envoycon

NORTH AMERICA

envoycon NORTH AMERICA

Extending Envoy: A Guide to Custom Extensions with Envoy Gateway

Guy Daich, SAP
Huabing Zhao, Tetrate

Background: Gateway API

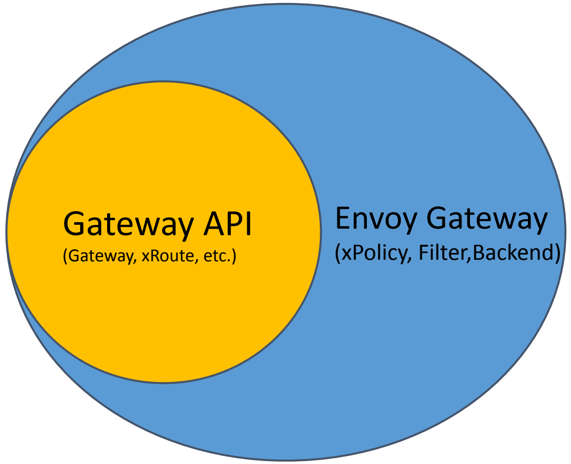
Gateway API: A Superior Alternative to Ingress API

- **API Features:** Offers a richer set of features compared to Ingress API
- **Gateway Standard:** Designed as a universal API standard to maximize compatibility across implementations
- **Extension Mechanism:** Introduces well-defined extension points, replacing the unstructured annotations in Ingress API
  - Policy Attachment: Enables attaching custom policies without altering the core API
  - Custom Filters: Supports rule-level custom traffic filters for granular control
  - Custom Backend: Routes traffic efficiently to various backends

Background: Envoy Gateway

Envoy Gateway: Empowering Gateway API with Envoy

- A Gateway controller fully compatible with Gateway API
- Simplifies deployment and management of Envoy as an API Gateway
- Handles traffic for both Kubernetes clusters and VM-based workloads
- Goes beyond the Gateway API with advanced features
  - Policy Attachment
    - ClientTrafficPolicy
    - BackendTrafficPolicy
    - SecurityPolicy
    - EnvoyPatchPolicy
    - EnvoyExtensionPolicy
  - HTTPRouteFilter
  - Backend
    - IP
    - HostName
    - Unix Domain Socket



Envoy Extension Policy

EnvoyExtensionPolicy allows loading custom extensions into Envoy to execute user-defined logic for request and response processing.

- Supported Extension types:
  - Wasm
  - Ext-Proc
  - More in future: Lua, Dynamic module …
- Configuration Flexibility:
  - Supported attachment targets: Gateway, HTTPRoute
  - Multiple Ext-Proc/Wasm extensions can be defined per policy
  - **EnvoyProxy** CR can be used to determine filter ordering setting, allowing users to control when Wasm/Ext-Proc filters are invoked in the filter chain

