AppSentinels API Security Platform

Istio Ingress Gateway Integration

Envoy Lua Filter

Table of Contents

Revision History

Revision	Date Modified	Author	Comments
1.0	11-Sep-25	Sachin	Initial release: Istio ingress gateway integration (filter v1.0.2)

1. Introduction

AppSentinels integrates with Istio service mesh deployments at the ingress gateway using a native Envoy extension. The integration is delivered as a single Kubernetes resource, an Istio `EnvoyFilter`, that adds a lightweight Lua HTTP filter to the gateway's Envoy proxy. No changes are required to application code, application pods, or the gateway container image.

The filter observes API traffic passing through the gateway and asynchronously sends request and response logs to the AppSentinels Edge Controller for security processing. The Edge Controller forwards security metadata to the AppSentinels Cloud for AI/ML-based analysis, API discovery and threat detection. Optionally, the filter can operate inline and enforce blocking decisions returned by the Edge Controller. Default behavior is out of band logging.

This document will help you:

- Understand how the AppSentinels filter works with the Istio ingress gateway
- Confirm the prerequisites and network connectivity
- Configure and deploy the AppSentinels `EnvoyFilter`
- Verify, troubleshoot and maintain the integration

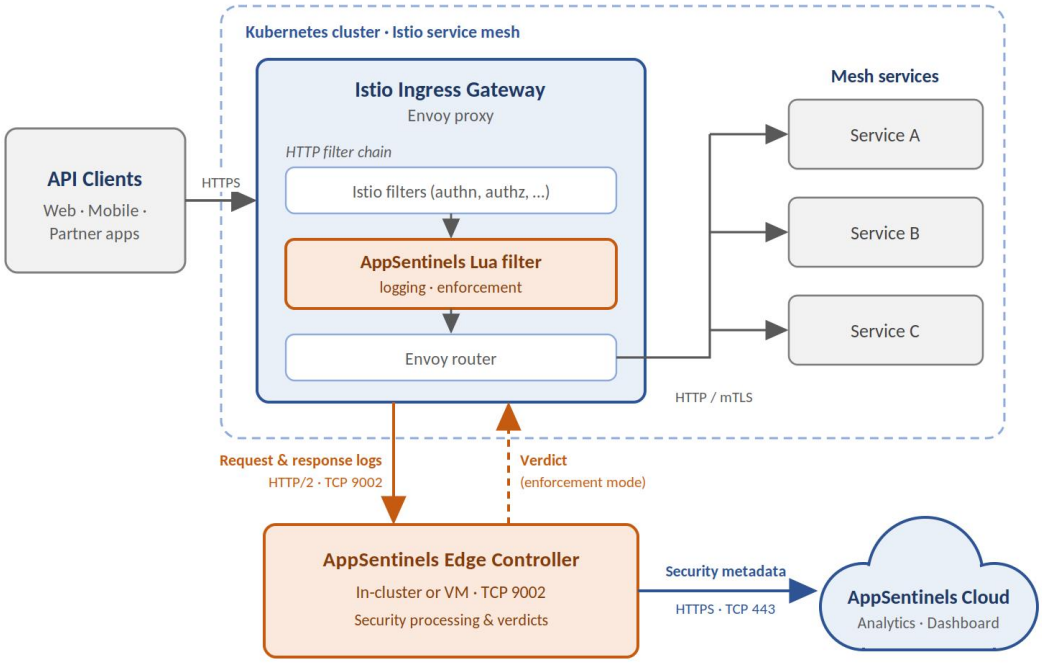

Figure 1 Istio ingress gateway deployment with AppSentinels

1. API clients send traffic to the Istio ingress gateway.
2. The AppSentinels Lua filter, running inside the gateway's Envoy proxy, captures the request and sends a copy to the Edge Controller over HTTP/2 on TCP port 9002.
3. The gateway routes the request to the destination service as usual. In enforcement mode, the gateway first waits (up to a configurable timeout) for the Edge Controller's verdict and blocks the request if instructed.
4. When the response passes back through the gateway, the filter sends a response log to the Edge Controller asynchronously.

