



Service Mesh with Consul

August 2022

Copyright © 2021 HashiCorp

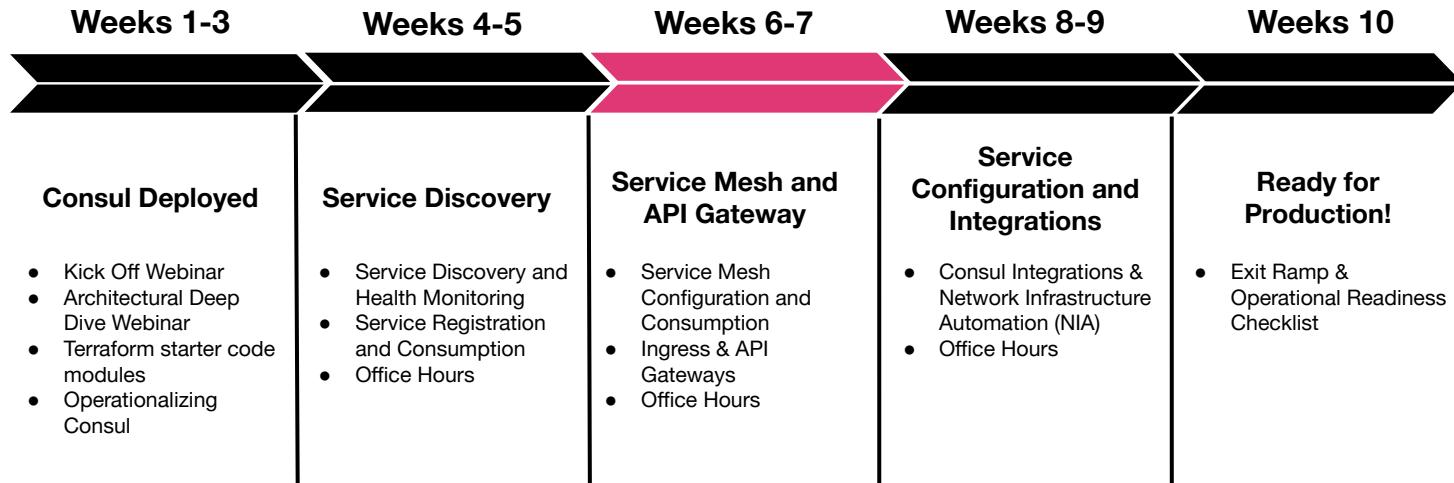




Agenda

1. What is a Service Mesh?
2. Service Mesh Components
3. Traffic Control
4. Service Resiliency
5. Observability
6. Gateways

Consul Enterprise Path to Production



— 01

What is a Service Mesh?



Method Call became Network Call

Network is reliable,
homogeneous and
secure

- Latency is zero
and bandwidth
is infinite



Countermeasures

- Load Balancing
- Traffic Management
- Health Checks
- Service Discovery
- Encryption

Re-Implementation is
expensive

- Global rules and configurations
are hard to enforce
- Originally Hystrix and Finagle, but
JVM-only



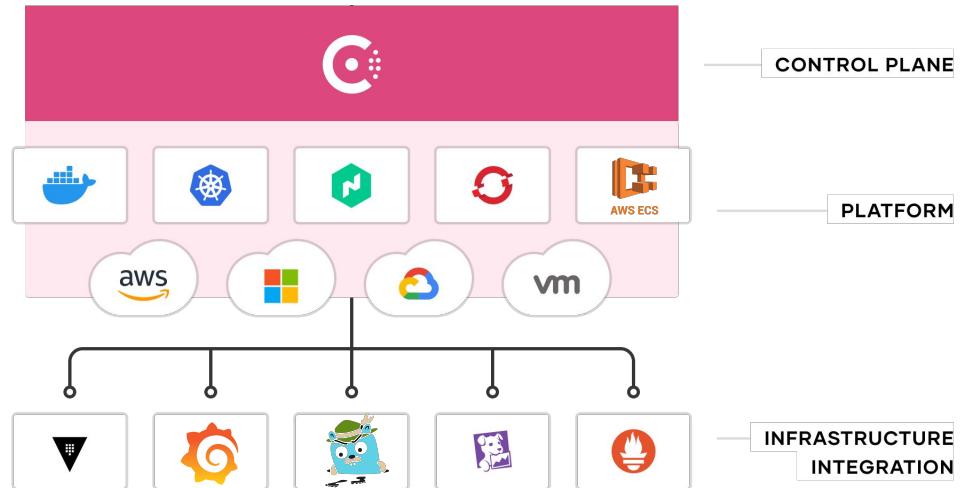
What is a Service Mesh?

- A dedicated infrastructure layer for facilitating service-to-service communication
- Features Include:
 - Consistent Discovery
 - Security
 - Tracing, Monitoring and Failure Handling
 - Does not require a shared asset such as an API gateway
- Consul Connect is used interchangeably with Consul Service Mesh



Multi-Platform Service Mesh

- Automatic load balancing
- Fine-grained control of traffic behavior with routing rules, retries and failovers
- Metrics, logs, and traces for all traffic within a cluster
- Identity-based authentication and authorization for service-to-service communication



— 02

Components



How Does It Work?



