My experiences with API gateways...

Mahesh Mahadevan Follow
Apr 7 · 11 min read ★

A while back, I was working on a project to implement API gateway for our product's cloud offering. The main motivation behind it was to have a single point of entry for all the external traffic and safeguarding the back-end services, but there were other reasons too :). I will assume that you have stumbled over this link with prior knowledge on API gateways, if not, you should read about them here and here before you go any further into this article.

Finding an API Gateway that matches your requirement is an overchoice, you might eventually suffer a psychological breakdown if you take it personally. Like so many other software stacks prevalent in the industry today, even API gateway comes with a variety of choices and flavors. To be honest, it is quite a struggle when it comes to making this decision without digging a bit deeper into each of these possible options and matching it to your specific requirements. This article is my humble attempt in making this task a little simpler, and provide you with a flowchart at the end of it all, to hopefully arrive at a decision.

As we began our search, we started evaluating a few options that came up based on our research. To do this, we mainly relied on various comparisons that already exist on the internet, then we shortlisted a few of them based on their current popularity and the feature sets they had to offer.

Disclaimer *— This article does not provide any sort of performance comparison of API gateways, even though, this was also a criterion on which we based our selection. I do not claim that implementations discussed here are the only options available, but these implementations were the most popular choices based on our product requirements at the time of writing this article.*

. . .

Lets list Requirements...

First, lets quickly list down what was the minimum that we expected from an API gateway solution. Please note, this is not an exhaustive list or something that might match a solution that you are looking for, but it should cover most of the scenarios where API Gateways are applicable.

1. Reverse Proxy — This is the single most important reason why most projects adopt a gateway sort of a solution. Any mature project, which has opened its APIs to the outside world would avoid exposing its back-end URLs for security reasons and abstracting the complexity of back-end services to client applications. This also gives a single point of entry to all clients accessing back-end APIs.
2. Load balancing — Gateways can route a single incoming URL to multiple back-end destinations. This is often useful in micro-services architecture when you want to scale up your application for high-availability or even other-wise if you are running some sort of cluster server setup.
3. Authentication and Authorization — Gateways should be able to successfully authenticate and allow only trusted clients to access APIs and also able to provide some sort of authorization layer using something like RBACs.
4. IP Listing — Allow or Block certain IP addresses to pass through. Provides an additional layer of security to your ecosystem, useful when you discover a malicious set of addresses trying to bring down your application by using DDoS sort of attacks.
5. Analytics — Provide a way to log usage and other useful metrics related to API calls. Should be able to disintegrate the API usages made per client for possible monetization.
6. Rate-Limiting — Ability to throttle API calls, for example, you only want to allow 10000 calls per minute for all consumers or 1000 calls per month for a particular consumer.
7. Transformation — Ability to transform either request and responses*(including header and body)* before forwarding them further.
8. Versioning — An option to use different versions of API at the same time or possibly provide a slow rollout of API in form of Canary release or Blue/Green deployments
9. Circuit Breaker — Useful with micro-services architecture pattern to avoid disruption in service
10. WebSocket support — Many dynamic and real-time capabilities can be addressed by using WebSockets, providing a WebSocket interface to end clients can really reduce overheads of multiple HTTP calls for frequent data transfers
11. gRPC support — Google's gRPC promises to further reduce load by making use of HTTP/2, can be used efficiently for inter-communication between services. I would recommend this is something you should definitely add to your list of requirements to make your solution future proof

12. Caching — Will further reduce the network bandwidth and round trip time consumption and improve performance if frequently requested data can be cached
13. Documentation — If you plan to expose your APIs to developers outside of your organization, then you must think about using API documentation such as Swagger or OpenAPI.

. . .

API Gateways and Service Mesh

Before going into comparisons on actual implementations, I must also talk about another pattern which you might run into while looking for gateways, Service Mesh. It might be confusing at first to know the difference between API gateway and Service Mesh and each of their purpose, so I am going to describe them in a little detail before we proceed.

API gateways are applied on Layer 7 of OSI model or you can say to manage traffic coming from outside network (sometimes also called north/south traffic), whereas Service Mesh is applied to Layer 4 of OSI model or to manage inter-services communications (sometimes also called as east/west traffic). Some examples of API Gateway features are Reverse Proxy, Load Balancing, Authentication and Authorization, IP Listing, Rate-Limiting, etc.

