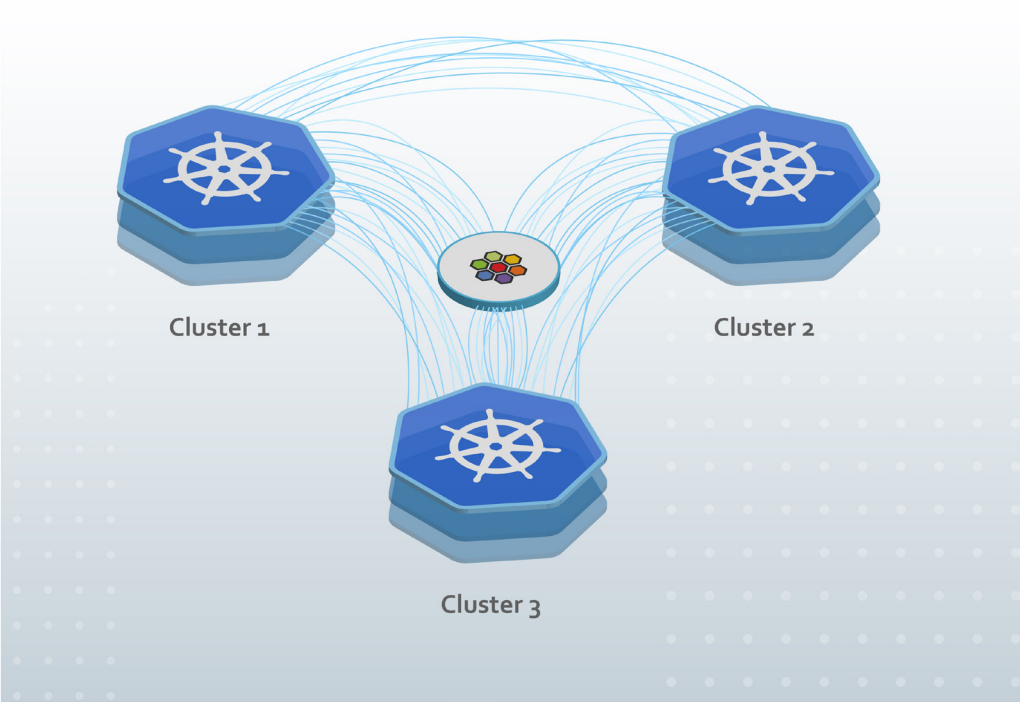
Multi-Cluster Kubernetes Networking with Cilium

Overcoming the Networking, Observability and Security challenges that slow down adoption of production Kubernetes platforms

Multi-Cluster Kubernetes Networking with Cilium

With the shift to microservices and cloud-native technologies like Kubernetes, it is possible to take advantage of near-limitless resources among other benefits that enable applications to scale with minimal downtime. Although Kubernetes makes it simple to scale distributed applications on a single cluster, its current networking model introduces significant operational complexity and performance overhead when connecting applications across multiple clusters.



Why multi-cluster?

In the early days of Kubernetes and microservices, clusters were seen as islands not easily connected using traditional Kubernetes networking. But as Kubernetes use matured, and technical requirements expanded, the need for resilient, and secure multi-cluster architectures became essential. Organizations now require the ability to control and observe multi-cluster architectures that can seamlessly discover services and load balance across clusters and zones, perform cross-cloud and data center disaster recovery, as well as deliver low-latency services, among other cases.

A secure and efficient multi-cluster strategy with built-in observability enables platform operators, SREs, and application developers to more easily meet the demands of today's hyper-distributed applications. Throughout this paper, we'll discuss what you need for an ideal multi-cluster network. We'll discuss how Cilium simplifies multi-cluster networking with its eBPF-based cluster mesh and reliable identity-based

networking technology and then deep dive into how it easily solves these common multi-cluster use cases:

- **High Availability and Disaster Recovery** - Maintain redundant workloads across zones or data centers for greater availability and application uptime while reducing single points of failure and operational complexity.
- **Controlled Migration -** Transition applications and infrastructure both legacy and modern cloud native from one cluster group to another with zero downtime while moving from one architecture to another.
- **Edge** - Improve latency and data access by delivering workloads or compute resources in zones that are closer to the end-user or that reside outside of the cloud and closer to the systems that are latency sensitive. At the same time meet data residency requirements.
- **Cloud Bursting** - Meet application demands during peak traffic spikes by dynamically accessing cloud-based services as needed while running workloads from private data centers or private clouds.

What you need for multi-cluster networking

The minimum requirement for a successful multi-cluster network is for Kubernetes pods to easily communicate with one another across clusters. Critical to this is a cross-cluster management layer or control plane. A multi-cluster control plane provides identity-based load balancing, service discovery, service to service network policy enforcement and encryption between services, nodes, pods and clusters. In addition to this, the network must be observable through a single pane of glass by both platform and development teams.

A multi-cluster control plane is responsible for the following:

- **Pod-to-pod communication** - Central to multi-cluster networking is pod IP routing implemented across multiple clusters so that pods from one cluster can communicate with pods in another cluster on the network.
- **Service discovery and service-aware load-balancing** - Services must be optionally discoverable and accessible by all clusters. Not only should services be available across all clusters, but they also need to be automatically rerouted to another if the local cluster services become unavailable.
- **Network policy** - Critical to a successful multi-cluster network is the ability to apply multi-cluster network policy to control access to pods across all clusters and namespaces. Network policies must be straightforward to implement and configurable at both the platform and the application layers.
- **Encryption** - Encryption between all endpoints, clusters, pods and services can be automatically and transparently applied at the platform level without requiring application modifications. This is particularly important between different clusters which may not be within the same Virtual Private Cloud (VPC).

- **Observability** - To reduce operational complexity and simplify troubleshooting, security, logging and tracing, all clusters on the network should be observable through a single pane of glass that can be accessed by all teams in the organization.

