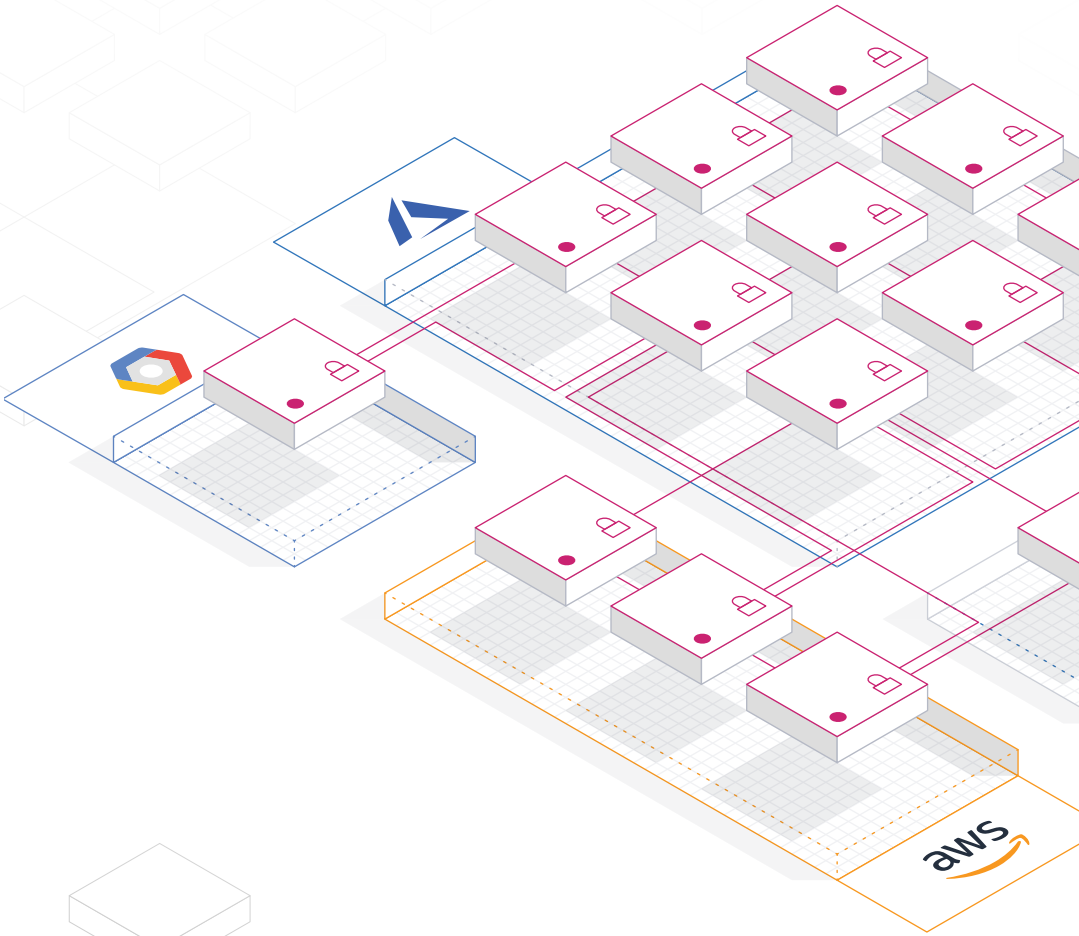
HashiCorp Consul

Service Mesh and Microservices Networking



WHITEPAPER

HashiCorp

Service mesh and microservice networking

As organizations adopt cloud infrastructure, there is a concurrent change in application architectures towards microservices. These two trends are both part of a larger goal—to increase developer efficiency and enable faster delivery of new features and capabilities. There are many success stories around these practices, such as Netflix's move to a cloud native infrastructure, but people often forget that there are new networking and operational challenges that come with this approach. This white paper covers the traditional, static approach to networking and why we need a different, dynamic approach with cloud-based microservices.

Contents

Traditional networking with monolithic applications

When discussing traditional networking we also need the context of traditional application architecture. Prior to microservices, the main pattern was the monolithic application. This is a single large application that consists of multiple discrete sub-systems or capabilities. As an example, a desktop banking application that contains a login portal, balance viewing, transfers, and foreign exchange capabilities compiled and deployed as a single application. Due to the interdependence of development teams, monolithic applications are harder to update, and many organizations could only manage to update these applications a few times annually.