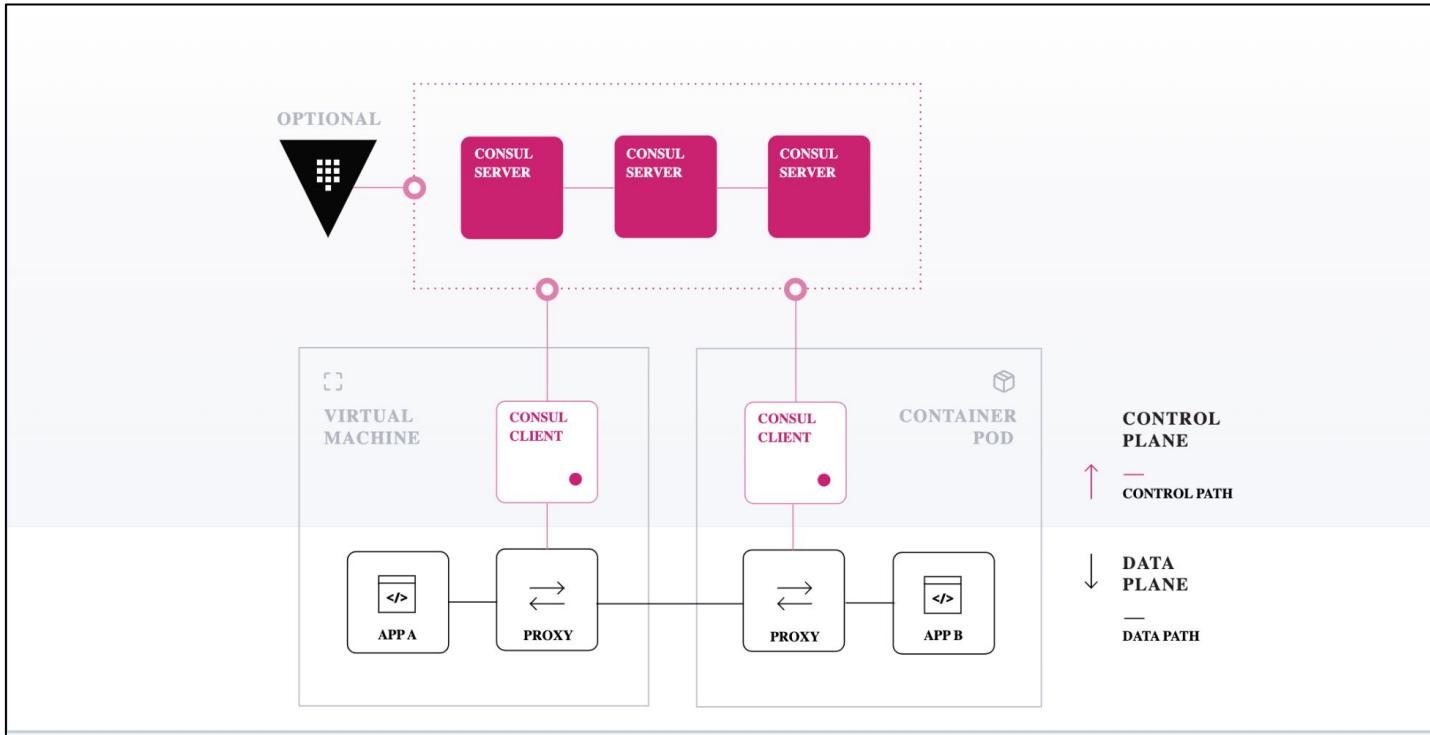
Control Plane

- Manages and configures the proxies to route traffic.
- Configures policies and collect telemetry

Data Plane

- Composed of a set of intelligent proxies (Envoy) deployed as sidecars
- Control all network communication between microservices

Control Plane



— 02 A

Component 1 - Control Plane

Consul Service Mesh Components



1. Consul internal certificate authority -

Signs services with certificates for mTLS communication

2. Service sidecar proxies

Control and encrypt service traffic

3. Consul intentions

Authorize communications between sidecar proxy enabled (Service Mesh) services



Certificate Authority (CA)

- Provides services with identities
- Certificates are SPIFFE compliant
- Used to establish mutual TLS
- Certificate authority options
 - Built-in CA (not encrypted on disk)
 - Vault
 - AWS ACM Private CA

— 02 B

Component 2 - Sidecar Proxy

Sidecar Proxies

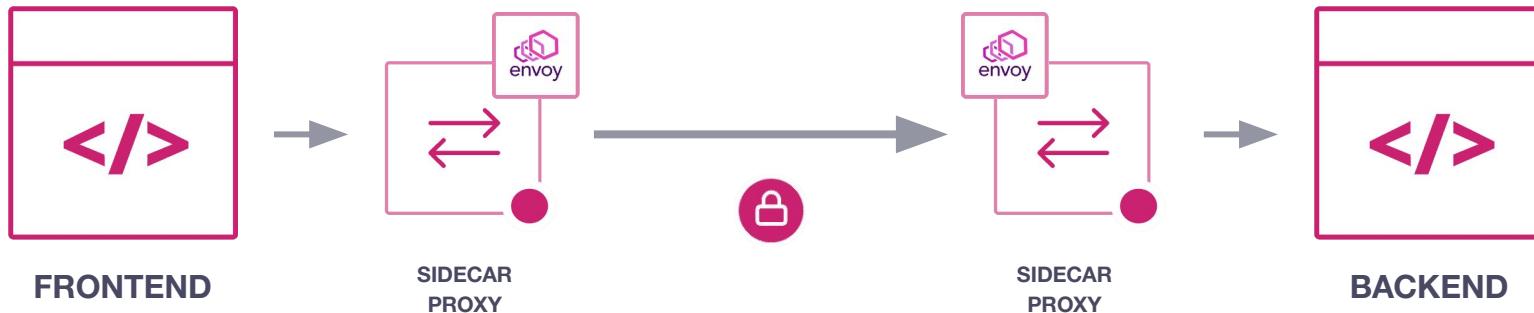


- Typically implemented with reverse-proxy processes deployed alongside each service process
- Communicate with service registries, identity providers, log aggregators, etc
- Automatically establish TLS connections
- Secure inbound and outbound connections
- Applications are unaware (connect via localhost)
- Supported Proxies
 - Envoy preferred - (enables Layer 7 policies)
 - Built-in L4 only for dev and testing, use Envoy in Prod deployments
 - Extensible for 3rd party proxies

Service Mesh Overview



Service Mesh Overview





Envoy - Preferred Proxy

- Service proxy written in C++
- Controls all inbound and outbound traffic
- Deployed as sidecar
- Consul leverages many built-in features, for example:
 - Load balancing
 - TLS termination
 - Health checks
 - Traffic Split
 - Metrics

— 02 C

Component 3 - Intentions



Consul Intentions



Intentions **control service-to-service communication**

- Allow or deny service traffic
- Logical service name (not IP)
- Scales independent of instances
- Consistency ensured with Raft
- Manage with web UI, CLI, API and Kubernetes CRD
- Multi-datacenter support

Intentions - UI

The screenshot shows the Intentions UI interface. The left sidebar contains navigation links: Admin Partition (selected), Namespace (selected), Overview, Services, Nodes, Key/Value, Intentions (selected), ACCESS CONTROLS (Tokens, Policies, Roles), and Auth Methods. The main content area displays the 'Intentions' page with 7 total items. The table has columns: Source, Destination, Permissions, and Actions. The first row shows an intention from 'ingress-gateway' to 'bird-frontend' with 2 permissions, managed by CRD. Subsequent rows show intentions from 'productpage' to 'details', 'ratings', 'reviews', and 'backend', all managed by CRD. The last row shows an intention from 'frontend' to 'backend' with a Deny permission.