Other characteristics of a multi-cluster network

As well as the capabilities listed above, an effective multi-cluster network also needs the following properties:

- **Efficiency -** To achieve native performance, multi-cluster network configuration should depend on standard network principles, and run with minimal latency without additional operational overhead.
- **Resiliency** - Clusters should not impact other clusters when something goes wrong in the network. In the case of one cluster failing, the remaining clusters should take up the extra resources and carry on without disrupting the rest of the network. The network must also not depend on a central data store that can be a central point of failure.
- **Simplicity** - Usability is a key component of any multi-cluster network. Users of the system whether that's Site Reliability Engineers (SREs), platform operators or application developers need to implement and deploy applications without understanding the intricate details of a subnet or routing. In addition to this, the network itself should be simple to debug and troubleshoot if there is an issue.

Why existing solutions aren't enough

As mentioned, traditional Kubernetes networking lacks several features that make creating a multi-cluster network challenging, primarily a lack of cross-cluster pod-to-pod communications as well as a control plane that spans clusters. To satisfy these requirements, additional functionality needs to be added to Kubernetes.

No cross-cluster pod-to-pod communications

Without a simple built-in mechanism for cluster to cluster communications, most organizations end up with a sea of cluster 'islands' that can increase operational overhead, costs and resources.

In a traditional networking model, and because of the dynamic nature of orchestration, pods can only communicate across clusters once the service on one cluster has been 'exposed' as an `ingress` on the network. This leaves application developers with the extra burden and responsibility of making applications available across clusters, instead of having it built-in at the platform level. Exposing internal shared services at the application level can cause security issues, possible duplication of efforts across teams and can make troubleshooting more difficult.

Cross-cluster control plane also not part of the solution

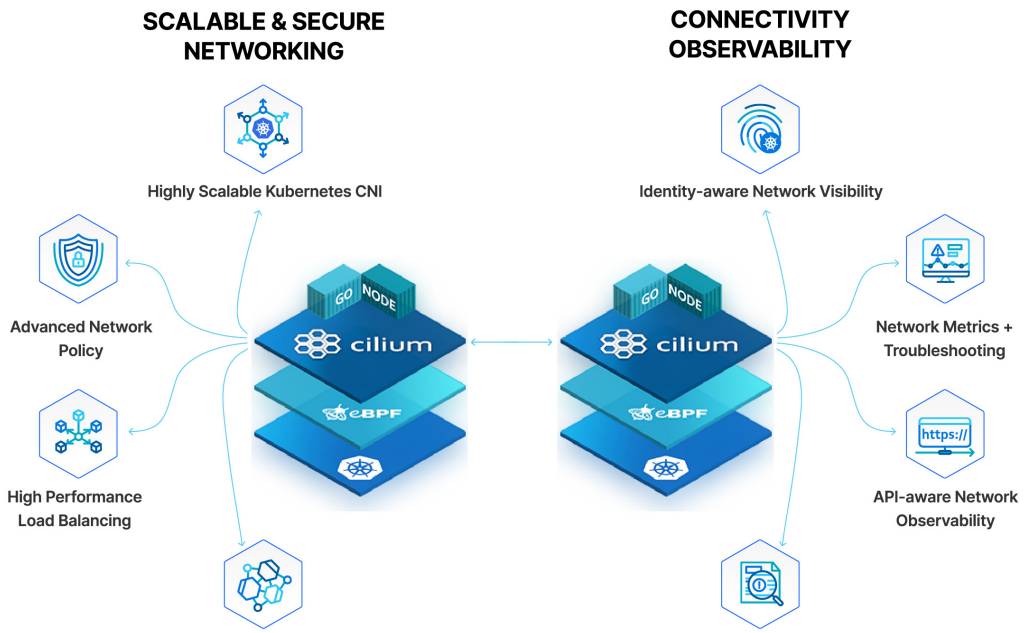
Besides the lack of pod-to-pod cross-cluster communication, the other missing component from out-of-the-box Kubernetes networking is a built-in control plane that can span clusters and implement routing and other services without going through gateways or proxies.

Today an external service mesh can implement all of these services. However an external service mesh increases operational complexity, and since it forwards packets through proxies and gateways, it also introduces overhead and latency that can result in an inefficient network. Also, stringing together multiple proxies can introduce points of failure which can make debugging network issues more complex. Not to mention that troubleshooting security issues is also not straightforward, since it requires your DevSecOps to pour over complex iptable rules and tcpdumps.

What is Cilium?

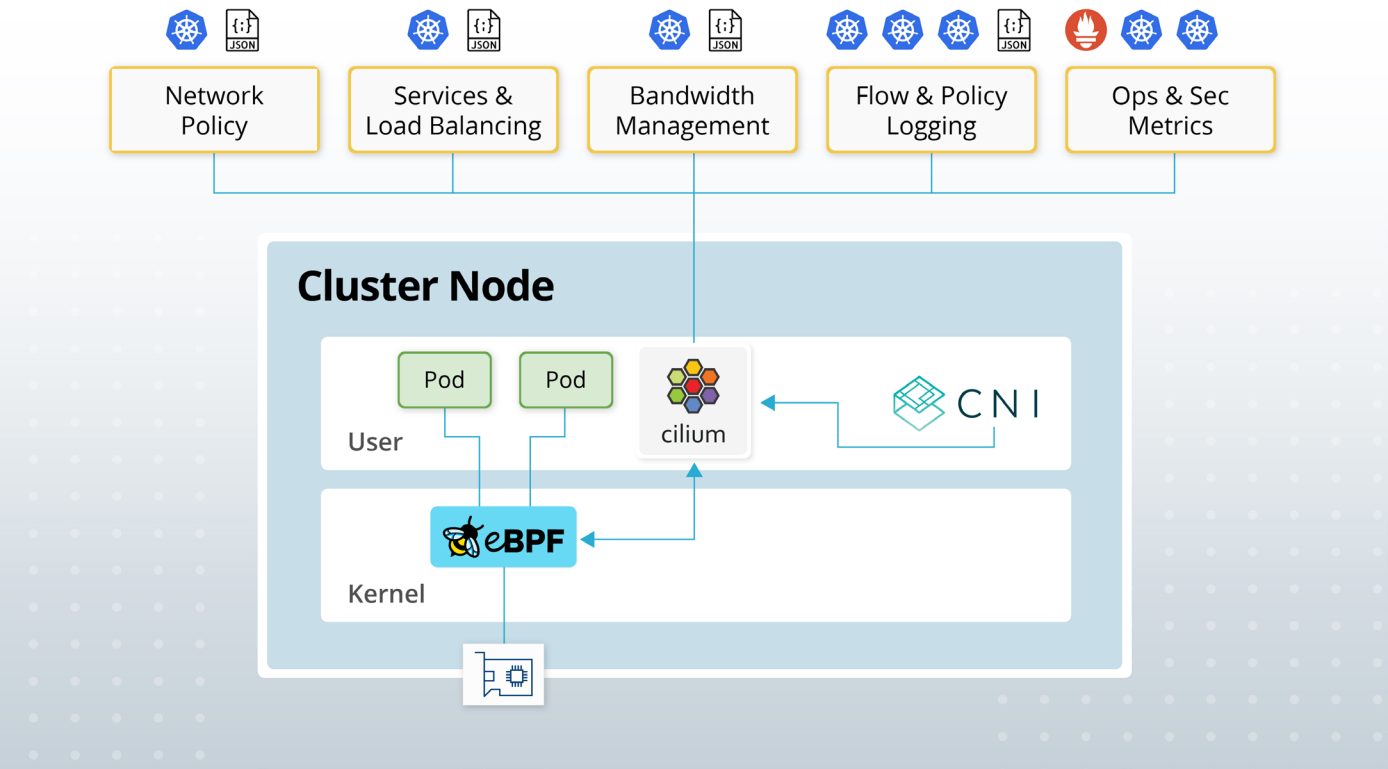
Cilium leverages eBPF, the powerful new Linux kernel technology, to build high-performance, identity-based networking, observability, and security. eBPF introduces programmability at the Linux kernel level to support an efficient implementation of smart networking that is ideal for the dynamic nature of entities in Kubernetes clusters. Since policy and routing are provided at the kernel level, it can be observed and monitored at run-time with tools like Prometheus without requiring changes to code or containers.

