

Multi-tenant management in HP OneCloud Platform

John Zheng/ john.zheng@hp.com



#IstioCon

Agenda

- HP OneCloud Platform
- API Rate Limit per tenant
- Authentication and Authorization for tenant
- Observability Enhancement for tenant
- Q & A



HP OneCloud Platform designed with Istio

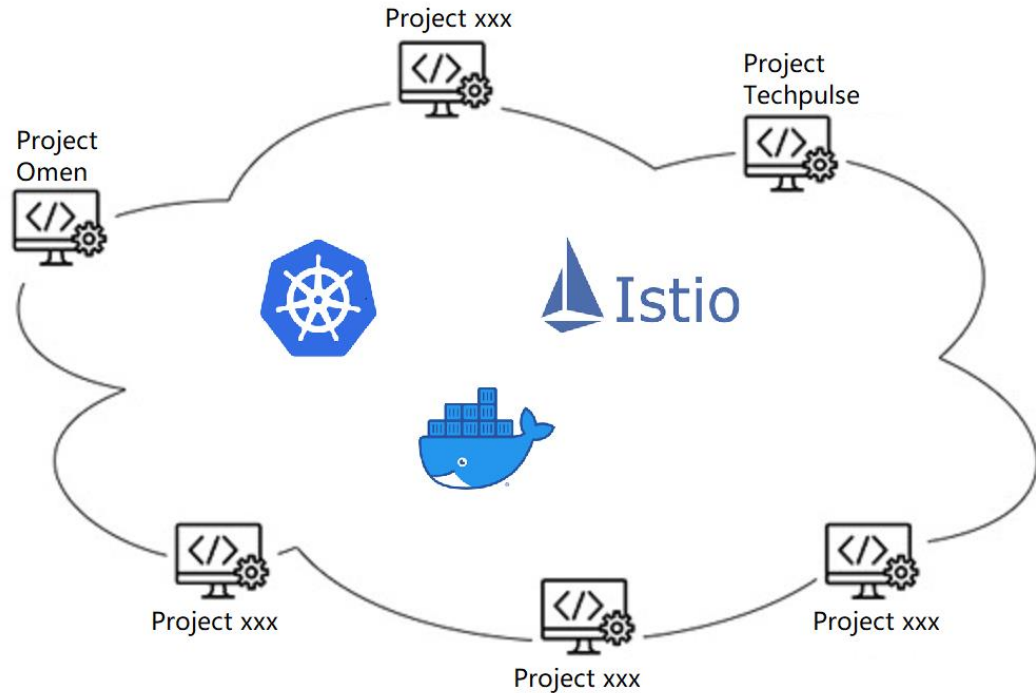
#IstioCon



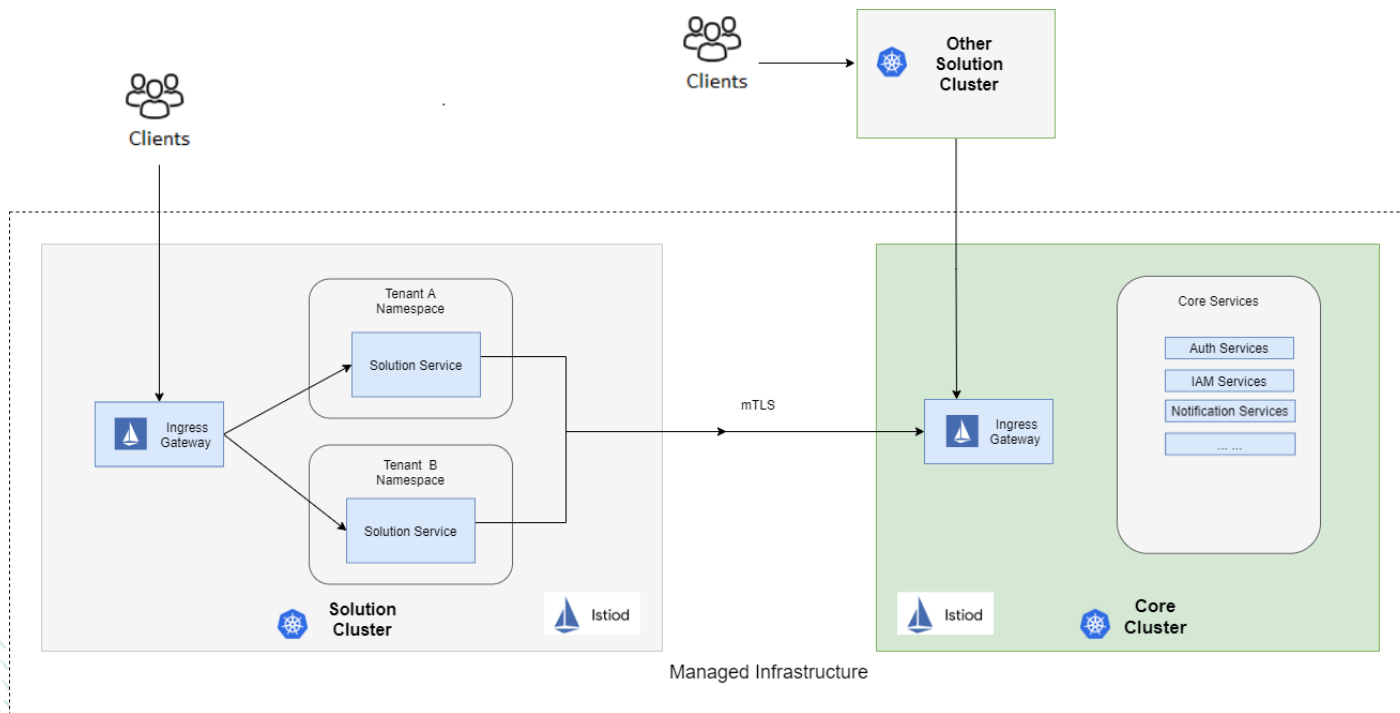
HP OneCloud Platform

HP has lots of projects, deployed on cloud. They have common features, also have project specified feature.

HP OneCloud platform is HP's largest cloud platform and is recognized as **HP's only official cloud platform**, includes all common features, connect all projects.



HP OneCloud Platform Connect With Istio



Common services are in core cluster

Projects shared solution cluster

- Different namespace
