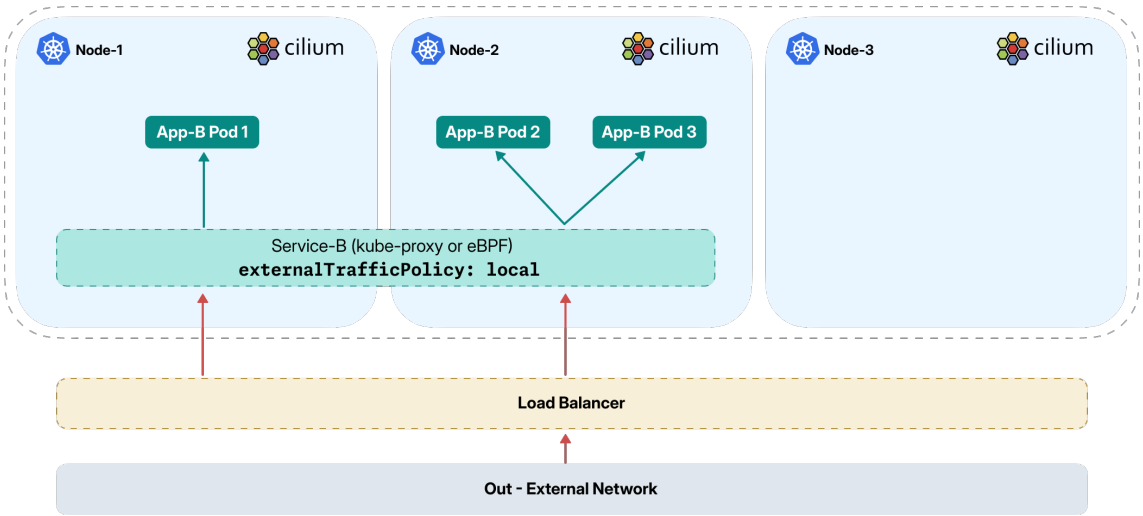


As the Service has `externalTrafficPolicy: Cluster`, the BGP Control Plane unconditionally advertises the ingress IPs of the selected Service from all nodes, achieving per-pod equal cost distribution of incoming traffic, including a rapid increase of the traffic volume (`requirement App A/2`).

The option `Cluster` is helpful if the application changes nodes frequently, for example, every minute. In small BGP topologies, it does not affect the convergence time. In more extensive networks, including the Internet, the BGP protocol requires time to update the edge devices about a change. With more frequent changes, a BGP dampening may kick in on the upstream devices to which Cilium is connected. This slows down convergence for the sake of stability. At the same time, in `Cluster` mode, the traffic is distributed evenly across all pods.

Unlike App A, Application B is exposed externally via Ingress Services with `externalTrafficPolicy: Local`. The BGP Control Plane keeps track of the endpoints for the service on the local node and stops advertisement when there's no local endpoint. Therefore, the connections initialized from outside to inside go from the External Network to selected nodes with dedicated TPU cards where the pods of application B exist (`requirement App B/2`).

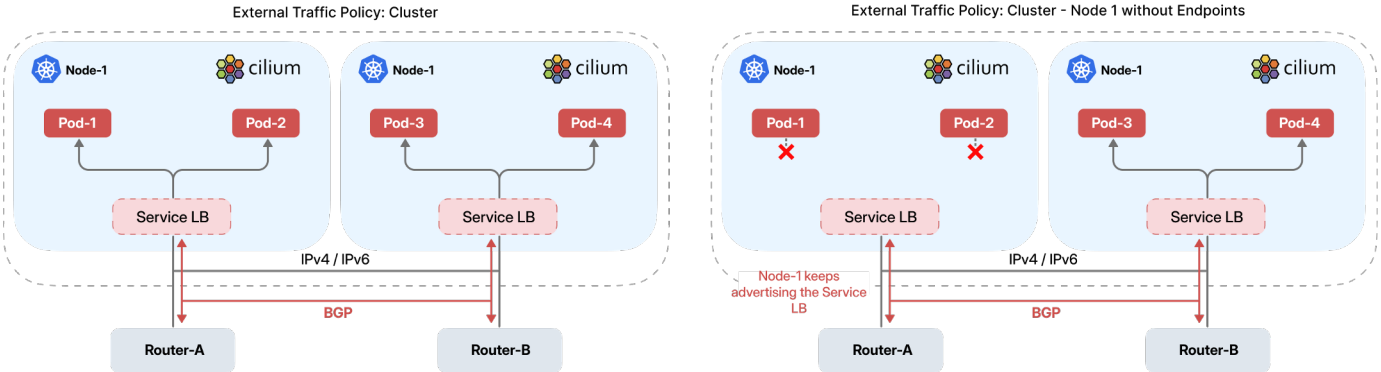
`Local` is used to preserve the original client IP and is more optimal for less dynamic pod re-scheduling or smaller networks. In this case, the traffic is directed to a worker where the application pods run, which conserves networking and host resources by avoiding sending traffic across nodes without the particular application. But it can lead to an uneven load distribution across the pods if the number of pods of a particular deployment is not the same on each worker. This situation is depicted below.