Cilium injects as a Container Network Interface (CNI) plugin that runs within large-scale, highly-dynamic Kubernetes clusters. It replaces kube-proxy in Kubernetes for load balancing, and provides an improved alternative to iptables rules which is the default implementation of the kube-proxy.



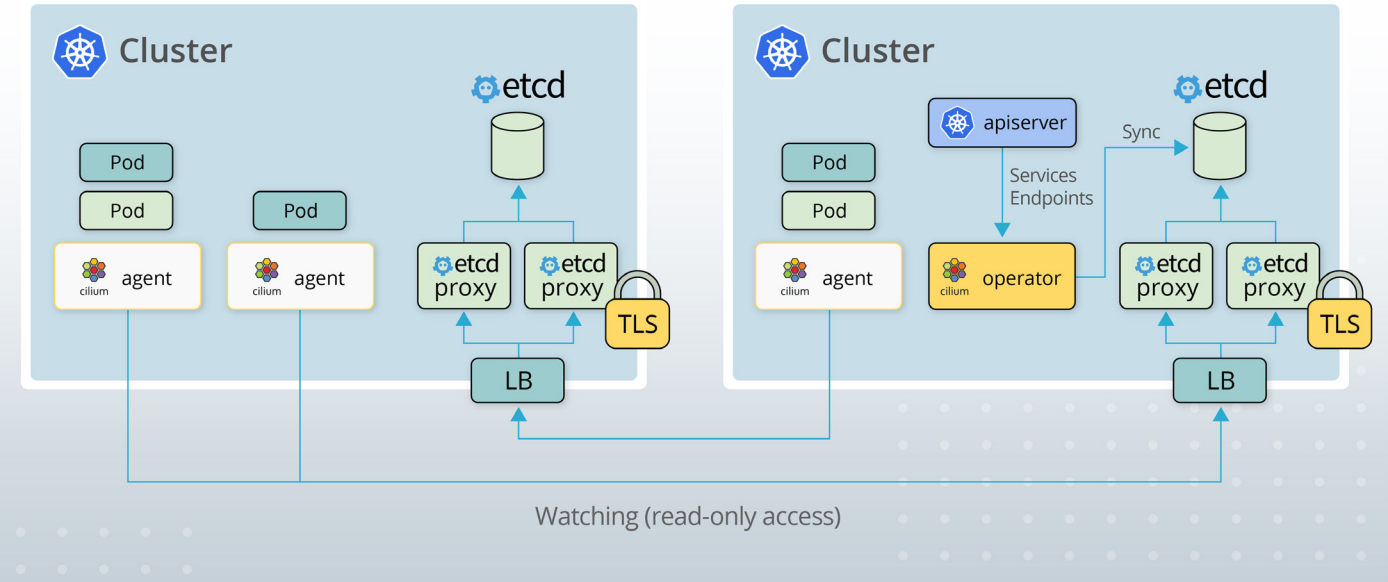
How Cilium secures and connects multiple clusters

Cilium's Cluster Mesh transparently implements encrypted multi-cluster pod to pod routing which means that developers do not have to implement it at the application layer. It provides platform and application teams with a single management plane for load-balancing, observability, and security enforcement between nodes from multiple Kubernetes clusters. Cluster Mesh connectivity is high performance, as data flows directly from one worker node to another without intermediate proxies. It preserves workload identity for cross-cluster traffic, meaning that network observability tooling, such as Cilium's Hubble, as well as network policies, continue to operate just as they do within a single cluster.



Cilium cluster mesh control plane

Cilium's Cluster Mesh uses etcd as an external key-value store to exchange information across multiple Cilium instances. Each cluster in the network operates its own etcd cluster with replication occurring on a read-only basis to ensure that failures in a cluster do not bring down other clusters.



Clusters are connected and routed through VPCs using either the standard API from any cloud provider or an on-premise infrastructure via regular IPSec-based VPN gateways and tunnels or directly through network headers or via hybrid routing mode.

The Cilium control plane is exposed to the VPCs using the service annotation `Internal` for load balancer type. TLS authenticates the client and server with the certificates and keys that are managed as Kubernetes secrets. Service load-balancing is set to either `internal` or `external` and available through standard Kubernetes annotations.