Source	Destination	Permissions	Actions
ingress-gateway ∅ default / ⚡ default	bird-frontend ∅ default / ⚡ default	2 Permissions Managed by CRD	...
bird-frontend ∅ default / ⚡ default	bird-backend ∅ default / ⚡ default	Allow Managed by CRD	...
productpage ∅ default / ⚡ default	details ∅ default / ⚡ default	Allow Managed by CRD	...
productpage ∅ default / ⚡ default	ratings ∅ default / ⚡ default	Allow Managed by CRD	...
productpage ∅ default / ⚡ default	reviews ∅ default / ⚡ default	Allow Managed by CRD	...
reviews ∅ default / ⚡ default	ratings ∅ default / ⚡ default	Allow Managed by CRD	...
frontend ∅ default / ⚡ default	backend ∅ default / ⚡ default	Deny	...

Intentions



```
Kind = "service-intentions"
Name = "web"
Sources = [
    {
        Name      = "ingress-service"
        Action    = "allow"
    },
    {
        Name      = "api"
        Action    = "deny"
    }
]
```

CODE EDITOR

Intentions (Application Aware)



```
Kind = "service-intentions"
Name = "web"
Sources = [{  
    Name = "ingress-service"  
    Permissions = [  
        {  
            Action = "allow"  
            HTTP {  
                PathExact = "/ui"  
                Methods = ["GET"]  
            }  
        }  
        {  
            Action = "deny"  
            HTTP {  
                PathExact = "/admin"  
                Methods = ["GET"]  
            }  
        }  
    ]  
}]
```