This application architecture was supported by a networking architecture that reflected that slow update rate and limited number of unique applications. There are several key challenges the network had to help solve:

- **Routing:** Requests coming from the Internet need to reach front-end servers, which need to reach back-end servers. This is the north-south traffic flow.

- **Segmentation:** To secure connected systems, networks need to segment traffic between logical zones.

To solve the request routing problem, traditional networks used load balancers. These were either hardware appliances or software load balancers. Traffic would flow north-south using multiple tiers of load balancers—one tier for each application or back-end system. Load balancers performed basic health checks to mitigate failures and balance traffic between multiple machines, enabling horizontal scalability.

Segmentation was solved by using a combination of firewalls and virtual networks. Firewalls were typically hardware appliances that were placed in critical network junctions to constrain traffic. Virtual networks were constructed using Virtual LAN (VLAN) to provide Quality of Service (QoS) and bulkheads between different groups. This approach allowed a network perimeter and IP-based access to be enforced between machines.

Cloud & microservice challenges

With the shift to microservices applications, a single large monolith may be decomposed into dozens of individual services. This decouples the development teams so that each service can be developed and deployed independently. By reducing the coordination required for a release, development velocity is improved. But this introduces the new challenge of having many more applications being updated more frequently than before.

Microservices applications also change the traffic flow. With monolithic applications, traffic was from primarily north-south. Microservice applications communicate with each other to compose and re-use functionality, adding more east-west traffic. This service-to-service traffic still needs to be routed, and typically load balancers are paired with each service. This results in a proliferation of load balancers, increasing cost and request latency.

As the infrastructure becomes larger and more dynamic, this puts pressure on networking and security teams to keep pace with more changes to load balancers and firewall rules. These teams often cannot keep up with the increasing demand, and change tickets can take weeks or months to complete, restricting the agility of application teams. Cloud infrastructure abstracts the underlying data center and promotes an on-demand consumption model. This abstraction removes our control of the underlying network, prevents the use of hardware devices, and restricts us to a limited set of capabilities. This means traditional hardware load balancers and firewalls cannot be used, and we cannot leverage segmentation techniques such as VLAN. The network topology is defined by API calls, not a physical layout, making it much harder to preserve the network perimeter with a limited number of ingress and egress points.

To simulate the controls of the traditional network, many organizations have very complex cloud networks. Virtual networks are recreated using hundreds of accounts or VPCs, to substitute for VLANs. Traffic flows are enforced via complex peering arrangements or VPN topologies. This creates a management challenge for operations teams who need to set up and administer the network and applications being deployed into the appropriate networks.

All these challenges point to a mismatch between the application architecture, cloud infrastructure, and traditional approaches to networking.

What is service mesh?

A service mesh is a software-driven approach to routing and segmentation. The goal is to solve the networking and security challenges of operating microservices and cloud infrastructure. Service mesh solutions bring additional benefits such as failure handling, retries, and network observability.

When we consider the traditional networking approach, it solved both routing and segmentation using hardware that was manually managed. Instead, we can solve these problems in software with automation.

A service mesh has two key components:

- **Control Plane:** The control plane holds the state of the system and plays a coordination role. It provides a centralized registry of where services are running, and the policies that restrict traffic. It must scale to handle tens of thousands of service instances, and efficiently update the data plane in real time.

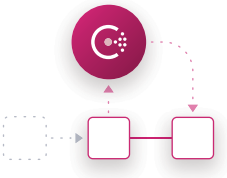
- **Data Plane:** The data plane is distributed and responsible for transmission of data between different services. It must be high-performance and integrate with the control plane.

The distributed nature of a service mesh allows us to push routing and segmentation to the edges of the network, rather than controlling the topology and imposing them through middleware. The network topology can be dramatically simplified, since the network is only responsible for connecting all the endpoints. We can also remove firewalls and load balancers from east-west traffic, since the data plane provides routing between services and enforces network policies.

