API GATEWAY : THE ADVANTAGES AND PROBLEMS NEED TO BE RESOLVED

NAME: LY TUAN ANH

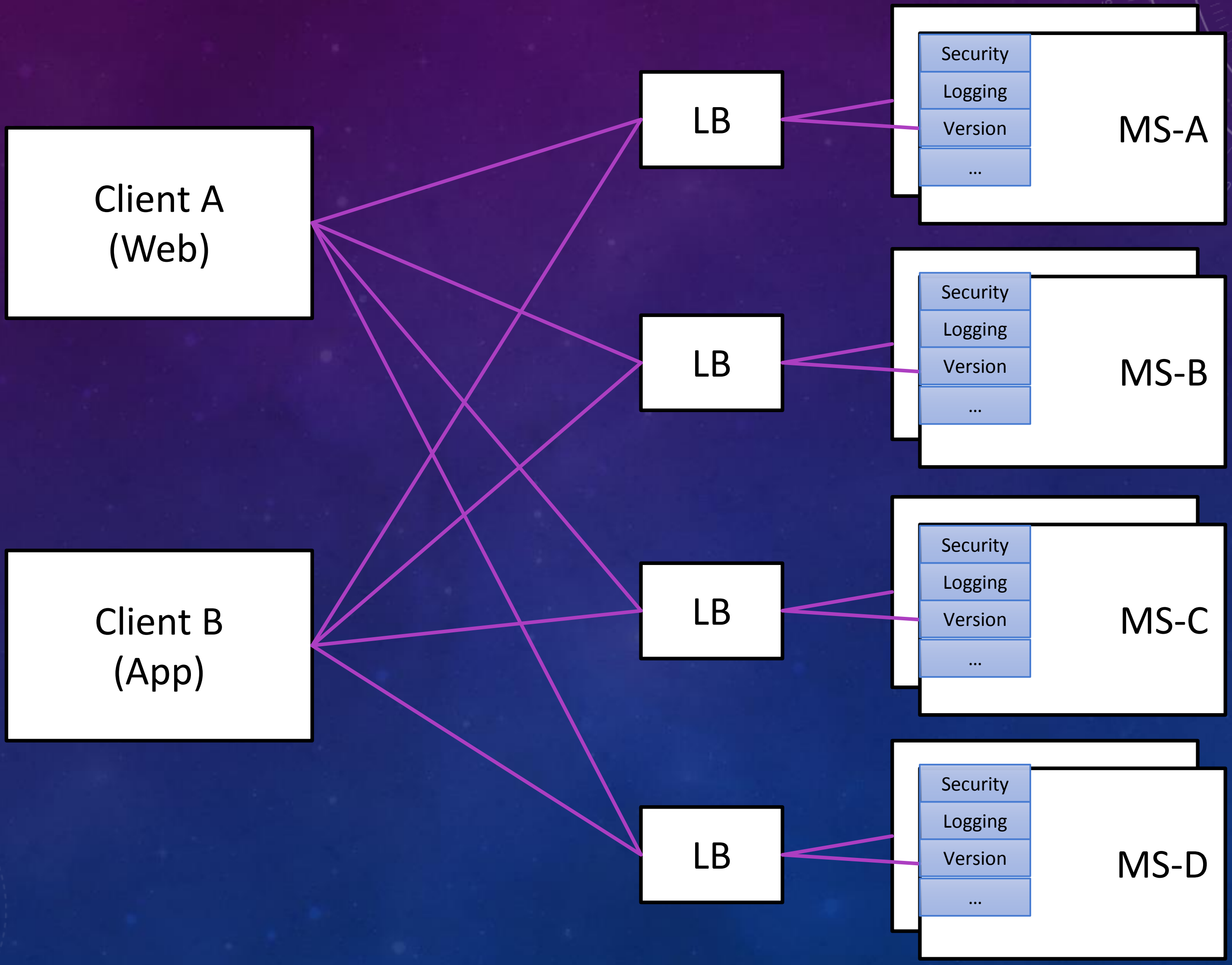
EMAIL: LYTUANANH2003@GMAIL.COM

JOB: TECHNOLOGY OFFICER OF FPT CORPORATION

SYNOPSIS

- You’re developing based on MSA(Micro-Services Architecture)
- How do the clients access the individual Micro-services?

