Kostenloses eBook

LERNEN microservices

Free unaffiliated eBook created from
Stack Overflow contributors.

#microservices

Inhaltsverzeichnis

Über

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: microservices

It is an unofficial and free microservices ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official microservices.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Kapitel 1: Erste Schritte mit Microservices

Bemerkungen

In diesem Abschnitt erhalten Sie einen Überblick darüber, was Mikrodienste sind und warum ein Entwickler sie verwenden möchte.

Es sollte auch alle großen Themen in Microservices erwähnen und auf die verwandten Themen verweisen. Da die Dokumentation für Microservices neu ist, müssen Sie möglicherweise erste Versionen dieser verwandten Themen erstellen.

Examples

Wichtige Checkliste für Microservices-Plattform

- CI / CD-Pipeline
- Zentralisierter Authentifizierungs- und Autorisierungsdienst
- API-Dokumentation
- API-Gateway
- Protokollverwaltungstool zentralisieren
- Service-Monitor
- Infrastrukturautomatisierung
- Zentraler Konfigurationsserver

API-Dokumentation

Verwenden Sie **Spring REST Docs** , um Ihre Dienste zu dokumentieren. Es ist ein leistungsfähiges Framework, das sicherstellt, dass die Servicelogik immer mit der Dokumentation übereinstimmt. Dazu müssen Sie Integrationstests für Ihre Services schreiben.

Wenn die Dokumentation und das Serviceverhalten nicht übereinstimmen, schlagen die Tests fehl.

Hier ist ein Beispiel zum Generieren der Dokumente für ein Maven-Projekt.

Fügen Sie diese Abhängigkeit in Ihrem Pom hinzu:

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

und fügen Sie das Plugin asciidoc unter dem Tag build.plugins hinzu:

```
<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>
```

Nun nehmen wir einen Beispielcontroller, den wir dokumentieren wollen:

```
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);
 }
}
```

Schreiben Sie nun einen Testfall für den Dienst:

```
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;
```

statische org.springframework.restdocs.operation.preprocess.Preprocessors.preprocessResponse importieren;

```
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.eyJzdWIiOiJiaGFyZHdhai5uaXRpc2gxOSIsInJvbGVzIjpbIkFETUlOIl0sImlzcyI6Imh0dHA6Ly9ob3

                .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()
                        )
                        ));
    }
```

}

Weitere Informationen finden Sie in dieser Referenz: http://docs.spring.io/spring-restdocs/docs/current/reference/html5/

Beispiel für die API-Dokumentation

Verwenden Sie **Spring REST Docs** , um Ihre Dienste zu dokumentieren. Es ist ein leistungsfähiges Framework, das sicherstellt, dass die Servicelogik immer mit der Dokumentation übereinstimmt. Dazu müssen Sie Integrationstests für Ihre Services schreiben.

Wenn die Dokumentation und das Serviceverhalten nicht übereinstimmen, schlagen die Tests fehl.

Hier ist ein Beispiel zum Generieren der Dokumente in einem Maven-Projekt:

Fügen Sie diese Abhängigkeit in der POM-Datei hinzu:

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

Fügen Sie außerdem das ASCII-Docs-Plugin hinzu, um Docs unter dem Tag build.puglin zu generieren

```
<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>
```

Als Beispiel erstellen wir einen Controller-Service, den wir dokumentieren möchten.

```
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);
}
```

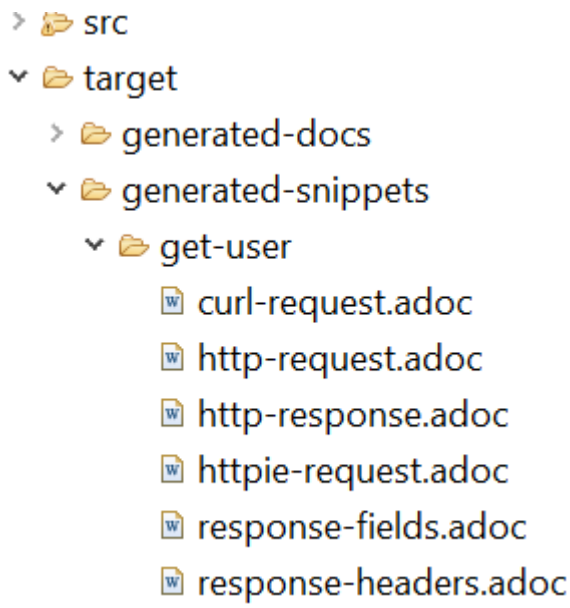
}

Schreiben wir einen Testfall, um diesen Service zu testen:

```
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.eyJzdWIiOiJiaGFyZHdhai5uaXRpc2gxOSIsInJvbGVzIjpbIkFETUlOIl0sImlzcyI6Imh0dHA6Ly9ob3

                .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()
                        )
                        ));
    }
}
```