CODE EDITOR

— 03

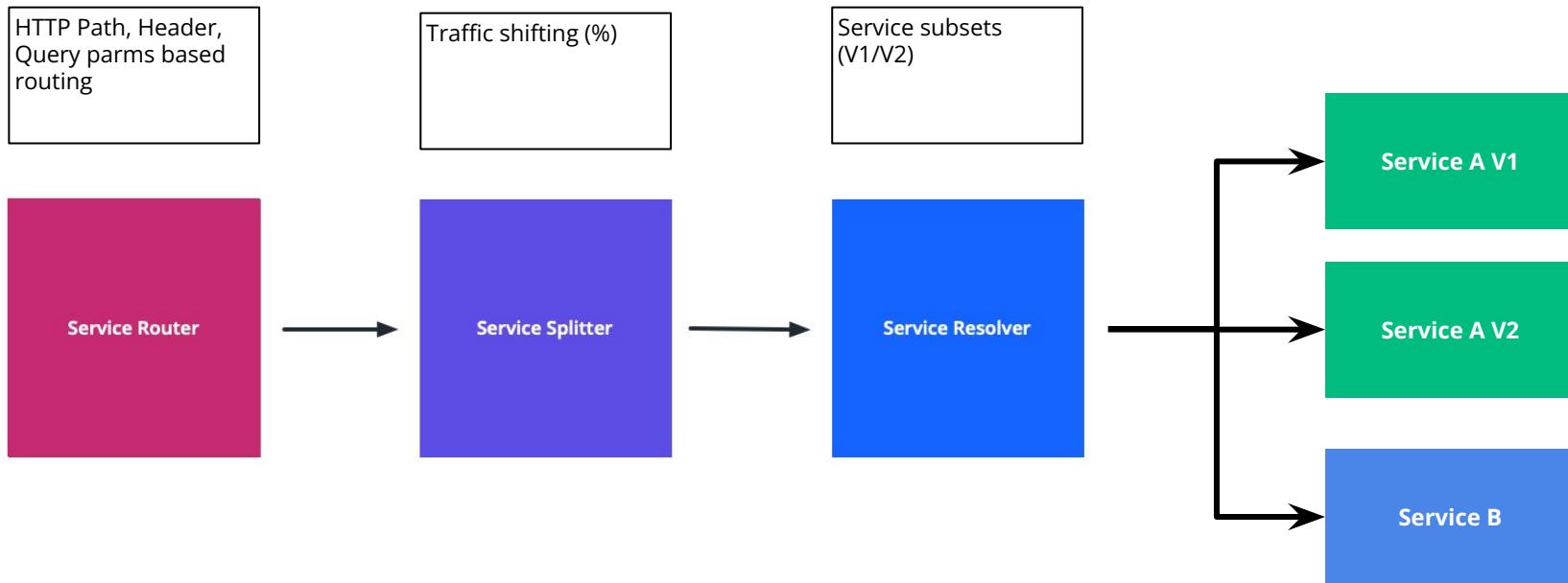
Traffic Control



Consul Discovery Chain

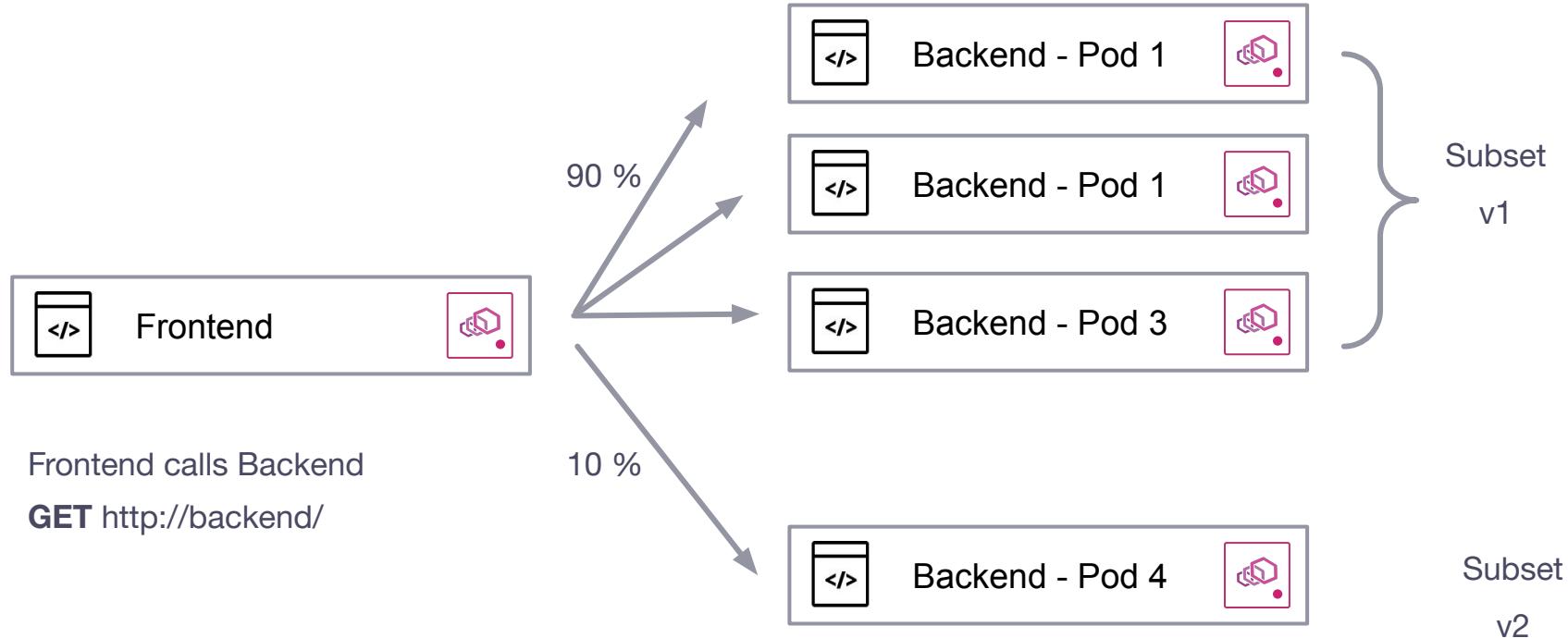


Consul Discovery Chain





Traffic Splitting - Weights



Service Resolver



```
kind = "service-resolver"
name = "backend"

default_subset = "v1"

subsets = {
    v1 = {
        filter = "Service.Meta.version == v1"
    }
    v2 = {
        filter = "Service.Meta.version == v2"
    }
}
```

CODE EDITOR

Service Splitter

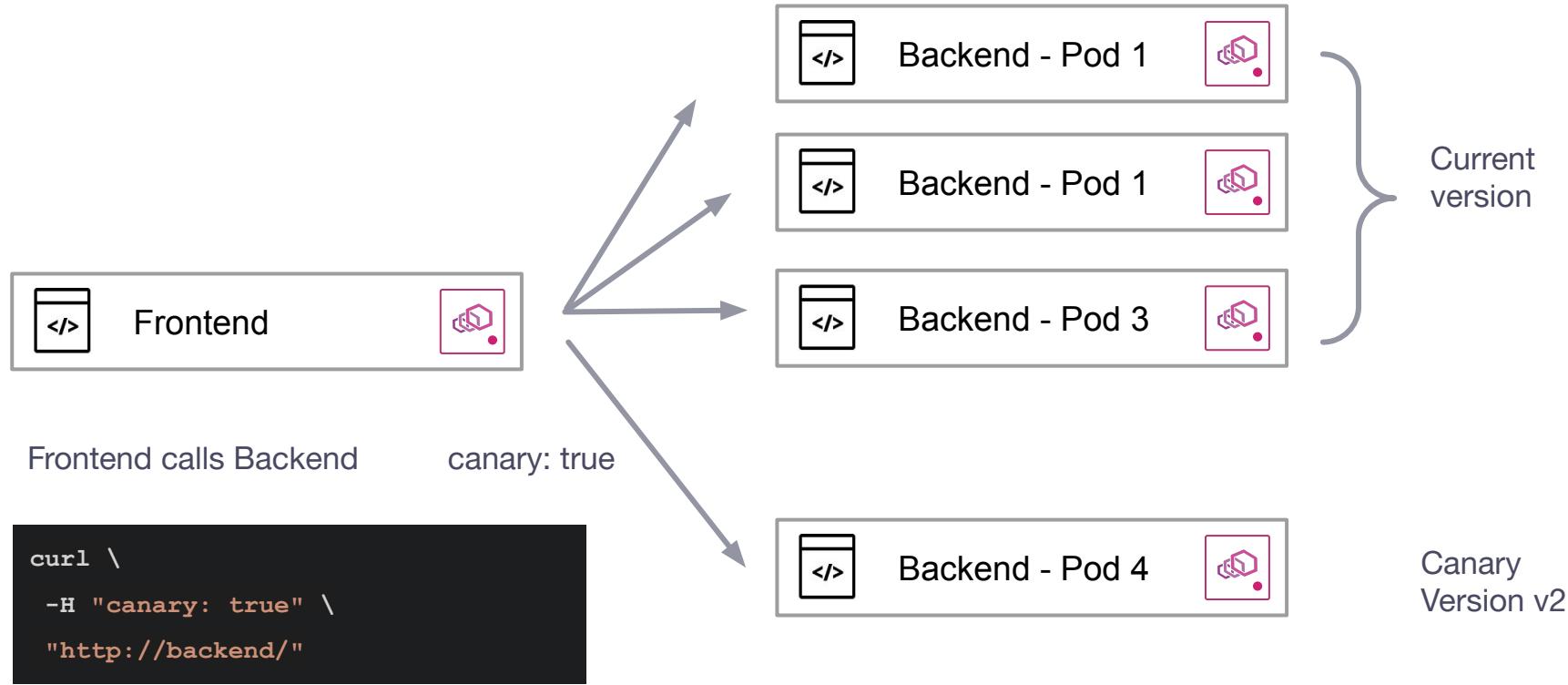


```
kind = "service-splitter",
name = "backend"

splits = [
    {
        weight = 90,
        service_subset = "v1"
    },
    {
        weight = 10,
        service_subset = "v2"
    }
]
```