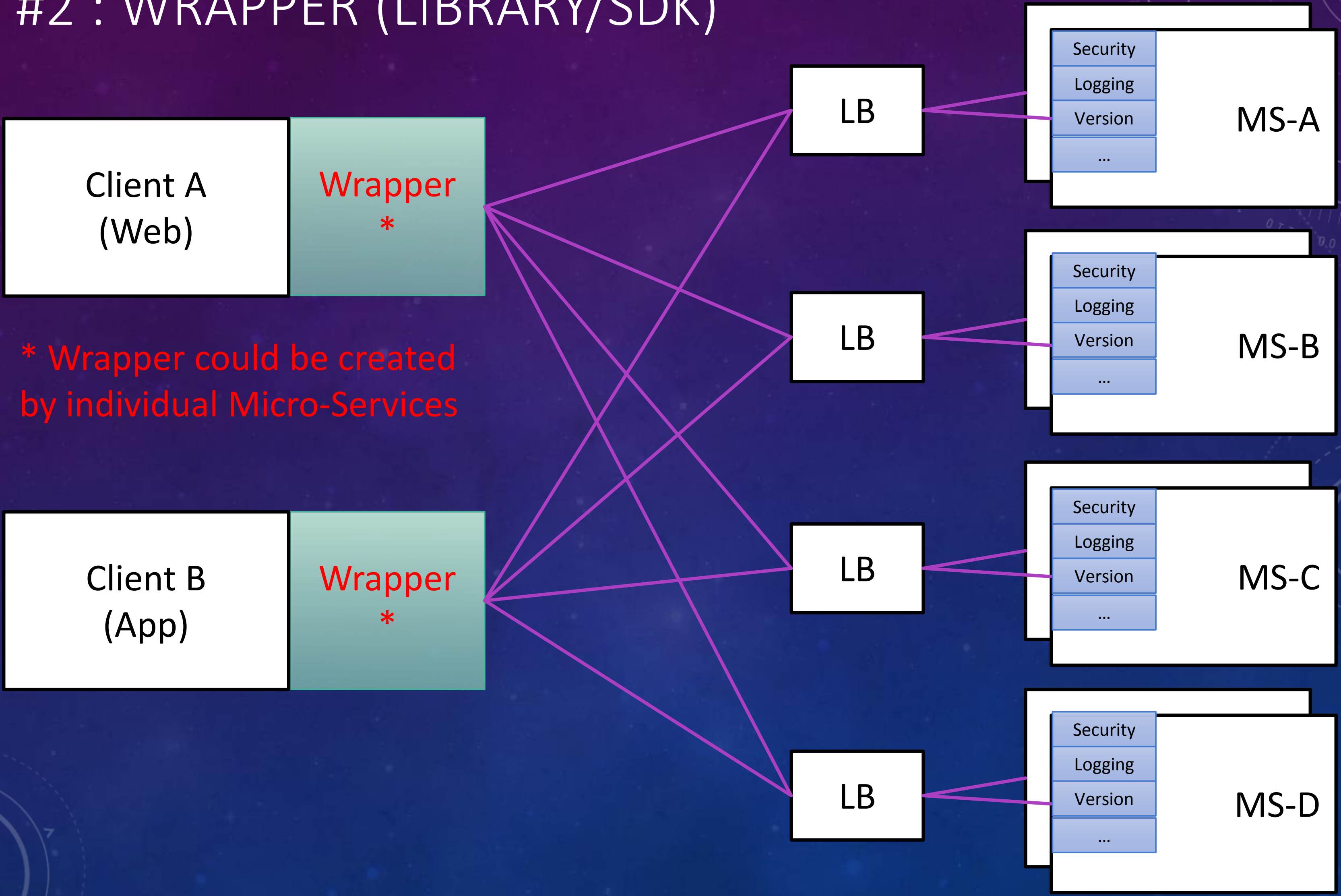
#1 : I DON’T CARE FOR CLIENTS, DIY



#1 : I DON'T CARE FOR CLIENTS, DIY

- Clients need to access individual Micro-Services by themselves
- Pros
  - No SPOF
  - No cost for developing API Gateway
- Cons
  - Clients need to know endpoints of Micro-Services
  - If Micro-Services changes, all clients need to update
  - Each Micro-Services needs to handle these by themselves
    - Securities to protect their APIs
    - Logging, Analytics, and any requirements from clients
  - whenever new Micro-Service is added
  - If there is no API standard, client will go to hell
  - Cannot handle REST "chattiness" problem
  - You need to place Load Balancer in front of each Micro-services

#2 : WRAPPER (LIBRARY/SDK)



* Wrapper could be created by individual Micro-Services

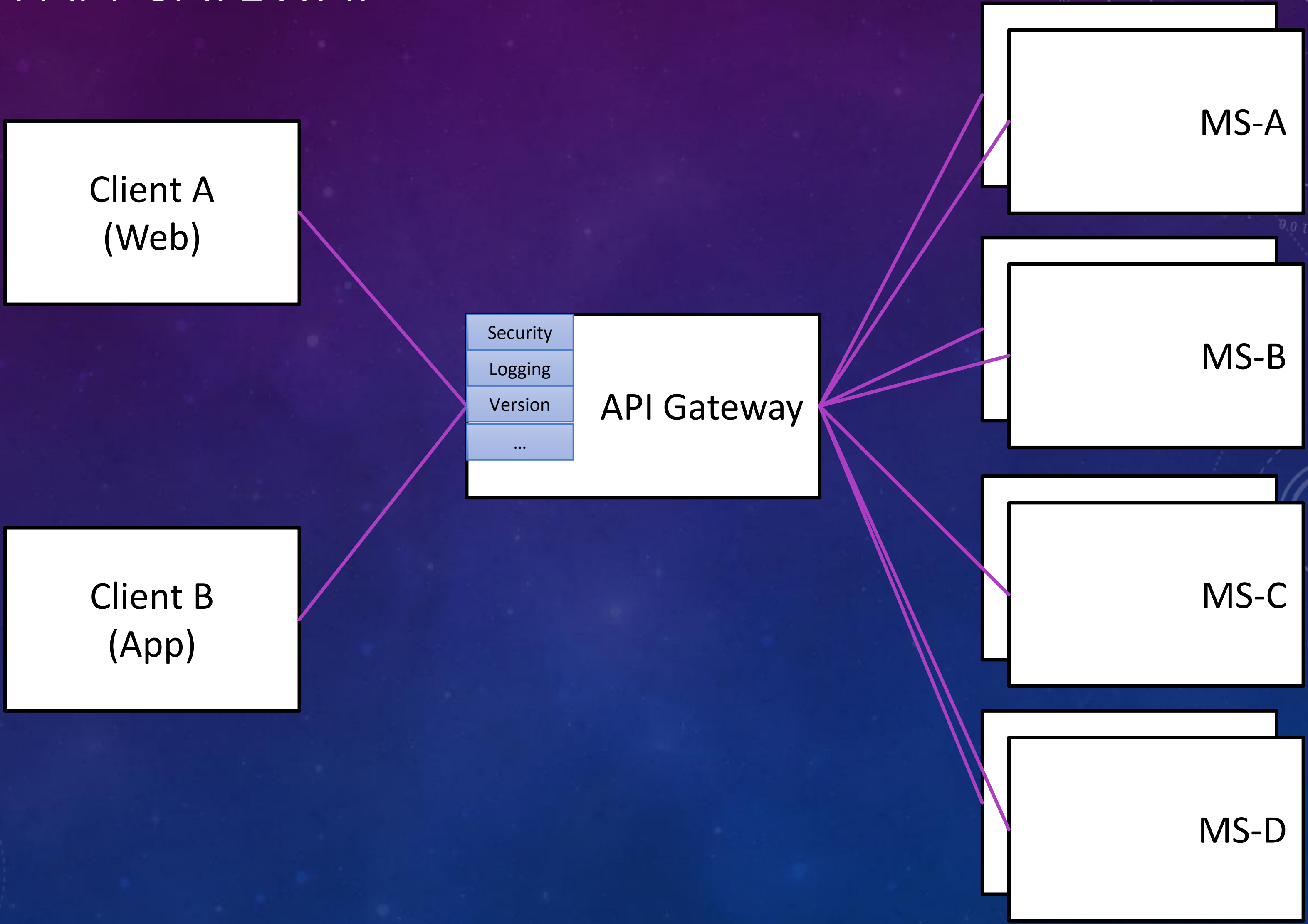
#2 : WRAPPER (LIBRARY/SDK)

Pros/Cons	#1	#2
No SPOF	Yes	Yes
No cost for developing API Gateway	Yes	Yes
Higher Abstraction than REST APIs, so easy to use	No	Yes
needs to know endpoints of Micro-Services	Client	Wrapper
If Micro-Services changes something(ex: LB VIP)	All clients	updated, QA, ,re-deployed
is responsible for backward compatibility	No	Yes
Each Micro-Services needs to handle these by themselves	Yes	Yes
You’re adding another security path whenever new Micro-Service is added	Yes	Yes
If there is no API standard, client will go to hell	Yes	Yes
handle composition scenario to prevent REST chattiness problem.	Cannot	Can
You need to place Load Balancer in front of each Micro-services and consider fail-over of LB, too	Yes	Yes
Becoming big burden if you need to support polyglot clients	No	Yes

CHECKPOINT

- It’s all about level of “Abstraction”
  - Provide it as REST APIs
  - Provide it as Wrapper (Library/Wrapper)
- Higher abstraction
  - Makes client happy (but only if you maintain versions/backward compatibility well)
  - Makes Wrapper developer unhappy
  - Even worst if API Provider != Wrapper developer
- Common RoR problems
  - If client fails, who’s responsible for investigate it?

