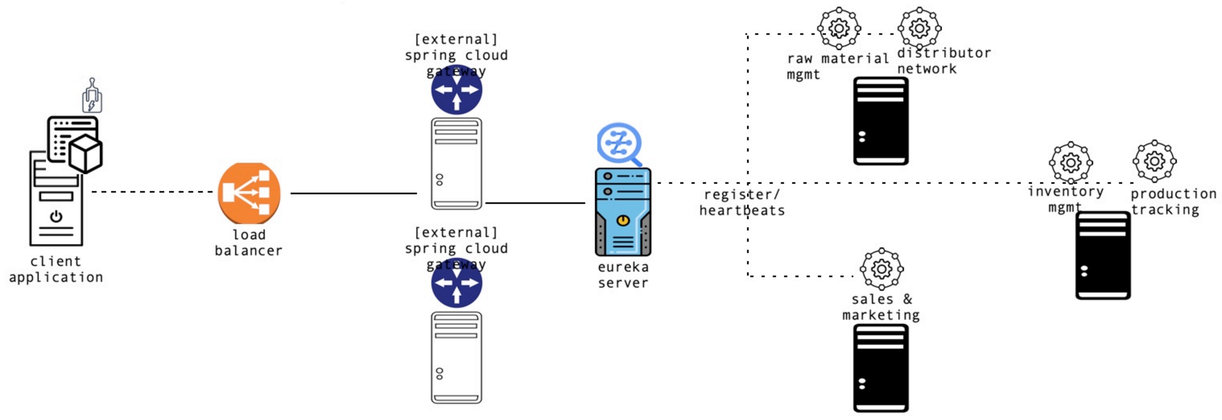
Spring CloudGateway

http://GATEWAYHOST:GATEWAYPORT/D0001/inventory…



Why do we need Gateway?
A big enterprise business system has been broken down into smaller microservices applications. Across these microservices applications we have common requirements that has to be implemented inorder to expose them:
1. oauth security
2. http logging
3. request tracebility
4. caching
5. routing
etc
each of these functionalities can be built into individual microservice applications independently. But the efforts of development and cost involved building these common functionalities into each application would be high and resulted into duplication as well. Inorder to overcome these problems in implementing such common services across the microservices the gateways are introdued.
There are several problems are there that are addressed through gateways as described below.

Problem#1
There are various different client applications built on different technologies like
1. server-side web applications built on java, .net, scala, ruby etc
2. ui technology based web applications build on reactJS, nodeJS, angular etc
all these applications want to access the backend services or microservices to fullfil the functionality. For these applications inorder to access the microservices they need to perform
1. discovery
2. loadbalancing

if every client application inorder to access the microservice has to perform discovery/loadbalancing seems at the client-side seems to be difficult job and may not be feasible for few client types.

So to implement such common functionality across the client applications and to let the clients easily adopt microservices we need to implement api gateway. It acts as an central component for the entire business system. It is an gateway of receiving the requests from the client application

1. all the microservices are exposed over one single URL through the help of api gateway. So the client applications can access the microservices by sending the request to the api gateway exposed URL irrespective of which microservice the client wants to access. The api gateway may perform additional responsibilities inorder to dispatch the request to the appropriate microservice
1. service discovery through eureka server
2. loadbalancing the requests across the nodes of the microservice
For eg.. the client will send the request to api gateway using endpointURL exposed as below
http://GATEWAYHOST:GATEWAYPORT/serviceName/..
upon receiving the request, the gateway looks up for the service the client wants to access based on serviceName within the request
1. if requires goes to eureka server to fetch the instances or nodes where the service is running
2. upon accessing the nodes, it would loadbalance and sends the request to the one of the replicas of the service

---

Problem#2
since all the microservices part of our system are made accessible only through one single-entry point which is gateway, it is easy to enforce common functionalities across the services like
1. security
2. http logging
3. request tracing
4. caching
etc

1. security
rather than implementing the security for each microservice application, we can easily enforce security across all the services at api gateway level. upon api gateway received the request, it will validate whether the client request has posses a valid oauth token or not and if the token is authorized to access then the request would be forwarded to underlying service otherwise would be denied.

2. http logging
each request to any microservice is being received by the api gateway, so it would be easy to central log all the http requests/responses being served that acts as an observability and monitoring metrics

3. request tracing
the api gateway can associate for each request it has received an traceid and propagate along each microservice invocations that are involved in serving the request. so that we can easily trace/debug the failures encountered during the time of serving the request by going through log entries based on that traceid

4. caching
as microservices are distributed across systems that has to be communicated over the network, there is a network latency in serving the client requests that impacts the performance and responsiveness of the overall system.
To address these common performance and latency issues, we can implement caching at api gateway level.

From the above usecases we can easily derive the common requirements in building an api gateway:
1. The api gateway should support routing the incoming requests to an appropriate microservice application (routing request)
2. The api gateway should have control over the requests/responses. Like if security has not been passed, reject the request and return an error response from the gateway itself.
3. The api gateway should be able to modify the request it has received before dispatching the request to the backend microservice or upon receiving the response from the microservice endpoint, it should be able to modify the response before dispatching to the client. (filtering)
4. In addition to support advanced routing strategies like
   For example based on the Locale of the client we want to route the requests to different backend-services like
   The client who is accessing the Sales & Marketing microservice from geographic location India should be routed to backend service IndiaSalesAndMarketing
   The client origin-region: US -> route the request to USSalesMarketing microservice
   here for the give requestURI the backend service to whom we need to route the request is not fixed, we need to route the request based on the request characteristics like queryParam=eq or pathParam=eq or headerParam=value which is called predicate/conditional based routing

From these we can derive #3 typical requirements to be handled by api gateway:
1. routing
2. filtering
3. predicates