CODE EDITOR

Traffic Splitting - Request Header



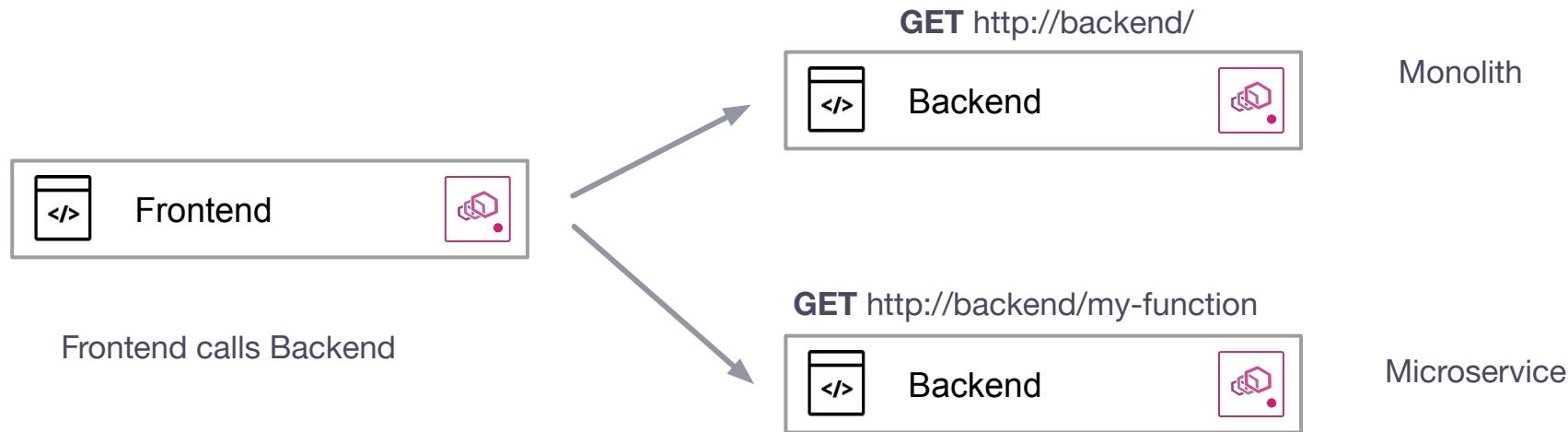
Service Router



```
# A/B test
kind = "service-router"
name = "backend"
routes = [
    {
        match {
            http {
                header = [
                    {
                        name = "canary"
                        exact = "true"
                    },
                ]
            }
        }
        destination {
            service = "backend"
            service_subset = "v2"
        }
    }
]
```

CODE EDITOR

Traffic Splitting - Request Path



Service Router



```
kind = "service-router"
name = "backend"
routes = [
    {
        match {
            http {
                path_prefix = "/my-function"
            }
        }

        destination {
            service = "my-function"
        }
    },
    {
        match {
            http {
                path_prefix = "/"
            }
        }

        destination {
            service = "backend"
        }
    }
]
```

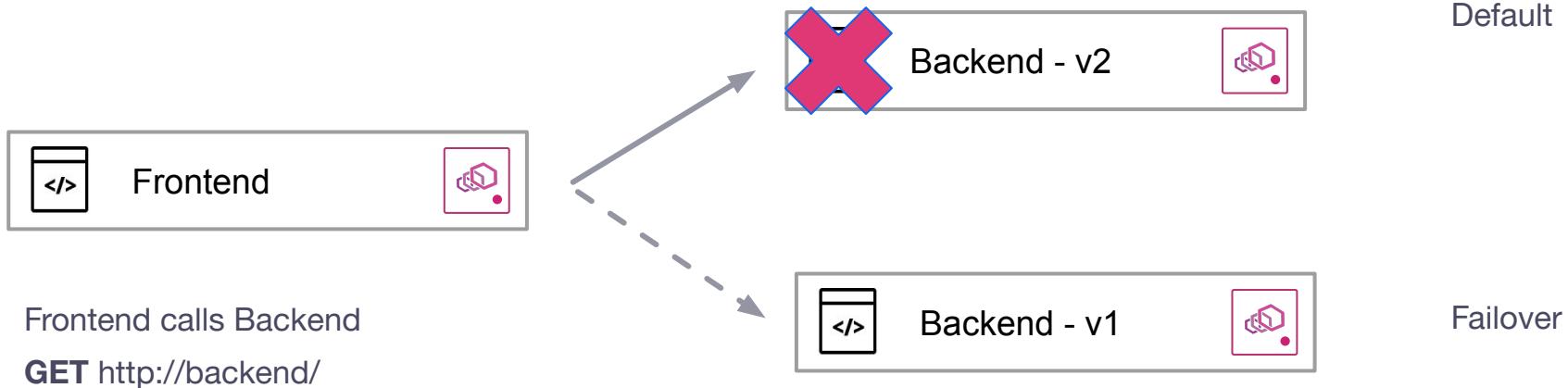
CODE EDITOR

— 04

Service Resiliency



Failover



Failover Subset



```
kind          = "service-resolver"
name         = "backend"

failover = {
    "*" = {
        Service = "backend"
        ServiceSubset = "v1"
    }
}
```

CODE EDITOR

Service Datacenter



```
● ● ● ● CODE EDITOR

kind          = "service-resolver"
name         = "backend"

failover = {
    "*" = {
        Service = "backend"
        Datacenters = ["dc2"]
    }
}
```

Retry and Timeout



```
kind = "service-router"
name = "frontend"

routes = [
    {

        destination {
            RequestTimeout = "10s"
            NumRetries = "3"
            RetryOnConnectFailure = true
        }
    }
]
```

CODE EDITOR

— 05

Observability



Observability



A measure of how well you can infer a system's internal state by observing its outputs