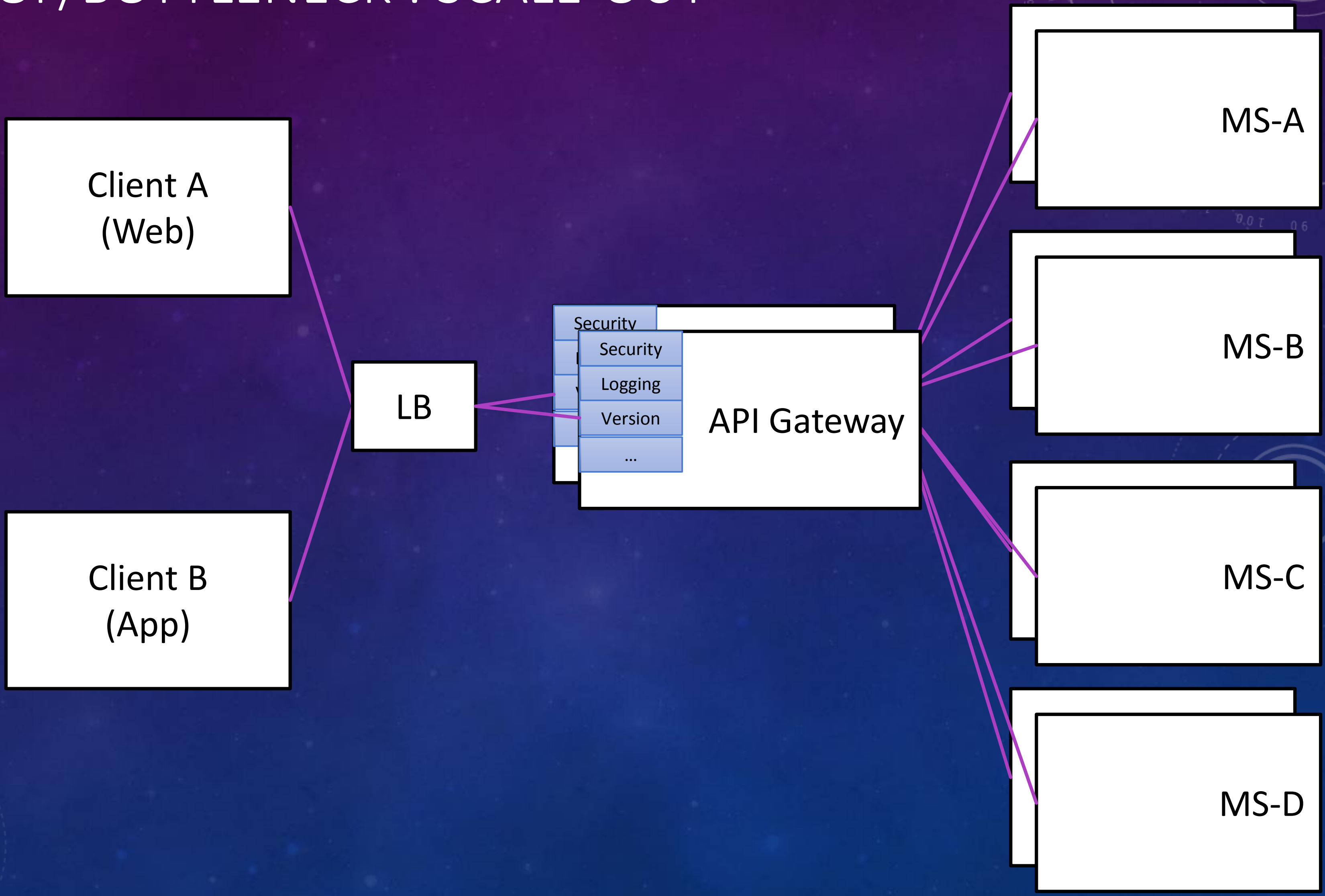
#3 : API GATEWAY



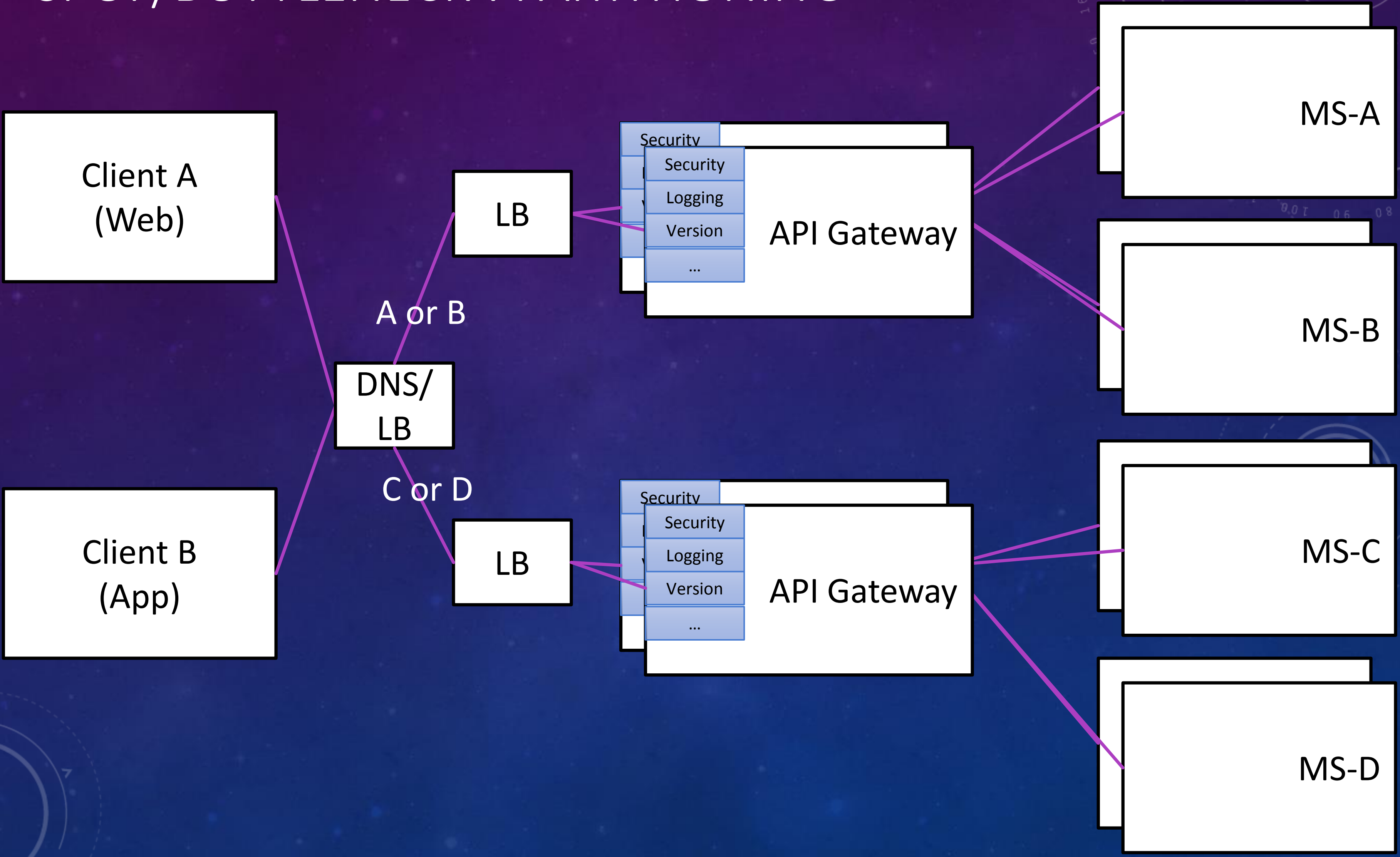
#3 : API GATEWAY

- Single endpoint for clients, handle requests proxied/routed to the appropriate service (or service instance)
- Pros
  - Can solve most problems
  - Separation of Concerns
    - Micro-Services focus on business features
    - API Gateway provides protection/common feature layer
  - Minimize/Isolate services' change impacts
- Cons
  - Possibility of SPOF/bottleneck
  - Performance tradeoff due to processing time in API Gateway and more network hops
  - Need to manage routing rule or APIs
  - Needs Service Discovery/Registry
  - Cost for developing API Gateway
  - Additional Hardware/Network/Management cost
  - Risk of management bottleneck