- Project runs as tenant, need control rights

Some standalone cluster without Istio can access core cluster also, as tenant.

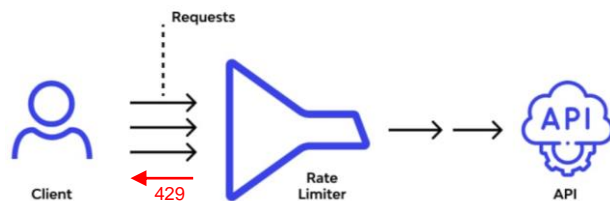


API Rate Limit per tenant

#IstioCon



Rate Limit Requirements



- Security Concern

- For each client IP, limit the number of requests per minute.
- For some IP from known partner cluster, we think it is securer, do not limit on this IP.
- All projects need to apply this security purpose ratelimit.

- Business Concern

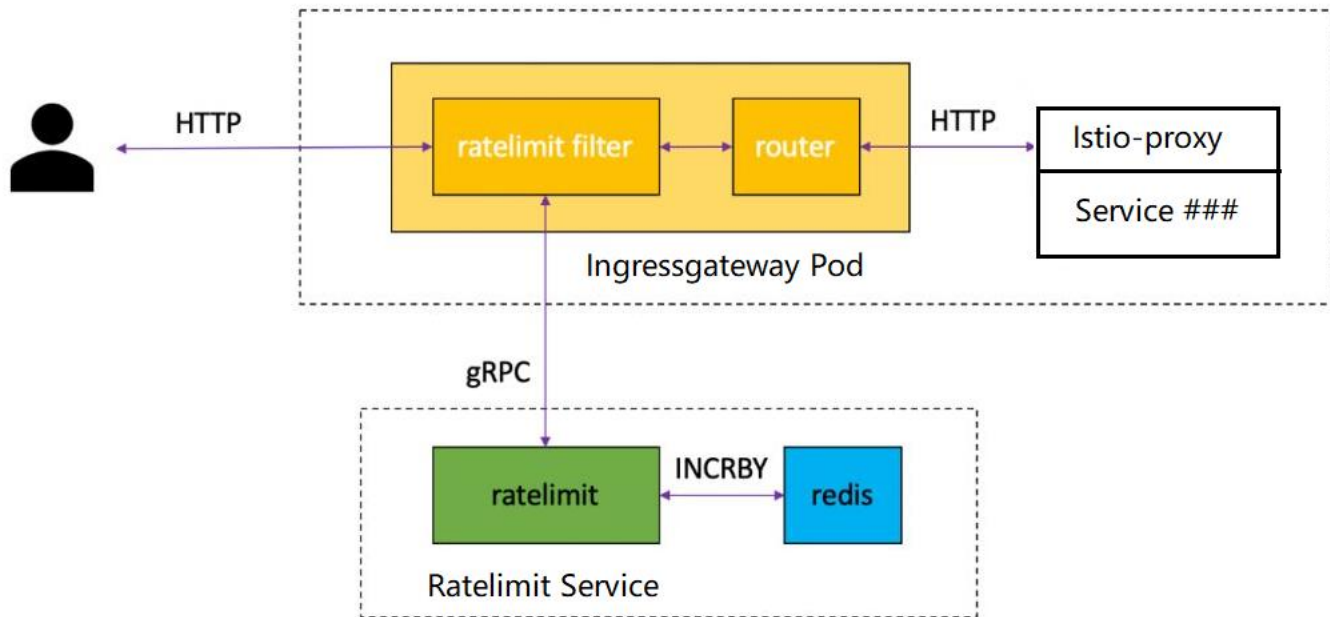
- For tenant who has paid, for each api, limit the number of requests per minute to a high value.
- For tenant who has not paid, for each api, limit the number of requests per minute to a low value.
- Project owner can choose whether project api apply business purpose ratelimit or not.