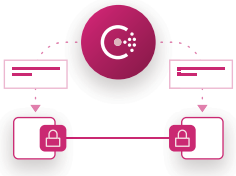
A service mesh is an abstract concept that solves multiple service networking challenges in an integrated way. To make this concrete, we introduce HashiCorp Consul.

Introduction to HashiCorp Consul

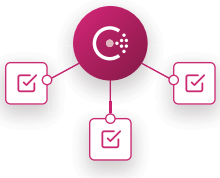
HashiCorp Consul is a service mesh that provides a solution to service discovery, segmentation, and configuration:



- **Service discovery:** Consul provides a centralized catalog of all the nodes and services, along with their health status. The catalog is automatically populated and kept up-to-date. An API allows the catalog to be used for automation and dynamic routing between services.

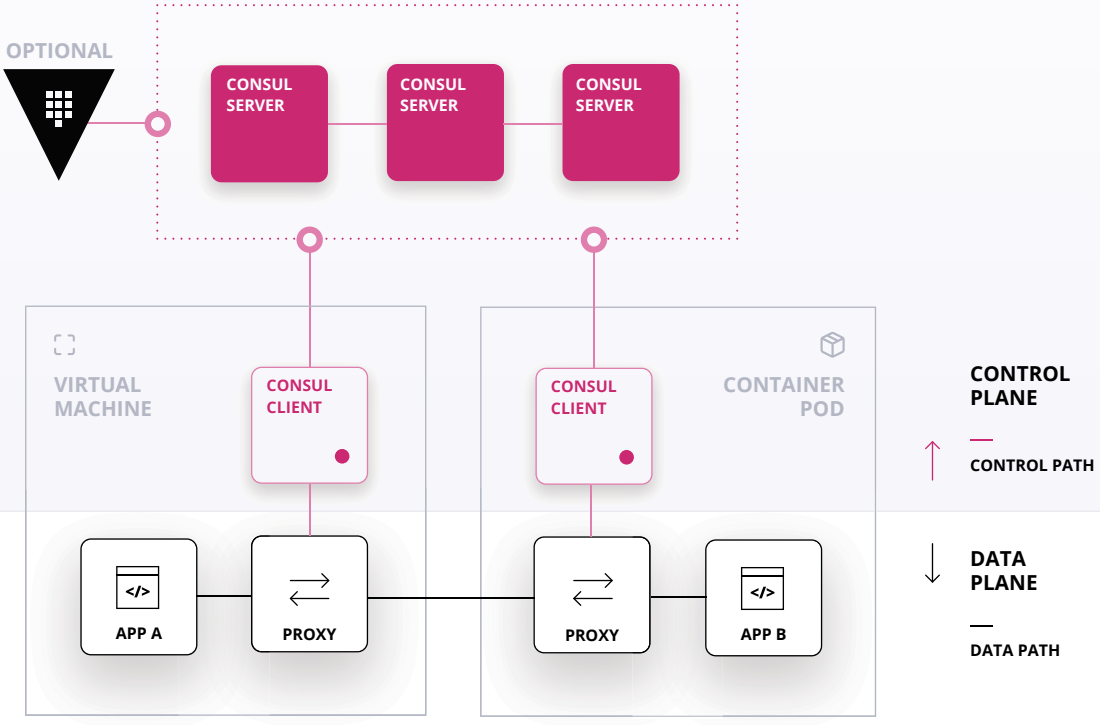


- **Service segmentation:** Consul defines rules governing which services can communicate. Instead of IP-based rules, Consul uses a logical service as the source and destination. This enables dynamic infrastructure: services can scale up and down without waiting for static firewall rules. Service identity is provided with TLS certificates that are generated and managed by Consul's built-in certificate authority (CA) or other CA providers, such as HashiCorp Vault. This provides cryptographic identity and encrypts all traffic over the network.



- **Service configuration:** Consul provides a hierarchical key/value store. The K/V store can be used to store application configuration such as runtime configurations, maintenance modes, and feature flags. This allows configurations to be centrally managed and updated in real time.

To provide these features, Consul has a client-server architecture and is the “control plane” for the service mesh. Multiple servers are deployed for high availability, and clients run on every host. Clients integrate with the proxies that provide the “data plane” for the service mesh. The clients are critical for scalability, as they cache policies and configurations and distribute the work of policy enforcement and health checking.