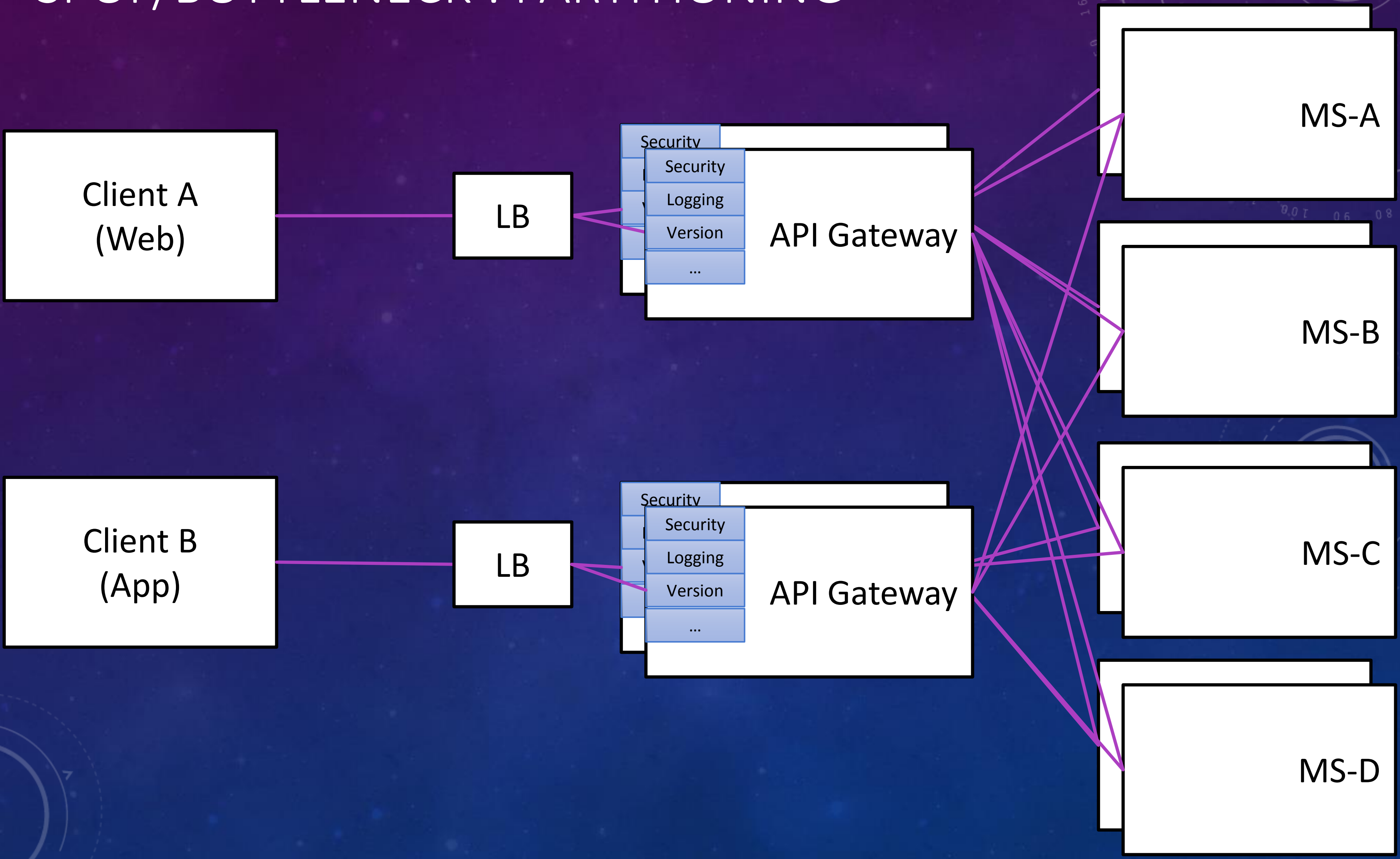
SPOF/BOTTLENECK : SCALE-OUT



SPOF/BOTTLENECK : PARTITIONING



SPOF/BOTTLENECK : PARTITIONING



PERFORMANCE TRADEOFF

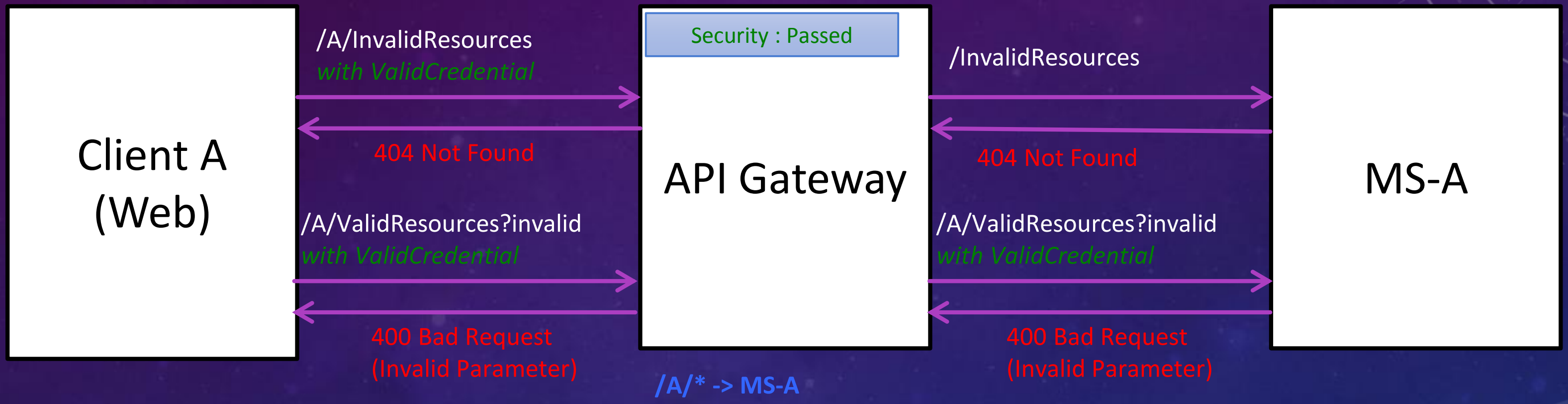
- Network hop/latency depends on network topology
- API Gateway processing time depends on what you want to do in API Gateway
- Consider Tradeoff : What’s more important?
- Some Tips
    - Don’t parse request/response body if you don’t need it
    - Caching on API Gateway

