EBook Gratuito

APPRENDIMENTO microservices

Free unaffiliated eBook created from
Stack Overflow contributors.

#microservices

Sommario

Di

Capitolo 1: Iniziare con i microservizi

Osservazioni

Questa sezione fornisce una panoramica dei microservizi e del motivo per cui uno sviluppatore potrebbe volerlo utilizzare.

Dovrebbe anche menzionare qualsiasi argomento di grandi dimensioni all'interno dei microservizi e collegarsi agli argomenti correlati. Poiché la Documentazione per i microservizi è nuova, potrebbe essere necessario creare versioni iniziali di tali argomenti correlati.

Examples

Checklist essenziale per piattaforma Microservices

- Pipeline CI / CD
- Autenticazione centralizzata e servizio di autorizzazione
- Documentazione API
- Gateway API
- Centralizza lo strumento di gestione dei log
- Monitor di servizio
- Automazione dell'infrastruttura
- Server di configurazione centralizzato

Documentazione API

Usa **Spring REST Docs** per documentare i tuoi servizi. È un potente framework che garantisce che la logica del servizio sia sempre in linea con la documentazione. Per fare ciò, dovresti scrivere test di integrazione per i tuoi servizi.

In caso di discrepanza nella documentazione e nel comportamento del servizio, i test falliranno.

Ecco un esempio di esempio per generare i documenti per un progetto maven.

Aggiungi questa dipendenza al tuo pom:

```
<dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-rest-webmvc</artifactId>
    <version>2.6.6.RELEASE</version>
</dependency>
```

e aggiungi il plugin asciidoc sotto il tag build.plugins:

```
<plugin>
        <groupId>org.asciidoctor</groupId>
        <artifactId>asciidoctor-maven-plugin</artifactId>
```

```
        <version>1.5.3</version>
        <executions>
            <execution>
                <id>generate-docs</id>
                <phase>prepare-package</phase>
                <goals>
                    <goal>process-asciidoc</goal>
                </goals>
                <configuration>
                    <backend>html</backend>
                    <doctype>book</doctype>
                </configuration>
            </execution>
        </executions>
        <dependencies>
            <dependency>
                <groupId>org.springframework.restdocs</groupId>
                <artifactId>spring-restdocs-asciidoctor</artifactId>
                <version>1.2.0.RELEASE</version>
            </dependency>
        </dependencies>
    </plugin>
```

Ora prendiamo un controller di esempio che vogliamo documentare:

```
package com.hospital.user.service.controller;

import org.springframework.hateoas.Resource;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

import com.hospital.user.service.entity.User;
import com.hospital.user.service.exception.UserNotFoundException;
import com.hospital.user.service.repository.UserCrudRepository;
import com.hospital.user.service.resource.assembler.UserResourceAssembler;

@Controller
@RequestMapping("/api/user")
public class SampleController {


final UserCrudRepository userRepository;


final UserResourceAssembler userResourceAssembler;

final BCryptPasswordEncoder  passwordEncoder;

public SampleController(UserCrudRepository userCrudRepository, UserResourceAssembler
userResourceAssembler, BCryptPasswordEncoder passwordEncoder){
    this.userRepository = userCrudRepository;
    this.userResourceAssembler = userResourceAssembler;
    this.passwordEncoder = passwordEncoder;
}
```

```
@RequestMapping(method = RequestMethod.GET, value = "/{userId}", produces = {
MediaType.APPLICATION_JSON_VALUE})
ResponseEntity<Resource<User>> getUser(@PathVariable String userId){
    User user = (User) this.userRepository.findOne(userId);
    if(user==null){
        throw new UserNotFoundException("No record found for userid"+ userId);
    }
    Resource<User> resource = this.userResourceAssembler.toResource(user);
    return new ResponseEntity<Resource<User>>(resource, HttpStatus.OK);
 }
}
```