1.1 Related Resources

This guide covers the Istio gateway integration only. You will also need the following resources:

Resource	Description and location
Edge Controller Pre-Requisites & Deployment guide (PDF)	Requirements, deployment options, secure logging and verification for the AppSentinels Edge Controller. Deploy the controller before starting this integration. https://downloads.appsentinels.ai/appsentinels-deployment/deployment-guides/AppSentinels-Controller-PreRequisites.pdf
AppSentinels Istio gateway filter (YAML)	The `EnvoyFilter` resource, including the complete AppSentinels Lua script. Its configuration is described in Section 4. https://downloads.appsentinels.ai/appsentinels-deployment/envoy-istio-gateway/appsentinels-envoy-istio-gateway-filter.yaml

2. Architecture & Design

2.1 Components

Component	Description
Istio ingress gateway	Your existing Envoy-based Istio gateway. It hosts the AppSentinels filter; no image change or sidecar is required.
AppSentinels EnvoyFilter	Istio resource named `request-response-filter`, created in the `istio-system` namespace. It contains two configuration patches: an upstream cluster for the Edge Controller and the Lua HTTP filter.
Controller cluster (`ext_authz-http-service`)	Envoy upstream cluster that points to the Edge Controller on port 9002. It uses HTTP/2, DNS-based endpoint discovery (refreshed every 50 seconds), round-robin load balancing and a 250 ms connect timeout.
AppSentinels Lua filter	Inserted immediately before the Envoy router in the gateway's HTTP filter chain. It captures request and response metadata and payloads, ships logs to the Edge Controller and, in enforcement mode, applies the controller's verdict.
AppSentinels Edge Controller	Deployed in your environment. It processes traffic logs, detects security events, returns enforcement verdicts and sends metadata to the AppSentinels Cloud.
AppSentinels Cloud	SaaS platform providing AI/ML analytics, the API catalogue and the AppSentinels Dashboard.

2.2 Operating Modes

The filter supports two operating modes, selected with the `ENFORCEMENT` parameter (see Section 4.4).

Aspect	Out-of-Band (OOB) mode, default	Enforcement (inline) mode
Setting	`ENFORCEMENT = false`	`ENFORCEMENT = true`

Aspect	Out-of-Band (OOB) mode, default	Enforcement (inline) mode
Purpose	Visibility, API discovery and threat detection	All OOB capabilities plus inline blocking
Request log delivery	Asynchronous; the gateway does not wait for the controller	Synchronous; the gateway waits up to `AUTHZ_TIMEOUT` (default 100 ms)
Latency impact	Minimal; requests are forwarded without waiting	Adds the controller's processing time to each logged request, bounded by `AUTHZ_TIMEOUT`
Request body handling	Streamed in chunks as it passes through; not held back	Buffered in memory so it can be inspected before forwarding
Blocking	None	Request is rejected when the controller returns a 4xx verdict
Response log delivery	Asynchronous	Asynchronous
Controller unavailable	Traffic unaffected; logs are not captured while unavailable	Governed by `FAILOPEN` (see Section 4.5)

Recommendation: Deploy in OOB mode first. Once API discovery and detections have been reviewed with the AppSentinels team, enable enforcement mode on the gateways where inline protection is required.

2.3 Request and Response Flow

Figure 2 shows the sequence of calls between the client, gateway, Edge Controller and upstream service. Steps 3 and 3a occur only in enforcement mode.