Envoy Solution for rate limit

Envoy can be used to [set up global rate limits](#) for your mesh.

Global rate limiting in Envoy uses a gRPC API for requesting quota from a rate limiting service



Envoy Rate limit

Quick Start

Go through <https://istio.io/latest/docs/tasks/policy-enforcement/rate-limit/>

Further Practice

Go through <https://github.com/johnzheng1975/istio-ratelimit-example>

Best Practice as HP OneCloud

Go through <https://github.com/johnzheng1975/istiocon2023/tree/main/samples/ratelimit>

- Use a pre-handler envoyfilter, handle the complex logic.
- Generate additional request headers in the pre-handler envoyfilter.
- Rate limit is based on these new generated request headers.



Design for Security Purpose Rate limit

- Get client IP from request header `x-forwarded-for` .
- Add this client IP as new request header `ratelimit-source-ip` .
- Define a list of partner cluster IP, if client IP is in this list, do not limit, set new request header `ratelimit-enabled-secure` with value `false` . Otherwise, set `ratelimit-enabled-secure` with value `true` .
- In ratelimit envoyfilter, based on `ratelimit-enabled-secure` , `ratelimit-source-ip` , configured as below

```
- key: header_match
  value: oc-ratelimit-enabled-secure # enable_ratelimit_secure
  descriptors:
    - key: CLIENTIP                  # ratelimit-source-ip
      rate_limit:
        unit: minute
        requests_per_unit: 12      # number here is for testing purpose
```



Design for Business Purpose Rate limit

› In ratelimit envoyfilter, based on `ratelimit-enabled-business`, `ratelimit-project`, `ratelimit-path`, `:method`, `ratelimit-tenantid`, `ratelimit-level` configured as below

```
descriptors:
- key: header_match
  value: oc-ratelimit-enabled-business          # ratelimit-enabled-business
  descriptors:
  - key: PROJECT                               # ratelimit-project
    descriptors:
    - key: PATH                                # ratelimit-path
      descriptors:
      - key: METHOD                             # header :method
        descriptors:
        - key: TENANTID                        # ratelimit-tenantid
          descriptors:
          - key: TLEVEL                         # ratelimit-level
            value: premium
            rate_limit:
              unit: minute
              requests_per_unit: 8              # number here is for testing purpose
        - key: TLEVEL
          value: trial
          rate_limit:
            unit: minute
            requests_per_unit: 4                # number here is for testing purpose
```



Good Code Structure

- Include below code:

```
- 00-namespace-ratelimit.yaml          # For namespace creation
- 10-configmap-ratelimit.yaml          # For configmap for ratelimit
- 20-service-ratelimit-redis.yaml      # For ratelimit deploy creation, redis creation.
- 30-envoyfilter-ratelimit.yaml        # For ratelimit envoyfilter
- 40-envoyfilter-ratelimit-svc.yaml    # For ratelimit envoyfilter
- 50-ef-ratelimit-pre-handle.yaml      # The pre-handler envoyfilter, make logic simple
```

- This code structure is simple, easy to maintain.
 - 00, 10, 20, 30, 40 need not change.
 - Only some variable in 50-ef-ratelimit-pre-handle.yaml need be changed, based on requirements change, as below:

```
-- Define a list of partner cluster IP, if client IP is in this list, do not enable secure limit
secure_platform_ips = { ["54.148.85.56"]=true, ["3.120.217.35"]=true, ["15.65.244.13"]=true, ["192.168.1.61"]=true }

-- Assume only three projects enabled ratelimit, includes project1, project3, project5
project_enabled_ratelimit = { ["project1"]=true, ["project3"]=true, ["project5"]=true }

-- Assume tenant01 and tenant02 are premium tenant, the others are trial tenant
premium_tenants = { ["tenant01"]=true, ["tenant03"]=true }
```



Apply & Test

Apply:

```
$ kubectl apply -f ./code
namespace/ratelimit created
configmap/ratelimit-config created
service/redis created
deployment.apps/redis created
service/ratelimit created
deployment.apps/ratelimit created
envoyfilter.networking.istio.io/filter-ratelimit created
envoyfilter.networking.istio.io/filter-ratelimit-svc created
envoyfilter.networking.istio.io/ef-ratelimit-pre-handle created
```

Test:

Test Scenario 01: Business ratelimit for premium tenants

Test Scenario 02: Business ratelimit for trial tenants

Test Scenario 03: Secure ratelimit only

Test Scenario 04: Both Secure ratelimit and Business ratelimit are disabled

Test Scenario 05: Both Secure ratelimit and Business ratelimit are enabled

*Test Result is
expected!*



Sample

Introduction for tutorial:

- <https://github.com/johnzheng1975/istiocon2023/blob/main/samples/ratelimit/readme.md>

Sample Code:

- <https://github.com/johnzheng1975/istiocon2023/tree/main/samples/ratelimit/code>

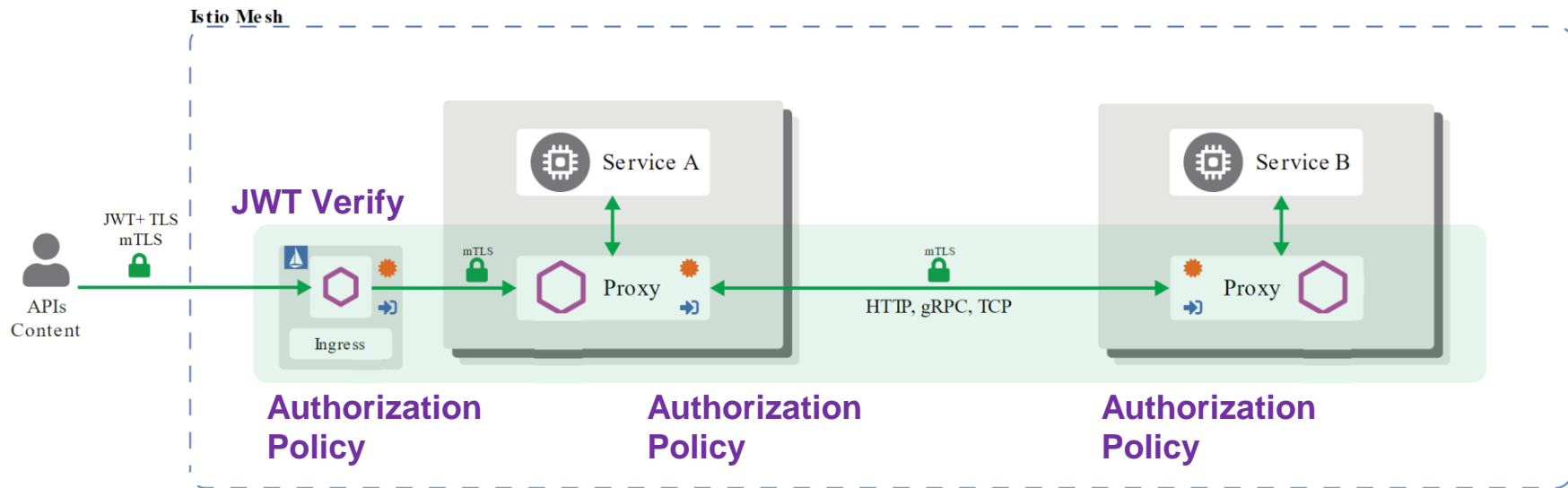


Authentication and Authorization for tenant

#IstioCon



Authentication and Authorization



Version 1

Authentication:

- Each tenant has its own JWKS file.
- JWT Verify is implemented with Istio [Request Authentication](#).

Authorization:

- Service-to-service Authorization is using [Authorization Policy](#).
- Based on
 - Method
 - path
 - request.auth.claims[tenant_id]
 - request.auth.claims[type]
 - source principals
 -

API Scope/ Role-based authorization :

- Invoke authz service with envoyfilter ext-authz



Version 1

Advantage:

- Both Request Authentication/ Authorization Policy are convenience, need not coding
- Both Request Authentication/ Authorization Policy are powerful, can handle all user cases

Disadvantage:

- When projects more and more, Authorization Policy/ Request Authentication are huge and difficult to manual maintain for DevOps.
- Authorization Policy logic is complex, easy to typo or error configuration



Version 1 Improvement

Further Solution for automation:

- We developed a tool called “Developer Portal”, let developer to configure on UI.
- “Developer Portal” generate request authentication/ authorization policy with Istio [client-go](#) lib ([example](#))

Further Disadvantage:

- When Istio upgrade, need upgrade both Cluster and “Developer Portal”. It brings incompatible since their upgradation time is different.
- Version control is not as good as github.



Version 2

Authentication:

- Each tenant has its own JWKS file.
- JWT Verify is implemented with Istio [Request Authentication](#).



Keep this. Maintained with *Helm Charts + Flux*

Authorization:

- Service-to-service Authorization is using [Authorization Policy](#).
- Based on
 - source principals
 - method
 - path
 - `request.auth.claims[tenant_id]`
 - `request.auth.claims[type]`
 -



Removed this.

Moved all validation to Authz Service, include:

- Old Authorization policy
- Token Blacklist Verify
- API Scope
- Role-based author
-

API Scope/ Role-based authorization :

- Invoke `authz` service with `envoyfilter ext-authz`

Using “External Authorization” to trigger validate.