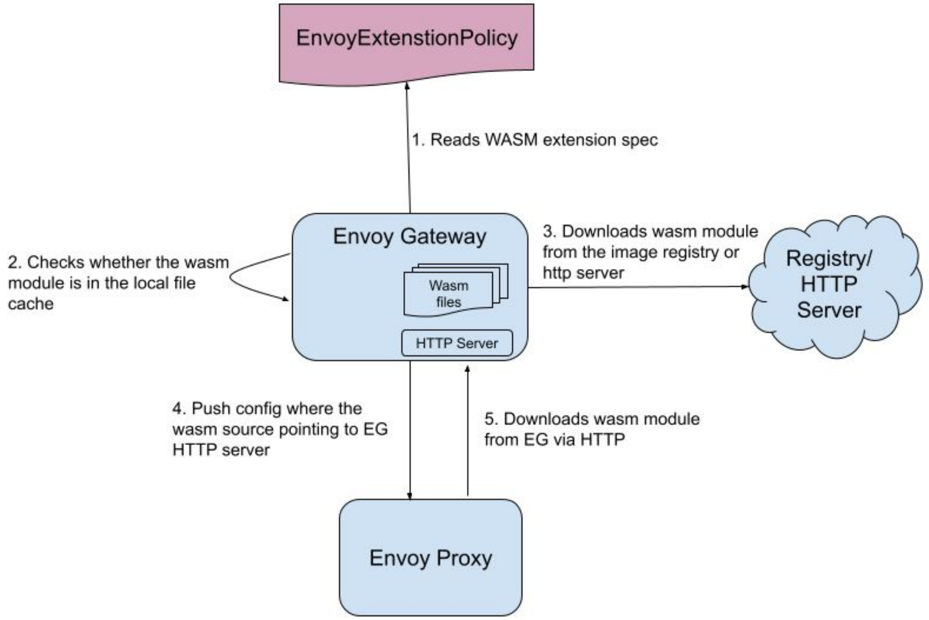
```yaml
apiVersion: gateway.envoyproxy.io/v1alpha1
kind: EnvoyProxy
metadata:
  name: custom-proxy-config
  namespace: envoy-gateway-system
spec:
  filterOrder:
    - name: envoy.filters.http.ext_proc
      after: envoy.filters.http.ratelimit
```

```yaml
apiVersion: gateway.envoyproxy.io/v1alpha1
kind: EnvoyExtensionPolicy
metadata:
  namespace: envoy-gateway
  name: my-custom-extensions
spec:
  targetRef:
    group: gateway.networking.k8s.io
    kind: Gateway
    name: my-gateway
  wasm:
  - name: wasm-filter-1
    code:
      type: HTTP
      http:
        url: https://www.example.com/wasm-filter-1.wasm
        sha256: 746df05c8f3a0b07a46c0967cfbc5cbe5b9d48d0
  - name: wasm-filter-2
    code:
      type: Image
      image:
        url: oci://www.example.com/wasm-filter-2:v1.0.0
        pullSecretRef:
          name: my-pull-secret
        sha256: a1efca12ea51069abb123bf9c77889fcc2a31cc5
  extProc:
  - backendRefs:
    - name: my-ext-proc-svc
      port: 8000
```

Wasm Extension - OCI Image Support

Envoy Gateway supports OCI image as a remote Wasm code source.

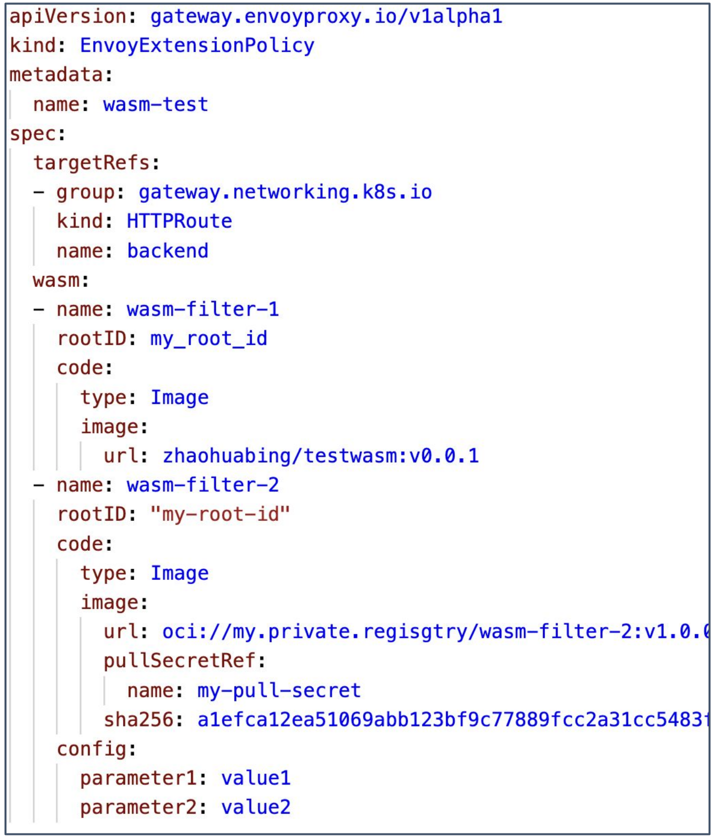
- **Versioning**: Users can use the tag of the OCI image to manage the version of the Wasm module.
- **Security**: Users can use private registries to store the Wasm module securely.
- **Distribution**: Users can use the existing tools of the OCI registry to distribute the Wasm module.