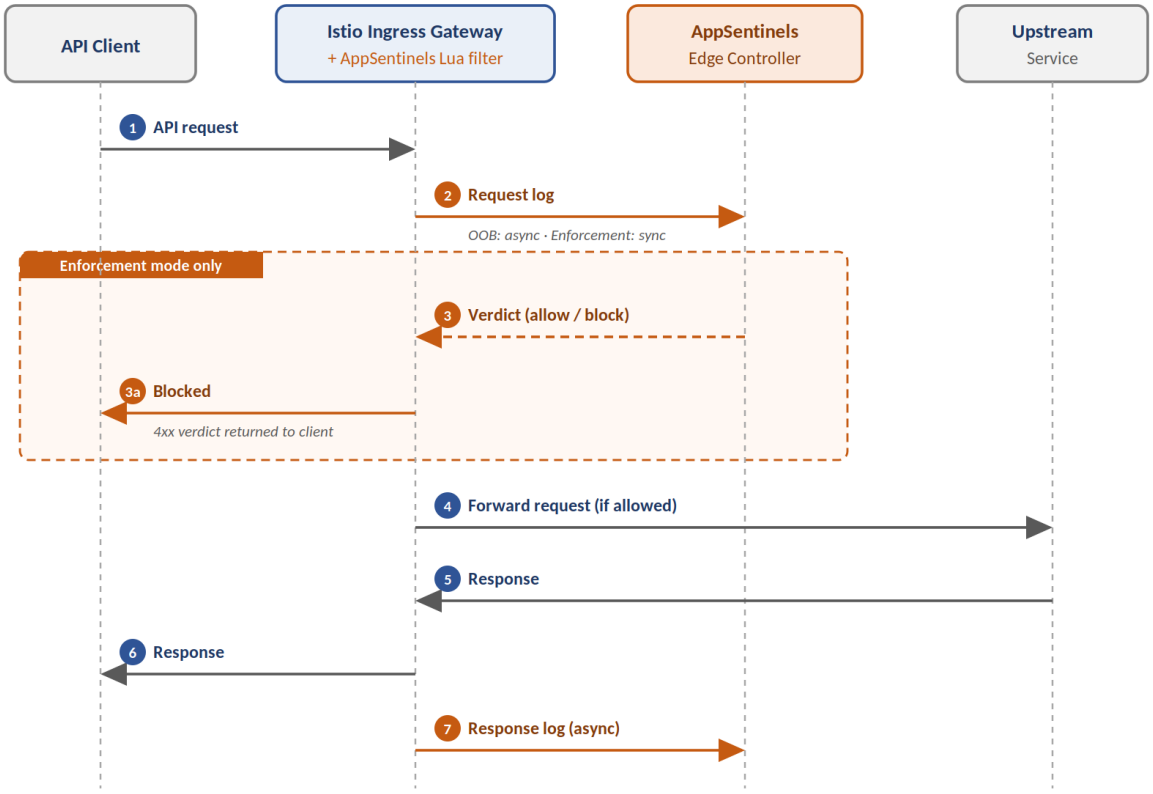

Figure 2 Request and response flow through the AppSentinels filter

2.4 What Traffic Is Captured

Item	Behavior
HTTP methods	All methods except `HEAD` and `OPTIONS`, which are skipped entirely (neither request nor response is logged). Configurable via `SKIP_METHODS`.
Headers	All request headers and all response headers, together with the method, path, authority (host) and status code.
Payloads	Captured when the content type contains `json`, `form`, `xml` or `graphql` (or no content type is set) and the body is smaller than 128 KB. Otherwise, only metadata is logged, and the log is flagged to indicate the body was not captured. Configurable via `LOGGING_CONTENT_TYPES` and `MAX_PAYLOAD_SIZE`.
Correlation	Request and response logs are correlated using the `x-request-id` header, which Istio gateways generate by default.
Filter placement	The filter runs after other gateway HTTP filters, such as Istio authentication and authorization. Requests rejected by those earlier filters (for example, an `AuthorizationPolicy` deny or a JWT validation failure) do not reach the AppSentinels filter and are generally not logged.

3. Prerequisites

3.1 Istio Environment

Requirement	Details
Istio gateway	Istio service mesh with an Envoy-based ingress gateway: either the classic `istio-ingressgateway` deployment or a gateway provisioned by Istio for the Kubernetes Gateway API.
Cluster access	`kubectl` access with permission to create and modify `EnvoyFilter` resources in `istio-system` (or your Istio root namespace, if customized).
Tools	`kubectl` (required) and `istioctl` (recommended, used for verification and debugging).
Version compatibility	Share the output of `istioctl version` with AppSentinels before deployment so that compatibility with your Istio and Envoy versions can be confirmed.

3.2 Edge Controller

This guide does not cover Edge Controller deployment. Before starting the integration, deploy the Edge Controller and confirm it is connected to the AppSentinels Cloud by following the AppSentinels Edge Controller Pre-Requisites & Deployment guide. For the Istio gateway integration, also ensure that:

- The controller exposes **TCP port 9002**, the port used by Envoy-based sensors.
- You have the controller's hostname or IP address, as reachable from the gateway pods. You will enter it in the filter (Section 4.1).
- If the controller runs inside the Istio service mesh, it accepts plaintext connections from the gateway on port 9002. The controller cluster defined in the filter does not use Istio mutual TLS.