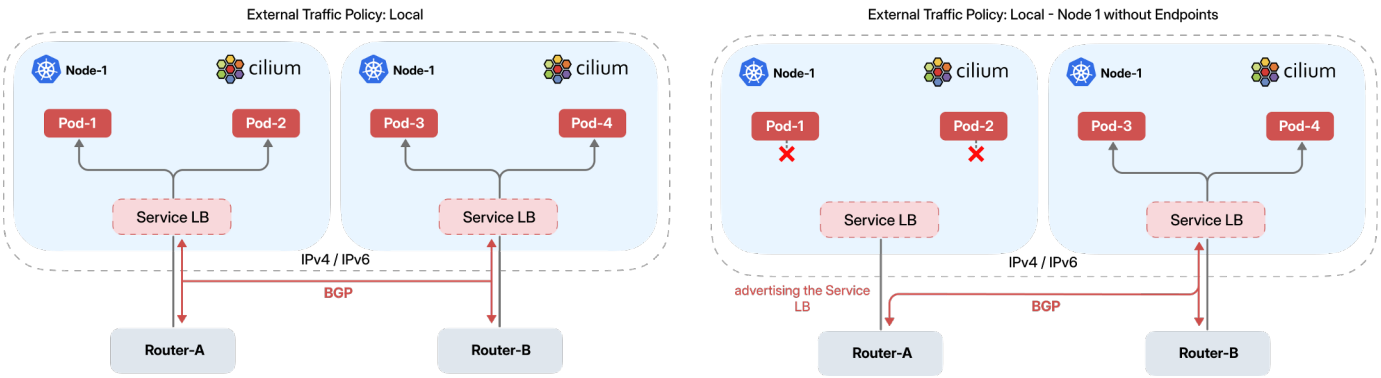
The traffic is distributed across workers with equal cost, but not evenly across the pods. If pods are distributed on all three workers or scale out to the even number, then roughly the same amount of the traffic is delivered to each pod. The uneven load distribution can be also mitigated through pod anti-affinity settings to distribute pods to as many nodes as possible.

The `externallTrafficPolicy` attribute affects also the way how BGP advertises prefixes related to services upon a failure of pods.

The following two diagrams compare a normal state of operations for the `externallTrafficPolicy=Cluster` and a failure or removal of pods from Node-1. In this case, Node-1 keeps advertising the service load balancer's IP address to Router-A and Router-B. Therefore, routers send traffic to both Service LBs located on Node-1 and Node-2.



The following two diagrams compare a normal state of operations for the `externallTrafficPolicy=Local` and a failure or removal of pods from Node-1.



In the case of a failure, Node-1 stops advertising the service load balancer's IP address to Router-A and Router-B. Therefore, routers send traffic to the Service LB located on Node-2 only.