Ora scrivi un test per il servizio:

```
package com.hospital.user.service;

import org.junit.Before;
import org.junit.Rule;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.restdocs.JUnitRestDocumentation;
import org.springframework.restdocs.mockmvc.RestDocumentationResultHandler;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
import org.springframework.test.context.web.WebAppConfiguration;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.setup.MockMvcBuilders;
import org.springframework.web.context.WebApplicationContext;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;


import static org.springframework.restdocs.operation.preprocess.Preprocessors.prettyPrint;
import static
org.springframework.restdocs.operation.preprocess.Preprocessors.preprocessRequest;
```

import static
org.springframework.restdocs.operation.preprocess.Preprocessors.preprocessResponse;

```
import static org.springframework.restdocs.mockmvc.MockMvcRestDocumentation.document;
import static
org.springframework.restdocs.mockmvc.MockMvcRestDocumentation.documentationConfiguration;

import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;

import static org.springframework.restdocs.mockmvc.RestDocumentationRequestBuilders.get;

import static org.springframework.restdocs.headers.HeaderDocumentation.headerWithName;
import static org.springframework.restdocs.headers.HeaderDocumentation.responseHeaders;

import static org.springframework.restdocs.payload.PayloadDocumentation.fieldWithPath;
import static org.springframework.restdocs.payload.PayloadDocumentation.responseFields;

@RunWith(SpringJUnit4ClassRunner.class)
@WebAppConfiguration
@EnableWebMvc
```

```
@ComponentScan( basePackages = { "com.hospital.user.service" } )
@SpringBootTest
public class SampleControllerTest {

private RestDocumentationResultHandler documentationHandler;

@Rule public final JUnitRestDocumentation restDocumentation = new
JUnitRestDocumentation("target/generated-snippets");

 @Autowired private WebApplicationContext context;
    private MockMvc mockMvc;

    @Before
    public void setUp(){

        this.documentationHandler = document("{method-name}", //this will create files with
the test method name
                preprocessRequest(prettyPrint()), // to print the request
                preprocessResponse(prettyPrint()));

            this.mockMvc = MockMvcBuilders.webAppContextSetup(this.context)
                .apply(documentationConfiguration(this.restDocumentation)
                        .uris()
                        //.withScheme("https") Specify this for https
                        .withHost("recruitforceuserservice") //Define the host name
                        .withPort(8443)
                    )
                .alwaysDo(this.documentationHandler)
                .build();
    }

    @Test
    public void getUser() throws Exception {
            // tag::links[]

this.mockMvc.perform(get("/api/user/"+"591310c3d5eb3a37183ab0d3").header("Authorization",
                    "Bearer
eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJiaGFyZHdhai5uaXRpc2gxOSIsInJvbGVzIjpbIkFETUlOIl0sImlzcyI6Imh0dHA6Ly9ob

                .andExpect(status().isOk())
                .andDo(this.documentationHandler.document(

                        responseHeaders(
                                headerWithName("Content-Type").description("The Content-Type
of the payload: `application/json`") // Asserts that the response should have this header.
                                ),

                        responseFields(
                                fieldWithPath("username").description("Unique name for the
record"), // Asserts that the response should have this field
                                fieldWithPath("password").description("password of the user"),
                                fieldWithPath("securityAnswer").description("Security answer
which would be used to validate the user while the password is reset."),
                                fieldWithPath("securityQuestion").description("Security
question to reset the password"),
                                fieldWithPath("email").description("Email of the user"),
                                fieldWithPath("roles").description("Assigned roles of the
user"),
                                fieldWithPath("id").description("Unique identifier of an
user"),
                                fieldWithPath("_links").ignored()
```

```
                        )
                        ));
    }
```

}

Si prega di seguire questo riferimento per maggiori dettagli: http://docs.spring.io/spring-restdocs/docs/current/reference/html5/

Esempio per la documentazione API

Usa **Spring REST Docs** per documentare i tuoi servizi. È un potente framework che garantisce che la logica del servizio sia sempre in linea con la documentazione. Per fare ciò, dovresti scrivere test di integrazione per i tuoi servizi.

In caso di discrepanza nella documentazione e nel comportamento del servizio, i test falliranno.

Ecco un esempio di esempio per generare i documenti in un progetto maven:

Aggiungi questa dipendenza nel file pom:

```
<dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-rest-webmvc</artifactId>
    <version>2.6.6.RELEASE</version>
</dependency>
```

Inoltre, aggiungi il plugin per documenti ASCII per generare documenti sotto il tag build.puglin

```
<plugin>
        <groupId>org.asciidoctor</groupId>
        <artifactId>asciidoctor-maven-plugin</artifactId>
        <version>1.5.3</version>
        <executions>
            <execution>
                <id>generate-docs</id>
                <phase>prepare-package</phase>
                <goals>
                    <goal>process-asciidoc</goal>
                </goals>
                <configuration>
                    <backend>html</backend>
                    <doctype>book</doctype>
                </configuration>
            </execution>
        </executions>
        <dependencies>
            <dependency>
                <groupId>org.springframework.restdocs</groupId>
                <artifactId>spring-restdocs-asciidoctor</artifactId>
                <version>1.2.0.RELEASE</version>
            </dependency>
        </dependencies>
    </plugin>
```