3.3 Network Connectivity

Source	Destination	Port / Protocol	Purpose
Istio gateway pods	Edge Controller	TCP 9002, HTTP/2 (cleartext by default; TLS optional)	Request and response logs; enforcement verdicts

The controller hostname configured in the filter (Section 4.1) must be resolvable by DNS from the gateway pods. If Kubernetes `NetworkPolicy` resources restrict egress from the gateway namespace, allow egress to the controller on TCP 9002. Network requirements of the Edge Controller itself, such as access to the AppSentinels Cloud, are listed in the controller guide.

3.4 Information to Share with AppSentinels

- Output of `istioctl version`
- Gateway type (classic ingress gateway or Kubernetes Gateway API) and the gateways to be onboarded

- Gateway pod labels, from `kubectl get pods -n <gateway-namespace> --show-labels`
- Expected peak requests per second and typical request/response payload sizes
- Intended operating mode (OOB or enforcement)

4. Configure the AppSentinels Filter

The AppSentinels filter, including the complete Lua script, is published as a single YAML file: appsentinels-envoy-istio-gateway-filter.yaml. Download it to a machine with `kubectl` access and check its version:

```
BASE=https://downloads.appsentinels.ai/appsentinels-deployment/envoy-istio-gateway
curl -O ${BASE}/appsentinels-envoy-istio-gateway-filter.yaml

grep "# Version" appsentinels-envoy-istio-gateway-filter.yaml
```

This guide describes filter version 1.0.2. If the downloaded file shows a different version, contact AppSentinels before proceeding.

Open the file in a text editor and update the values below before applying it. Change only these values; the remaining Lua code should not be modified.

Value	Where in the file	Required	Section
Edge Controller hostname or IP address	The `address:` line under `socket_address:` in the first patch (`applyTo: CLUSTER`). Replace the placeholder `<IP/hostname of edge controller>`.	Yes	4.1
Edge Controller port	The `port_value:` line, directly below the address	Only if not 9002	4.1
Instance name	`INSTANCE_NAME` in the Lua configuration section	Recommended	4.2
Gateway scope	`workloadSelector` block, added under `spec:`	Recommended	4.3
Mode, timeouts and capture settings	Lua configuration section	Optional	4.4, 4.5

4.1 Set the Edge Controller Hostname or IP Address

The Edge Controller address is set in **one place only**: the `address` field under `socket_address`, inside the first configuration patch (`applyTo: CLUSTER`). In the file as supplied, this line contains the

placeholder `<IP/hostname of edge controller>` (line 52 in filter version 1.0.2). The highlighted line shows its location:

```
apiVersion: networking.istio.io/v1alpha3
kind: EnvoyFilter
metadata:
  name: request-response-filter
  namespace: istio-system
spec:
  configPatches:
  - applyTo: CLUSTER
    match:
      context: GATEWAY
    patch:
      operation: ADD
      value: # cluster specification
        name: "ext_authz-http-service"
        ...
        load_assignment:
          cluster_name: ext_authz-http-service
          endpoints:
          - lb_endpoints:
            - endpoint:
                address:
                  socket_address:
                    address: <IP/hostname of edge controller>
                    port_value: 9002
```

Replace the placeholder with the controller's hostname or IP address. For example, for a controller running as a Kubernetes Service, the lines become:

```
                  socket_address:
                    address: appsentinels-controller.appsentinels.svc.cluster.local
                    port_value: 9002
```

Use the value that matches where the Edge Controller runs:

Controller deployment	Value to enter	Example
Kubernetes Service in the same cluster	Service DNS name, in the form `<service>.<namespace>.svc.cluster.local`	`appsentinels-controller.appsentinels.svc.cluster.local`
VM or Docker host with a DNS name	Fully qualified hostname	`as-controller.example.com`
VM or Docker host without a DNS name	IP address	`10.20.30.40`
Multiple controllers behind a load balancer	Load balancer hostname or IP address	`as-controller-lb.example.com`

> **Format rules:** Enter only the hostname or IP address. Do not include a scheme (`http://` or `https://`), a port (`:9002`) or a path. The port is set separately in `port_value`.
>
> Keep `port_value: 9002` unless the controller is exposed on a different port.
>
> The hostname must resolve, and the address must be reachable, from the gateway pods (see Section 3.3).