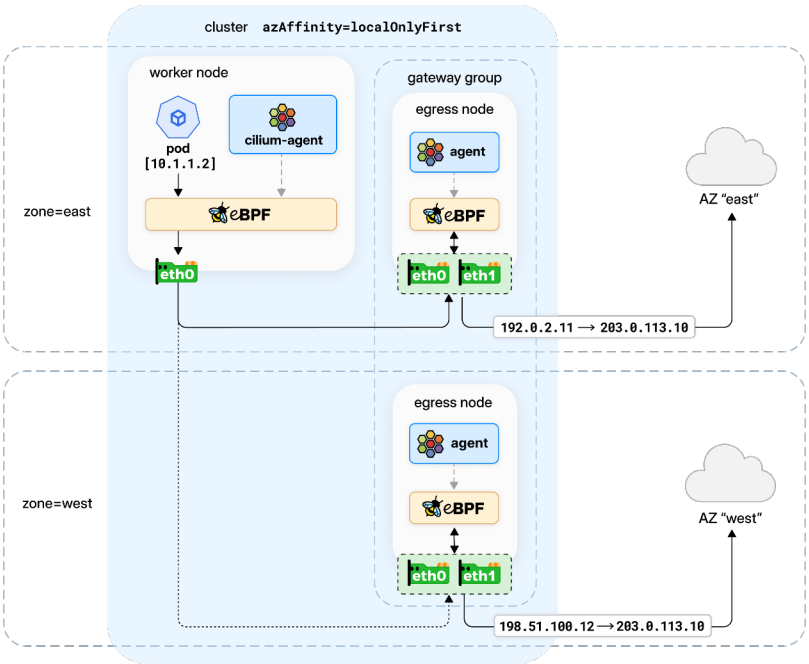
Similar considerations are applicable for more distributed topologies using Cluster Mesh. In such cases, there are two traffic policy levels:

- Cluster-mesh load balancing & service discovery
- In-cluster internal traffic policy

Configuring outbound exit preference - App C

It is recommended that the egress gateway traffic be controlled with topology-aware settings for a Kubernetes cluster spread across multiple cloud availability zones (AZ).

Without the topology-aware Egress Gateway feature, the traffic is load-balanced across all egress nodes regardless of the AZ location of a source pod. With this capability, Cilium can steer the traffic from Pods to the egress nodes, with a preference for those in the same AZ. Such configuration can optimize resources, decrease latency, and avoid inter-AZ charges (`requirement App C/4`).