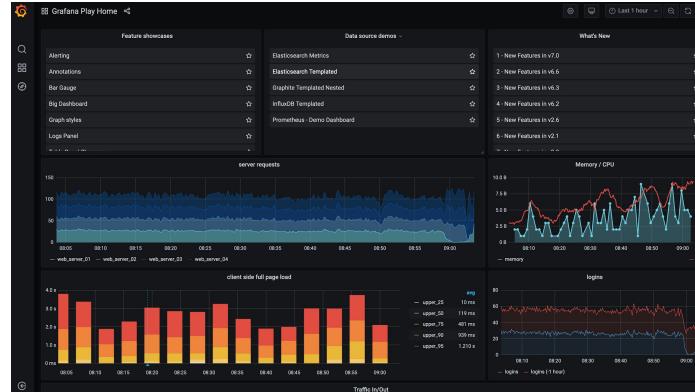
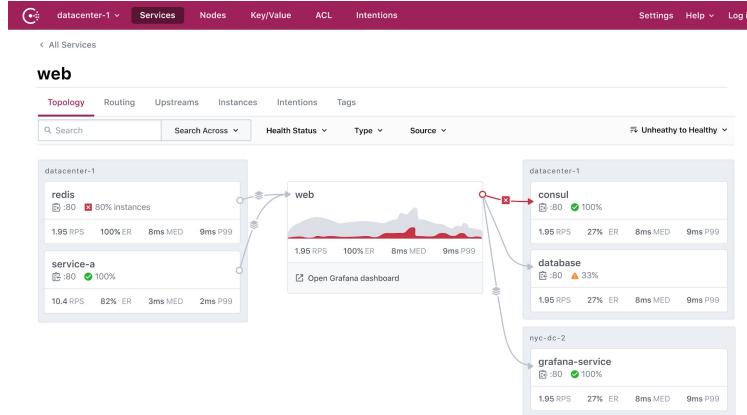
- Checks that increase observability include:
 - HTTP status codes
 - Response latency
 - Tracing data
- Traditionally, these methods were built into services themselves.

Service Mesh Visualization



Ability to see what services are communicating with which services within the service mesh.

The side-car proxies are emitting metrics, and you can scrape these via Prometheus and visualize them via Grafana dashboards.

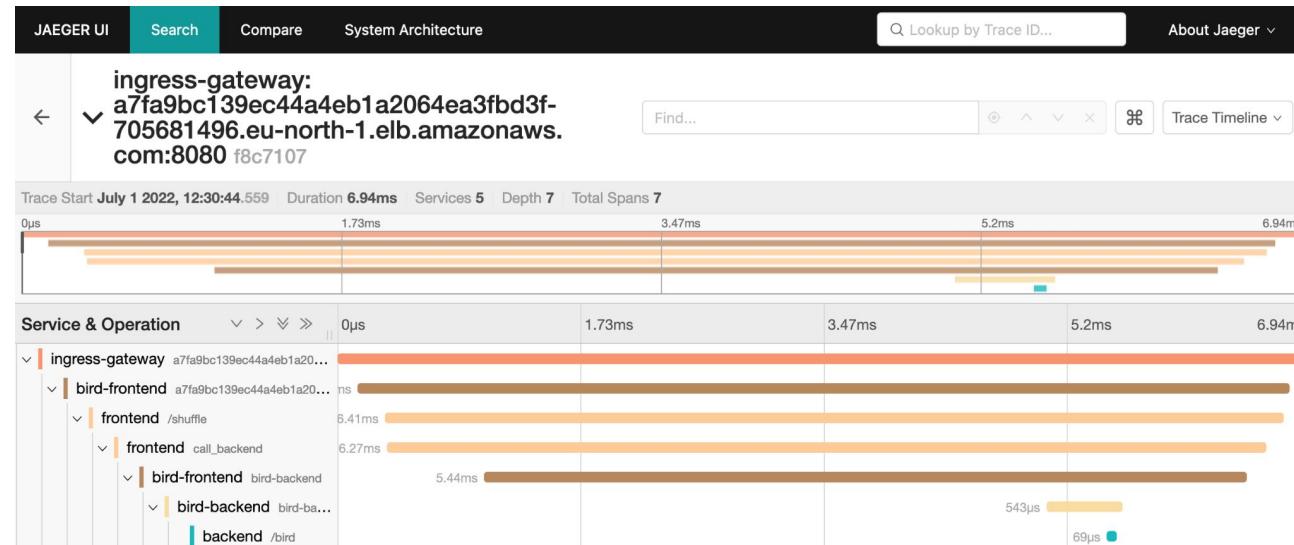




Distributed Tracing

Observe requests that span multiple microservices.

Integration with Tracing Collectors like Jaeger by emitting spans via the Zipkin format.



