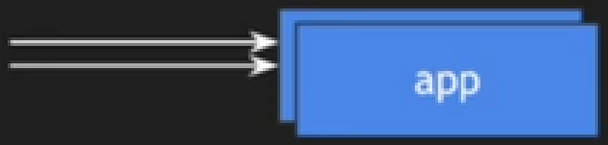
Rate Limiting vs Load Shedding

Microservices/Distributed Systems Strategy

Rate Limiting



GET /products
POST /products
GET /products/{id}
PUT /products/{id}

HTTP 429 Too many requests

Limit the calls per API

client

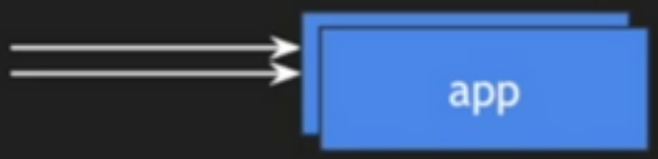
GET	/products	100 RPS
POST	/products	5 RPS
GET	/products/{id}	300 RPS
PUT	/products/{id}	5 RPS

across-clients

GET	/products	3000 RPS
POST	/products	100 RPS
GET	/products/{id}	10000 RPS
PUT	/products/{id}	100 RPS

API level threshold to control incoming requests to the application

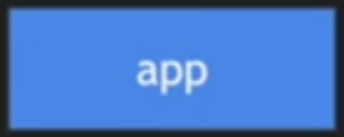
Load Shedding



GET /products
POST /products
GET /products/{id}
PUT /products/{id}

HTTP 503 Server unavailable

Limit the calls per Service



CPU utilization
Concurrent in-flight requests
Resource isolation

Make decisions based on the state of the system rather than the client

Source: Stripe

Rate Limiting

Let's say we can multiple instances of application which can do auto-scaling and we've multiple api's which this particular application is exposing.

We can safe-guard our service so that one particular client cannot consume all our resources at particular service.

We can limit the calls per api per client specific to these api's.

For e.g. : for a individual client we can give only 100 (requests per second) for fetching the data from a particular api . Similarly , we can enforce such restrictions across clients.

Hence , rate limiting can be defined as api level threshold to control incoming requests to the application in a particular moment of time.

Load Shedding

Assume we've 2 services and auto scaling configured. Apart from number of incoming requests, we can limit the calls to the service via parameters such as CPU Utilisation, Resource isolation , and Concurrent in-flight requests.

Google uses CPU Utilisation to identify and control the requests in order to load shed their service to safeguard the things from go wrong.

Facebook uses Concurrent in-flight requests in order to load shed their servers

Amazon Lambda uses Resource isolation in order to segregate and reduce the load shed which is created per service.

These parameters will be at the service level and not at the user level or request level.

If the service degrades, service performance is going to degrade. So the decisions are made on the state of the system rather than the client.

We'll send HTTP 503 Server Unavailable or Retryable for load shedding if its bombarding the system , we can't call the service.

Stripe has identified critical and non critical requests. They try to reduce the non critical requests using load shedding, in this way critical traffic falls within the threshold and can be served as and when required.

When to use?

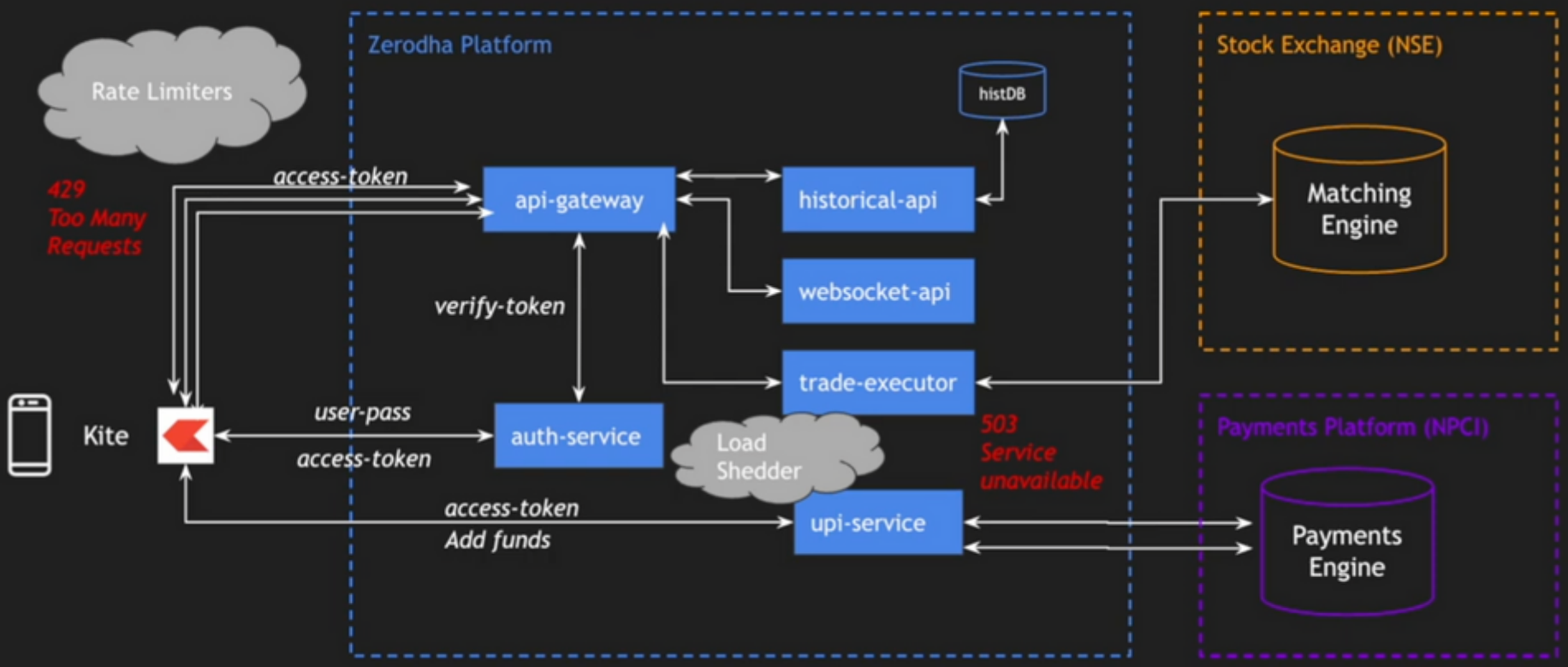
Quality of Service for your APIs

Noisy neighbour

Low priority request vs High priority request

If users can change the pace at which they hit our servers

Zerodha: Trading application



Let's say at Zerodha trading application , we have a platform which connect to the mobile application (in zerodha , it's known to be as Kite.)
And the system is mentioned in the former picture.

Since that was a trading application , there can be multiple requests coming via Kite and there will be a lot of traffic coming via Api gateway and hitting the trade executor or the api.

We can add Rate Limiters at the Api Gateway Layer itself to control the incoming requests via client. We can safeguard the service from more requests hitting the platform.

Rate Limiting Algorithms will help Zerodha if they expose API Gateway to other developers because in addition to their mobile application there are lot of third party api connected to zerodha to pull the real time data via the developer api , so rate limiting can help them reduce the unwanted calls and bringing the servers down.

To make sure that ASG limit isn't exceeded we can define a limit for each and every individual service via concurrent in-flight requests and return 503 to retry the service after a while