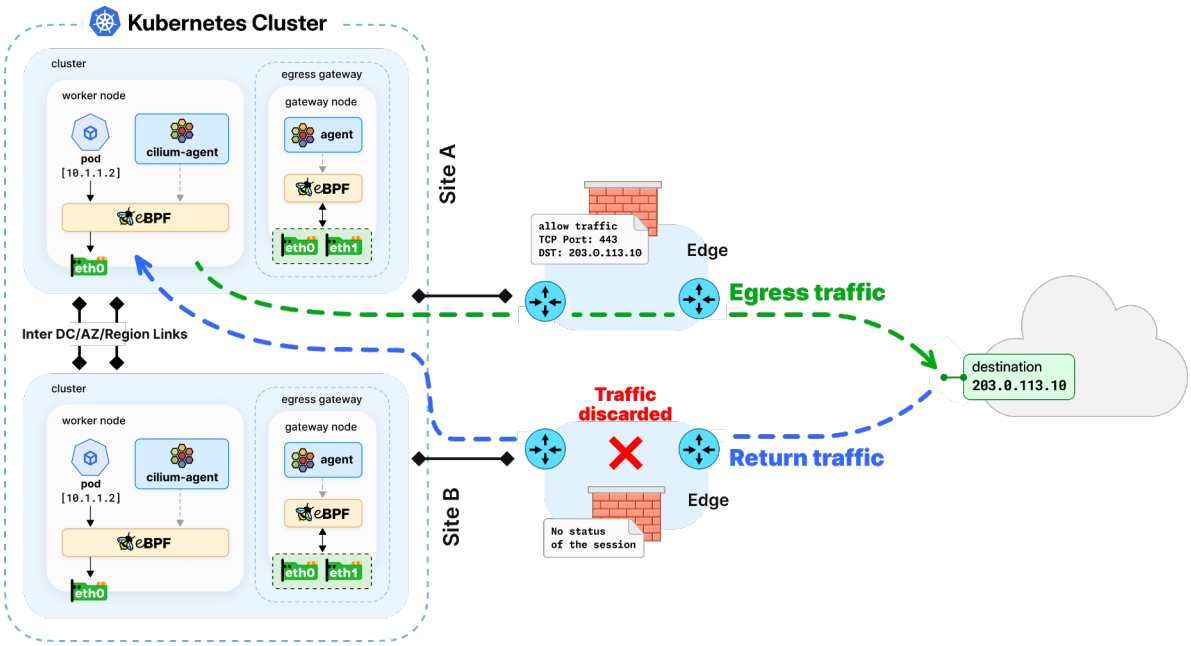


As depicted in the diagram above, pod 10.1.1.2 is located in the zone east with Egress Gateway `azAffinity` set to `localOnlyFirst`. Local gateways will be preferred in this mode as long as at least one gateway is available in a given AZ.

Avoiding asymmetric traffic - App B

For App-B when the app is connecting to the external database, a performance degradation and packet drop should be avoided by managing a traffic asymmetry (`requirement App B/4`).

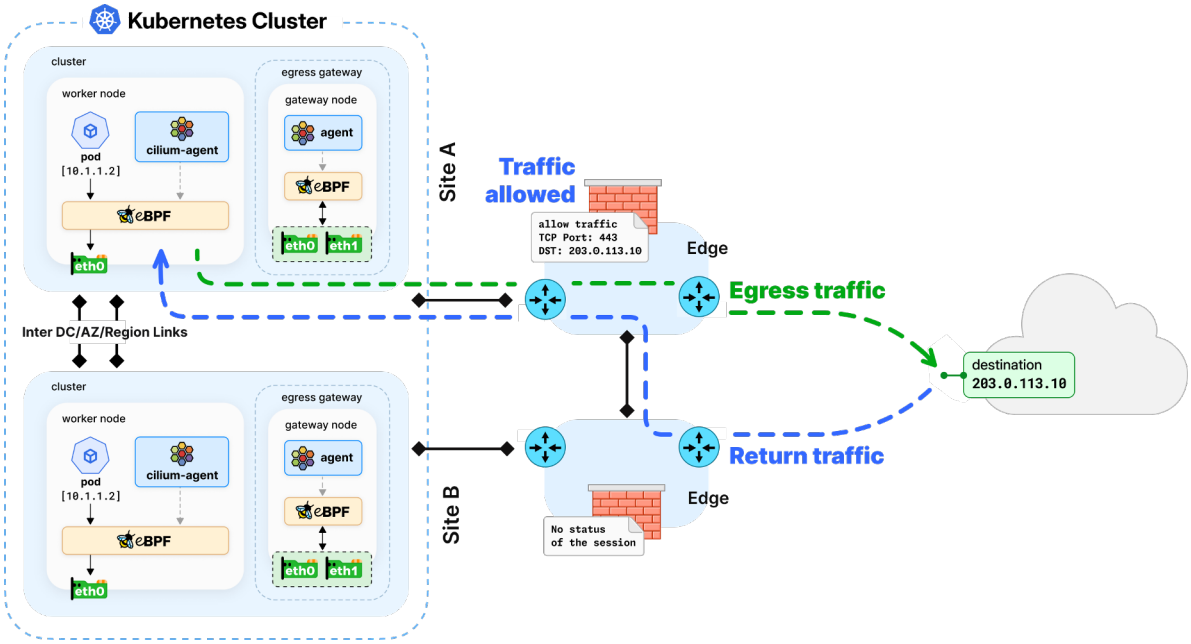
Traffic can become asymmetric when IP prefixes are advertised through multiple peering points. This can lead to issues with firewalls that operate between the source and destination, as these firewalls may drop packets if they do not share connection status. For example, if the initial packet bypasses the firewall but the SYN/ACK reply goes through it, the firewall, without connection status, will discard the traffic.