— 06

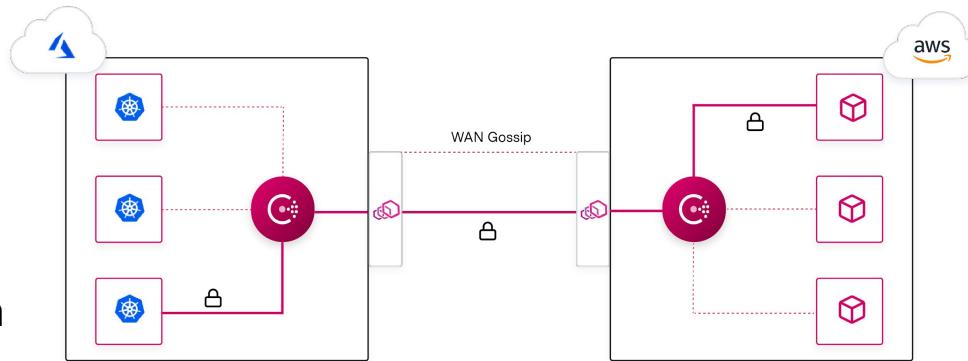
Gateways





Mesh Gateway

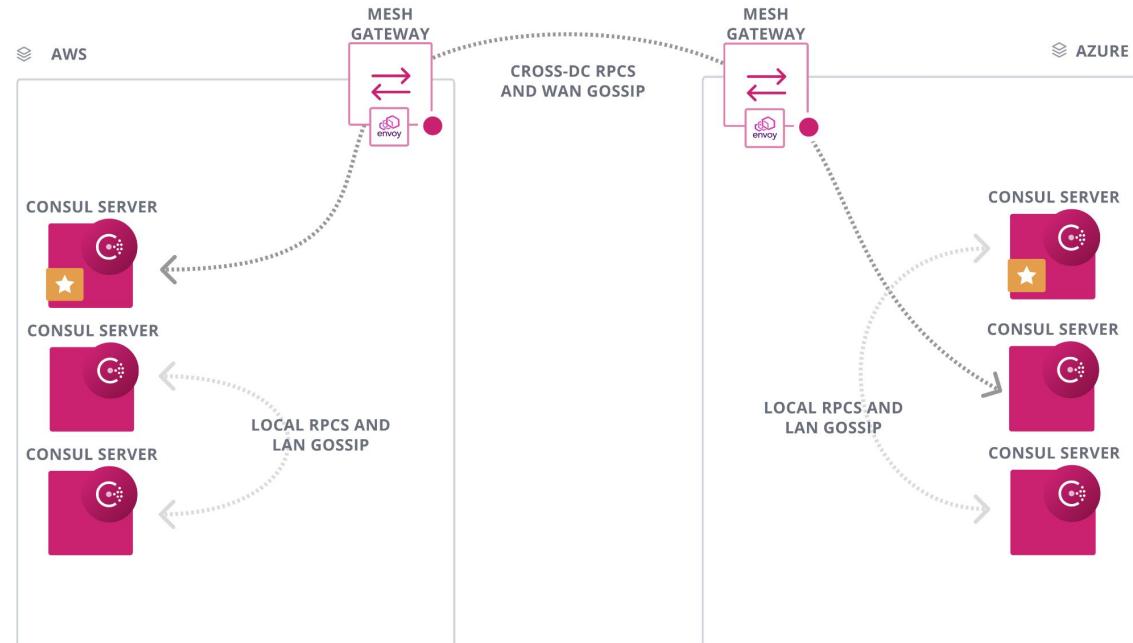
- Extend Consul service mesh across potentially complex environments:
 - Isolated networks
 - Overlapping IP addresses
 - Kubernetes environments
- Automatic mTLS Encryption between Consul datacenters
- Layer 7 Traffic Policies between environments



WAN Federation over Mesh Gateways



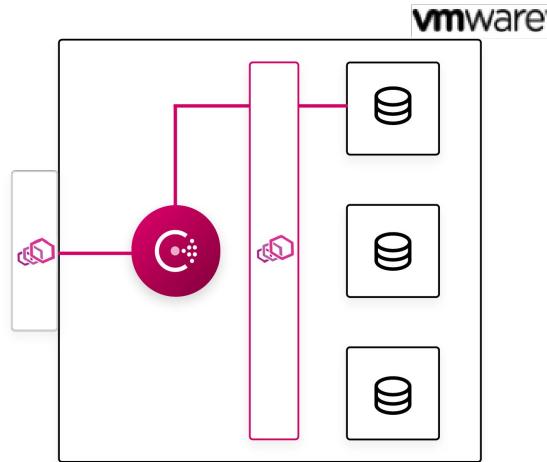
- All Consul traffic passes through Mesh Gateway - including WAN Gossip
- Secure service communication between clouds without VPN or cloud specific services
- Deliver redundancy and lifecycle capabilities across cloud environments





Terminating Gateway

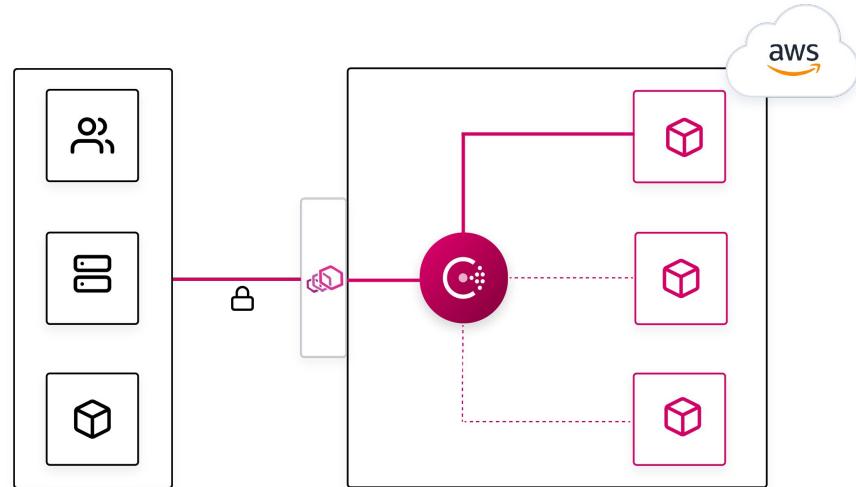
- Outbound access from service mesh workloads
- Extend Consul security and traffic policies to external infrastructure and managed services
- Acts as an egress proxy for the service mesh environment



Ingress Gateway



- Inbound access to service mesh workloads
- Enables one-to-many access between ingress gateway and service mesh services
- Logical point of layer 7 traffic policies
- Route traffic to services based on URI path, header, etc...



Example: Ingress Gateway



```
Kind = "ingress-gateway"
Name = "my-ingress"
Namespace = "default"

Listeners = [
{
    Port      = 8080
    Protocol = "http"
    Services = [
{
        Name   = "frontend"
        Namespace = "web"
        Partition = "web-team"
        Hosts = ["*"]
    }
]
}
]
```

