Department of Information Engineering and Computer Science

Fog and Cloud Computing

Group 07

Project architecture draft

A scalable federated architecture

Academic year 2020/2021

2021/05/04

Student
Carlo Corradini
223811

Student
Giovanni Zotta
223898

1 Mandatory fields

- Course: Fog and Cloud Computing 2021
- Architecture: A scalable federated architecture
- Group Number: 07
- Authors:
  - Corradini, Carlo carlo.corradini@studenti.unitn.it
  - Zotta, Giovanni giovanni.zotta@studenti.unitn.it

2 Idea

Our idea is to have a GraphQL[1] web service that exposes a single public endpoint. The endpoint is managed by the Apollo Gateway[2], which is responsible to answer all the queries made by the final users by combining the microservices that are hidden in the backend.

To be more precise, the gateway formally is not a worker node since it should only receive queries and understand how to gather the needed information by querying the hidden services that can provide that information. The other microservices present in our architecture are workers that process requests as they come, and are able to communicate with the gateway in case they need information from other services.

A possible representation of the idea is shown in Figure 2.1. We have an online shop that has a

[1] https://graphql.org/

[2] https://www.apollographql.com/docs/federation/gateway

Products service, a Reviews service and a Users service. Each service is specialized to only resolve queries related to its context.

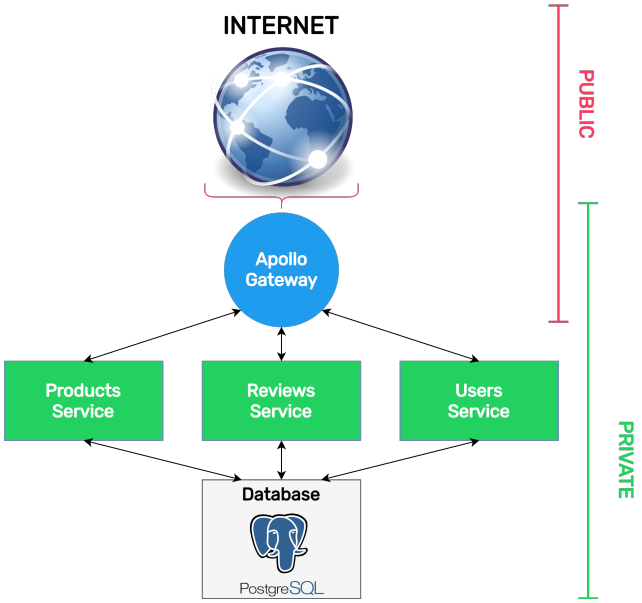

Figure 2.1: High level representation of the architecture

```
query GetCurrentUserReviews {
  me {
    username
    reviews {
      body
      product {
        name
      }
    }
  }
}
```

Figure 2.2: GraphQL query to obtain the current user's reviews

As an example of a complex query take a look at Figure 2.2. The current user requests: username, all reviews and for each review the body and the correlated product information.

The gateway needs to retrieve information from all the three services, since it will need to know all of the products that the user bought along with the associated reviews.

The implementation of this architecture is called Apollo Federation[3].

This kind of web service is keen to work well as a scalable architecture. Indeed, every component of our idea can ideally be scaled horizontally as much as needed, with a little caveat on the database. Following the pet-cattle scheme, we will probably treat all of our services as cattles, apart from the database in order to avoid to deal with consistency concerns. We think we will make the database scale only vertically for the scope of this project.

We think that this model of web service would fit well as a project topic, as it allows us to experiment with various different components that we have learnt during the IaaS and PaaS labs.

3 Architecture

Since the gateway is the most heavily used component of our architecture, it needs to be able to scale dynamically depending on the workload. This holds for all the components of our architecture, but it

[3]https://www.apollographql.com/docs/federation/

is especially true for the gateway, since it would otherwise be a single-point-of-failure.

We will mainly talk about Kubernetes for the moment. Every component shown in Figure 3.1 apart from the database will be a Service (in Kubernetes terms) composed of a Deployment which will provide declarative updates to a Replica Set of different Pods, all running the image of their respective service. Thanks to this architecture, it would be possible to have an highly scalable and fault-tolerant cluster. Another reason to use Services and Deployments is to be able to separate environments via Namespaces and Labels to be able to have rolling-updates, for example it would be possible to spin up a development version of our services in production without any significant downtime.

Every Service will be hosted in a Node, and will be upscaled or downscaled by the Control Plane when needed. If we are able to setup this Kubernetes cluster, we would like to integrate it with OpenStack if time allows. The final goal would be to be able to host every node on an OpenStack instance, for which we would need to setup an OpenStack project and understand which images and flavors would suit our goals in the best way.

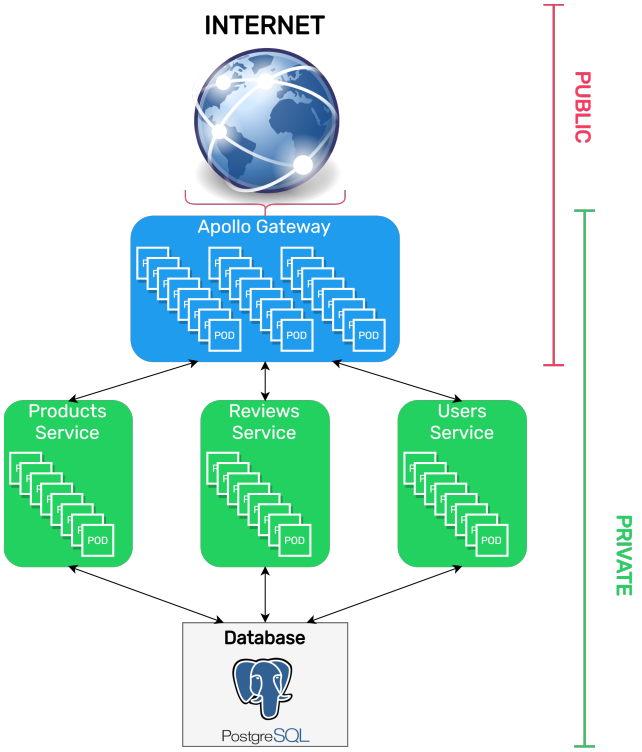

Figure 3.1: Representation of our architecture

For completeness, the Gateway must expose a Playground (in combination with the API endpoint) at */playground*. The GraphQL Playground[1] is a website that can be used to interact directly with the GraphQL Apollo Gateway. The Playground has multiple features but one of the most useful is the query autocomplete action made possible by reading the merged IDL (Interface Description Language)[2] of the overall architecture.

[1]https://github.com/graphql/graphql-playground
[2]https://en.wikipedia.org/wiki/Interface_description_language