There are several ways to improve this design:

- **Reduce BGP Session Redundancy**:
  - Egress nodes should advertise their IPs only on the local site rather than on other sites. This approach reduces unnecessary redundancy.
  - Routing redundancy can be achieved by connecting two core networks together.
- **Optimize Firewall Placement**:
  - Position firewalls closer to the Kubernetes cluster where aggregation points are more distinct.
- **Enable SNAT on Firewalls**:
  - Turn on SNAT on firewalls for source IPs from Egress Gateway nodes. This makes traffic symmetric as the return traffic is diverted through the SNAT-performing firewall.
  - Double SNAT (first on Egress Gateway nodes and second on firewalls) is not an issue for typical HTTP applications.

Design improvements are shown in the following diagram.



General recommendations to avoid issues with asymmetric traffic on stateful firewalls:

- **Advertise Prefixes Strategically**:
  - Advertise prefixes over actively used peering points. In case of a site failure, switch advertisement and traffic to another site.
- **Policy Configuration for Symmetric Traffic**:
  - Make traffic symmetric by advertising prefixes over one site with higher preference and a backup site with lower preference.
- **Optimal Firewall Placement**:
  - Place firewalls at strategic points to aggregate traffic effectively. Depending on the topology, this can be at peering points, within the core network, at the edge of data centers/cloud, inside the data center/cloud, or closer to the application layer with distributed functions.
- **Use Status Synchronization or Clustering**:
  - Implement status synchronization or clustering between firewall units participating in traffic inspection.
  - Common practice includes using firewall clusters.
- **Active/Standby Configuration**:
  - Configure firewalls as active/standby, with the active one handling outbound and return traffic.
- **Disable Connection Tracking or Use Stateless Firewalls**:
  - Disable connection tracking or use stateless firewalls to avoid issues with asymmetric traffic on stateful firewalls.

Inbound traffic engineering with BGP - App B

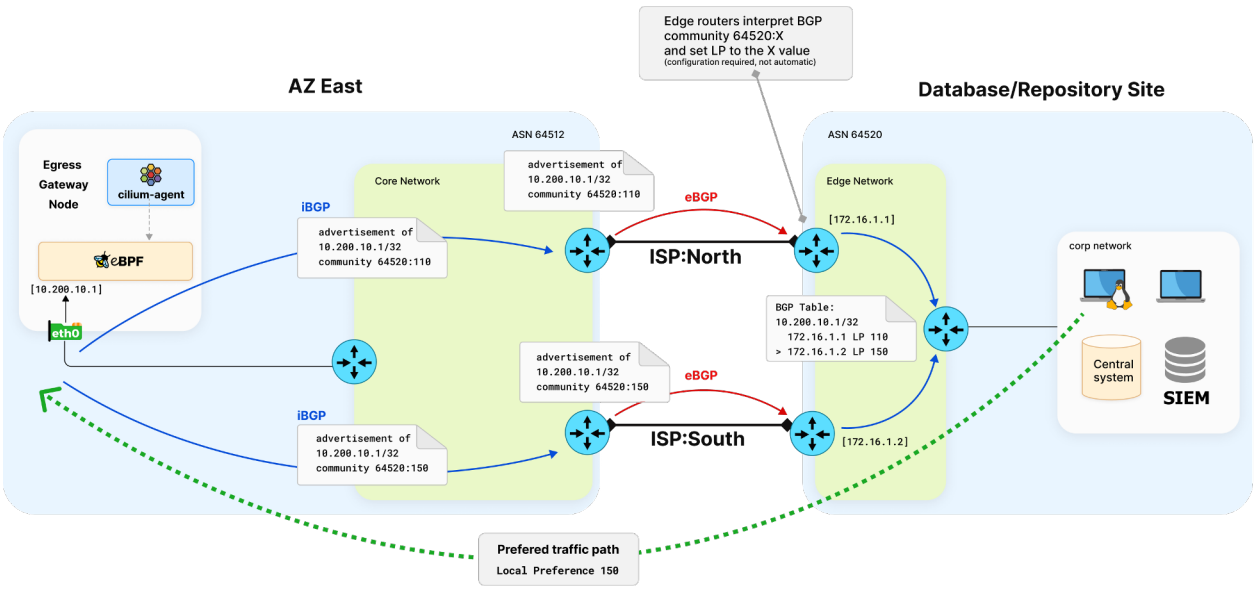
The traffic originating from the external network to a Kubernetes cluster can be steered along the path by BGP attributes. BGP devices receive prefixes from multiple neighbors and can decide which route is preferable based on the BGP Best Path selection process.