Cilium Cluster Mesh vs Sidecar-based Service Meshes

Cilium Cluster Mesh and service meshes like Istio have similar functionality such as service discovery, load balancing, security and traffic management and routing. The main difference between the two is in how they route traffic within the cluster. A service mesh like Istio is deployed to the cluster in a sidecar configuration and runs all the traffic through gateways and proxies which can add significant network latency to your application. Cilium Cluster Mesh, on the other hand, routes pod IPs via tunneling, directly through network headers, or via hybrid routing mode enabling direct pod to pod communication. The table below also highlights some other key differences between Cilium and sidecar-based service mesh:

Functionality	Cilium cluster mesh	Sidecar-based service mesh
Management complexity	Low Cilium is easy to install and operate with no code to instrument.	High Steep learning curve and difficult to implement and operate.
Service proxy	No IP forwarding is without proxies or gateways.	Yes But adds overhead and latency to the network. Multiple ingress and egress proxies and gateways.
Identity-based networking	Yes Fundamental component of design and does not route through a proxy.	No
Encryption implemented at the platform layer	Yes Does not require application changes.	No Requires changes to the application to offload encryption to the sidecar.
High Availability control plane	Yes The network is fully observable without additional code changes.	Yes Requires configuring replicated control planes across all clusters and also makes use of horizontal pod auto scalers for HA.
Network layer metrics	Yes The network is fully observable without additional code changes.	No L7 application layer only.
Built-in observability dashboard	Yes Hubble is built on top of Cilium and eBPF to enable deep visibility into the communication and behavior of services as well as the networking infrastructure.	No L7 application layer only.

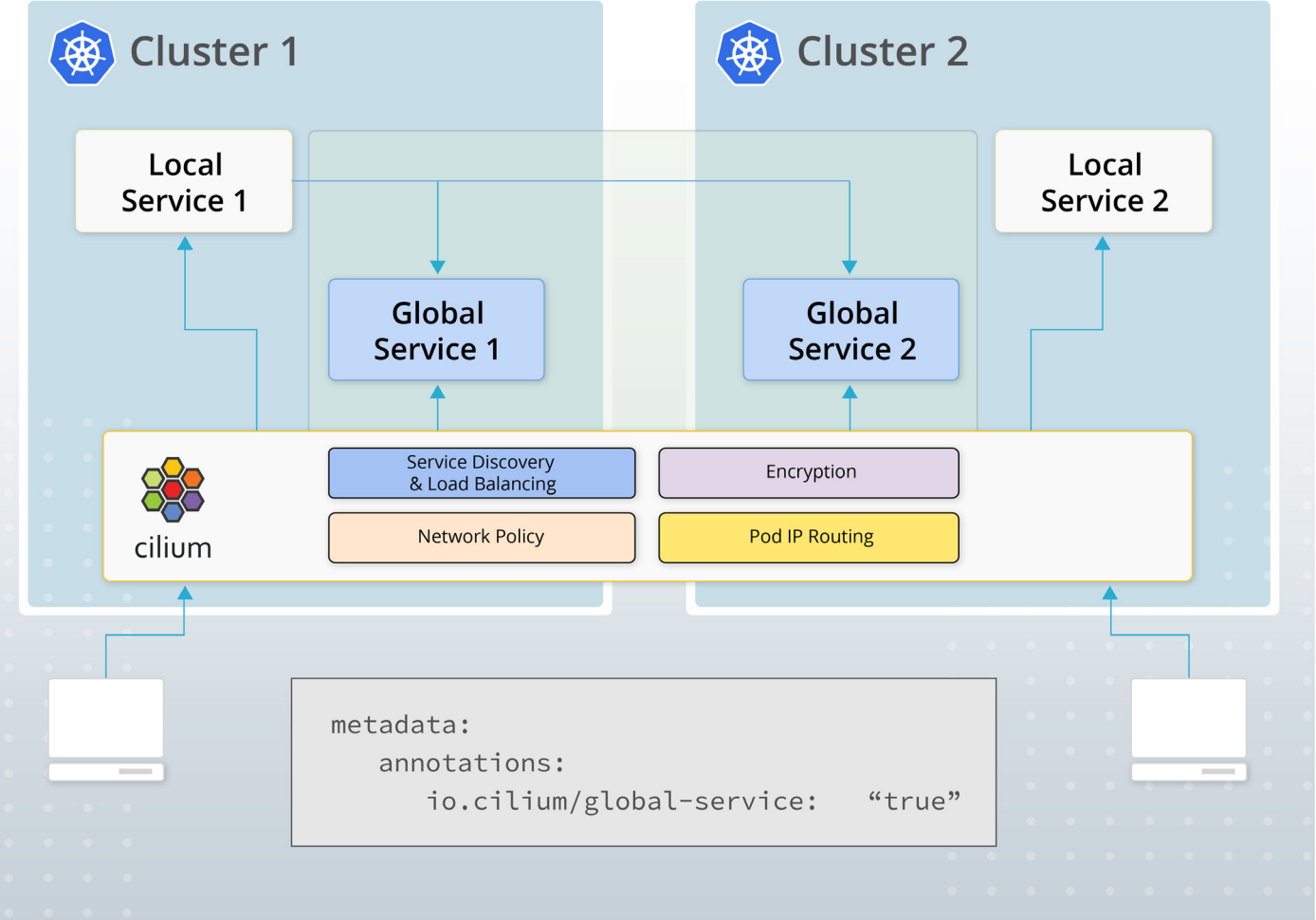
Because of the differences in stack routing and monitoring, Cilium Cluster Mesh and sidecar-based service meshes are complementary in the following ways:

1. A sidecar-based service mesh together with Cilium can provide deep observability at both the application and network layer.
2. Cilium can run alongside a sidecar-based service mesh to enforce HTTP L7 policies for the service mesh's sidecar proxies.
3. You can use Cilium's multi-cluster Pod IP routing layer to fulfill one of the requirements of a service mesh like Istio where all pod CIDRs in every cluster must be routable to each other.
4. You may also want to run a mix of global services from both the sidecar-based service mesh and Cilium side by side. In this case, Cilium's global services (see below) are reachable from the sidecar-based service mesh's managed services as they get discovered by DNS just like any other regular service.

Service discovery with the Global Service annotation

Cilium routes connections between pods on multiple clusters if `global services` annotation is added to a Kubernetes service deployment. Implementing this annotation immediately solves the pod to pod communication problem, and eliminates the need for internal load balancers and DNS records with cross-cluster identity-based networking.

