eBook Gratuit

APPRENEZ microservices

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#microservices

Table des matières

À propos

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

Chapitre 1: Démarrer avec les microservices

Remarques

Cette section fournit un aperçu de ce que sont les microservices et pourquoi un développeur peut vouloir l'utiliser.

Il devrait également mentionner tous les sujets importants dans les microservices, et établir un lien avec les sujets connexes. La documentation des microservices étant nouvelle, vous devrez peut-être créer des versions initiales de ces rubriques connexes.

Examples

Liste de vérification essentielle pour les microservices

- Pipeline CI / CD
- Service d'authentification et d'autorisation centralisé
- Documentation API
- Passerelle API
- Centralisation de l'outil de gestion des journaux
- Moniteur de service
- Automatisation de l'infrastructure
- Serveur de configuration centralisé

Documentation API

Utilisez **Spring REST Docs** pour documenter vos services. C'est un framework puissant qui garantit que la logique de service est toujours en ligne avec la documentation. Pour ce faire, vous devez écrire des tests d'intégration pour vos services.

Si le comportement de la documentation et du service ne correspond pas, les tests échoueront.

Voici un exemple d'exemple pour générer les documents pour un projet Maven.

Ajoutez cette dépendance dans votre pom:

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

et ajoutez le plug-in asciidoc sous la balise 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>
```

Prenons maintenant un exemple de contrôleur que nous voulons documenter:

```
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);
 }
}
```

Maintenant, écrivez un scénario de test pour le service:

```
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 statique
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()
```

```
                        )
                        ));
    }
```

}

Veuillez suivre cette référence pour plus de détails: http://docs.spring.io/spring-restdocs/docs/current/reference/html5/

Exemple de documentation de l'API

Utilisez **Spring REST Docs** pour documenter vos services. C'est un framework puissant qui garantit que la logique de service est toujours en ligne avec la documentation. Pour ce faire, vous devez écrire des tests d'intégration pour vos services.

Si le comportement de la documentation et du service ne correspond pas, les tests échoueront.

Voici un exemple d'exemple pour générer les documents dans un projet maven:

Ajoutez cette dépendance dans le fichier pom:

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

Ajoutez également le plug-in docs ASCII pour générer des documents sous la balise 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>
```

Maintenant, à titre d'exemple, créons un service de contrôleur que nous voulons documenter.

```
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);
}
```

}

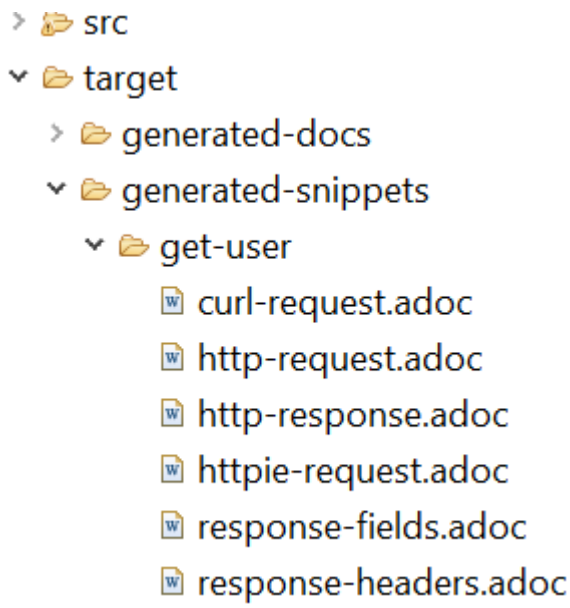
Écrivons un cas de test pour tester ce service:

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

Exécutez le test unitaire et certains fichiers sont générés sous le dossier cible.



Créez un dossier source en tant que **src / main / asciidocs** et créez un **fichier** doc **avec un préfixe adoc** pour documenter vos détails de service. Exemple de fichier 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']
```

La **balise include** du fichier doc doit inclure les extraits. Vous pouvez spécifier le format que vous souhaitez générer la doc.

Une fois que vous avez tout en place, **lancez Maven Build** . Il exécutera vos tests et le plug-in asciidoc générera votre fichier HTML sous le dossier **target.generate-docs** . Votre fichier généré devrait ressembler à ceci:

Table of Contents

Retrieve User

A `GET` request gets a

Response fields

Path
username
password
securityAnswer
securityQuestion
email
roles
id

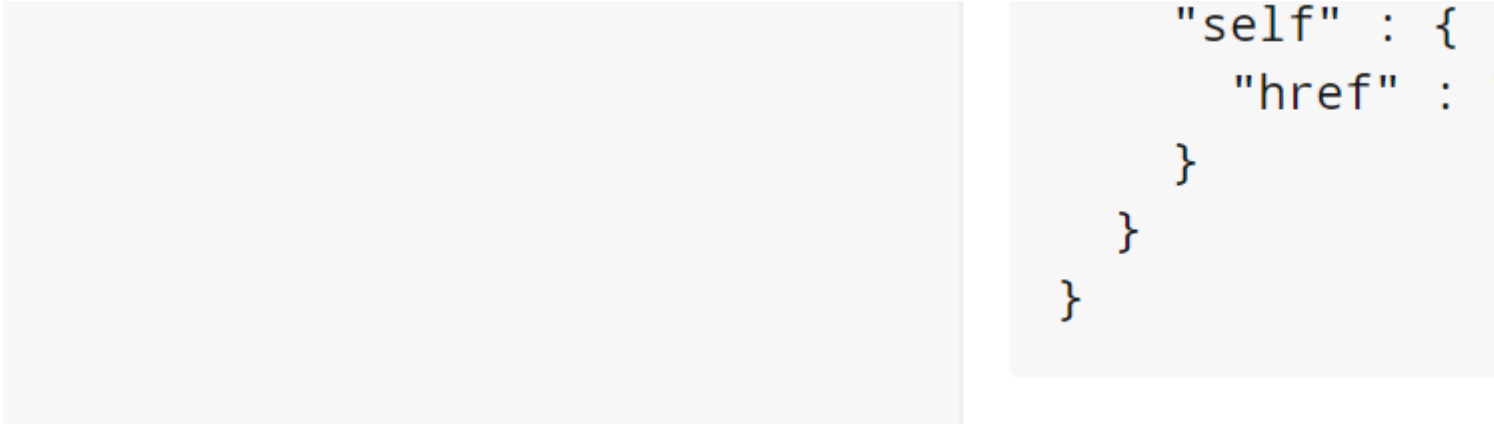
Table of Contents

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" : {
```