The following examples show how BGP attributes can affect routing decisions.

- Networking devices can filter out prefixes based on specific values of BGP attributes, like community.
- A higher BGP Local Preference can be set to prefer one path over another in the same BGP Autonomous System (AS).
- BGP AS-PATH prepending or Local Preference can be used in other Autonomous System (AS). Both parameters can be set through BGP communities with the upstream device or a service provider. Below is an example of setting up Local Preference (LP) through the BGP community.

Example 1: The traffic to Egress Gateway IPs should go via ISP:South links (`requirement App B/5`).

For simplicity, let's assume that a Cilium node in AS 64512 has two iBGP sessions to internal routers. It advertises the IP address 10.200.10.1/32 allocated to a load balancer service using two BGP community values: 64520:110 and 64512:150.

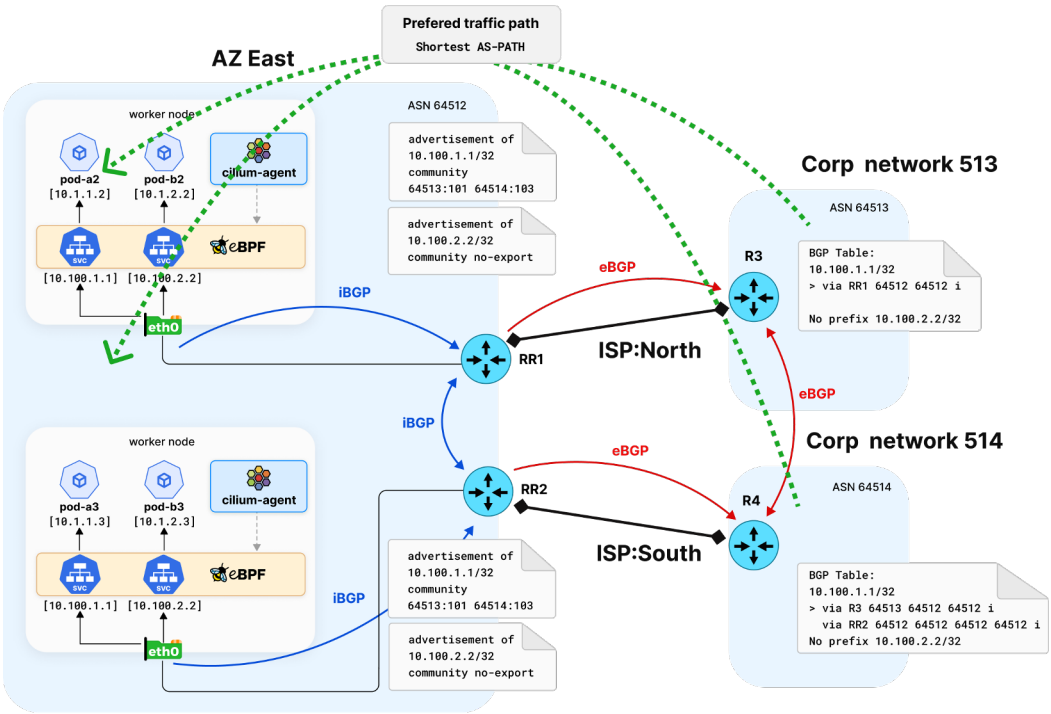


The sessions are received by the edge routers in AS 64520. The BGP communities are then interpreted according to a local BGP policy. In this case, a configuration sets the Local Preference (LP) values to 110 and 150, respectively. This approach enforces the routing in the remote site to send the traffic through the ISP:South link as the higher LP wins.

Example 2: The traffic to Ingress ServicesIPs should go via ISP:North links (`requirement App B/6`).

In this example, there are many corporate networks with IoT/probe devices sending data to App B. They are connecting to the IP address 10.100.1.1 identifying App B externally. The goal is to prefer the ISP:North link over ISP:South. There is also another IP address 10.100.2.2, but this won't be advertised at this time.