Alternatively, replace the placeholder from the command line. Set `CONTROLLER` to your controller's hostname or IP address, run the replacement, and confirm the change:

```
CONTROLLER="appsentinels-controller.appsentinels.svc.cluster.local"

sed -i "s|<IP/hostname of edge controller>|${CONTROLLER}|" \
  appsentinels-envoy-istio-gateway-filter.yaml

grep -n -A2 "socket_address:" appsentinels-envoy-istio-gateway-filter.yaml
```

On macOS, use `sed -i ''` in place of `sed -i`. Do not rename the cluster (`ext_authz-http-service`) unless you also update `CONTROLLER_CLUSTER_NAME` in the Lua configuration to the same value.

4.2 Set the Instance Name

Set `INSTANCE_NAME` in the Lua configuration section to a meaningful name for this gateway. It identifies the traffic source in the AppSentinels Dashboard. Each gateway pod additionally reports its own hostname, so individual replicas can be distinguished.

```
local INSTANCE_NAME = "prod-ingress-gateway"
```

4.3 Target Specific Gateways

As supplied, the filter is created in `istio-system` without a workload selector. Istio applies an `EnvoyFilter` in its root namespace to all matching proxies in the mesh, so the filter will attach to **every** gateway, including egress gateways and any additional ingress gateways. We recommend limiting it to the intended gateway by adding a `workloadSelector` under `spec`.

For the classic Istio ingress gateway:

```
spec:
  workloadSelector:
    labels:
      istio: ingressgateway
  configPatches:
  ...
```

For a gateway provisioned by Istio for the Kubernetes Gateway API:

```
spec:
  workloadSelector:
    labels:
      gateway.networking.k8s.io/gateway-name: <gateway-name>
  configPatches:
  ...
```


Confirm the labels on your gateway pods with `kubectl get pods -n <gateway-namespace> --show-labels` before applying. To onboard multiple gateways with different settings (for example, different instance names or modes), create one `EnvoyFilter` per gateway, each with a unique `metadata.name` and its own `workloadSelector`.

4.4 Configuration Reference

The Lua configuration section appears near the top of the `inline_code` block:

```
-- Needed for visibility, pls update this to your filter instance name
local INSTANCE_NAME = "unknown"
-- End of visibility

-- Config section start
local ENFORCEMENT = false
local AUTHZ_TIMEOUT = 100
local ACCESSLOG_TIMEOUT = 100
local FAILOPEN = true
local CONTROLLER_CLUSTER_NAME = "ext_authz-http-service"
local SKIP_METHODS = {"HEAD", "OPTIONS"}
local LOGGING_CONTENT_TYPES = {"json", "form", "xml", "graphql"}
local MAX_PAYLOAD_SIZE = 131072  --- 128k
local AS_DEBUG = false
-- Config section end
```

Parameter	Default	Description
`INSTANCE_NAME`	`"unknown"`	Name of this gateway integration, shown in AppSentinels for visibility. Set a descriptive value.
`ENFORCEMENT`	`false`	`false` runs in OOB mode; `true` runs in enforcement (inline) mode.
`AUTHZ_TIMEOUT`	`100`	Timeout in milliseconds for sending the request log. In enforcement mode, this is the maximum time the gateway waits for a verdict.
`ACCESSLOG_TIMEOUT`	`100`	Timeout in milliseconds for sending the response log.
`FAILOPEN`	`true`	Enforcement mode only. Controls behavior when the controller returns a server error, times out or is unreachable (see Section 4.5).
`CONTROLLER_CLUSTER_NAME`	`"ext_authz-http-service"`	Must match the cluster name defined in the `CLUSTER` patch.
`SKIP_METHODS`	`{"HEAD", "OPTIONS"}`	HTTP methods excluded from logging and enforcement.
`LOGGING_CONTENT_TYPES`	`{"json", "form", "xml", "graphql"}`	Content-type substrings for which payloads are captured. Metadata is logged for all content types.
`MAX_PAYLOAD_SIZE`	`131072` (128 KB)	Maximum body size, in bytes, captured per request and per response. Larger bodies are not captured; metadata is still logged.
`AS_DEBUG`	`false`	Writes diagnostic messages to the gateway proxy log. Keep disabled in production (see Section 7.1).

After any change, re-apply the `EnvoyFilter`. Istio pushes the updated configuration to the gateway pods dynamically.