Once a service with an identical name and namespace is added for each cluster that includes the 'global' annotation, it is securely made available to multiple clusters. Cilium connects clusters without proxies or gateways by representing each global workload as if the workload runs as a pod.



Because Cilium leverages the underlying capabilities of Kubernetes, there is no need to change any of the fundamental architectures of Kubernetes. For example, services that are available to other clusters are kept healthy using Kubernetes' liveness and readiness probes where services are added and removed as necessary when pods scale up and down or they become unhealthy.

Network policy across multiple clusters

Once the services are made available to other clusters, you can create and implement network policies for which services can communicate across clusters. Because network policies are implemented at the kernel level, all traffic can be observed and monitored with Prometheus and visualized with Hubble without requiring developers to instrument their code. The key to seamless pod to pod networking is Cilium's implementation of identity-based networking that identifies which pods and services can communicate using labels instead of IP addresses to enforce access controls. Policies are also DNS and API-aware, and can be visualized and edited in the Network Policy Editor, see "Network Policy" for more information.

Cilium and Multi-Cluster use cases

Now that you have an idea of how Cilium works and what its advantages are over traditional Kubernetes networking options, let's examine some common use cases and discuss how Cilium solves them.

High Availability and Disaster Recovery

High Availability and Disaster Recovery are two of the most common use cases for multi-cluster networks. With Cilum and Cilium Mesh both of these scenarios can be easily achieved with little effort.

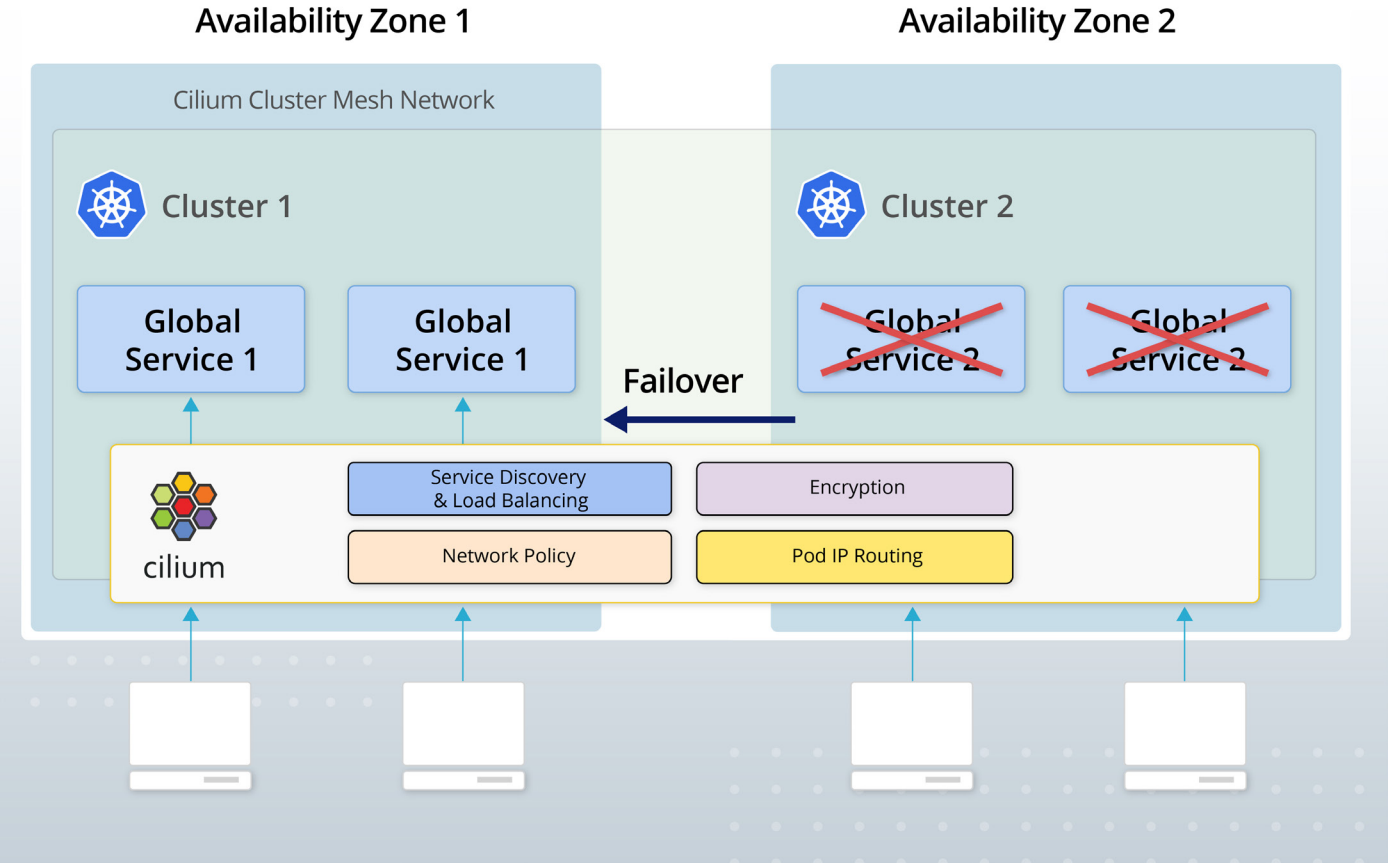
High Availability
Organizations need to replicate the same service across multiple availability zones within a single cloud provider or sometimes across multiple cloud providers. Cilium's cluster mesh built-in capabilities like load balancing, identity-based service discovery, increase resiliency and reliability against potential service failure and disruptions.

Because a Cilium multi-cluster network views multiple clusters as one, if a service or pod in the network goes down in one zone, it can leverage Kubernetes self-healing capabilities where pods will fail over and start-up in another cluster, resulting in zero downtime for your application.

Disaster Recovery

Kubernetes pods and services can self-heal at the application level by spinning up new and healthy pods. But what if something more serious occurs and one of your nodes goes down? Kubernetes doesn't self-heal nodes or other infrastructure and you will need another plan for such a scenario.

There are many topologies for replicating and backing up your data center. However, Cilium's ability to seamlessly connect clusters between regions and datacenters vastly simplifies recovery from disaster in the case of a partial or total meltdown.



