CONTINUOUS INTEGRATION & CONTINUOUS DELIVERY

newt|global
Aiming At Customer Delight

BUILDING A PROFICIENT MICRO SERVICES USING API GATEWAY

The intent to develop applications using Micro Services fails when we do not decide how the application clients interact with the micro services restful API's. This is not a concern in building a monolithic style application, since there is just one set of endpoints that are typically replicated with a load balancer to distribute traffic and a circuit breaker to monitor traffic. The primary reason for using services as components in a micro service architecture is that services are independently deployable. This means that each Service has to communicate with the Application client each time there is a request.

INTRODUCTION

Consider an example of developing an Order Management application in a Micro Service pattern that will be predominantly used by third party vendors. It is likely that you need to implement an Order detail page that displays the information on any selected product by the vendor. In a Micro Service architecture, the data displayed on the order detail page is owned by multiple micro services. So, each time when there is a request to fetch the product details on the order page, the client communicates with each of the service to display the results with a load balancer in place that distributes requests across the available instances.

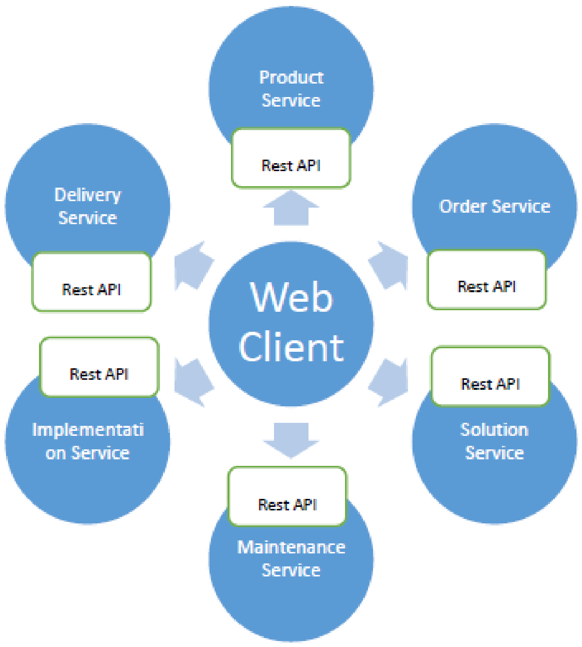

Figure: Mapping the client need to relevant micro services

There are many challenges and limitation with these requests. Services consists of multiple processes that will be developed and deployed together. Some of these services might not be web friendly or might use different messaging protocol that is not browser or firewall friendly, thus eventually resulting as 'no response' whenever there is a call to that particular service. In a complex application the client has to communicate with several separate services during each request that might result in a mismatch between the needs of the client and the API's exposed by each micro service. The granularity of APIs provided by micro services is often different than what a client needs.

PURPOSE OF AN API GATEWAY

An approach to tackle these kind of issues is to use an API Gateway with the micro service architecture. An API Gateway is a single point entry for all client calls. A single API gateway can create multiple api's, one for each platform. For example, we can support mobile applications, multiple browsers, server side applications at a single instance without downtime. An API gateway creates custom API for each of these clients and takes the responsibility of request routing, composition and protocol translation. The gateway essentially aggregates the results to provide a single point of view by invoking multiple micro services. An API gateway works well with Eureka server and when Spring cloud supports other discovery client then those as well. Other advantages with API gateway in terms of centralization and control: rate limiting, authentication, auditing, logging and implementing a simple reverse proxy with Spring Cloud.

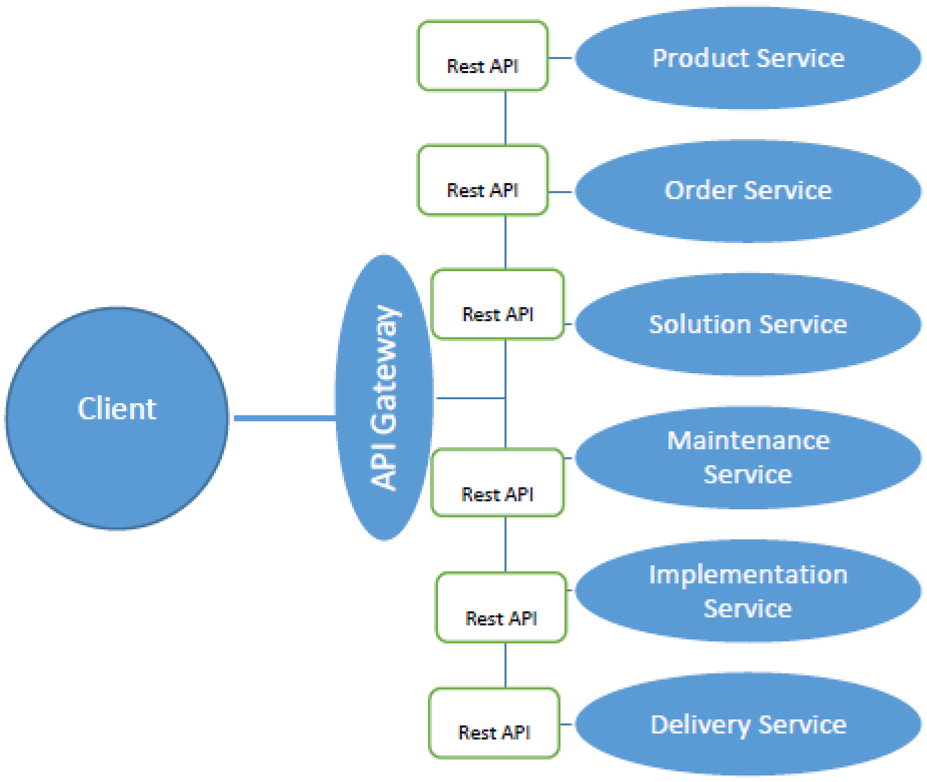

Fig.: Using an API gateway with micro services

There are many challenges and limitation with these requests. Services consists of multiple processes that will be developed and deployed together. Some of these services might not be web friendly or might use different messaging protocol that is not browser or firewall friendly, thus eventually resulting as ‘no response’ whenever there is a call to that particular service. In a complex application the client has to communicate with several separate services during each request that might result in a mismatch between the needs of the client and the API’s exposed by each micro service. The granularity of APIs provided by micro services is often different than what a client needs.

TESTING API’S

What may be the dream of the API delivery team is to have bug free API gateway. As one might expect, using an API gateway has both benefits and drawbacks. One of the major benefit is the encapsulation process. The API gateway should never become a developer’s bottle neck in terms of deployment and maintenance. It is always important that the gateway is as light as possible and best practice is to create API virtualization at the time of development, so that the testing team can start testing the API during the process. API virtualization simulates the minimum behaviors of one or more API endpoints instead of API production. This enables frequent and comprehensive testing while the Development is still under process. This is similar to creating a Sandbox environment for the testing team to access the data and server.

Another way of testing is by creating API mocks that imitates the software components used by the Developers. However, this is a traditional method of testing when virtualization is not possible.

CONCLUSION

It makes sense to have an API gateway developed for most of the applications that are built on a Micro Service architecture, as the main objective is to make sure a single unresponsive service should not fail the entire request. The main feature of the API gateway is that it provides a custom API for each of the application's clients that calls for different platform. Though it is additional effort to develop an API gateway, the long term sustainability and benefits overcomes the barrier of using Micro Service architecture and ensures complex applications run on a simple back end process. Access Management, Authentication & Validation, Manageability & Resilience and the ease of integrating with third party tools are some of the parameters that need to consider to choose a reliable API gateway.