4.5 Enabling Enforcement Mode

1. Validate the deployment in OOB mode (Section 6) and confirm with AppSentinels that detections are ready for enforcement.
2. Set `ENFORCEMENT = true`.
3. Choose the failure behavior with `FAILOPEN`, using the table below.
4. Review `AUTHZ_TIMEOUT` against your latency budget. The default is 100 ms.
5. Re-apply the `EnvoyFilter` and verify (Section 6).

Edge Controller outcome	FAILOPEN = true (default)	FAILOPEN = false
2xx (allow)	Request forwarded to the service	Request forwarded to the service
4xx (block)	Request blocked; controller response returned to the client	Request blocked; controller response returned to the client
5xx, timeout or controller unreachable	Request forwarded (fail-open)	Request rejected with an error status (fail-closed)

Recommendation: Use `FAILOPEN = true` in production so that controller maintenance or network disruption does not affect application availability.

Request body buffering: In enforcement mode, the gateway buffers the request body in memory before contacting the controller so that it can be inspected. This increases gateway memory use per in-flight request, and very large request bodies that exceed Envoy's buffer limits may be rejected by Envoy (typically with HTTP 413). Review this for upload-heavy APIs before enabling enforcement.

4.6 Secure Logging over HTTPS (Optional)

By default, traffic from the gateway to the controller is cleartext HTTP/2. This is suitable when the controller runs in the same cluster or on a trusted private network. When this traffic crosses network boundaries, enable TLS:

1. Enable HTTPS on the Edge Controller as described under "Secure Logging" in the AppSentinels Edge Controller Pre-Requisites & Deployment guide.

2. Add a `transport_socket` to the cluster definition in the `CLUSTER` patch, at the same level as `load_assignment`:

```
transport_socket:
  name: envoy.transport_sockets.tls
  typed_config:
    "@type": type.googleapis.com/envoy.extensions.transport_sockets.tls.v3.UpstreamTlsContext
    sni: <controller hostname>
    common_tls_context:
      alpn_protocols: ["h2"]
      validation_context:
        trusted_ca:
          filename: <path to CA bundle in the gateway pod>
```

3. Set `sni` to the controller hostname configured in Section 4.1. Set `trusted_ca` to a CA bundle that can validate the controller certificate. For certificates issued by a private CA, mount the CA bundle into the gateway pods and reference its path. Omitting `validation_context` disables certificate verification and is not recommended.
4. Re-apply the `EnvoyFilter` and verify connectivity (Section 6.1).

5. Deploy the Filter

1. Apply the configured filter:

```
kubectl apply -f appsentinels-envoy-istio-gateway-filter.yaml
```

2. Confirm the resource was created:

```
kubectl get envoyfilter request-response-filter -n istio-system
```

3. Confirm the applied controller address is your controller's hostname or IP address, not the placeholder:

```
kubectl get envoyfilter request-response-filter -n istio-system -o yaml | grep -A2
socket_address
```

4. Identify a gateway pod (adjust the namespace and label for your gateway):

```
kubectl get pods -n istio-system -l istio=ingressgateway
```

5. Confirm the gateway is synchronized with the latest Istio configuration (status `SYNCED`):

```
istioctl proxy-status
```

6. Confirm the controller cluster is present on the gateway:

```
istioctl proxy-config cluster <gateway-pod> -n istio-system | grep ext_authz-http-service
```

7. Confirm the Lua filter is present in the gateway listeners (a non-zero count is expected):

```
istioctl proxy-config listener <gateway-pod> -n istio-system -o json | grep -c
"envoy.filters.http.lua"
```

Note: No gateway restart is required. Istio delivers the configuration to gateway pods dynamically. Listener updates cause existing long-lived connections to be drained gracefully, so apply changes during an appropriate change window.

6. Verify Deployment

6.1 Gateway Connectivity to the Edge Controller

Generate some API traffic through the gateway, then inspect the controller cluster's endpoint statistics from the gateway pod:

```
kubectl exec -n istio-system <gateway-pod> -c istio-proxy -- \
  pilot-agent request GET clusters | grep ext_authz-http-service
```

Check the output for the following:

- The controller's resolved IP address and port 9002 are listed
- `rq_total` increases as traffic flows through the gateway
- `cx_connect_fail` and `rq_timeout` remain at or near zero

6.2 Traffic at the Edge Controller

On the Edge Controller host, confirm that packets from the gateway are arriving on port 9002:

```
tcpdump -i <interface, e.g. eth0 or ens3> -nn 'tcp port 9002'
```

Because the gateway sends logs over HTTP/2, the payload is binary-encoded. Use `tcpdump` to confirm connectivity rather than to read log content. For controller-side health checks, see "Verify Deployment" in the controller guide.