CODE EDITOR

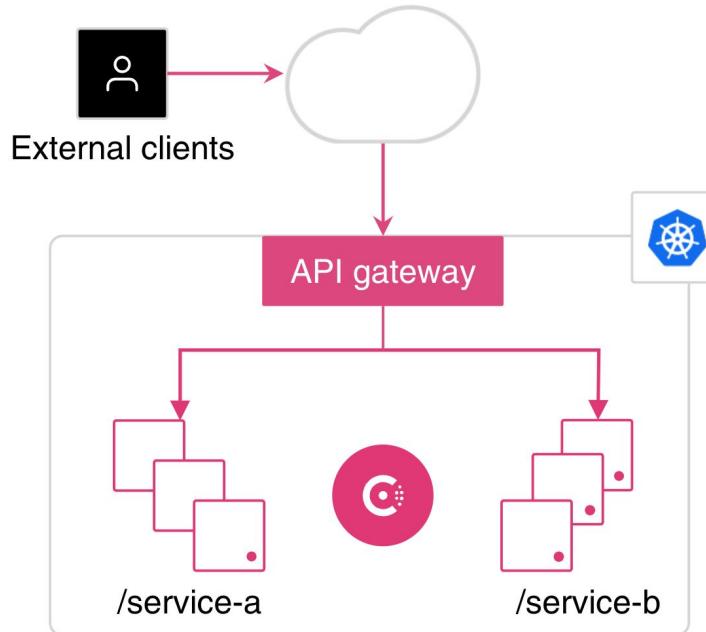
API Gateway



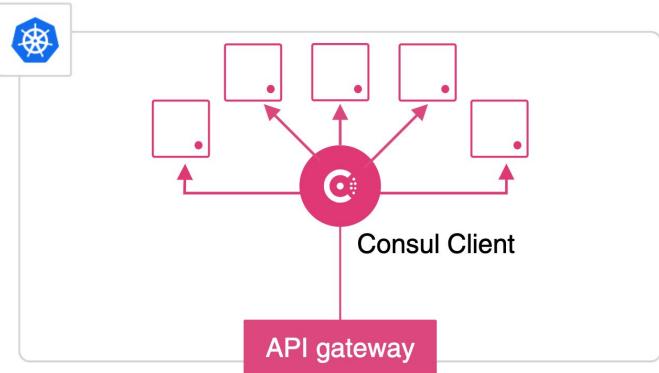
Control access to the service mesh

Consul API Gateway is designed to be deployed alongside Consul service mesh helping users control access into and out of the service mesh. It provides organizations the ability to:

- Control access at the point of entry
- Centralize traffic management
- Route requests based on metadata



Consul API Gateway Installation



Step 1: Install CRDs

```
$ kubectl apply  
--kustomize="github.com/hashicorp/consul-api-gateway/config  
/crd?ref=v0.1.0"
```

Step 2: Configure & Install via Helm

```
apiGateway:  
  enabled: true  
  image: hashicorp/consul-api-gateway:0.1.0
```

```
$ helm install consul ... --version "0.40.0"
```

API Gateway Components



Feature	Description
Gateway	Defines the main infrastructure resource that links API gateway components. It specifies the name of the <code>GatewayClass</code> and one or more <code>Listeners</code> , which specify the logical endpoints bound to the gateway's addresses.
GatewayClass	Defines a class of gateway resources that you can use as a template for creating gateways.
GatewayClassConfig	Describes additional Consul API Gateway-related configuration parameters for the <code>GatewayClass</code> resource.
Routes	Specifies the path from the client to the services. They attach to <code>Listeners</code> on the Gateways.
API Gateway Controller	Controller required for deploying and managing the API Gateway service. Retrieves configuration and authentication credentials from the <code>GatewayClassConfig</code> definition.

API Gateway Supported Features



- Multiple Listeners (HTTP, HTTPS, TCP, and TCP + TLS)
- Load Balancing
- Client Traffic Routing based on matching criteria
(URL, Hostname, Headers & values, HTTP Method and Query string)
- Use TLS Certs signed by external CA
- Split traffic based on weighted ratios
- Add/Delete/Modify HTTP headers and values
- Kubernetes

Example: Gateway



```
apiVersion: gateway.networking.k8s.io/v1alpha2
kind: Gateway
metadata:
  name: example-gateway
spec:
  gatewayClassName: consul-api-gateway
  listeners:
    - protocol: HTTP
      port: 9090
      name: http
      allowedRoutes:
        namespaces:
          from: Same
```

CODE EDITOR

Example: HTTPRoute



```
apiVersion: gateway.networking.k8s.io/v1alpha2
kind: HTTPRoute
metadata:
  name: productpage-route
spec:
  parentRefs:
  - name: example-gateway
  rules:
  - matches:
    - path:
        type: PathPrefix
        value: /
  backendRefs:
  - kind: Service
    name: frontend
    namespace: default
    port: 9090
```

CODE EDITOR

— 06

Next Steps





Learn

<https://learn.hashicorp.com/consul>

Step-by-step guides to accelerate deployment of Consul

The screenshot shows the HashiCorp Learn platform interface for Consul. The left sidebar contains navigation links for 'GET STARTED' (Consul on HCP, Consul on Kubernetes, Consul on VMs) and 'USE CASES' (Kubernetes Service Mesh, Microservices, NIA, Service Discovery & Health, Service Mesh & Gateways). The main content area features a prominent banner with the text 'Deploy a fully managed service mesh' and a call-to-action 'Sign up for HCP Consul'. Below the banner, there are three main sections: 'Learn Consul fundamentals' (7 TUTORIALS), 'HashiCorp Cloud Platform (HCP) Consul' (5 TUTORIALS), and 'Get Started on VMs' (9 TUTORIALS).

Consul

GET STARTED

- Consul on HCP
- Consul on Kubernetes
- Consul on VMs

USE CASES

- Kubernetes Service Mesh
- Microservices
- NIA
- Service Discovery & Health
- Service Mesh & Gateways

CERTIFICATION PREP

- Associate

Learn Consul fundamentals

7 TUTORIALS

HashiCorp Cloud Platform (HCP) Consul

Quickly get hands-on with HashiCorp Cloud Platform (HCP) Consul using the HCP portal quickstart deployment, experiment with the...

5 TUTORIALS

Get Started on Kubernetes

Setup Consul service mesh to get experience deploying service sidecar proxies and securing service with mTLS.

9 TUTORIALS

Get Started on VMs

Consul is a networking tool that provides a fully featured service mesh and service discovery. Try Consul locally.



Resources

- [Consul Service Mesh](#)
- [Consul Service Mesh with Kubernetes](#)
- [Certificate Management](#)
- [Intentions](#)
- [Discovery Chain](#)
- [Observability](#)
- [Observability with Kubernetes, Tutorial](#)
- [Distributed Tracing](#)
- [Multi-Cluster Federation via Mesh Gateways](#)
- [API Gateway](#)

Need Additional Help?



Customer Success

Contact our Customer Success Management team with any questions. We will help coordinate the right resources for you to get your questions answered.

customer.success@hashicorp.com

Technical Support

Something not working quite right? Engage with HashiCorp Technical Support by opening a ticket for your issue at support.hashicorp.com.

Discuss

Engage with the HashiCorp Cloud community including HashiCorp Architects and Engineers

discuss.hashicorp.com

Upcoming Onboarding Webinars



Office Hour

An interactive open forum to discuss specific questions about your environment and Use Cases.

Please bring your questions!

Webinar:

Service Configuration and Integrations
Topics include: Network Infrastructure Automation & Consul Terraform Sync

Webinar:

Program Closing
A wrap up of the program content and Consul Production Readiness Checklist

Q & A



Thank You

customer.success@hashicorp.com
www.hashicorp.com/customer-success