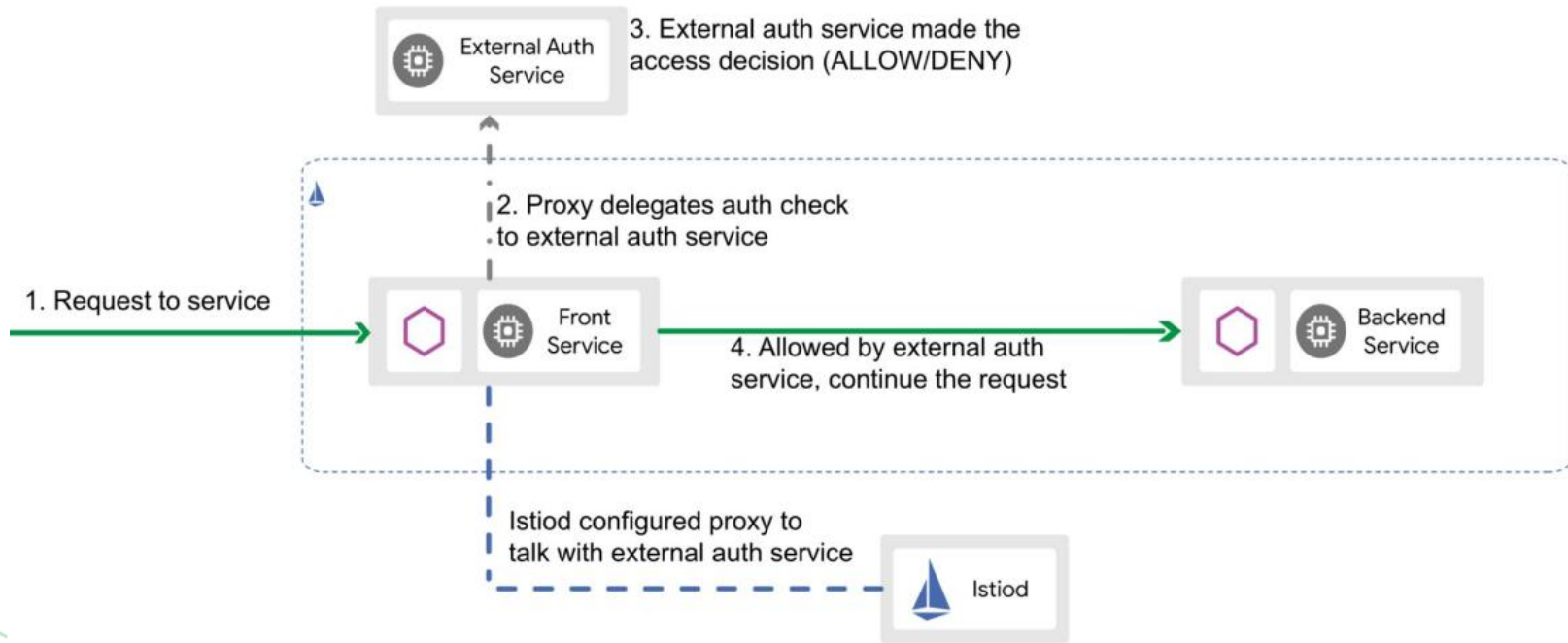
Version 2

Advantage:

- Do not use *Developer Portal* with Istio [client-go](#) , thus no upgradation incompatible issue.
- Use *Helm Charts template + Flux* for request authentication automation, flexible and less error-prone.
- **Put all authorization logic into single check method of Authz service**, logic is simpler.
- Use "Istio External Authorization" instead of "Envoy ext-auth envoyfilter"
 - Easier usage and simpler troubleshooting
 - Triggering the ext-authz flow conditionally






Istio External Authorization



Samples & Best Practice

Envoyfilter Ext-authz

AuthenticationAndAuthorization > ExtAuthz-Envoyfilter

<input type="checkbox"/> Name	Date modified
 envoyfilter-ext-authz-httpservice.yaml	9/17/2023 5:09 PM
 envoyfilter-ext-authz-simple.yaml	9/17/2023 5:09 PM
 readme.md	9/17/2023 9:49 PM

 readme.md

This sample is about ext-authz envoyfilter usage.

Before you begin

Before you begin this task, do the following:




- Read the [Istio authorization concepts](#).
- Follow the [Istio installation guide](#) to install Istio.
- Deploy test workloads:

This task uses two workloads, `httpbin` and `sleep`, both deploy in the `foo` namespace and workloads with the following commands:

```
$ kubectl create ns foo
$ kubectl label ns foo istio-injection=enabled
$ kubectl apply -f samples/httpbin/httpbin.yaml -n foo # und
$ kubectl apply -f samples/sleep/sleep.yaml -n foo # und
```

Istio External Authorization

AuthenticationAndAuthorization > ExtAuthz-AuthzCustom

<input type="checkbox"/> Name	Date modified
 Better External Authorization.docx	9/16/2023 6:12 PM
 BetterExternalAuthorization-YangminZhu...	9/16/2023 6:01 PM
 readme.md	9/20/2023 10:41 AM

This sample is about External Authorization usage.