6.3 AppSentinels Dashboard

- The Edge Controller deployed in your environment is listed on the **System Health** page.
- After traffic is generated through the gateway, discovered APIs appear in the **API Catalogue**.

7. Troubleshooting & Debugging

7.1 Enable Filter Debug Logs

1. Set `AS_DEBUG = true` in the filter configuration and re-apply the `EnvoyFilter`.

2. Raise the Lua log level on the gateway pod:

```
istioctl proxy-config log <gateway-pod> -n istio-system --level lua:debug
```

3. View AppSentinels filter messages, which are prefixed with `as:`:

```
kubectl logs -n istio-system <gateway-pod> -c istio-proxy | grep "as:"
```

4. When finished, set `AS_DEBUG = false`, re-apply, and restore the log level:

```
istioctl proxy-config log <gateway-pod> -n istio-system --level lua:warning
```

7.2 Common Issues

Symptom	Likely cause	Resolution
`ext_authz-http-service` cluster not present on the gateway	Filter not applied, created in the wrong namespace, or `workloadSelector` labels do not match the gateway pods	Check `kubectl get envoyfilter -A`; compare selector labels with the gateway pod labels; run `istioctl analyze`.
Gateway log shows Lua or configuration errors after editing	YAML indentation or Lua syntax error introduced during editing	Re-check the edited file against the original; change only the documented values.
No controller IP address in the Section 6.1 output, or no logs reach the controller	Placeholder `<IP/hostname of edge controller>` not replaced, or the hostname does not resolve from the gateway pods	Check the applied value with `kubectl get envoyfilter request-response-filter -n istio-system -o yaml | grep -A2 socket_address` and correct it as described in Section 4.1.
`cx_connect_fail` increasing	Controller unreachable: DNS resolution, firewall or security group, `NetworkPolicy`, or port 9002 not exposed	Confirm the controller hostname resolves from the cluster, the controller is listening on 9002, and network policies allow the traffic.
Connections to an in-mesh controller are reset	STRICT mTLS enforced on the controller pods	Run the controller without Istio sidecar injection, or allow plaintext (PERMISSIVE mTLS) on port 9002 for the controller pods.
`rq_timeout` increasing	Controller overloaded, high network latency, or timeouts set too low	Check controller resources against the sizing guidance in the controller guide; review network latency; adjust `AUTHZ_TIMEOUT` / `ACCESSLOG_TIMEOUT` with AppSentinels guidance.
Logs reach the controller, but no APIs appear on the Dashboard	Controller not connected to the AppSentinels Cloud	Check the System Health page and the controller's cloud connectivity, as described in the controller guide.
APIs visible, but payloads missing	Content type not in `LOGGING_CONTENT_TYPES`, or body larger than `MAX_PAYLOAD_SIZE`	Expected behavior. Adjust these parameters with AppSentinels if required.

Symptom	Likely cause	Resolution
Some requests not logged	`HEAD`/`OPTIONS` requests, or requests rejected by an earlier gateway filter	Expected behavior (Section 2.4).
Request and response not correlated	`x-request-id` generation disabled or the header stripped at the gateway	Ensure the gateway generates `x-request-id` (Istio default).
Higher latency after enabling enforcement	Gateway waits synchronously for the controller verdict	Expected, bounded by `AUTHZ_TIMEOUT`. Review controller sizing; keep latency-critical gateways in OOB mode if needed.
HTTP 413 on large uploads after enabling enforcement	Request body buffering exceeds Envoy buffer limits	Review gateway buffer limits for these APIs, or keep the affected gateway in OOB mode.

8. Operational Considerations

8.1 Performance and Resource Usage

- In OOB mode, requests are forwarded without waiting for the controller, and logs are sent asynchronously.
- Each request through the gateway results in up to two log calls to the controller (request and response), sent over persistent HTTP/2 connections.
- The filter holds a copy of captured payloads (up to `MAX_PAYLOAD_SIZE` per request and per response) while each log is assembled. Account for this in gateway memory limits for high-concurrency, large-payload APIs.
- In enforcement mode, each logged request additionally waits for the controller verdict, bounded by `AUTHZ_TIMEOUT`.