Führen Sie den Gerätetest aus, und einige Dateien werden im Zielordner generiert.



Erstellen Sie einen Quellordner als **src / main / asciidocs** und erstellen Sie eine Doc- **Datei mit einem Präfix von adoc** , um Ihre **Servicedetails** zu dokumentieren. Beispieldokumentdatei

```
[[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']
```

Das **Include-Tag** in der doc-Datei enthält die Ausschnitte. Sie können angeben, welches Format Sie für die Erstellung des Dokuments verwenden möchten.

Sobald Sie alles eingerichtet haben, **starten Sie Maven Build** . Es führt Ihre Tests aus und das asciidoc-Plugin generiert Ihre Dokument-HTML-Datei im Ordner **target.generate-docs** . Ihre generierte Datei würde ungefähr so aussehen:

Table of Contents

Retrieve User

A `GET` request gets a

Response fields

Path
username
password
securityAnswer
securityQuestion
email
roles
id

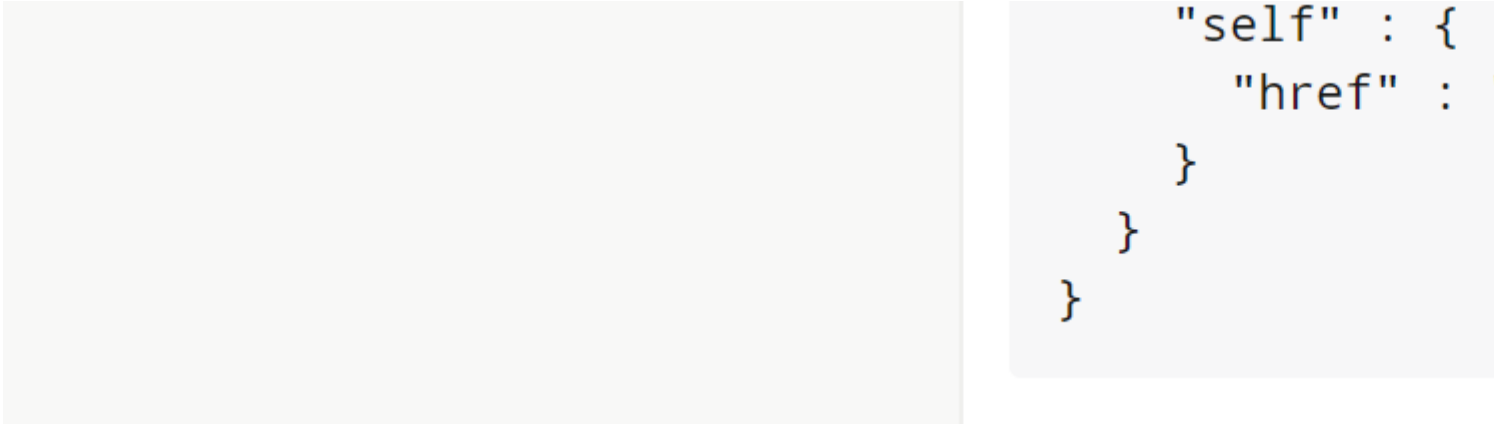
Table of Contents

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" : {
```



Wenn Ihr Maven-Build ausgeführt wird, wird Ihr aktuelles Dokument generiert, das immer mit Ihren Diensten übereinstimmt. Veröffentlichen Sie dieses Dokument einfach den Verbrauchern / Kunden des Dienstes.

Glückliche Kodierung

Erste Schritte mit Microservices online lesen:
https://riptutorial.com/de/microservices/topic/7719/erste-schritte-mit-microservices