EnvoyExtenstionPolicy
1. Reads WASM extension spec
2. Checks whether the wasm module is in the local file cache
Envoy Gateway
Wasm files
HTTP Server
3. Downloads wasm module from the image registry or http server
Registry/ HTTP Server
4. Push config where the wasm source pointing to EG HTTP server
5. Downloads wasm module from EG via HTTP
Envoy Proxy

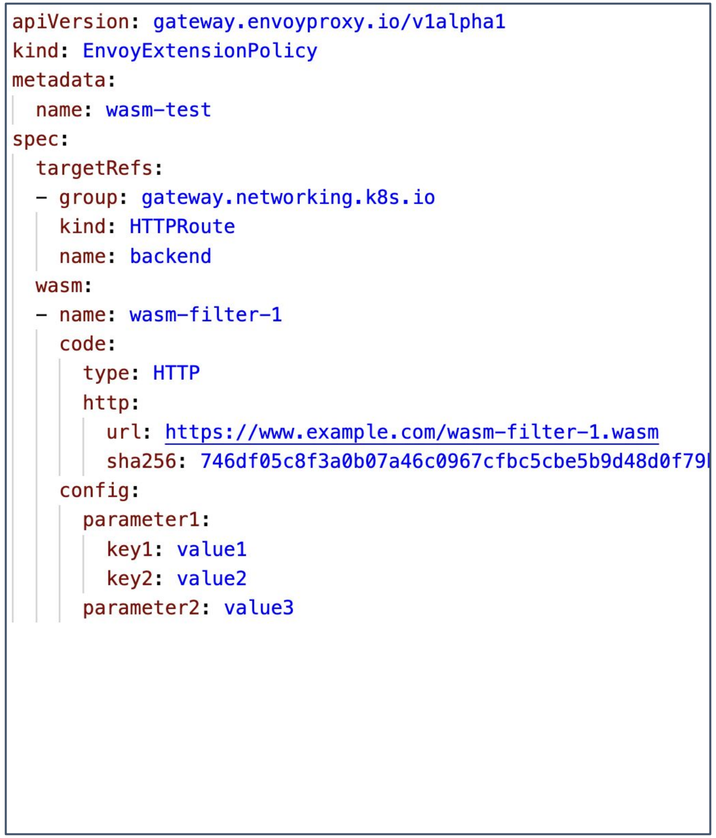
EnvoyExtensionPolicy - Wasm Extension

OCI Image Wasm source



```yaml
apiVersion: gateway.envoyproxy.io/v1alpha1
kind: EnvoyExtensionPolicy
metadata:
  name: wasm-test
spec:
  targetRefs:
  - group: gateway.networking.k8s.io
    kind: HTTPRoute
    name: backend
  wasm:
  - name: wasm-filter-1
    rootID: my_root_id
    code:
      type: Image
      image:
        url: zhaohuabing/testwasm:v0.0.1
  - name: wasm-filter-2
    rootID: "my-root-id"
    code:
      type: Image
      image:
        url: oci://my.private.regisgtry/wasm-filter-2:v1.0.
        pullSecretRef:
          name: my-pull-secret
        sha256: a1efca12ea51069abb123bf9c77889fcc2a31cc5483
    config:
      parameter1: value1
      parameter2: value2
```

HTTP Wasm source



```yaml
apiVersion: gateway.envoyproxy.io/v1alpha1
kind: EnvoyExtensionPolicy
metadata:
  name: wasm-test
spec:
  targetRefs:
  - group: gateway.networking.k8s.io
    kind: HTTPRoute
    name: backend
  wasm:
  - name: wasm-filter-1
    code:
      type: HTTP
      http:
        url: https://www.example.com/wasm-filter-1.wasm
        sha256: 746df05c8f3a0b07a46c0967cfbc5cbe5b9d48d0f79
    config:
      parameter1:
        key1: value1
        key2: value2
      parameter2: value3
```

External Processing in Envoy Gateway

envoycon NORTH AMERICA

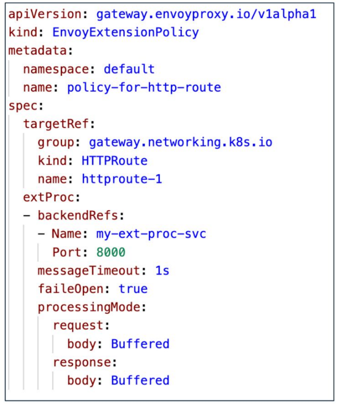
The **External Processing** extension is an HTTP filter that supports out-of-process extensibility using a callout to an **external gRPC service**.