MANAGING ROUTING RULE OR APIS

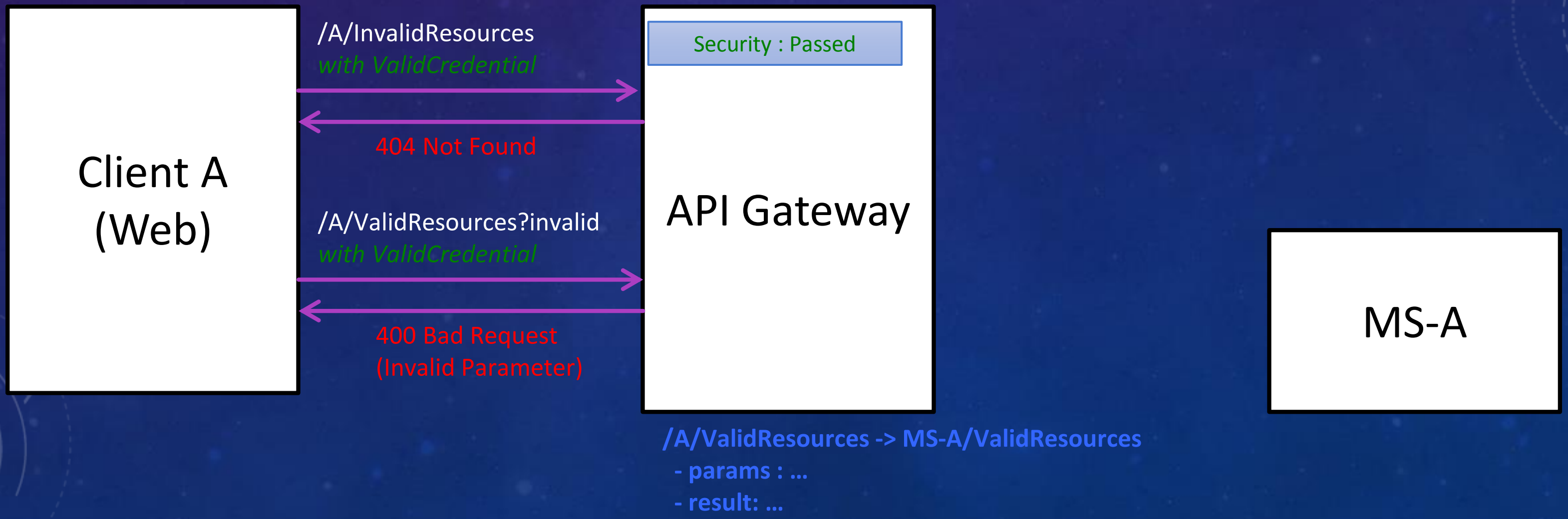
- Routing Rule-based Control
  - Define Coarse-grained routing rule
  - Gateway knows MSs but don’t care for specific APIs
  - Micro-Services need to resolve APIs and validate whether they are valid request
- API-based Control
  - Register APIs want to be managed in Gateway
  - API Gateway resolve APIs and validate request/response with exact match
  - Gateway should know APIs

MANAGING ROUTING RULE OR APIS

Routing Rule Based Control(per MS)



API Based Control (per API)



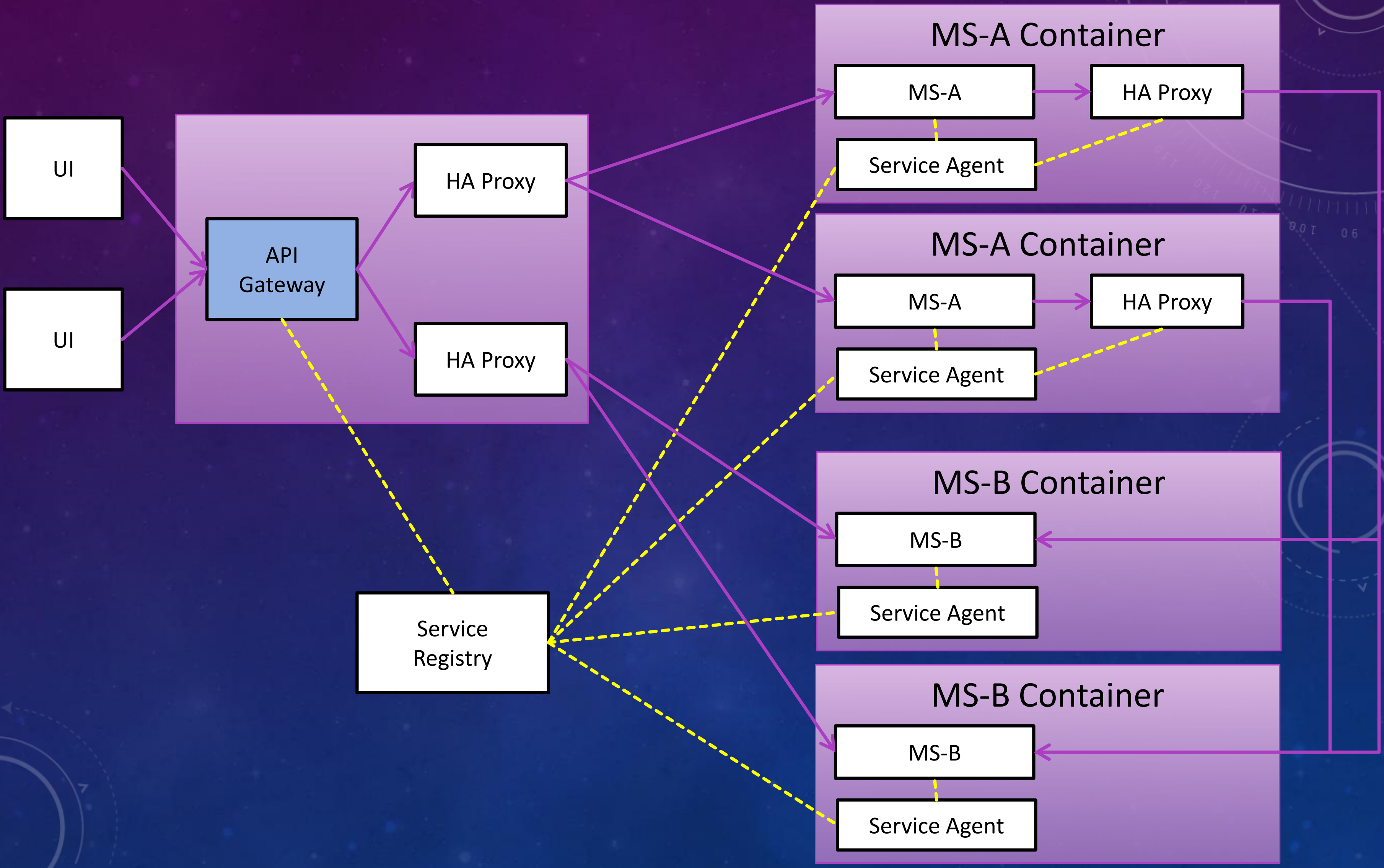
MANAGING ROUTING RULES OR APIS

- Routing rule based is preferred when
    - Clients are 1st parties
    - Coarse-grained control is enough
    - You can provide API spec/document from Micro-Services directly
    - API is changed frequently
- API based is preferred when
    - Clients are including 3rd parties
    - Minimize Micro-Services’ overhead from invalid request
    - Fine-grained control is needed
    - If you require mediation or some manipulation per APIs
    - You need to provide API spec/document from API Gateway
- Recommendations
    - Use routing rule based control primarily, then append API-based control as you need

MANAGING API SPECIFICATION

- You can manage it
  - Deeply coupled with API Gateway
    API-based Control requires for API Gateway to know API specification
  - Externally (ex : Swagger, ProtocolBuffer)
    Both Routing Rule-based and API-based control
- If you have a API spec,
  - Client developer can create client codes (even wrapper)
  - Server developer can create server codes