Ora, come esempio, creiamo un servizio controller che vogliamo documentare.

```
package com.hospital.user.service.controller;

import org.springframework.hateoas.Resource;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

import com.hospital.user.service.entity.User;
import com.hospital.user.service.exception.UserNotFoundException;
import com.hospital.user.service.repository.UserCrudRepository;
import com.hospital.user.service.resource.assembler.UserResourceAssembler;

@Controller
@RequestMapping("/api/user")
public class SampleController {


final UserCrudRepository userRepository;


final UserResourceAssembler userResourceAssembler;

final BCryptPasswordEncoder  passwordEncoder;

public SampleController(UserCrudRepository userCrudRepository, UserResourceAssembler
userResourceAssembler, BCryptPasswordEncoder passwordEncoder){
    this.userRepository = userCrudRepository;
    this.userResourceAssembler = userResourceAssembler;
    this.passwordEncoder = passwordEncoder;
}

@RequestMapping(method = RequestMethod.GET, value = "/{userId}", produces = {
MediaType.APPLICATION_JSON_VALUE})
ResponseEntity<Resource<User>> getUser(@PathVariable String userId){
    User user = (User) this.userRepository.findOne(userId);
    if(user==null){
        throw new UserNotFoundException("No record found for userid"+ userId);
    }
    Resource<User> resource = this.userResourceAssembler.toResource(user);
    return new ResponseEntity<Resource<User>>(resource, HttpStatus.OK);
}
```

}

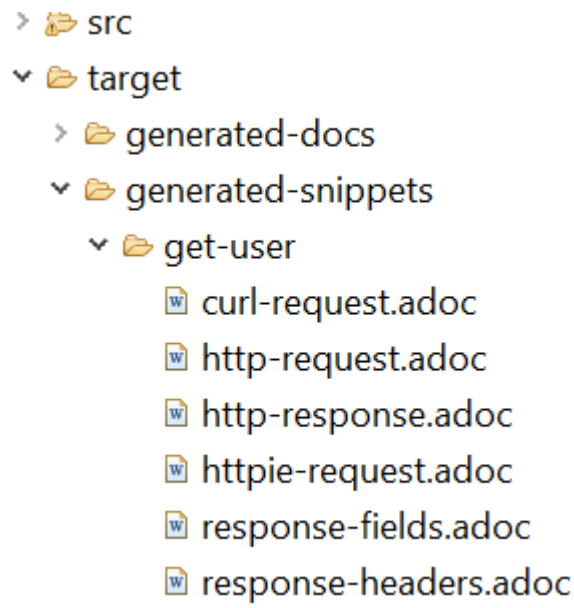
Scriviamo un test per testare questo servizio:

```
package com.hospital.user.service;

import org.junit.Before;
import org.junit.Rule;
import org.junit.Test;
import org.junit.runner.RunWith;
```

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.restdocs.JUnitRestDocumentation;
import org.springframework.restdocs.mockmvc.RestDocumentationResultHandler;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
import org.springframework.test.context.web.WebAppConfiguration;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.setup.MockMvcBuilders;
import org.springframework.web.context.WebApplicationContext;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;


   import static org.springframework.restdocs.operation.preprocess.Preprocessors.prettyPrint;
import static
org.springframework.restdocs.operation.preprocess.Preprocessors.preprocessRequest;
import static
org.springframework.restdocs.operation.preprocess.Preprocessors.preprocessResponse;

import static org.springframework.restdocs.mockmvc.MockMvcRestDocumentation.document;
import static
org.springframework.restdocs.mockmvc.MockMvcRestDocumentation.documentationConfiguration;

import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;

import static org.springframework.restdocs.mockmvc.RestDocumentationRequestBuilders.get;

import static org.springframework.restdocs.headers.HeaderDocumentation.headerWithName;
import static org.springframework.restdocs.headers.HeaderDocumentation.responseHeaders;

import static org.springframework.restdocs.payload.PayloadDocumentation.fieldWithPath;
import static org.springframework.restdocs.payload.PayloadDocumentation.responseFields;