The external gRPC service can inspect and mutate Headers and Bodies of the HTTP stream by registering to relevant **stream hooks.**

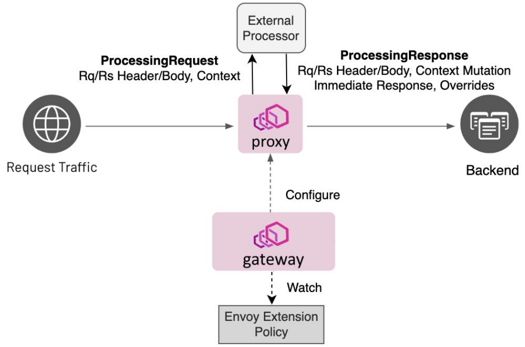
Ext-Proc Protocol:

- **Processing Request**: Created by Envoy and sent to external processor for a registered hook, containing one of HTTP Request/Response headers or body (chunks).

- **Processing Response**: Created by the grpc service in response to a Processing Request, containing mutation of HTTP request/response headers or body or Immediate Response.



```yaml
apiVersion: gateway.envoyproxy.io/v1alpha1
kind: EnvoyExtensionPolicy
metadata:
  namespace: default
  name: policy-for-http-route
spec:
  targetRef:
    group: gateway.networking.k8s.io
    kind: HTTPRoute
    name: httproute-1
  extProc:
  - backendRefs:
    - Name: my-ext-proc-svc
      Port: 8000
    messageTimeout: 1s
    faileOpen: true
    processingMode:
      request:
        body: Buffered
      response:
        body: Buffered
```



Advanced Deployment Patterns

Implications of an out-of-process extension:

- **Security**: Ensure identity of external processor and confidentiality/integrity of messages
- **Resilience**: communication to a different component should consider network instability, decoupled lifecycle of components, possible unavailability, etc.
- **Performance**: communication with a remote service, additional (un)marshaling, additional encryption

Envoy Gateway offers advanced integration options for the external grpc service:

- **Security**: Use **BackendTLSPolicy** and **EnvoyProxy** CRs to establish MTLS between Envoy Proxy and External Processing
- **Resilience**: Use Envoy Gateway's Extended **BackendRef** to configure a variety of behaviors for the External Processor connection pool, such as: Load Balancing, Circuit Breaking, HealthChecks (incl. grpc), Failover (primary, secondary), Timeouts, Buffer Sizes, KeepAlive and more
- **Performance**: Use **Backend** and **EnvoyProxy** CRs to integrate an External Processor that is exposed with Unix domain sockets in the same Pod (sidecar) or Host (decoupled deployment and scaling).

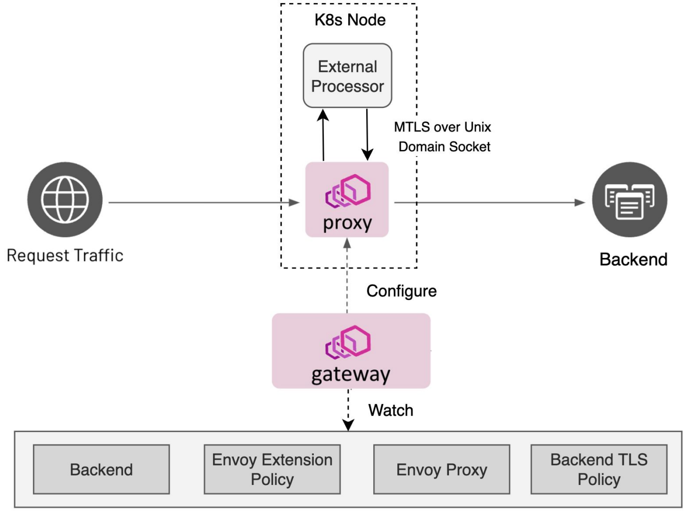
Advanced Deployment Patterns

```yaml
apiVersion: gateway.envoyproxy.io/v1alpha1
kind: Backend
metadata:
  name: grpc-uds-ext-proc
spec:
  endpoints:
  - unix:
      path: /var/run/ext-proc/extproc.sock
```

```yaml
apiVersion: gateway.envoyproxy.io/v1alpha1
kind: EnvoyExtensionPolicy
metadata:
  name: policy-uds-ext-proc
spec:
  targetRefs:
  - group: gateway.networking.k8s.io
    kind: HTTPRoute
    name: http-with-extproc-uds-tls
  extProc:
  - backendRefs:
    - kind: Backend
      group: gateway.envoyproxy.io
      name: grpc-uds-ext-proc
    processingMode:
      request: {}
      response: {}
    backendSettings:
      healthCheck:
        active:
          healthyThreshold: 1
          type: GRPC
      circuitBreaker:
        maxConnections: 2048
      timeout:
        tcp:
          connectTimeout: 15s
```

```yaml
apiVersion: gateway.networking.k8s.io/v1alpha3
kind: BackendTLSPolicy
metadata:
  name: policy-btls-uds-extproc
spec:
  targetRefs:
  - group: gateway.envoyproxy.io
    kind: Backend
    name: grpc-uds-ext-proc
  validation:
    caCertificateRefs:
    - name: grpc-ext-proc-ca
```

```yaml
apiVersion: gateway.envoyproxy.io/v1alpha1
kind: EnvoyProxy
metadata:
  name: proxy-config
spec:
  provider:
    type: Kubernetes
    kubernetes:
      envoyDeployment:
        container:
          volumeMounts:
          - mountPath: /var/run/ext-proc
            name: socket-dir
        pod:
          volumes:
          - name: socket-dir
            hostPath:
              path: /var/run/ext-proc
              type: ""
  backendTLS:
    clientCertificateRef:
      kind: Secret
      name: example-client-cert
```