Chaque fois que votre build Maven s'exécute, votre dernier document sera généré et sera toujours en ligne avec vos services. Il suffit de publier ce document auprès des consommateurs / clients du service.

Heureux codage

Lire Démarrer avec les microservices en ligne:
https://riptutorial.com/fr/microservices/topic/7719/demarrer-avec-les-microservices

Chapitre 2: API Gateway

Introduction

L'architecture des microservices offre une grande flexibilité pour découpler les applications et développer des applications indépendantes. Un Microservice devrait toujours pouvoir être testé et déployé indépendamment.

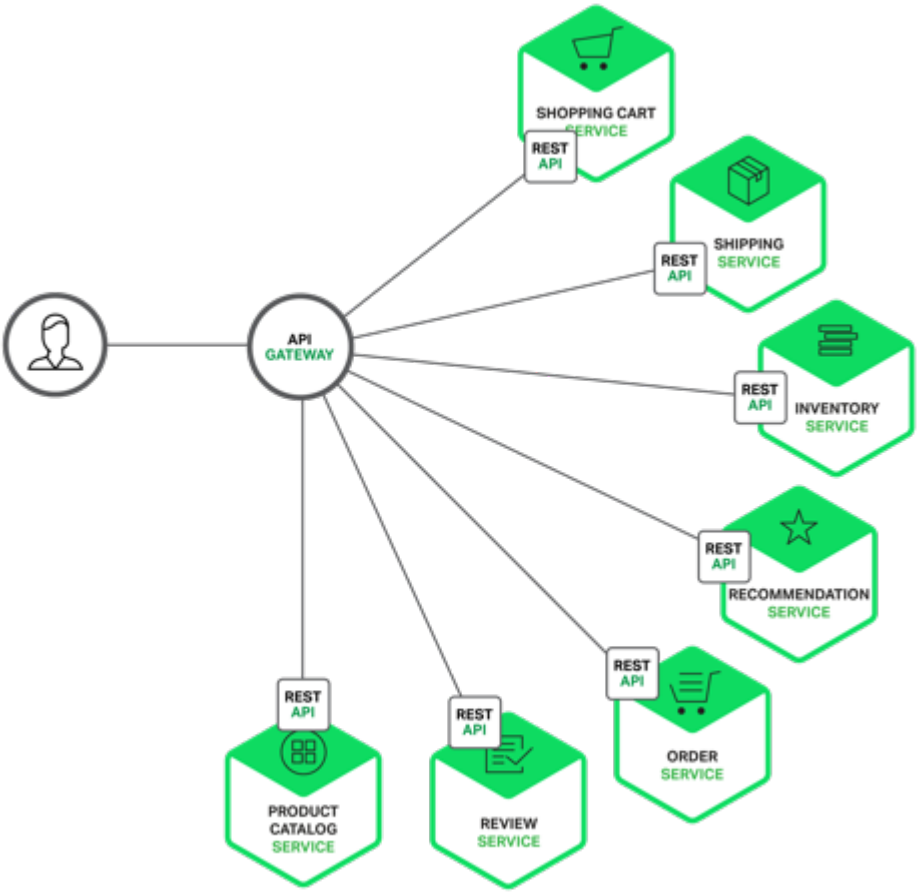
Mais comme vous continuez à avoir trop de services, il est **nécessaire d'avoir une passerelle API** .

Vous ne pouvez pas exposer tous vos services à des clients externes. Vous devez disposer d'une couche d'abstraction qui agit en tant que gatekeeper pour tous vos microservices. Un point d'entrée pour tous vos services.

Examples

Vue d'ensemble

Supposons que vous ayez un cloud de commerce électronique avec différents microservices comme service de panier, service de commande, service d'inventaire, etc. Vous devez disposer d'une passerelle API en tant que service Edge pour le monde extérieur.



La passerelle API extrait les détails (hôte et port) des microservices soulignés. Maintenant, tous vos clients ont juste besoin de connaître une URL de serveur qui est votre passerelle API. Toute modification apportée à un autre Miroservice ne nécessiterait aucune modification de votre application client. Chaque fois qu'une passerelle API obtient une requête, **elle achemine la demande vers un microservice spécifique** .

Lire API Gateway en ligne: https://riptutorial.com/fr/microservices/topic/10904/api-gateway

Crédits

S. No	Chapitres	Contributeurs
1	Démarrer avec les microservices	Community, Dinusha, mohan08p, Nitish Bhardwaj
2	API Gateway	Nitish Bhardwaj