Controlled Migration

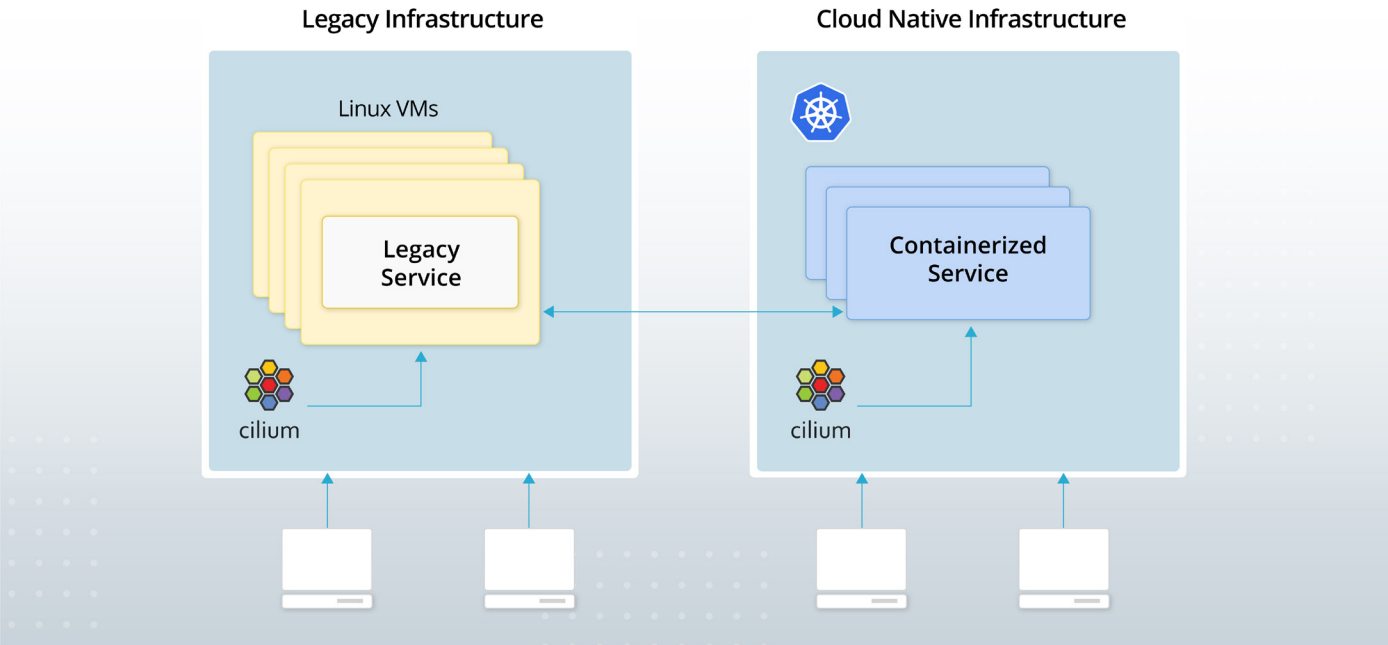
Whether you're just embarking on a Kubernetes journey and beginning to move your mission critical applications to a more modern cloud native stack or you need to upgrade clusters, a fast and seamless migration between clusters, environments both legacy and new is essential.

Migrating applications from Legacy Infrastructure to Kubernetes
Starting from scratch with a greenfield application that is 100% cloud native is often not possible for most organizations. Instead many need to incorporate their existing infrastructure while re-architecting their monolithic applications to microservices for the cloud native world.

Because Cilium can recognize and transparently connect to non-containerized Linux VMs and bare metal Linux workloads in addition to containerized services on Kubernetes, it bridges the two infrastructures. This not only saves valuable time and resources for when you're ready to go cloud native, but it also reduces costs by leveraging existing legacy infrastructure while developing and running your applications.

With all of your infrastructure on the same network, observability and monitoring is also much simpler since it can be viewed from within a single pane of glass rather than switching between multiple dashboards.

In the diagram below, Cilium runs on both legacy Linux VMs and Kubernetes connecting them so that they can be accessed by containerized services in Kubernetes and vice versa. Because of Cilium's identity-based networking, non-containerized services are load balanced by Kubernetes worker nodes and recognized as another service by the cloud native infrastructure.