Service Mesh, on the other hand, works like a proxy or a side-car pattern which de-couples the communication responsibility of the service and handles other concerns such as Circuit breaker, timeouts, retries, service-discovery, etc. Istio is a very well known implementation of service mesh at the time of this article being published.

You must have noticed that I have included in my list requirements some of the features provided by Service Mesh as well. Quite a few of API gateway implementations today can work at both Layer 4 and 7 and handle requirements of service mesh as well. It would be nice if we could get an implementation which can also handle some of the service mesh requirements even though it is not a must. Here is a good article citing differences between the two in little detail.

. . .

Comparisons

TL;DR;

I am going to compare the following API gateways …*(read disclaimer)*

1. NGINX
2. Kong
3. Tyk
4. Ambassador
5. AWS API gateway

Nginx

Nginx has been one of the best choices for L7 proxy, load-balancing and creating a single point entry for your back-end applications for quite some time now. It has already been used and tested in many different production environments and has replaced many existing hardware load-balancers with very low footprint and reducing a lot of cost to companies. Lot of CDNs use Nginx as their engine for caching data at edge locations.

The biggest advantage of using Nginx as a gateway is its ability to range from simple to complex features, allowing you to cherry-pick only the required features as you progress. For example, if you only need load-balancing and reverse-proxy, to begin with, Nginx does this very easily with minimal overheads and you can eventually upgrade to other features as your product matures. You can also start with a full blown API gateway using their commercial offering, NGINX Plus, even though it is possible to achieve this with its open source offering using its wide range of available plugins.

Nginx is known for its small footprint and ability to meet high performance with low latency. You also get a lot of 3rd party custom Nginx plugins which can cover a wide range of custom scenarios, and of course, you could always seek help from a huge network of its developer community in case you are stuck somewhere. The only possible cons that you might encounter while working with Nginx is the configuration might be a little difficult to get hang of until and unless you have got your hands dirty working with it, you will have to go over few pages of their documentation till you can claim your mastery over it.

Kong

Kong is an Nginx and OpenResty based API gateway plus Service Mesh which caters for most of our requirements listed above. It was quite an easy install following the provided docker installation instructions.

Kong architecture is quite simple to understand and is made up of a few components…

- Kong base-module which wraps OpenResty and Nginx and is the engine which does the actual work
- Database layer with choice of Cassandra or Postgres to store all the configuration so it can be retrieved easily in case of failures
- Dashboard which provides User-Interface for API administration and viewing analytics *(part of the enterprise offering only, though kong provides REST APIs for managing services, upstream APIs and its consumers)*

Kong has both Open-source and Enterprise versions of its implementation, both work very well with sub-millisecond latency as it is powered by nginx itself. Imagine using Nginx for your gateway operations but with REST APIs and database layer for managing configurations easily.

Kong comes with a variety of plugins, which can cater for most of your crosscutting concerns ranging from access controls, security, caching to documentation. It also lets you write and use custom plugins using Lua language. Kong open-source is a good start to get familiar with their stack. Though it does not come with a Web-UI dashboard, there are few open source dashboards available which help you manage its configuration, otherwise if you are quite comfortable with REST you can work with their admin APIs directly.

Kong can also be deployed as Kubernetes Ingress and supports gRPC and Websockets proxy. Kong's advantage is its underlying engine is made up of lightweight yet powerful Nginx + OpenResty engine, which in itself can be built as a full-fledged API gateway. You can think of Kong as an auto-shift version of Nginx. A possible disadvantage of their implementation is not all the features come out-of-box, rather has to be manually configured by activating each of its respective plugins, which might need initial setup time and resources, but then this might not really a big hurdle for a lot of mature engineering teams.

Tyk

Tyk is another open-source gateway which promises excellent performance and is created in Go programming language. Tyk offers multiple features which are part of our requirements list and beyond. Tyk's web dashboard is one of the best in class and lets you control almost all aspects of API configuration provides excellent analytics of API usages.

Tyk has a rich feature set along with nice Web-UI dashboard making it a good choice for projects having a complicated API management scheme. What makes Tyk stand out is it includes API dashboard, analytics, security,

versioning, developer portal, rate-limiting and other gateway features out-of-box for free if you only intend to use this for non-commercial purpose. However, for commercial usage, you will need to buy their commercial license which also includes support.