Quick Start

- Please refer to: <https://istio.io/latest/docs/tasks/security/authorization/authz-custom/>

Introduction from its developer

- Video
 - <https://www.youtube.com/watch?v=B5mckzCAuk8>
- PDF
 - <https://events.istio.io/istiocon-2021/slides/g8p-BetterExternalAuthorization-YangminZhu.pdf>
- Design docs
 - <https://docs.google.com/document/d/1V4mCQCw7mlGp0zSQXYoBdbKMDnkPOjeyUb85U07ISj>

Best Practice

Here are some thoughts/ tips for ext-authz envoyfilter:

1. Comparing with ext-authz envoyfilter, it can support ext-authz flow conditionally, enable/disable for a specific path/host/IP/etc.

#IstioCon



Observability Enhance for tenant

Customize Access Logs

#IstioCon



Access Log

Istio(envoy) can generate access logs for service traffic in a configurable set of formats.

- We know API count/ latency/ error from access log
- We monitor and set alert based on access log

```
[2019-08-06T16:30:11.746Z] GET /backend/debug/ HTTP/1.1 200 - 0 1124 0 0 192.168.28.113 curl/7.63.0 6e9e455f-fe18-4511-9c52-aa517af0edff a70851b76b86511e9b8c60ebd9abcaa2-807989241.us-east-1.elb.amazonaws.com 10.100.216.61:8080
```



Access Log Customization

Requirements:

1. Add tenant_id for each API log
 - Calculate API usage for each tenant, to know how much each tenant should pay.
 - Before deprecating the old version of the API, you can know which tenants are still using the old version of the API and notify them to use the new version.
 - Know impact to tenant for each accident.
2. Know grpc status since half of our services are grpc services.
 - Show grpc status in access log, to know grpc api error rate, trigger alert in case error rate is high.



Access Log Customization

Step 1: Create an envoyfilter, get tenant_id from token, insert it into http header. [[code](#)]

Step 2: `kubectl edit configmap -n Istio-system istio`

```
apiVersion: v1
data:
  mesh: |-
    accessLogEncoding: JSON
    accessLogFile: /dev/stdout
    accessLogFormat: |
      {
        "authority": "%REQ(:AUTHORITY)%",
        "bytes_received": "%BYTES_RECEIVED%",
        "bytes_sent": "%BYTES_SENT%",
        "connection_termination_details": "%CONNECTION_TERMINATION_DETAILS%",
        "downstream_local_address": "%DOWNSTREAM_LOCAL_ADDRESS%",
        "downstream_remote_address": "%DOWNSTREAM_REMOTE_ADDRESS%",
        "duration": "%DURATION%",
        "method": "%REQ(:METHOD)%",
        "path": "%REQ(X-ENVOY-ORIGINAL-PATH?:PATH)%",
        "protocol": "%PROTOCOL%",
        "request_id": "%REQ(X-REQUEST-ID)%",
        "requested_server_name": "%REQUESTED_SERVER_NAME%",
        "response_code": "%RESPONSE_CODE%",
        "response_code_details": "%RESPONSE_CODE_DETAILS%",
        "response_flags": "%RESPONSE_FLAGS%",
        "route_name": "%ROUTE_NAME%",
        "start_time": "%START_TIME%",
        "upstream_cluster": "%UPSTREAM_CLUSTER%",
        "upstream_host": "%UPSTREAM_HOST%",
        "upstream_local_address": "%UPSTREAM_LOCAL_ADDRESS%",
        "upstream_service_time": "%RESP(X-ENVOY-UPSTREAM-SERVICE-TIME)%",
        "upstream_transport_failure_reason": "%UPSTREAM_TRANSPORT_FAILURE_REASON%",
        "user_agent": "%REQ(USER-AGENT)%",
        "x_forwarded_for": "%REQ(X-FORWARDED-FOR)%",
        "grpc_status_number": "%GRPC_STATUS_NUMBER%",
        "tenant_id": "%REQ(TENANT_ID)%",
      }
  defaultConfig:
    discoveryAddress: istiod.istio-system.svc:15012
```

#IstioOps



Access Log Customization

Test:

```
$ curl -I $gateway_url/productpage?id=13570 -H "Authorization: Bearer $token"
```

Token:





PAYLOAD: DATA

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "tenant_id": "tenant0002_fromtoken",
  "iat": 1516239022
}
```

Logs:

```
$ kubectl logs -n istio-system istio-ingressgateway-7485484874-gzz9s | grep "id=13570" | grep "{" | jq .
{
  "user_agent": "curl/7.68.0",
  "response_code": 200,
  "upstream_transport_failure_reason": null,
  "authority": "a37a73937493a427db222f8deecd65ca-1310205472.us-east-1.elb.amazonaws.com",
  "protocol": "HTTP/1.1",
  "upstream_host": "192.168.30.230:9080",
  "x_forwarded_for": "192.168.1.61",
  "start_time": "2023-09-21T16:21:02.809Z",
  "connection_termination_details": null,
  "response_flags": "-",
  "tenant_id": "tenant0002_fromtoken",
  "upstream_local_address": "192.168.48.162:37810",
  "route_name": null,
  "request_id": "19f38af1-b668-4837-9448-8f49fb57c52a",
  "upstream_service_time": "27",
  "downstream_remote_address": "192.168.1.61:54975",
  "bytes_sent": 0,
  "grpc_status_number": 2,
  "bytes_received": 0,
  "upstream_cluster": "outbound|9080||productpage.default.svc.cluster.local",
  "method": "HEAD",
  "duration": 29,
  "requested_server_name": null,
  "response_code_details": "via_upstream",
  "path": "/productpage?id=13570",
  "downstream_local_address": "192.168.48.162:8080"
}
```

Samples Code

samples > customizeLogs		Search customizeLogs
<input type="checkbox"/> Name	Date modified	
 ef-sample-add-tenant_id.yaml	9/21/2023 3:20 PM	
 readme.md	9/21/2023 11:55 PM	

This sample is about customize logs, adding tenant_id, grpc_status.

Before you begin

1. Setup Istio in a Kubernetes cluster by following the instructions in the [Installation Guide](#).
2. Deploy the [Bookinfo](#) sample application.
3. Had better remove envoyfilter from ratelimit sample, to make logs clear.

Customize Access Logs For tenant

1. Create envoyfilter to generate tenant_id from token.

```
kubectl apply -f ./ef-sample-add-tenant_id.yaml
```

2. Edit service mesh in istio configmap.

```
$ kubectl edit configmap -n istio-system istio

data:
  mesh: |-
    ### Add logs config begin
    accessLogEncoding: JSON
    accessLogFile: /dev/stdout
```

[View Details](#)

#IstioCon



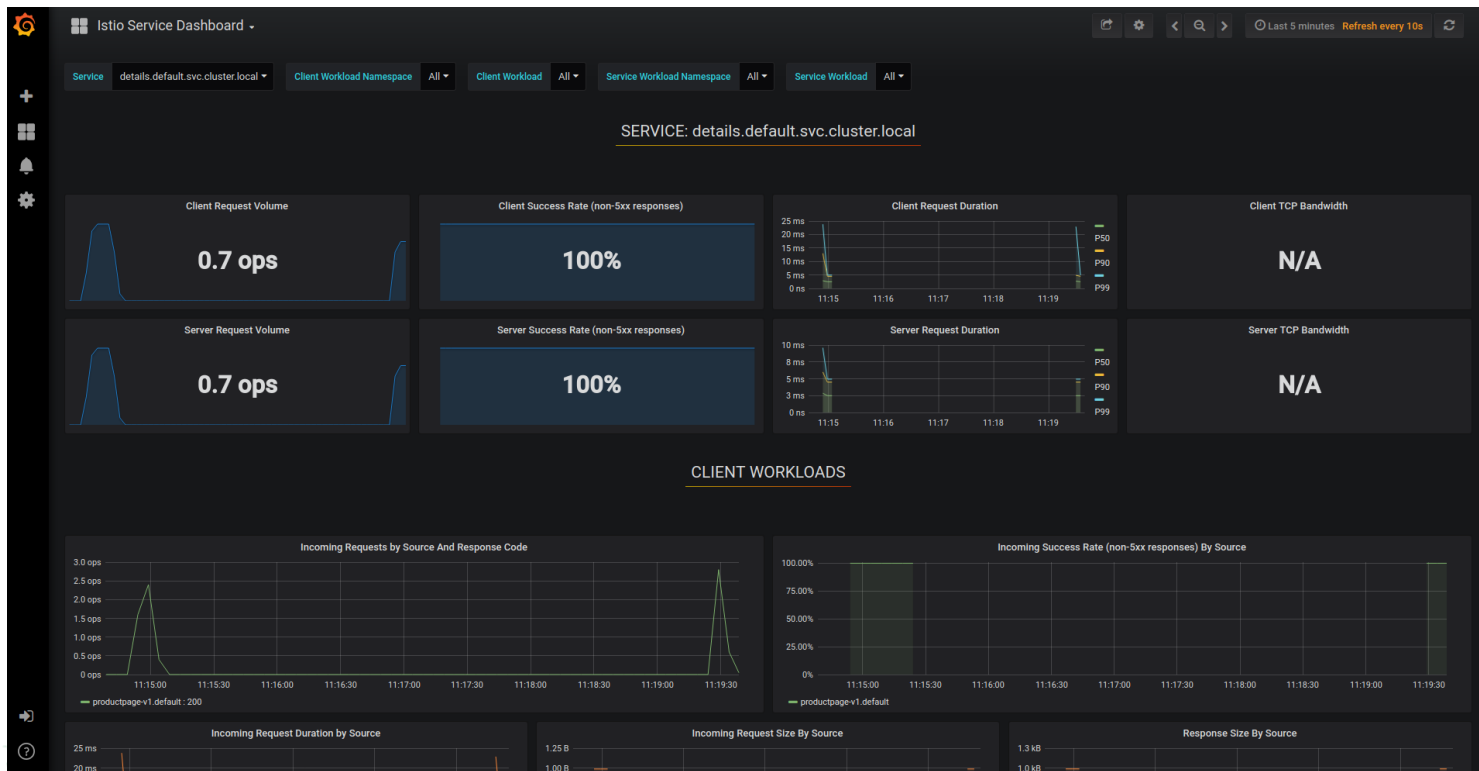
Observability Enhance for tenant

Customize Istio Metrics

#IstioCon



Istio Metrics



#IstioCon



Istio Metrics Customization

Requirements:

- Add `tenant_id` to `istio_requests_total` metrics



Implement for Metrics Customize

Step 1: Create an envoyfilter, get tenant_id from token, insert it into http header.

Step 2: `kubectl edit cm -n Istio-system Istio`