To deploy this scenario, the BGP communities and AS-PATH can be utilized to enforce the traffic engineering policy at the edge devices RR1 and RR2 as depicted below.



Cilium nodes advertise two services:

- IoT service with the IP address: 10.100.1.1 - to be advertised
- A new service with the IP address: 10.100.2.2 - to be filtered out at this time

There are BGP communities attached by Cilium to advertised prefixes:

- IoT service - the BGP communities: 64513:101 64514:103. The intention is to steer the traffic through AS 64514 and RR2 as a preferable path from external networks. The BGP policy at the edge can be configured in the following way

Each BGP community received in the format of [ASN:10x] is interpreted with one of the actions:

- x=0, do not announce to these peers
- x=1, announce to ASN peers with one prepend
- x=2, announce to ASN peers with two prepends
- x=3, announce to ASN peers with three prepends

Typically, routers prepend their own ASN to prefixes advertised to eBGP peers.

In this case, 64513:101 64514:103 means that the BGP AS_PATH attribute is additionally prepended three times with ASN 64512 when sending the prefix to AS 64514 and only once to AS 64513. So, ASN 64512 in AS_PATH is four and two, respectively. A shorter AS_PATH is preferred by the BGP routing path selection algorithm over a longer AS_PATH. That's why the traffic goes from "Corp network 513" to "East AZ" directly via the link ISP:North, whereas the path from "Corp network 514" to "East AZ" is preferable over "Corp network 513", not directly.

For the other service 10.100.2.2 the BGP community is set to `no-export`, which means for BGP, the prefix with this community must not be advertised to other eBGP peers.

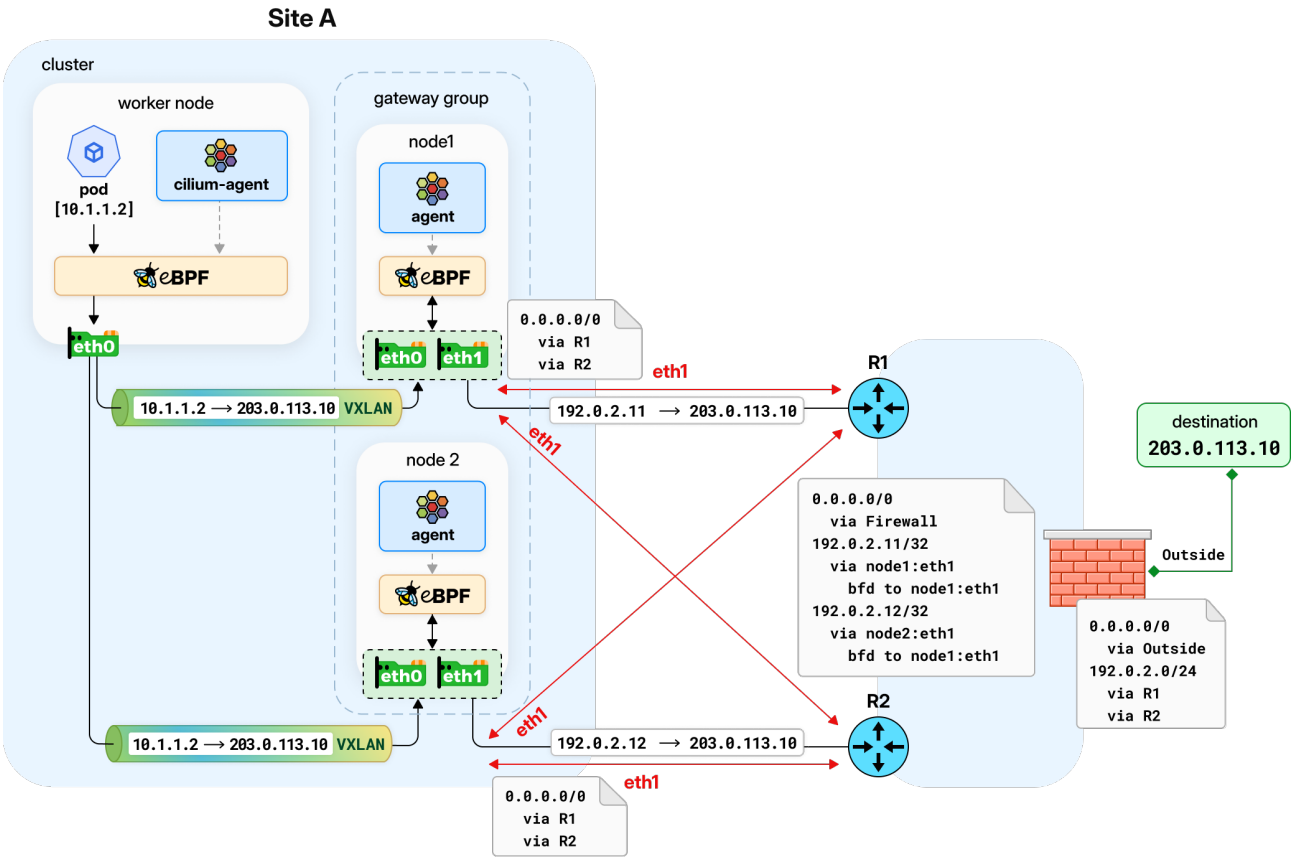
As an effect, "Corp network 513" and "Corp network 514" receive the prefix related to IoT service 10.100.1.1 only. Service 10.100.2.2. is advertised within AS 64512 and is not exposed outside.