@RunWith(SpringJUnit4ClassRunner.class)
@WebAppConfiguration
@EnableWebMvc
@ComponentScan( basePackages = { "com.hospital.user.service" } )
@SpringBootTest
public class SampleControllerTest {

private RestDocumentationResultHandler documentationHandler;

@Rule public final JUnitRestDocumentation restDocumentation = new
JUnitRestDocumentation("target/generated-snippets");

 @Autowired private WebApplicationContext context;
    private MockMvc mockMvc;

    @Before
    public void setUp(){

        this.documentationHandler = document("{method-name}", //Documents would be generated
by the test method name.
                preprocessRequest(prettyPrint()), //To print request
                preprocessResponse(prettyPrint()));

            this.mockMvc = MockMvcBuilders.webAppContextSetup(this.context)
                .apply(documentationConfiguration(this.restDocumentation)
                        .uris()
                        //.withScheme("https") Specify this for https
                        .withHost("recruitforceuserservice") //To use the hostname
```

```
                        .withPort(8443)
                    )
                .alwaysDo(this.documentationHandler)
                .build();
    }

    @Test
    public void getUser() throws Exception {
            // tag::links[]

this.mockMvc.perform(get("/api/user/"+"591310c3d5eb3a37183ab0d3").header("Authorization",
                    "Bearer
eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJiaGFyZHdhai5uaXRpc2gxOSIsInJvbGVzIjpbIkFETUlOIl0sImlzcyI6Imh0dHA6Ly9ob

                .andExpect(status().isOk())
                .andDo(this.documentationHandler.document(

                        responseHeaders(
                                headerWithName("Content-Type").description("The Content-Type
of the payload: `application/json`")//Asserts that the response has this header.
                                ),

                        responseFields(
                                fieldWithPath("username").description("Unique name for the
record"), //Asserts that the response has this field.
                                fieldWithPath("password").description("password of the user"),
                                fieldWithPath("securityAnswer").description("Security answer
which would be used to validate the user while the password is reset."),
                                fieldWithPath("securityQuestion").description("Security
question to reset the password"),
                                fieldWithPath("email").description("Email of the user"),
                                fieldWithPath("roles").description("Assigned roles of the
user"),
                                fieldWithPath("id").description("Unique identifier of an
user"),
                                fieldWithPath("_links").ignored()
                        )
                        ));
    }
}
```

Esegui il test unitario e alcuni file vengono generati nella cartella di destinazione.



Creare una cartella di origine come **src / main / asciidocs** e creare un **file** doc **con un prefisso di adoc** per documentare i dettagli del servizio. Esempio di file doc

```
[[resources]]
= Resources

User: Have the information about an User. It has following fields:

include::{snippets}/get-user/response-fields.adoc[]

[[resources-user]]
== User

The User resources has all the information about the application user. This resource
is being used to perform all CRUD operations on User entity.


[[resources-user-retrieve]]
=== Retrieve User

A `GET` request gets a User.

operation::get-user[snippets='response-fields,curl-request,http-response']
```

Il **tag include** nel file doc deve includere i frammenti. Puoi specificare qualsiasi formato desideri generare il documento.

Una volta che hai tutto a posto, **esegui Maven Build** . Eseguirà i test e il plugin asciidoc genererà il file html **del documento nella** cartella **target.generate-docs** . Il tuo file generato sarebbe simile a questo:

Table of Contents

Retrieve User

A `GET` request gets a

Response fields

Path
username
password
securityAnswer
securityQuestion
email
roles
id

Table of Contents

Example request

```
$ curl 'http://r
'Authorization:
eyJhbGciOiJIUzUx
h0dHA6Ly9ob3NwaX
lHdJjaaodsIzKX4y
```

Example respons

```
HTTP/1.1 200 OK
Content-Type: ap
Content-Length:

{
  "password" : "
  "securityQuest
  "securityAnswe
  "id" : "591310
  "username" : "
  "email" : "bha
  "roles" : [ "A
  "_links" : {
    "self" : {
```