Tyk can be really a good fit for projects looking for these features out-of-box from day one and are ready to spend for it*(i.e you are planning to use it for commercial purpose)*, rather than explore options such as Nginx and Kong which can take little bit of your developers time to get all the required features working. Where Tyk falls short in comparison to the previous two implementations, are ease of installation and cost. Tyk has too many components which do not make it that easy to install and manage on-premise. It also has a cloud and hybrid installation options which decreases installation and management overheads, but then it comes with its own pricing and increases your project costs.

Ambassador

Ambassador is an open-source, Kubernetes native microservices gateway built on top on Envoy. It can be used as Kubernetes Ingress and load balancer and is built to publish and test microservices easily and rapidly on top on Kubernetes environment.

Ambassador is very lightweight and all of its states is stored in Kubernetes, so there is no need for a database. Ambassador was built keeping the developer in mind, so all of its operations and concepts are more developer-centric, for example - the most recommended technique for adding a route on Ambassador is via annotations on Kubernetes services yamls. Its free version comes with features such as versioning, gRPC, and WebSockets support, Authentication, Rate-Limiting and Integration with Istio to work as Service Mesh, whereas features such as OAuth, single sign-on, JWT authentication, access control policies, and filtering are part of its paid version called Ambassador Pro.

Its advantages are its ability to serve large traffic with low footprint and minimal setup configuration on Kubernetes environments, whereas where it lacks is not being as feature rich in comparison to previously discussed gateways as it is missing out-of-box dashboard and integration with analytics which will require some setup.

Amazon API Gateway

Amazon API gateway is an AWS cloud offering of managed API management which lets you create, publish and manage APIs in a matter of few clicks. You can currently expose REST or WebSocket endpoint, import

new APIs using Swagger or Open API, Route your URL to various back-ends including AWS EC2, AWS lambda or even your in premise endpoints. The cost of routing million APIs through the gateway is very low and predictable as there is fixed pricing model for a given number of requests *(though you should be careful about costs incurred due to external data transfers)*.

AWS API gateway requires no setup as it is all managed, you can quickly create or route APIs in few clicks, secure them using SSL, provide Authentication and Authorization, create API Keys for external clients of your APIs, manage versioning of your APIs and also generate client SDKs if you wish to do so. AWS API gateway really packs a punch with its offering, making it really easy to setup an API gateway in a matter of few minutes. So if you are already on AWS or planning to go to AWS, you must really think about using this gateway as your first choice unless you have some mandatory feature requirements that it cannot fulfill as of yet. The obvious disadvantage being that you might get locked in with AWS service and this dependency might make your migration task to some other framework difficult at a later point in time.

. . .

Comparison Matrix

Here is a summary of feature comparison in the form of a table for above five API gateways.

- Green Tick: Feature is part of its open source version
- Yellow Tick: Feature is only part of its paid version
- Red Cross: Feature not present **yet** *(visit their respective portals for the current set of supported features)*

	Nginx	Kong	Tyk[2]	Ambassador	AWS API Gw
Basic[1]	✓	✓	✓	✓	✓
OAuth	✓	✓	✓	✓	✓
JWT	✓	✓	✓	✓	✓
Ip Listing	✓	✓	✓	✗	✓
Analytics	✓	✓	✓	✓	✓
Rate Limiting	✓	✓	✓	✓	✓
Transformation	✓	✓	✓	✓	✓
gRPC	✓	✓	✓	✓	✗
Websockets	✓	✓	✓	✓	✓
Service Mesh	✗	✓	✓	✓ 4	✗
Kubernetes Ingress	✓	✓	✓	✓	✗
Caching	✓	✓	✓	✗	✓
Documentation	✓	✓	✓	✗	✓
Admin UI	✓	✓ 3	✓	✗	✓
Ease of Setup	3.5	4	2.5	3	5