8.2 Data Handling and Security

- The filter forwards request and response headers (including authorization headers and cookies) and captured payloads to the Edge Controller, which runs in your environment. Restrict network access to controller port 9002 to the gateway pods.
- Enable HTTPS (Section 4.6) when gateway-to-controller traffic traverses untrusted networks.
- The Edge Controller sends security metadata to the AppSentinels Cloud over HTTPS.
- Restrict Kubernetes RBAC for `EnvoyFilter` resources in `istio-system`, since these resources can change gateway behavior.

9. Rollback, Upgrades & Maintenance

9.1 Rollback

To remove the integration completely:

```
kubectl delete envoyfilter request-response-filter -n istio-system
```

Istio removes the filter and controller cluster from the gateways dynamically, and traffic continues unaffected. To disable only inline blocking, set `ENFORCEMENT = false` and re-apply.

9.2 Filter Updates

Updated filter versions are published at the same download location (Section 1.1). The version is shown in the `# Version:` line of the file header; this guide describes version 1.0.2. Download the new file, carry over your environment-specific values (controller address, `INSTANCE_NAME`, mode settings, `workloadSelector` and any TLS settings), then apply it.

9.3 Istio Upgrades

- `EnvoyFilter` patches depend on Envoy's internal configuration structure, which Istio does not guarantee to remain compatible across releases. Validate the filter in a non-production environment before upgrading production gateways.
- After an upgrade, repeat the checks in Sections 5 and 6.
- Share the target Istio version with AppSentinels ahead of production upgrades.
- For revision-based (canary) upgrades, confirm the filter is applied to the new revision's gateway pods before shifting traffic.

Appendix A – Log Metadata Added by the Filter

In addition to the original request and response headers, the filter adds the following headers to the logs it sends to the Edge Controller. These headers are sent only to the controller; they are not added to traffic forwarded to your services or returned to clients.

Header	Sent with	Description
`X-As-Req-Timestamp`	Request and response logs	Time (milliseconds) the request was received at the gateway
`X-As-Resp-Timestamp`	Response log	Time (milliseconds) the response was received at the gateway
`X-As-Status`	Response log	HTTP status code of the response
`X-As-Sensor`	Response log	Sensor type identifier (`envoy-http-access`)
`X-As-Sensor-Instance`	Response log	Value of `INSTANCE_NAME`
`X-As-Sensor-Hostname`	Response log	Hostname of the gateway pod

Header	Sent with	Description
`x-request-id`	Request and response logs	Correlates the request and response
`x-as-req-truncated-body` `x-as-resp-truncated-body`	Request / response log	Set when the body was not captured due to size or content type
`x-as-content-length`	Request / response log	Original body size, when the body was not captured
`X-As-TE`	Request and response logs	Original `transfer-encoding` value
`X-As-Error`	Request and response logs	Diagnostic list of missing attributes (for example, a missing request ID, path or host)

The path in the response log also carries the query parameters `sensor=envoy-lua` and `cap=nohb`, which identify the sensor type to the Edge Controller.

Appendix B – Command Quick Reference

Task	Command
Download the filter	`curl -O <filter URL from Section 1.1>`
Apply the filter	`kubectl apply -f appsentinels-envoy-istio-gateway-filter.yaml`
List EnvoyFilters	`kubectl get envoyfilter -A`
Check configured controller address	`kubectl get envoyfilter request-response-filter -n istio-system -o yaml | grep -A2 socket_address`
Check gateway config sync	`istioctl proxy-status`
Check controller cluster	`istioctl proxy-config cluster <gateway-pod> -n istio-system | grep ext_authz-http-service`
Check Lua filter in listeners	`istioctl proxy-config listener <gateway-pod> -n istio-system -o json | grep -c envoy.filters.http.lua`
Controller endpoint statistics	`kubectl exec -n istio-system <gateway-pod> -c istio-proxy -- pilot-agent request GET clusters | grep ext_authz-http-service`
Enable Lua debug logging	`istioctl proxy-config log <gateway-pod> -n istio-system --level lua:debug`
View filter logs	`kubectl logs -n istio-system <gateway-pod> -c istio-proxy | grep "as:"`
Analyze Istio configuration	`istioctl analyze -n istio-system`
Remove the filter	`kubectl delete envoyfilter request-response-filter -n istio-system`