Kapitel 2: API-Gateway

Einführung

Microservices-Architektur bietet große Flexibilität, um Anwendungen zu entkoppeln und unabhängige Anwendungen zu entwickeln. Ein Microservice sollte immer unabhängig testbar und einsatzbereit sein.

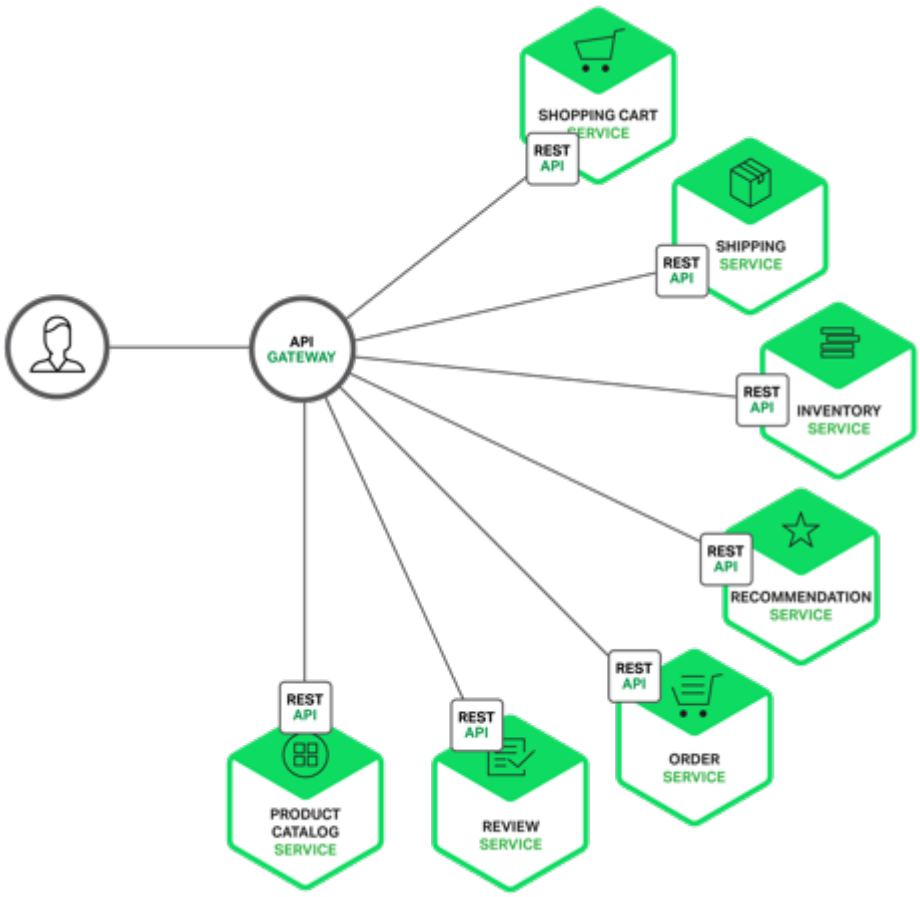
Da Sie jedoch zu viele Dienste anbieten, ist **ein API-Gateway erforderlich** .

Sie können nicht alle Ihre Dienste externen Kunden zur Verfügung stellen. Sie benötigen eine gewisse Abstraktionsebene, die als Gatekeeper für alle Ihre Microservices fungiert. Ein Einstiegspunkt für alle Ihre Dienstleistungen.

Examples

Überblick

Angenommen, Sie verfügen über eine E-Commerce-Cloud mit verschiedenen Microservices, z. Sie benötigen ein API-Gateway als Edge-Service für die Außenwelt.



Das API-Gateway abstrahiert die Details (Host und Port) über die unterstrichenen Microservices.

Jetzt müssen alle Ihre Clients nur eine Server-URL kennen, die Ihr API-Gateway ist. Alle Änderungen an anderen Miroservice erfordern keine Änderungen an Ihrer Client-App. Immer wenn ein API-Gateway eine Anforderung erhält, **leitet es die Anforderung an einen bestimmten Microservice weiter** .

API-Gateway online lesen: https://riptutorial.com/de/microservices/topic/10904/api-gateway

Credits

S. No	Kapitel	Contributors
1	Erste Schritte mit Microservices	Community, Dinusha, mohan08p, Nitish Bhardwaj
2	API-Gateway	Nitish Bhardwaj