SERVICE DISCOVERY/REGISTRY



COST FOR DEVELOPING API GATEWAY

- Depends on what you want to do with API Gateway
- Simple requirements = Simple API Gateway
  (nginx/HA proxy might be enough for you)
- Node.js is a good start point to implement
- But going complex
  - If you need to consider 3rd parties and Open API since Developer portal and Onboarding process is required
  - If you want some GUI and management console (= Publisher portal)
  - Consider API Gateway as Silver Bullet (ESB?)…

ADDITIONAL HARDWARE/NETWORK/MANAGEMENT COST

- Another tradeoff : What’s more important?
- Depends on how you implement it and what you want to do
- Cost could be issue
  - If you consider adopting commercial products
  - If you consider doing a lot of manipulation in API Gateway

RISK OF MANAGEMENT BOTTLENECK

- If API Gateway is managed by single team, there are risks of management bottleneck
  - API Gateway team has primary responsibility for changes/failure/backward compatibility, …
  - API Gateway team could be a bottleneck (going worse if you do a lot of manipulations in it)
- Recommendation : separate managements
  - API Gateway itself (API Gateway team)
  - Services on the API Gateway (each service teams)

API GATEWAY

For free

KONG

API Umbrella

APIMAN

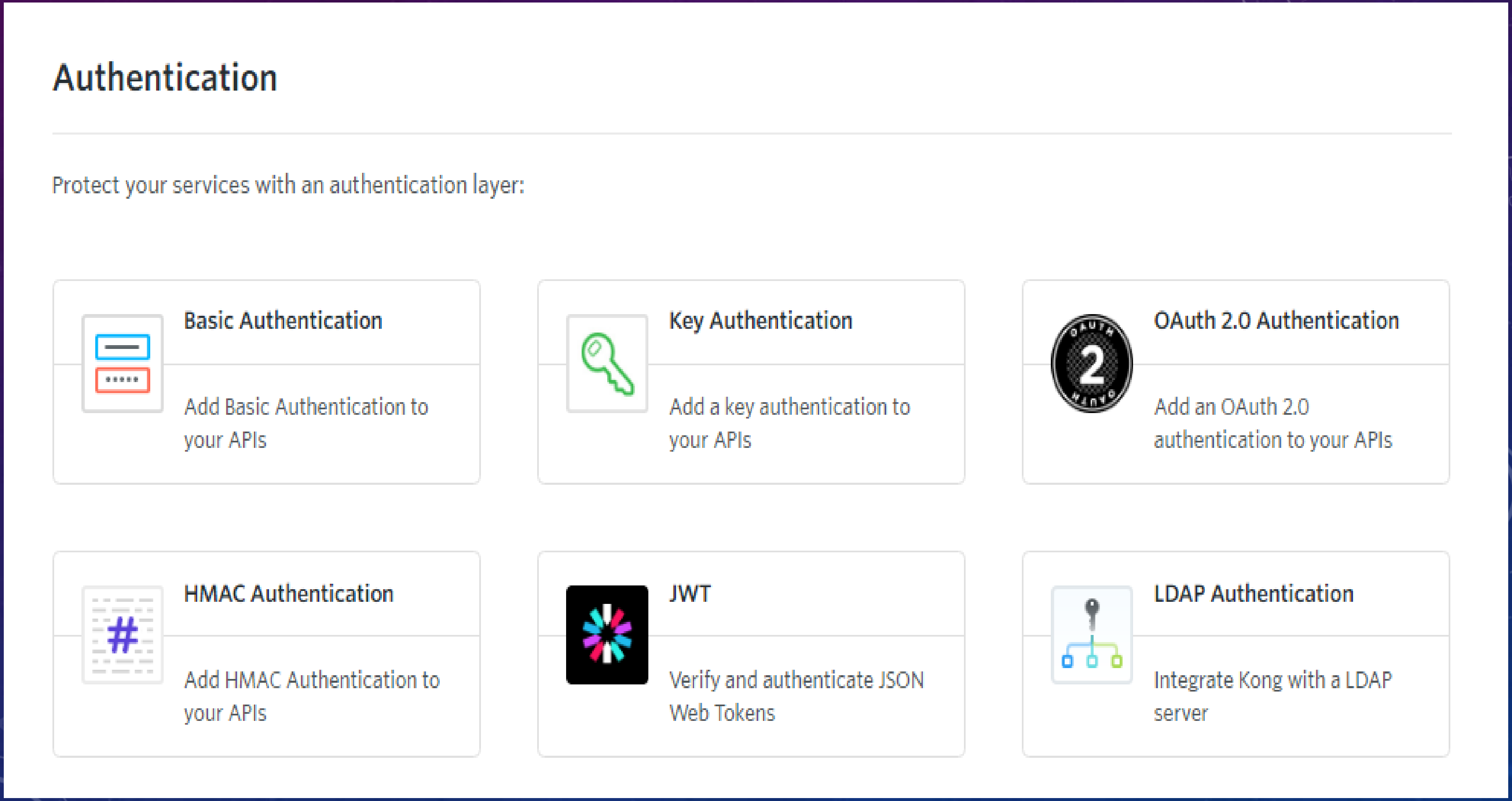
for Enterprice

Microsoft Azure

amazon web services

Tyk

API GATEWAY:PLUGINS



Authentication

Protect your services with an authentication layer:

Basic Authentication
Add Basic Authentication to your APIs

Key Authentication
Add a key authentication to your APIs

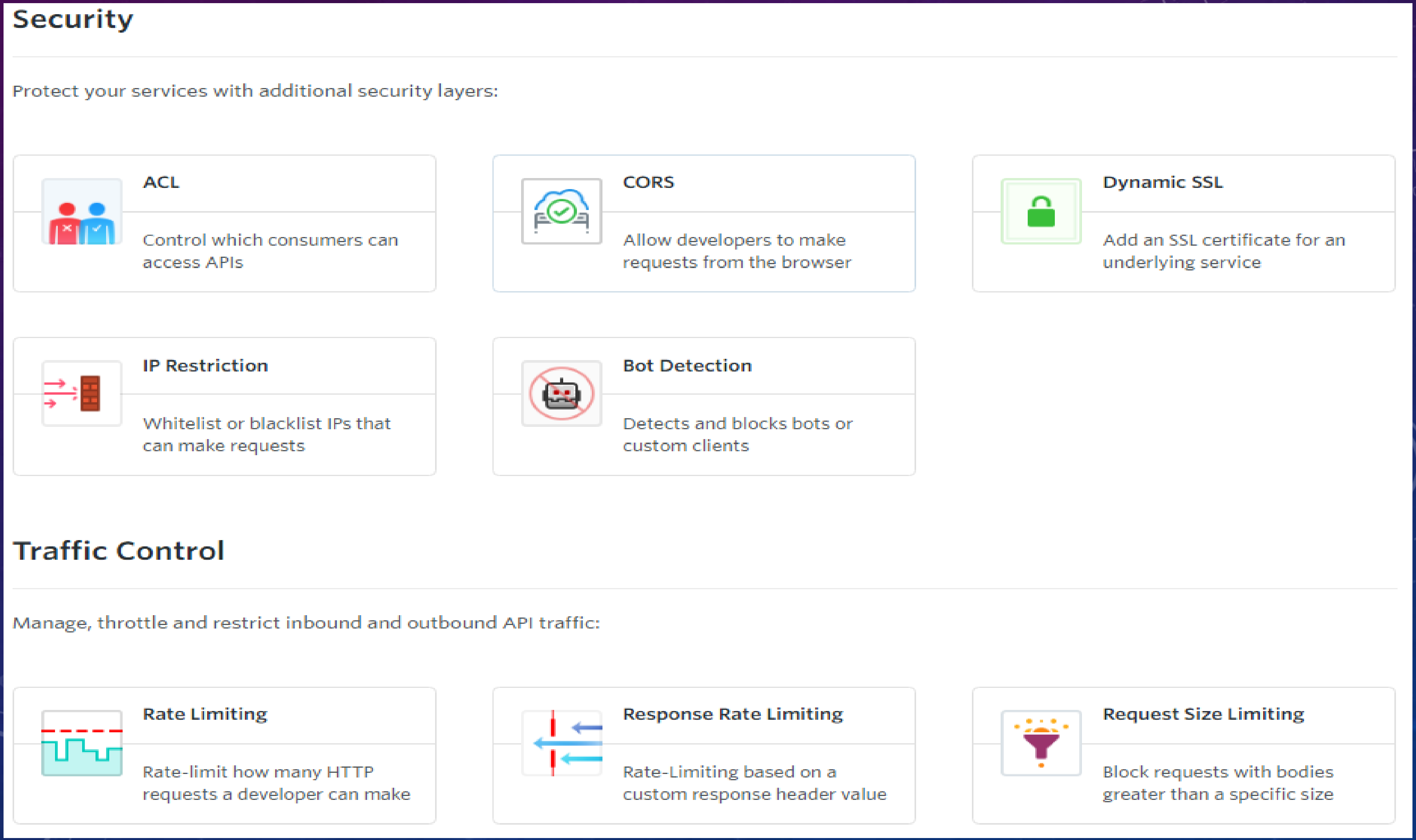
OAuth 2.0 Authentication
Add an OAuth 2.0 authentication to your APIs