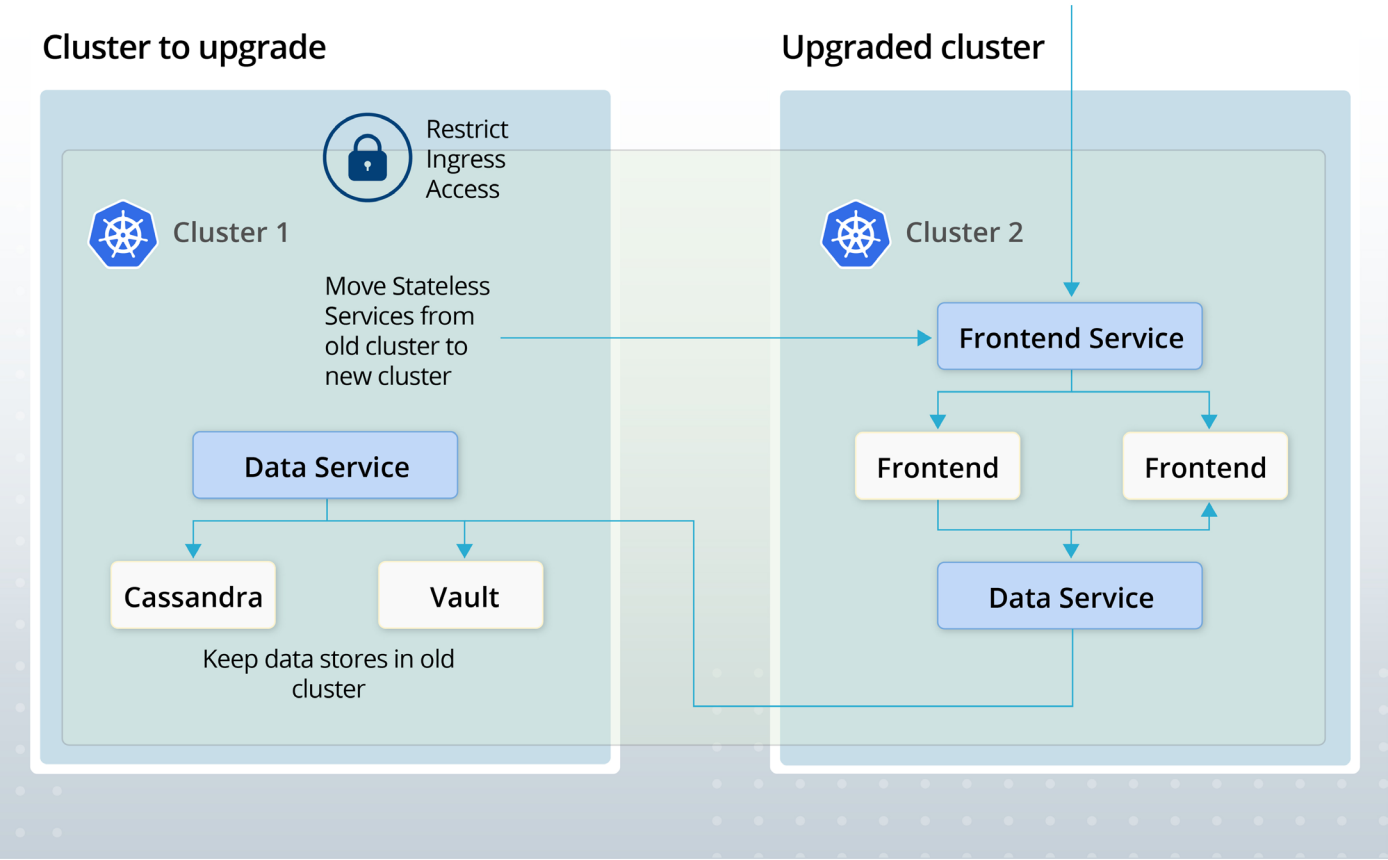


Migration and critical cluster upgrades

In this scenario, the platform team recently received notice of a mission critical CVE (Common Vulnerabilities and Exposures) requiring that all clusters be patched immediately. The data science team is working on a set of clusters that need the upgrade, but they are in the middle of a project and can't afford any downtime or worse having to waste cycles spinning up and configuring a whole new environment.

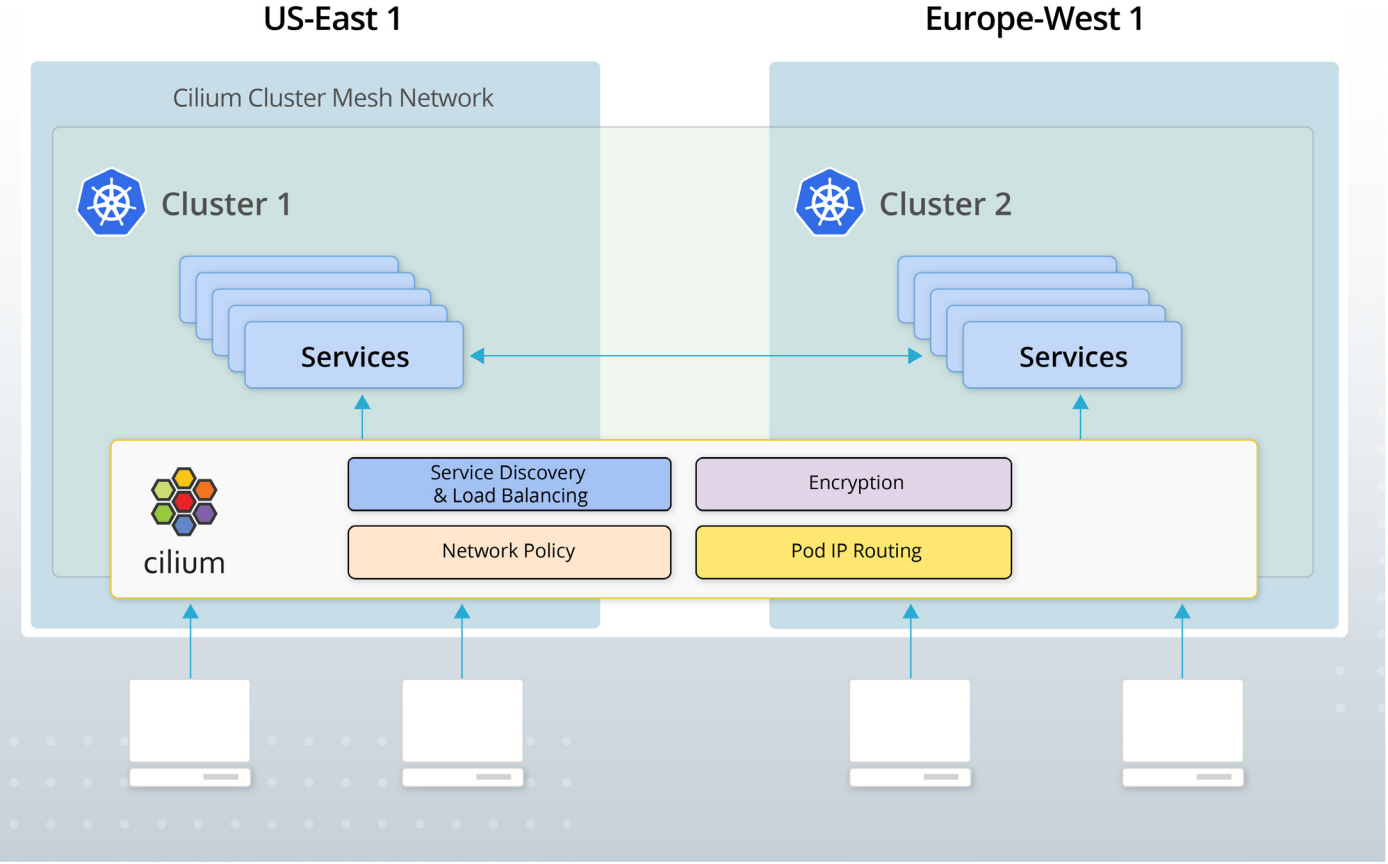
The ability to conduct a controlled migration is greatly assisted by Cilium's multi-cluster networking and single control plane which makes the process transparent, simple and secure. The diagram below is an example showing how your team can continue to work while a cluster is being patched.