When to Use Wasm/ExtProc

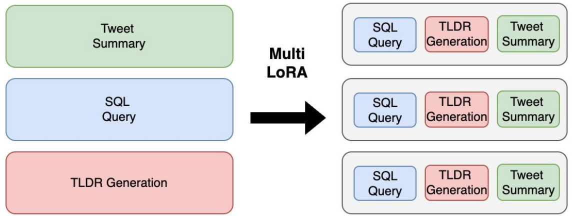
Wasm extensions excel at **lightweight, in-path data processing**, while External Process extensions can handle more **complex logic requiring external system interaction**.

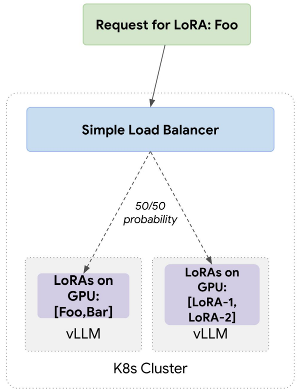
Consider the below factors when choosing between them:

- **Performance:**
  - Wasm extensions offer superior performance since they run directly within the Envoy process.
  - External Process extensions rely on network calls, which can slightly reduce performance due to additional latency. Sidecar deployment can mitigate this to some degree.
- **Functionality:**
  - Wasm operates in a sandbox, restricting system calls and access to external resources.
  - External Process extensions have no such limitations and can be built in any language, with full access to system resources.
- **Deployment:**
  - Wasm extensions can be dynamically loaded by Envoy from an OCI registry or HTTP URL.
  - External Process extensions require managing a separate process, adding complexity to deployment.
- **Security:**
  - Wasm runs within Envoy, meaning bugs in the extension could impact Envoy's stability.
  - External Process extensions run independently, so failures won't directly affect Envoy's operation.
- **Scalability:**
  - Wasm extensions are embedded in Envoy and scale together with the Envoy instances.
  - External Process extensions can scale independently, with separate resource management.

Demo: K8s LLM Instance Gateway

- **LLMs** can be **tuned** for specific tasks.
- For LLM operators, operating a fleet of single-purpose (tuned) LLM servers is **costly** and often leads to **underutilization of resources**.
- **Low-Rank Adaptation (LoRA)**, a popular tuning technique, can be used to run **multiple adapters with a shared model** in a single LLM Server instance, **improving resource utilization**.
- However, the number of **concurrently loaded adapters is limited**, and **naive round-robin** load balancing can lead to significant performance impacts, due to delays for **requests queued for an unloaded adapter.**
- A tailored Load Balancing algorithm has the potential to address these concerns.