HMAC Authentication
Add HMAC Authentication to your APIs

JWT
Verify and authenticate JSON Web Tokens

LDAP Authentication
Integrate Kong with a LDAP server

API GATEWAY:PLUGINS



Security

Protect your services with additional security layers:

ACL

Control which consumers can access APIs

CORS

Allow developers to make requests from the browser

Dynamic SSL

Add an SSL certificate for an underlying service

IP Restriction

Whitelist or blacklist IPs that can make requests

Bot Detection

Detects and blocks bots or custom clients

Traffic Control

Manage, throttle and restrict inbound and outbound API traffic:

Rate Limiting

Rate-limit how many HTTP requests a developer can make

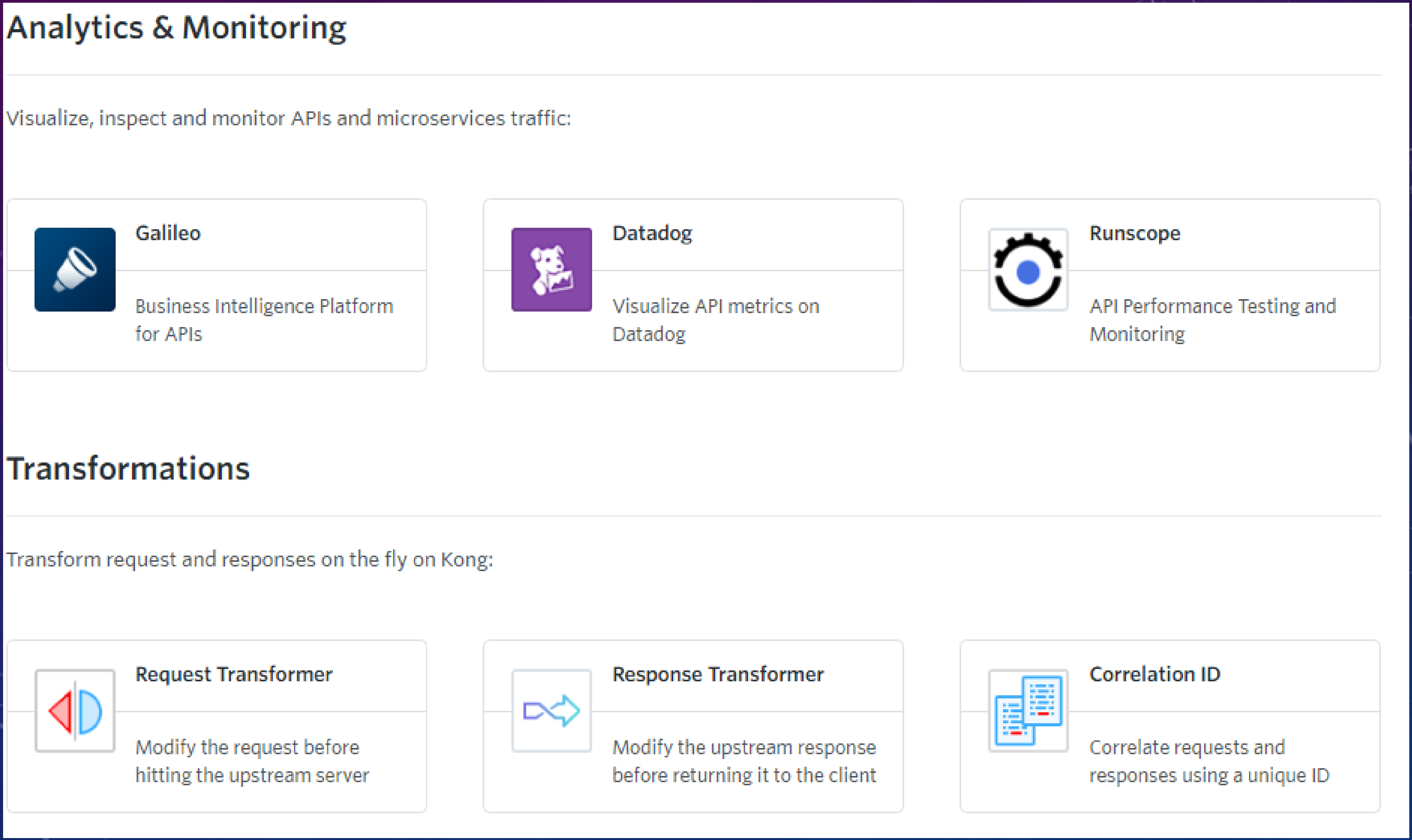
Response Rate Limiting

Rate-Limiting based on a custom response header value

Request Size Limiting

Block requests with bodies greater than a specific size

API GATEWAY:PLUGINS



Analytics & Monitoring

Visualize, inspect and monitor APIs and microservices traffic:

Galileo

Business Intelligence Platform for APIs

Datadog

Visualize API metrics on Datadog

Runscope

API Performance Testing and Monitoring

Transformations

Transform request and responses on the fly on Kong:

Request Transformer

Modify the request before hitting the upstream server

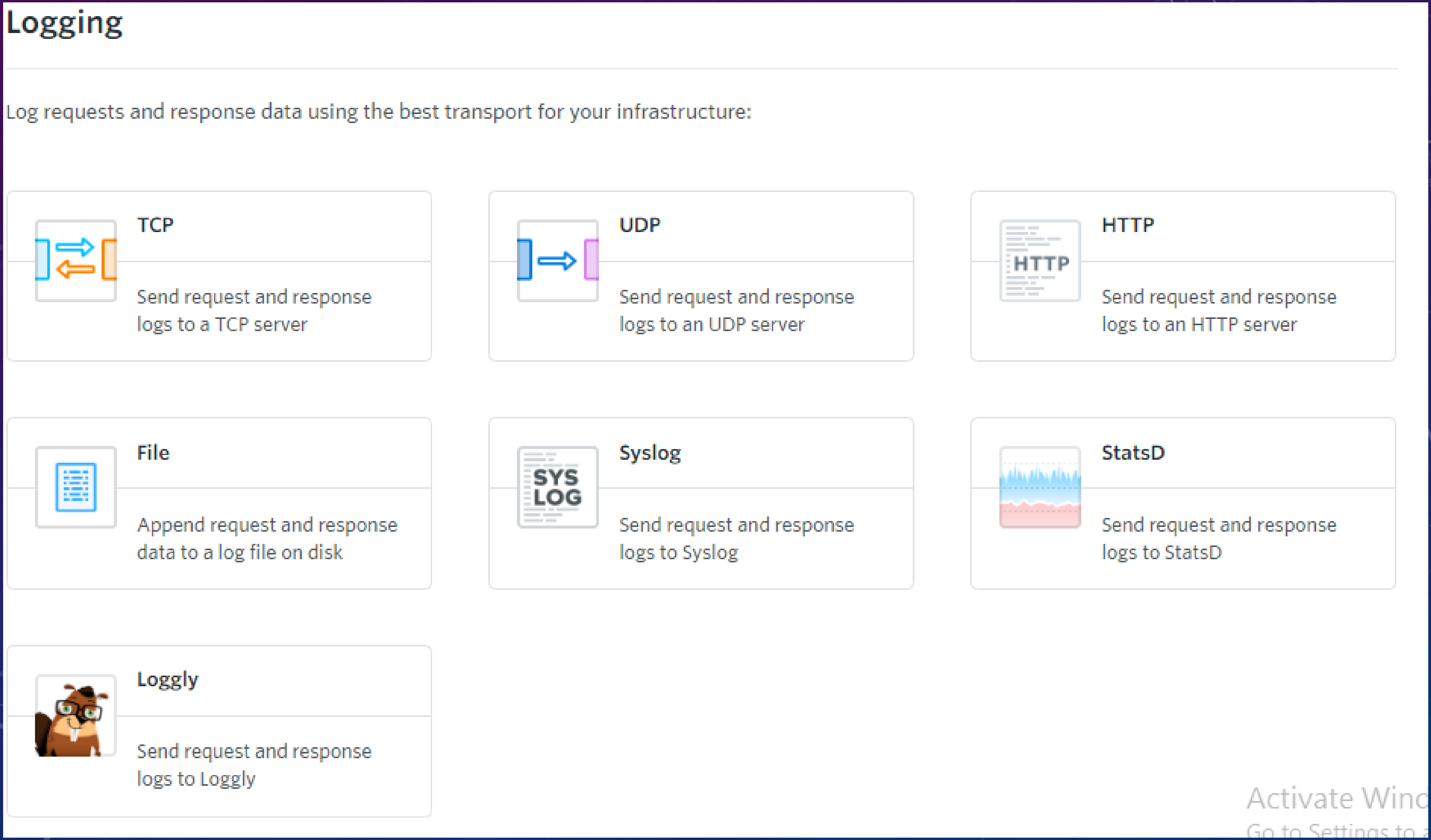
Response Transformer

Modify the upstream response before returning it to the client

Correlation ID

Correlate requests and responses using a unique ID

API GATEWAY:PLUGINS



Logging

Log requests and response data using the best transport for your infrastructure:

TCP

Send request and response logs to a TCP server

UDP

Send request and response logs to an UDP server

HTTP

Send request and response logs to an HTTP server

File

Append request and response data to a log file on disk

Syslog

Send request and response logs to Syslog

StatsD

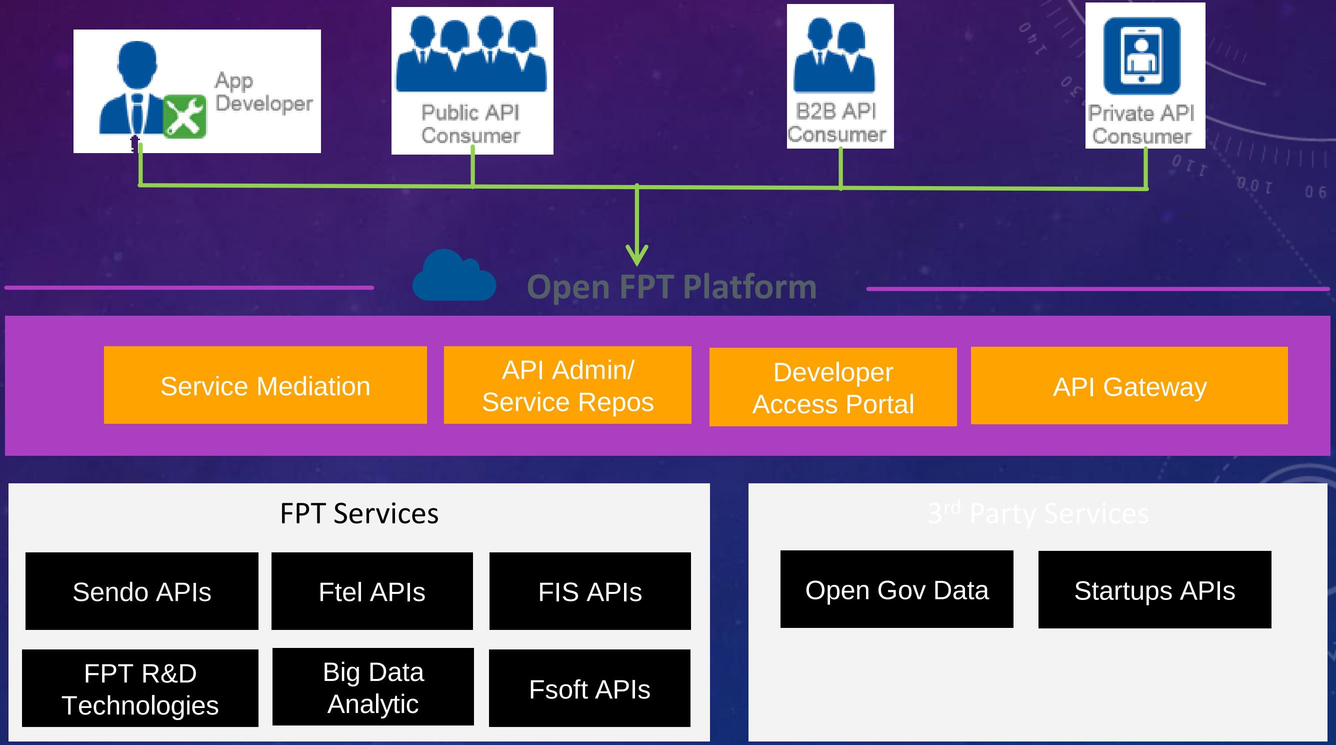
Send request and response logs to StatsD

Loggly

Send request and response logs to Loggly

Open FPT Initiative

- Open FPT acts as common API platform for FPT companies to expose their services for both external and internal usages.
- Create API ecosystem especially for startups
- IoT application ready
- Open Source to contribute back to tech community and built on Open Source
- Leverage latest Cloud technologies
- Can be used to open up Gov Data



THANK YOU