Similarly, as described above, the BGP attributes can help steer the traffic to external destinations. The difference is that Cilium cannot influence the outbound routing policy, but the upstream devices can enforce it.

Applying static routes - App C

It is possible to configure static routes and redistribute them in the external network at the edge or in a specific place in the topology. In the case of static routes, it is recommended to add IP probes or BFD, which can automatically detect a neighbor or a remote failure and remove a stale path from the routing table (`requirement App C/3`).

Static and BGP routes can be redistributed on network devices to other dynamic routing protocols.



Final Note

Wrapping Up: Your Path to Kubernetes Networking Excellence

Effective traffic engineering is the key to unlocking performance, security, and cost efficiency. This guide has shown how Cilium, with its advanced features, can transform your approach to traffic management, making your infrastructure more resilient and adaptable.

From optimizing data flows to securing communication between services, Cilium provides the tools you need to handle the complexity of modern cloud-native environments. But mastering these tools and tailoring them to your unique needs requires more than just know-how—it requires deep expertise.

Take the Next Step: Enterprise Traffic Engineering with Isovalent

Navigating the complexities of traffic engineering in cloud native environments can be challenging, even with the right tools. This is where Isovalent's expertise shines. As the creators of eBPF and Cilium, Isovalent is uniquely positioned to help your team deploy and optimize Cilium for your specific needs.

Whether you're just starting with Kubernetes networking or looking to scale your existing architecture, Isovalent offers the enterprise-grade solutions and 24×7 support that ensure your deployment is secure, efficient, and future-proof.

To explore how Isovalent can help you maximize the benefits of Cilium in your Kubernetes environment, reach out to us today. Our team of Cilium, Tetragon, and eBPF experts is ready to guide you through the intricacies of traffic engineering and help you build a resilient, high-performance network that meets your organization's needs.

Contact Isovalent to learn more about our enterprise solutions and how we can support your journey to a fully optimized Kubernetes network.

Connectivity Visibility with Hubble

This track primarily focuses on Hubble Flow events that provide label-aware, DNS-aware, and API-aware visibility for network connectivity within a Kubernetes environment using Hubble CLI, Hubble UI and Hubble Timescape, which provides historical data for troubleshooting.

Isovalent Enterprise for Cilium: Connectivity Visibility with Hubble

Discovery: Cloud Network Engineer

In this short hands-on discovery lab designed for Cloud Network Engineers, you will learn, in 15 minutes, several Cilium networking features, including: Dual Stack IPv4/IPv6 support with Cilium, BGP, Load-Balancer IPAM, L2 Service Announcement, Egress Gateway and more!

Discovery: Cloud Network Engineer