Introduction to the API Gateway

In this lesson you will learn about the challenges of using direct client to microservice communication and the multiple benefits of introducing an API Gateway into your microservices architecture.

What is an API Gateway?

What is an API Gateway? In its simplest form, an API gateway is a service that's the entry point into the application REST API from the outside world. It's main concern in a microservices environment is mapping requests from external parties to internal microservices, but it can deal with many other cross cutting concerns too.

To get a better understanding of the API gateway purpose, let's explore a few of the issues you might find when using direct client to microservice communication.

Direct client-to-microservice communication

As you might remember, you configured a load balancer service in your Kubernetes cluster to give a public IP to your Identity microservice, which gives your clients access to it from anywhere. This is working fine for one microservice, but it will present issues for future microservices.

This is because if you add more load balancer services for your other microservices, each one will acquire a new public IP and, since your clients will likely access your microservices using DNS addresses instead of IPs, you will get into the business of mapping all those DNS addresses to IPs.

This clearly doesn't scale and, depending on your cloud provider, each new public IP will introduce additional costs that will only keep climbing with new microservices.

Next, having your microservices directly reachable from the public Internet exposes them to different types of attackers. Some of them may be sending multiple requests without any form of authentication or authorization.

This will never result in the attackers to get unauthorized access to your microservices, since all your REST APIs are properly configured to request authorization. However, it does consume resources that your microservices could be using for something useful.

The attackers might also try things like DDoS attacks, which could overwhelm one or more of your microservices and prevent your regular clients from making good use of them.

You will certainly want to encrypt all communication using https, but since all your microservices are directly exposed to the internet, you will need to setup a TLS certificate for each of them, plus all your microservices will have to decrypt https traffic, which impacts the performance of all of them.

Finally, direct client to microservice communication will introduce undesired coupling. For instance, today your clients can start a purchase using the purchase operation of the Trading microservice and

they can also get a view of the items that can be purchased using the store operation on the same microservice.
Consider what would happen if eventually you decide that the Trading microservice is doing too much and it needs to be split into multiple microservices, with the purchase operation now becoming part of a new Purchase microservice and the store operation now moved into the Store microservice.

Since the Trading microservice no longer exists, all existing clients will be unable to invoke the purchase and store operations and will need to be updated be able to call the new microservices, which is really challenging and might even be impossible in some cases.

Let’s now see how an API gateway can help to tackle these issues.

Using an API Gateway

As mentioned before, once you add an API gateway, it becomes the entry point into your microservices REST API from the outside world. This means that nobody other than the gateway has access to your microservices. This introduces multiple benefits, starting with the key ability to perform request routing.

Let’s say one of your clients wants to get the list of Catalog items. The client invokes the catalog/items path on the gateway and this one in turn routes that request to the items operation in the Catalog microservice. Same thing with trading/store path, which will be routed to the store operation in the Trading microservice.

Notice that thanks to this ability, a public IP is needed only for the API gateway itself, but not for any of the microservices, which will only have internal IPs and DNS addresses inside the Kubernetes cluster.

This will significantly reduce your costs of managing public IPs and keeps things simple for your clients.

Next, your can configure your API gateway to ensure authorized access to your microservices. So, if a client makes a call to the trading/purchase path, and it doesn’t present the required access token, the gateway can initiate the flow that will request an access token via your identity provider and only once the access token has been acquired, it can use it to route the original request to the Trading microservice.

This way none of your microservices have to deal with unauthorized requests, saving their precious resources for more interesting tasks.

The API gateway can also enforce rate limits. So, for instance, if an attacker starts sending a huge amount of request to the identity/users path, turning it into a DDoS attack, the rate limiter component of the gateway can make sure that only the requests that match some criteria can reach your microservices, which makes sure they don’t get overwhelmed and remain available for future requests.

Another important benefit is the ability to perform TLS termination. You can now require all traffic to use HTTPS by installing a TLS certificate only on the API gateway. Then, after the gateway receives the request, it descripts it and sends a new request to the corresponding microservice, this time using

simple http traffic, which is ok because no external entity has access to services inside your Kubernetes cluster.
Since TLS encryption is terminated at the API gateway, the microservices don’t need to spend resources on decrypting traffic anymore, which again saves their resources for more interesting tasks.

The issue of client to microservices coupling is also gone with an API gateway. Let’s come back again at the case of the trading/purchase path that maps to the purchase operation on the Trading microservice, and the trading/store path, which maps to the store operation on the same microservice.

If we eventually decide to split Trading into three new microservices, all you have to do is modify the routing configured in the API gateway so that the purchase and store paths now point to the new microservices. Your clients don’t even need to know that such internal configuration has been modified and they can keep invoking the operations on the same paths as before for as long as needed.

Finally, you could also enable monitoring and logging in your API gateway to get better view of the requests going through your system. For instance, with monitoring you could tell that a request that took 400 milliseconds to process overall, only took 150 milliseconds to get processed by the Trading microservice, leading you to investigate what else could have caused the additional 250 milliseconds of delay.

And with a logging component, you could get a clear record of each of the requests that go through your system, including many details that you could later use for troubleshooting issues.

In the next lesson you will install the HELM tool, in preparation to later install the API Gateway for the Play Economy system.