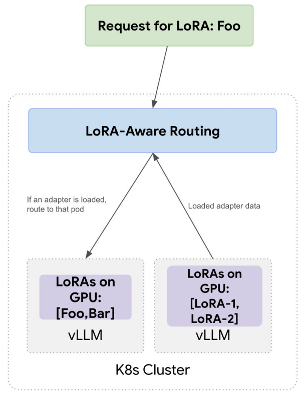


Generic Load Balancing

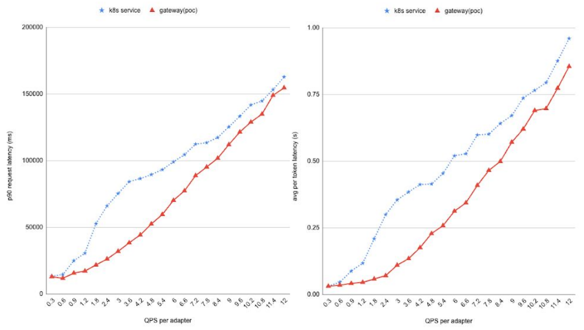
Source: https://docs.google.com/presentation/d/1I1XDf6fQQEtHxJtZxFdIaUcUA3lLBC7neW823diWS78/edit#slide=id.g2f118dd8646_0_40

Demo: K8s LLM Instance Gateway

- The **K8s Serving WG** is working to improve K8s as a platform for LLM inference. Together with the **Gateway API WG**, they have explored a solution to this issue based on **Gateway extensions**.
- Gateways provide a **comprehensive view** of system-wide and **instance-specific LLM resource utilization and adapter state**. They are also independent of any specific LLM server implementation
- Gateways can implement **LoRA-aware load balancing** to select pods with the appropriate **LoRA adapter loaded, shortest queue, and lowest KV cache** utilization.
- Early results from PoCs showed that **throughput increased by 85%** and latency decreased significantly.



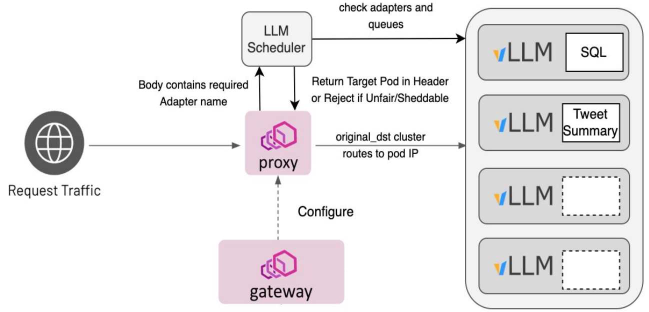
LoRA-Aware Load Balancing



Source: https://docs.google.com/document/d/11ALHEF-9yOaLdbHbDjBoTY6fzejoEKiSYHzWpWqe8ZY/edit?tab=t.0#heading=h.4zo9ni80x3j5

Demo: K8s LLM Instance Gateway

- The LLM scheduler is an External Processing service which is registered to request body and header hooks.
    - Scheduler **monitors** the state of vLLM pods, consuming **metrics** such as Loaded adapters, queue sizes, KV Cache utilization.
    - Incoming requests are buffered and the desired LoRA adapter is **extracted from the body**.
    - The **optimal pod** for serving the request is selected and provided to Envoy via a dedicated header added with **header mutation**.
    - If load shedding or rate limiting is required an **immediate response** is returned.
- Upstream routing leverages the Envoy **origianl_dst** cluster, capable of dynamically routing to an address specified by the scheduler.


LLM Scheduler
check adapters and queues
Body contains required Adapter name
Return Target Pod in Header or Reject if Unfair/Sheddable
Request Traffic
proxy
original_dst cluster routes to pod IP
Configure
gateway
vLLM
SQL
vLLM
Tweet Summary
vLLM
vLLM

Demo: K8s LLM Instance Gateway

Learn more about LLM Instance Gateway and WG-Serving:

- LLM Instance Github, Proposal
- WG-Serving: Github, KubeCon NA 24 Project Talk

Get involved in Envoy Gateway AI related topics:

- Slack: #envoy-ai-gateway
- Envoy AI Gateway Webinar, KubeCon NA 24

Demo

- K8s and Envoy Gateway manifests
- Implementation of an External Processing service
- Demonstrate LoRA-Aware routing, where requests are routed to vLLM pods that have the desired LoRA adapter loaded

Thank You

envoycon NORTH AMERICA

Q&A

Get involved in Envoy Gateway:

- https://github.com/envoyproxy/gateway
- https://gateway.envoyproxy.io/docs/

Session Feedback: