UNIVERSITÀ DI PARMA

AUTENTICAZIONE TRAMITE TOKEN IN UN'ARCHITETTURA A MICROSERVIZI: REALIZZAZIONE DI UN PLUGIN CUSTOM PER L'API GATEWAY KONG

Relatore:
Prof. Roberto Alfieri

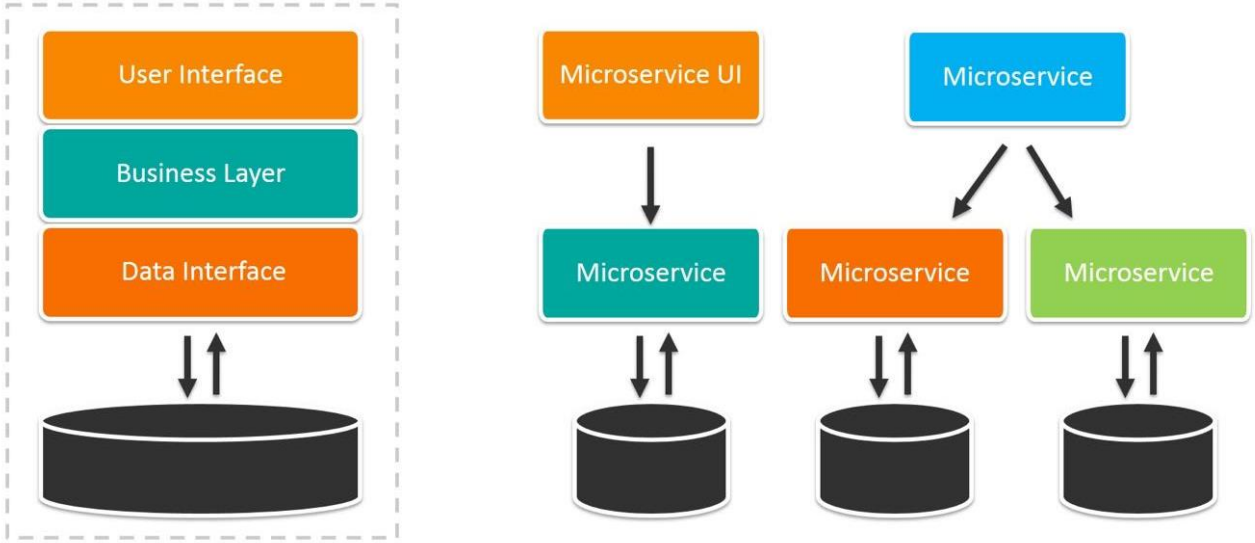
Correlatore:
Gregorio Palamà

Anno Accademico
2022 – 2023

Candidato:
Valerio Desiati

Cosa sono i microservizi

Nelle architetture a microservizi l'obiettivo è quello di **scomporre l'applicazione** da realizzare nelle sue funzioni (**servizi**) di base.
Ogni servizio può essere compilato e distribuito in modo **indipendente**; quindi i singoli servizi possono funzionare o non funzionare senza compromettere gli altri.

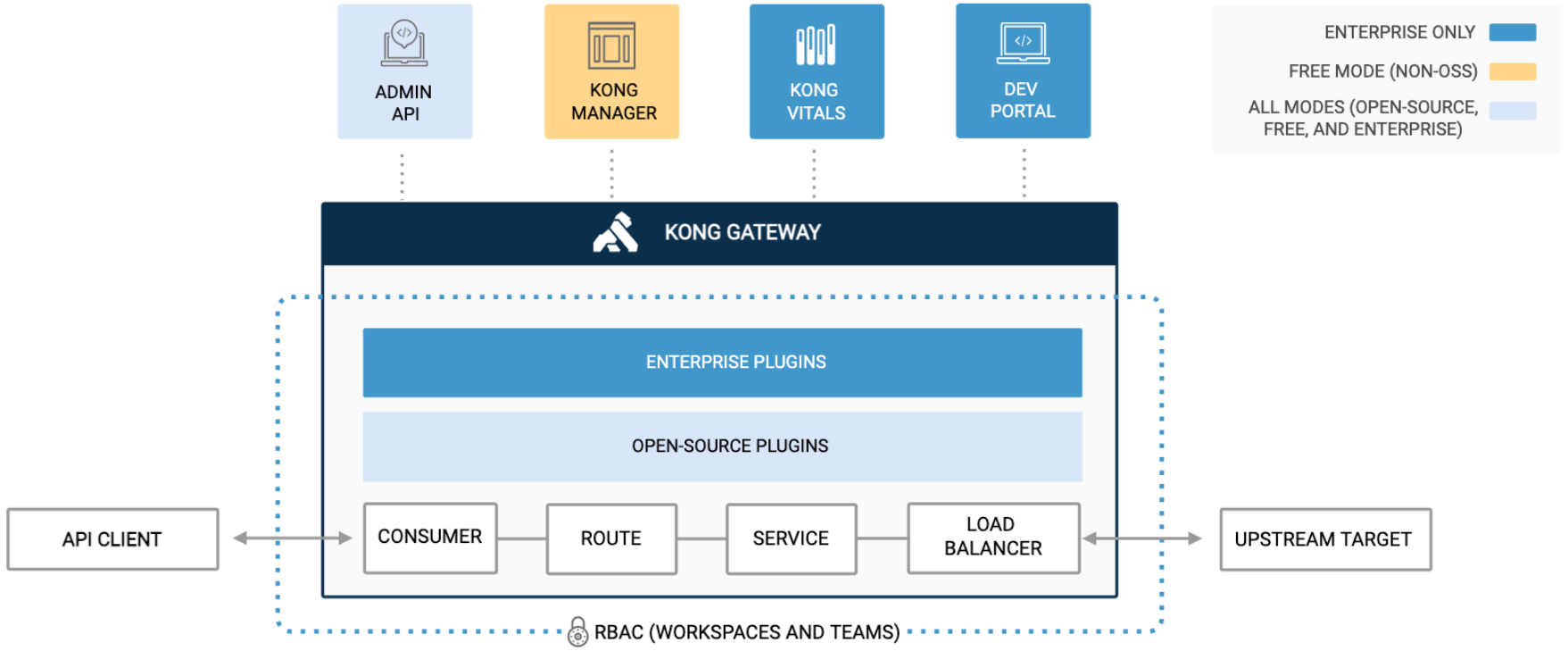


Confronto tra architettura monolitica e a microservizi

Caratteristiche dei microservizi

Cos'è Kong Gateway API?

Kong Gateway è un API gateway cloud-native che si interpone tra un client e un back – end per la **gestione delle API** che si comporta come un **proxy inverso** accettando tutte le richieste indirizzate alle API gestite, consentendo di configurare **services**, **routes** e **plugin**.



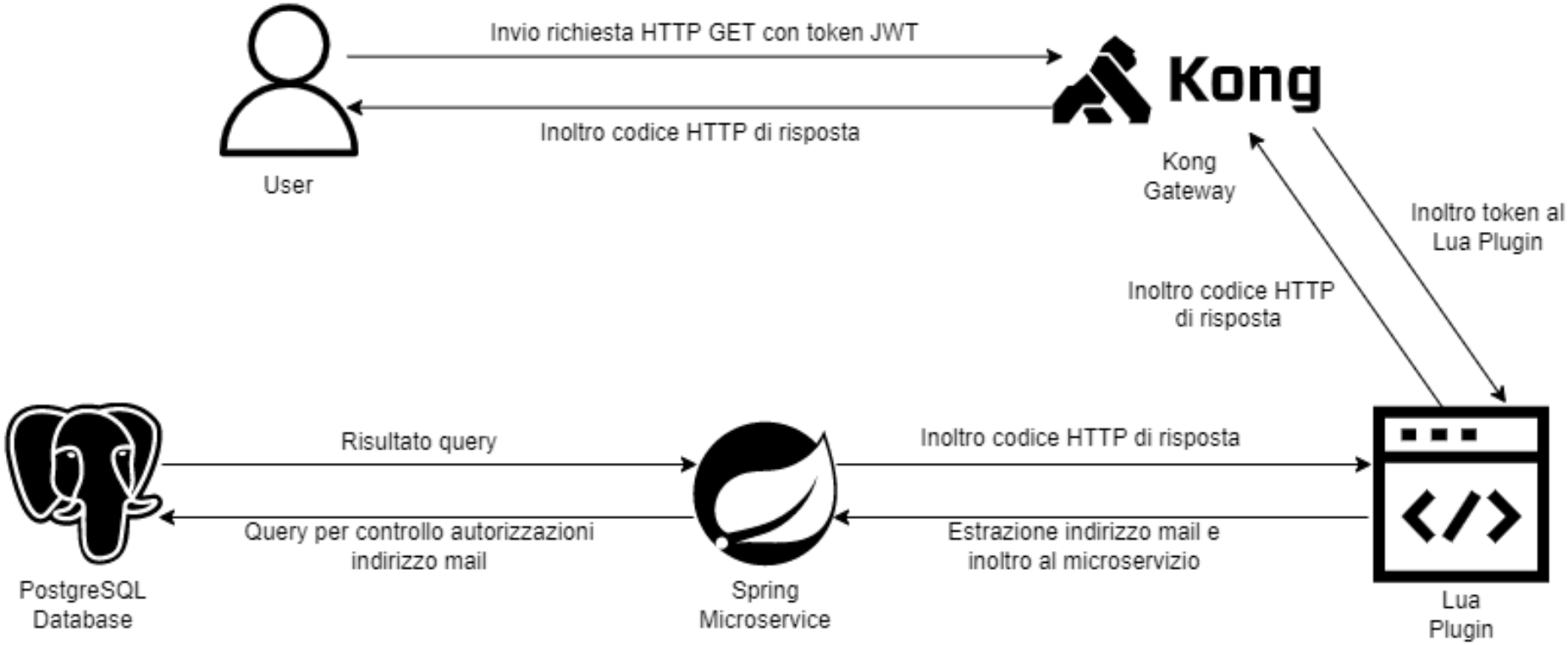
Scopo del progetto

L'obiettivo del progetto è quello di realizzare un software che consenta l'autenticazione di un utente all'interno di una piattaforma generica tramite un token da includere all'interno della richiesta inviata, senza quindi l'utilizzo di password o altri strumenti di autenticazione.

Il progetto è strutturato in 4 macro – componenti:

- Database
- Microservizio in Java utilizzando Spring Framework
- Kong Gateway
- Plugin custom per Kong Gateway in Lua

Schema delle comunicazioni



Struttura del Database PostgreSQL

Contiene due tabelle:

- `users`
  Contiene le informazioni di tutti gli utenti abilitati ad accedere ad un determinato servizio.

- `email`
  Contiene gli indirizzi e-mail di tutti gli utenti della piattaforma, quindi non è garantito che tutti avranno accesso a tutti i servizi.

email	
PK	**id integer SERIAL NOT NULL**
	email text NOT NULL

users	
PK	**id integer SERIAL NOT NULL**
	email text NOT NULL name text surname text

Modello Entità – Relazione del Database

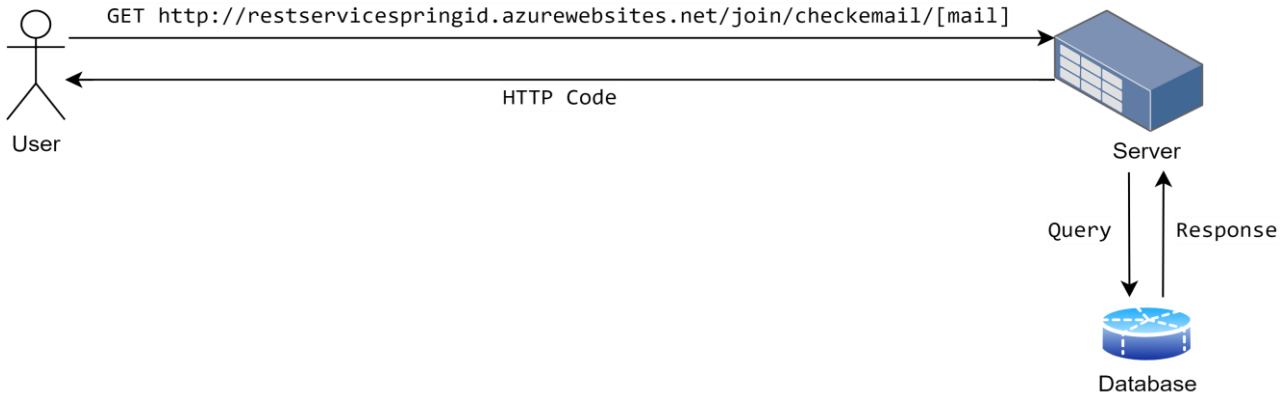
Obiettivi del microservizio

Lo scopo principale del microservizio è quello di ricevere una richiesta HTTP con all'interno del body un indirizzo e-mail.

Si effettua una query all'interno del Database per controllare che l'indirizzo sia presente nella tabella degli utenti abilitati all'utilizzo del servizio.

Si restituisce infine il codice HTTP adatto.

Codice HTTP	Significato
200	OK
402	PAYMENT REQUIRED
500	INTERNAL SERVER ERROR
503	SERVICE UNAVAILABLE



La classe `main`

La classe principale del progetto è composta da un solo comando, che si occupa di avviare l'applicazione.
Il metodo `run()` è il metodo principale di Spring Framework, serve ad avviare tutti i processi necessari per l'esecuzione dell'applicazione.

```java
package com.aesys.valeriodesiati.mail;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class EmailApplication {

    public static void main(String[] args) {
        SpringApplication.run(EmailApplication.class, args);
    }

}
```

Le classi `@Entity`

Una classe annotata come `@Entity` indica che questa servirà per la creazione di una tabella all'interno del Database specificato, utilizzando come campi gli attributi della classe.
Si noti come ad una normale classe Java siano aggiunte solo le annotazioni Spring:

- `@Id` e `@GeneratedValue()` per definire l'attributo come id autogenerato
- `@Column()` per definire il nome di un campo (colonna) della tabella ed eventuali altre proprietà

```java
package com.aesys.valeriodesiati.mail.model;
//imports..
@Entity
@Table (name="users")
public class Users {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int id;
    @Column(name = "email", nullable = false)
    private String email;
    @Column(name = "name")
    private String name;
    @Column(name = "surname")
    private String surname;

    public Users(int id, String email, String name, String surname) {
        this.id = id;
        this.email = email;
        this.name = name;
        this.surname = surname;
    }

    //setters, getters, toString()...
}
```

La classe `JoinController.java`

```java
package com.aesys.valeriodesiati.mail.controller;
//imports..
@RestController
public class JoinController {
    @Autowired
    EntityManagerFactory emf;

    @Transactional
    @GetMapping("/join/checkemail/{mail}")
    public ResponseEntity<Integer> checkEmail(@PathVariable("mail") String mail) {
        EntityManager em = emf.createEntityManager();
        em.getTransaction().begin();
        String result = null;

        try {
            result = (String) em.createQuery("SELECT u.email FROM Users u, Email e WHERE u.email = e.email AND u.email = :email")
                                .setParameter("email", mail).getSingleResult();
        }
        catch(NoResultException | NullPointerException e) {
            return ResponseEntity.status(HttpStatus.PAYMENT_REQUIRED).body(402);
        }
        if(result == null)
            return ResponseEntity.status(HttpStatus.PAYMENT_REQUIRED).body(402);
        else
            return ResponseEntity.status(HttpStatus.OK).body(200);
    }
}
```

Obiettivi del plugin Lua

È richiesto il seguente comportamento dal plugin:

1. Analizzare il token ricevuto

```
local function SplitToken(token)
        local segments = {}
        for str in string.gmatch(token, "([^\\.]+)") do
                table.insert(segments, str)
        end
        return segments
end
```

Obiettivi del plugin Lua

È richiesto il seguente comportamento dal plugin:

1. Analizzare il token ricevuto
2. Parse del token

```lua
local function ParseToken(token)
        local segments = SplitToken(token)
        if #segments ~= 3 then
                return nil, nil, nil, "Invalid token"
        end

        local header, err = cjson_safe.decode(basexx.from_url64(segments[1]))
        if err then
                return nil, nil, nil, "Invalid header"
        end

        local body, err = cjson_safe.decode(basexx.from_url64(segments[2]))
        if err then
                return nil, nil, nil, "Invalid body"
        end

        local sig, err = basexx.from_url64(segments[3])
        if err then
                return nil, nil, nil, "Invalid signature"
        end

        return header, body, sig
end
```

Obiettivi del plugin Lua

È richiesto il seguente comportamento dal plugin:

1. Analizzare il token ricevuto
2. Parse del token
3. Inviare una richiesta HTTP al microservizio CheckEmail

```
local body, code, headers, status =
                    http.request("http://restservice-springid.azurewebsites.net
                                 /join/checkemail/"..bodyTok.email)
```

Obiettivi del plugin Lua

È richiesto il seguente comportamento dal plugin:

1. Analizzare il token ricevuto
2. Parse del token
3. Inviare una richiesta HTTP al microservizio CheckEmail
4. Ottenere risposta dal microservizio e inoltrarla al Gateway

```lua
if code == 200 then
    return kong.response.exit(200, "Success")
end

if code == 402 then
    return kong.response.error(402, "Payment Required")
end

    --response codes 500 and 503...
```

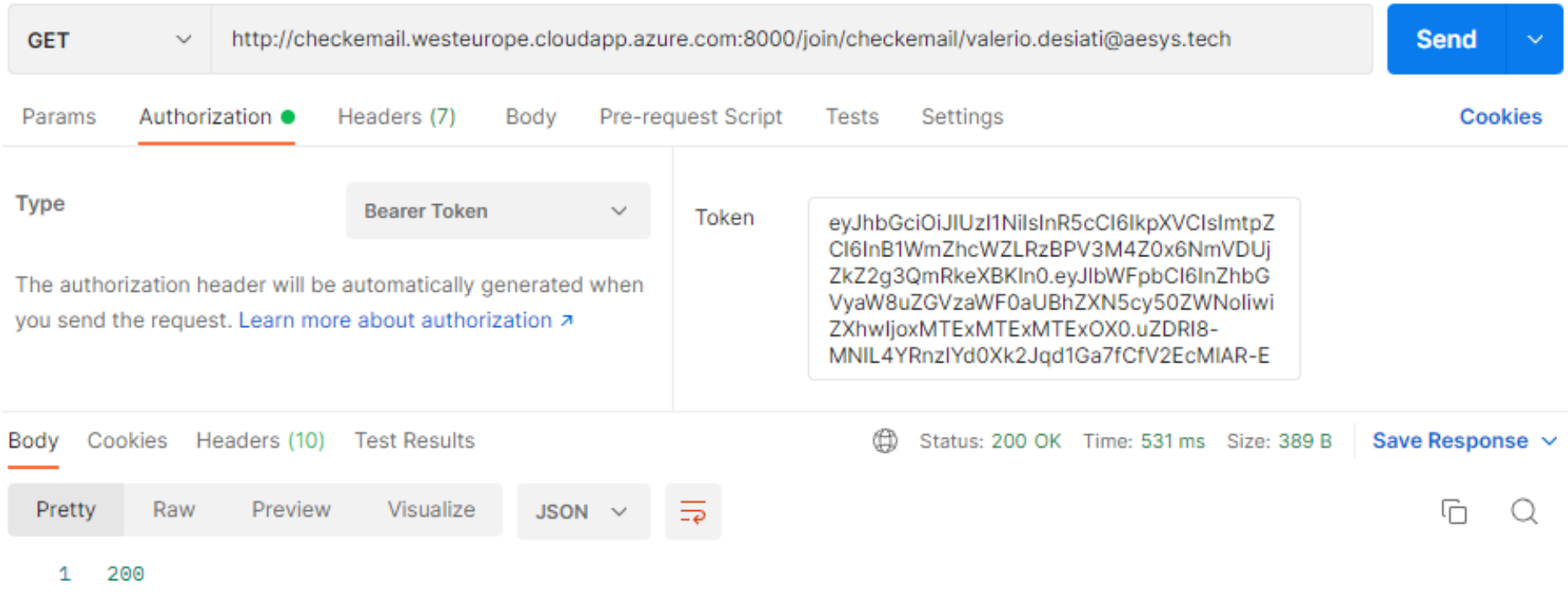
Script di configurazione

Tutta la configurazione è effettuata tramite lo script bash **`addplugin`** che ha i seguenti compiti:

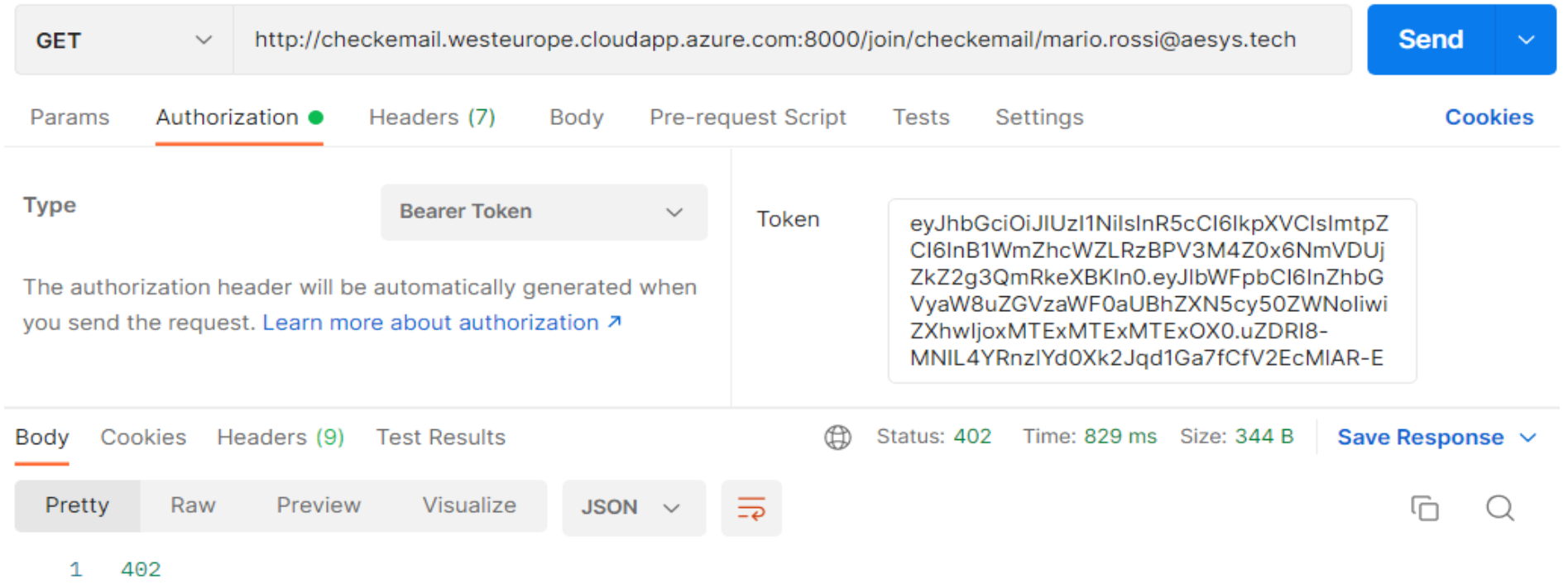
1. Avviare il container
2. Copiare al suo interno i file relativi ai plugin custom
3. Riavviare il container
4. Eseguire il comando **`curl`** per tutti i file JSON presenti relativi a tutte le altre configurazioni necessarie

```
sudo docker exec -it --user root $container rm -rf
                              /usr/local/share/lua/5.1/kong/plugins/$dir
sudo docker exec -it --user root $container mkdir
                              /usr/local/share/lua/5.1/kong/plugins/$dir
sudo docker cp . $container:/usr/local/share/lua/5.1/kong/plugins/$dir/
curl -s -X POST -H "Content-Type: application/json"
    -d @./config/checkemail/services.json
    http://checkemail.westeurope.cloudapp.azure.com:8001/services > /dev/null
```

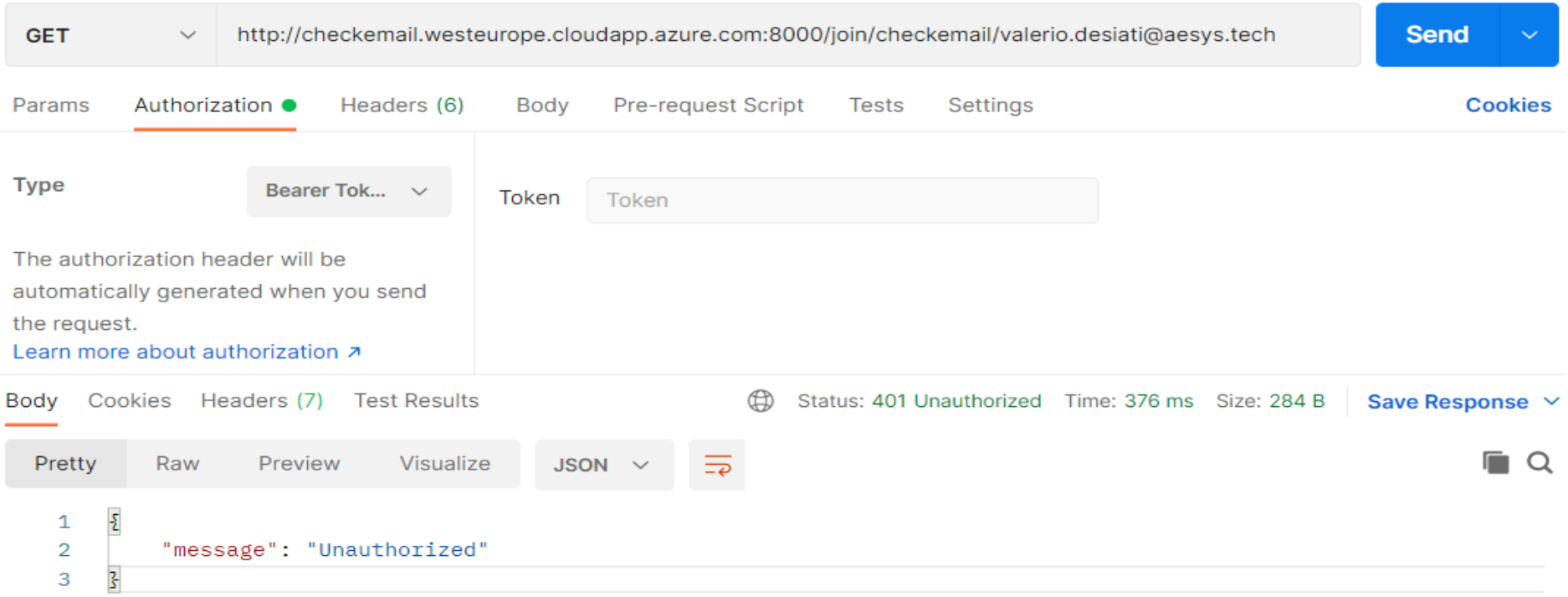
Risultati – OK



Risultati – Payment Required



Risultati – Unauthorized



UNIVERSITÀ DI PARMA

GRAZIE PER L'ATTENZIONE!