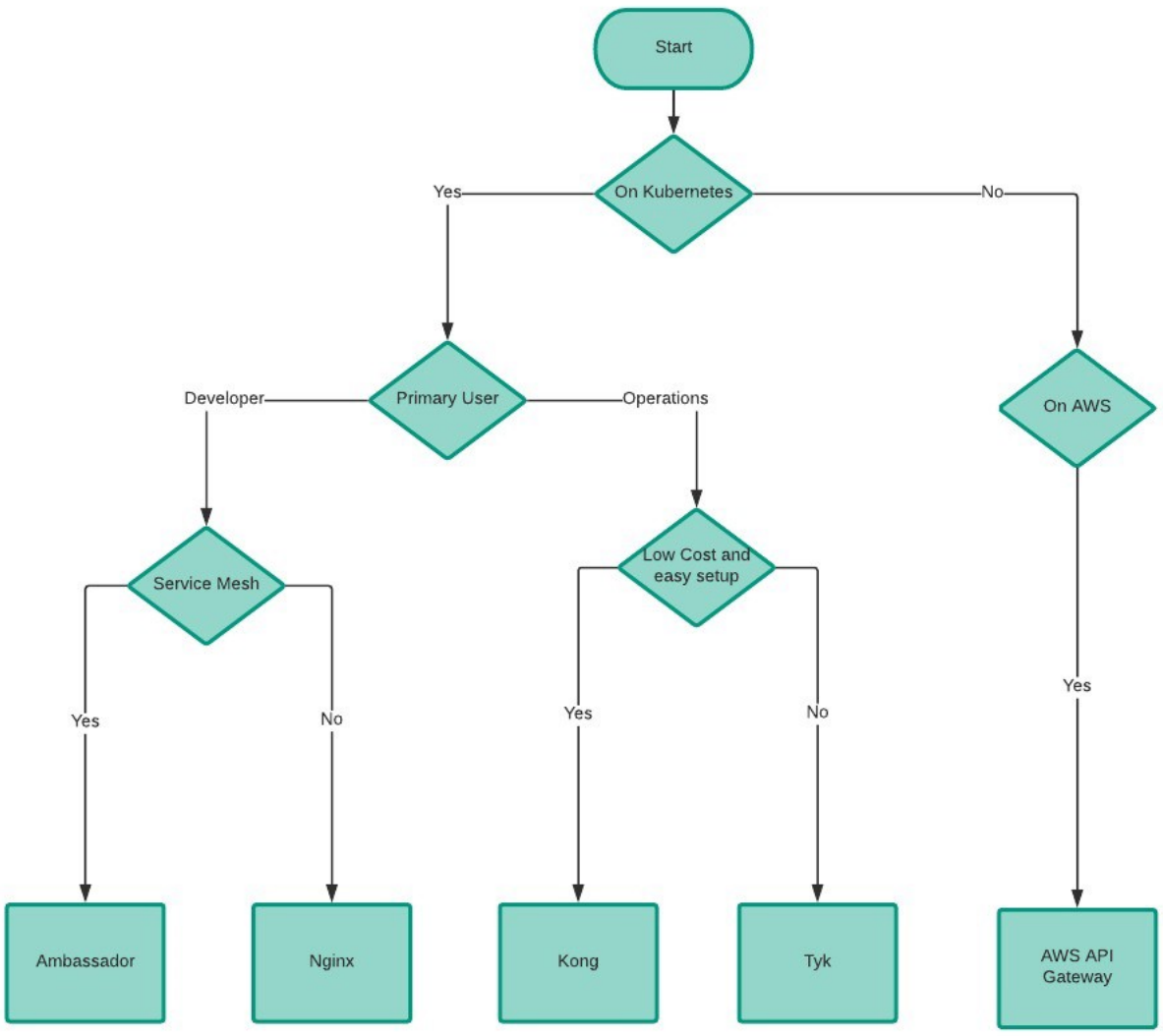
Comparison matrix

1. Basic features include reverse proxy, load balancing.
2. Tyk comes with developer license for non-commercial usages which includes all the features, for production usage, you need to buy their commercial license and it can be installed as Saas or a Hybrid approach.
3. Kong Open source edition does not come with its dashboard, but there are third party open source dashboards available which let you manage API and plugins through Web-UI.
4. Ambassador can be installed along with Istio to cover roles of service mesh.

. . .

Time for Decision

Finally, here is a flowchart, I have kept it simplistic on purpose. You should use the below chart alongside the above feature comparison matrix to narrow down your choice.



. . .

Implementations that are worth mentioning

- Apigee
- WSO2
- Zuul

Api Gateway Nginx Kong Service Mesh Compare