The diagram shows an architectural overview. The centralized servers hold the service catalog and access policies, which are efficiently transferred to the distributed clients in real time. The clients manage certificates for the applications and configure the local proxies. Applications communicate with the local proxies, that communicate directly with the destination services. The data plane is kept fully distributed, using end-to-end TLS and pushing enforcement to the edges for scalability and availability.

Consul provides a number of advantages for microservices and cloud infrastructure. There is no need for firewalls or load balancers in the service-to-service path. Instead,

the proxies manage dynamic routing and enforcement. This avoids any manual updates and allows services to be deployed and receive traffic in real time. The network needs to ensure traffic flows between services, but authorization is done on the edge, avoiding the need for complex network topologies simulating network segments. Applications can be frequently deployed and dynamically scaled, and Consul allows for end-to-end automation of the required service networking.

Using local proxies enables simple application integration, since they are transparent to the application. Instead of communicating with a static load balancer, applications communicate with the local proxy, which dynamically routes traffic, imposes mutual TLS to provide service identity and encryption of traffic over the wire, and enforces network policies to restrict traffic. For low-latency or throughput-sensitive applications, native integration is possible to avoid the overhead of network proxies. Natively integrated applications use a Consul SDK to fetch TLS certificates and verify that incoming TLS connections are authenticated against a trusted CA and authorized by network policies.

Competitive differentiation

The adoption of microservices architectures and cloud infrastructure is forcing new approaches to networking. There are many different vendors and tools, each attempting to solve the problem in different ways. Consul makes no assumptions about the underlying network and uses a pure software approach with a focus on simplicity and broad compatibility.

Scalable Distributed Architecture

Consul was designed to scale to data centers with tens of thousands of machines and support a multi-data center topology with hundreds of sites. This is done by ensuring a completely distributed data path and minimizing churn in the control plane. Data flows directly between proxies or applications without interacting with the central control plane. This avoids creating a scalability bottleneck.

The network policies managed by the control plane use a source and destination service based on logical identity. This unit is scale independent, unlike an IP or MAC address. As an example, suppose a rule authorizes "web server can reach the database." With 50 web servers and 5 databases, that would mean 250 individual rules for a firewall, but for Consul, it's just one. As services scale up and down, the Consul rule remains static, unlike firewall rules which need to be updated.

Using the service as the logical management unit significantly reduces churn on the control plane and allows for very large clusters.

Multi-Data Center and Multi-Cloud

Consul supports multi-data center and multi-cloud topologies. The architecture is client/server, with 3-5 servers and potentially thousands of clients per data center. The servers in each data center can be federated together to form a larger cluster. These data centers can be in either public or private clouds, allowing for a multi-cloud topology.

Consul models each data center as a failure domain, meaning the system is designed to allow a data center to fail without impacting other data centers. This ensures high availability of the entire cluster and allows us to scale globally by spanning additional data centers.

Multi-Platform Support

The network has long been the “common denominator” that allows various generations of technology to interconnect. Containers can interact with mainframes because they speak common networking protocols. Consul is designed to operate on bare metal, virtualized, and containerized environments, including all major operating systems. For environments that cannot run a Consul agent, either because they are legacy or black boxes, additional tooling is provided to integrate them. This allows containerized applications to discover and securely communicate with any legacy system.

Universal Protocol Compatibility

Network policies in Consul use a logical source and destination service. This allows Consul to enforce policy at Level 4 traffic, such as TCP. The advantage of focusing on L4 is universal protocol compatibility because application-level protocols, such as HTTP, do not need to be parsed. This allows both greenfield and legacy applications to be easily integrated and secured without needing to re-tool the applications to use specific protocols or extend Consul to be protocol aware.

Network Observability vs APM

Consul aims to provide a control plane for service networking, managing routing, and segmentation. For observability, Consul allows telemetry to be exported to best of breed monitoring and application performance management (APM) tools. Consul will provide basic integrated metrics such as the number of connections, bandwidth, and latency between services. Integration with external monitoring and observability tools allows existing solutions to be used, which provide fine-grained data retention, querying, alerting, and visualization.


HashiCorp