Since stateless applications are much easier to recreate, they can be shut down and easily redeployed onto a newly upgraded cluster. With the help of Cilium, and its global services annotation, newly deployed stateless applications pick up where they left off and access critical backend datastores like shared secrets or Kafka datasets on the old cluster while the cluster API is being upgraded and fixed.



Edge

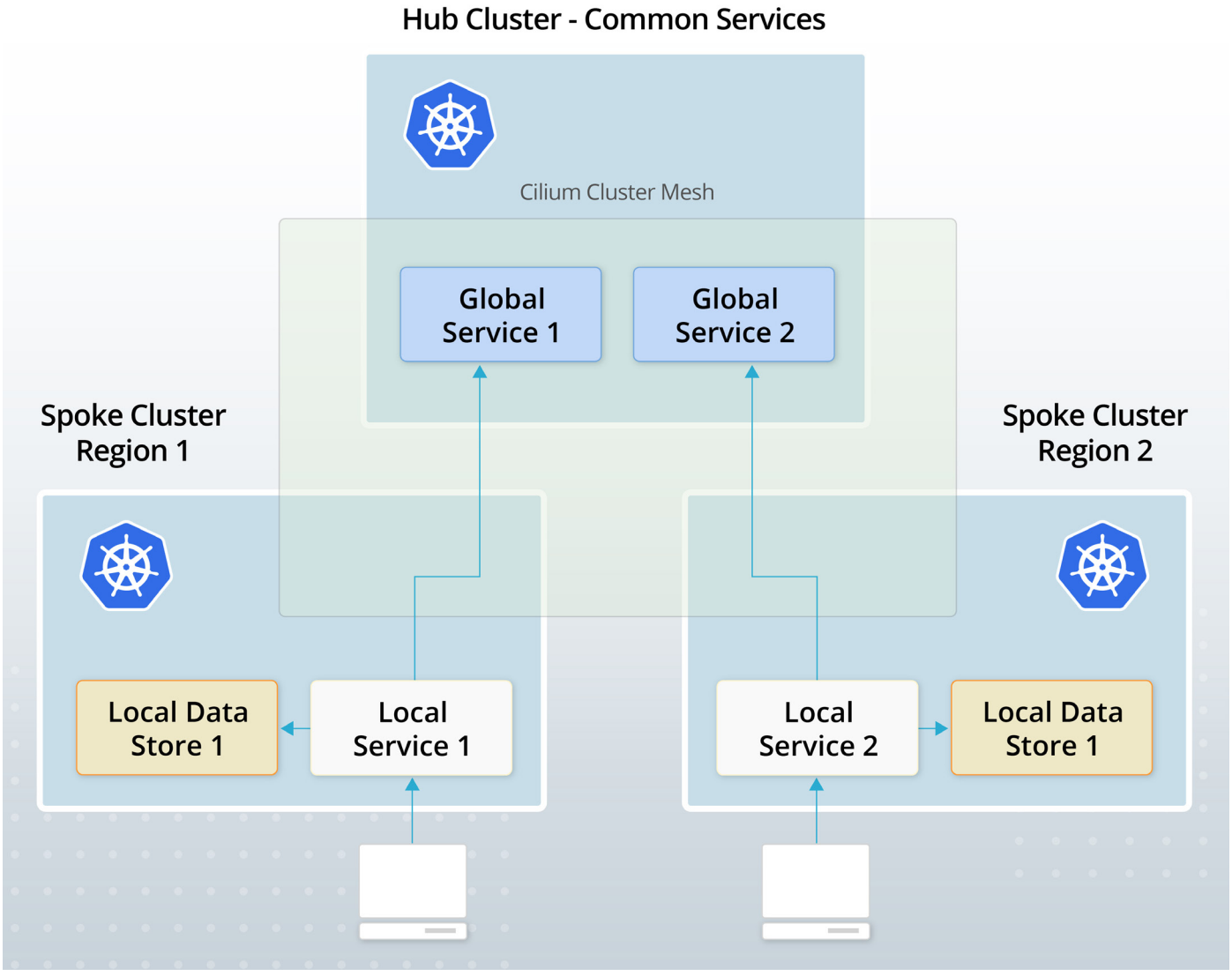
In edge scenarios, the multi-cluster network spans both regions and availability zones. Traffic is shifted closer to the user's location to a more efficient zone in order to reduce latency or bring data sources closer to a systems' location. Like the High Availability scenario, services running in a Cilium network are easily made available through identity-based addressing, which means that applications can span zones and data centers with nothing but policy applied to the load balancer.



Hub and spoke topology

A typical hub and spoke topology powered by Cilium can also meet the demands of both edge workloads and data residency requirements. In this configuration, services that depend on zero to no latency are made available to users in their respective regions (the spokes); whereas, centralized services are made available to all services from the hub. In this configuration, it's also possible to meet data sovereignty laws and governance regulations that are increasingly required by nations around the world.

Cilium Cluster Mesh provides cloud architects and platform operators with the flexibility to centralize commonly used services like shared secrets, DNS and other global services from a hub. And with the addition of policy, access to latency-dependent services and at edge deployments or access for users in specific particular regions or zones can be easily met. Also this configuration simplifies securing regional data to meet data sovereignty rules and regulations.



About Isovalent

Isovalent builds software to connect, observe and secure cloud native workloads via its Cilium open source project and Cilium Enterprise product.

Cilium Open Source

Cilium Open Source provides eBPF-based networking, observability, and security with optimal scale and performance for platform teams operating Kubernetes environments across cloud and on-prem infrastructure.

Cilium Enterprise

Cilium Enterprise addresses the complex workflows related to security automation, forensics, compliance, role-based access control, and integration with legacy infrastructure that arise as platform teams engage with application and security teams within an enterprise organization.

US HEADQUARTERS

444 Castro St. STE 730
Mountain View, CA 94041 USA

SWISS HEADQUARTERS

Industriestrasse 25 8604
Volketswil Zurich, Switzerland

ISOVALENT | cilium