```
apiVersion: v1
data:
  mesh: |-
    defaultConfig:
      extraStatTags:
        - tenant_id
```

Step 3: `kubectl apply -f telemetry.yaml`

```
apiVersion: telemetry.istio.io/v1alpha1
kind: Telemetry
metadata:
  name: global-metrics
  namespace: istio-system
spec:
  metrics:
    - overrides:
        - tagOverrides:
            tenant_id:
              operation: UPSERT
              value: request.headers["tenant_id"]
  providers:
    - name: prometheus
```



Test for Metrics Customize

Test:

```
$ curl -I $gateway_url/productpage?id=13570 -H "Authorization: Bearer $token"
```

Token:

PAYLOAD: DATA

```
{  
  "sub": "1234567890",  
  "name": "John Doe",  
  "tenant_id": "tenant0002_fromtoken",  
  "iat": 1516239022  
}
```

Metrics:

```
$ k exec -ti productpage-v1-75875cf969-nsv55 -c istio-proxy -- curl -sS 'localhost:15000/stats/prometheus' | grep istio_requests_total
```

```
istio_requests_total{reporter="destination",source_workload="istio-ingressgateway",source_canonical_service="istio-ingressgateway",source_canonical_revision="latest",source_workload_namespace="istio-system",source_principal="spiffe://cluster.local/ns/istio-system/sa/istio-ingressgateway-service-account",source_app="istio-ingressgateway",source_version="unknown",source_cluster="Kubernetes",destination_workload="productpage-v1",destination_workload_namespace="default",destination_principal="spiffe://cluster.local/ns/default/sa/bookinfo-productpage",destination_app="productpage",destination_version="v1",destination_service="productpage.default.svc.cluster.local",destination_canonical_service="productpage",destination_canonical_revision="v1",destination_service_name="productpage",destination_service_namespace="default",destination_cluster="Kubernetes",request_protocol="http",response_code="200",grpc_response_status="",response_flags="-",connection_security_policy="mutual_tls" tenant_id="tenant0002_fromtoken"} 4
```



Samples Code

samples > customizeMetrics		Search customizeMetri
<input type="checkbox"/> Name	Date modified	
ef-sample-add-tenant_id.yaml	9/21/2023 3:20 PM	
readme.md	9/22/2023 12:33 AM	
telemetry.yaml	9/22/2023 12:32 AM	

[View Details](#)

This sample is about customize metrics, adding tenant_id.

Before you begin

1. Setup Istio in a Kubernetes cluster by following the instructions in the [Installation Guide](#).
2. Deploy the [Bookinfo](#) sample application.
3. Had better remove envoyfilter from ratelimit sample, to make logs clear.

Customize Access Logs For tenant

1. Create envoyfilter to generate tenant_id from token.

```
kubectl apply -f ./ef-sample-add-tenant_id.yaml
```

2. Edit serice mesh in istio configmap.

```
$ kubectl edit configmap -n istio-system istio

data:
  mesh: |-

  defaultConfig:
    ### Add metrics config begin
    extraStatTags:
      - tenant_id
    ### Add metrics config end
    ... ..
```

3. Create telemetry for new added metrics.

```
$ kubectl apply -f telemetry.yaml
```



Q & A

Thank you!

Email: john.zheng@hp.com
WeChat: johnzhengaz
Github: [johnzheng1975](https://github.com/johnzheng1975)

#IstioCon