```
    "self" : {
      "href" :
    }
  }
}
```

Ogni volta che viene eseguita la generazione del tuo maven, verrà generato il tuo ultimo documento che sarebbe sempre in linea con i tuoi servizi. Basta pubblicare questo documento per i consumatori / clienti del servizio.

Buona programmazione.

Leggi Iniziare con i microservizi online: https://riptutorial.com/it/microservices/topic/7719/iniziare-con-i-microservizi

Capitolo 2: Gateway API

introduzione

L'architettura dei microservizi offre una grande flessibilità per disaccoppiare le applicazioni e sviluppare applicazioni indipendenti. Un servizio di microservizio dovrebbe essere sempre testabile e distribuibile indipendentemente.

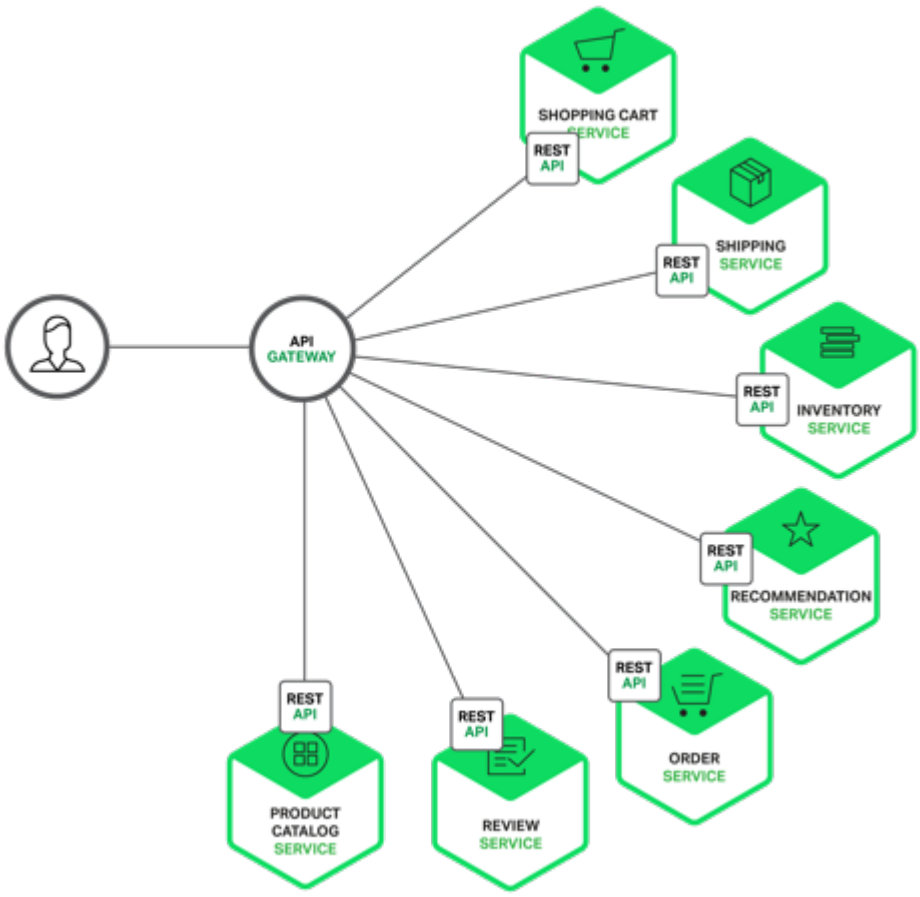
Ma, man mano che continui ad avere troppi servizi, è **necessario disporre di un gateway API** .

Non puoi esporre tutti i tuoi servizi a client esterni. È necessario disporre di uno strato di astrazione che funga da gatekeeper per tutti i tuoi microservizi. Un punto di accesso per tutti i tuoi servizi.

Examples

Panoramica

Supponiamo di disporre di un cloud E-commerce con vari microservizi come servizio carrello acquisti, servizio ordini, servizio inventario e così via. È necessario disporre di un gateway API come servizio Edge per il mondo esterno.



Il gateway API astrae i dettagli (host e porta) relativi ai microservizi di sottolineatura. Ora, tutti i tuoi clienti devono solo conoscere un URL del server che è il tuo gateway API. Eventuali modifiche in qualsiasi altro Miroservice non richiederebbero alcuna modifica nell'app client. Ogni volta che un gateway API riceve una richiesta, **inoltra la richiesta a uno specifico Microservice** .

Leggi Gateway API online: https://riptutorial.com/it/microservices/topic/10904/gateway-api

Titoli di coda

S. No	Capitoli	Contributors
1	Iniziare con i microservizi	Community, Dinusha, mohan08p, Nitish Bhardwaj
2	Gateway API	Nitish Bhardwaj