Deloitte.

Microservices

USI CBO CR LAUNCHPAD TRAINING PROGRAM

Microservices

Context, Objectives, Agenda

Context

- Microservices are an architecture style and an approach for software development to satisfy modern business demands.
- It is an evolution from previous architecture styles – Monolithic and SOA.
- Microservices are a style of distributed architecture with several loosely coupled services providing functionality by collaborating.
- Typically these services tend to be smaller and take one single responsibility, and hence they are developed over a few weeks rather than months.

Objectives

- You will learn
  - What is Microservice architecture and how it is different from the way enterprise applications were previously developed
  - Usage of tools like Spring Boot, Spring Cloud, Eureka, Zuul to understand the process of creating and deploying a microservice to Cloud

Agenda

Topic	Content
Introduction	• Introduction to Monolithic Architecture • Drawbacks of Monolithic Architecture • Service Oriental Architecture • Issue with SOA
Microservices	• Evolution and Characteristics of Microservices • Benefits and Challenges • Monolithic vs Microservices Architecture
Spring Cloud	• Postman – Usage and Installation • Spring Cloud & Service Discovery • API Gateway • Microservice Ecosystem & Tools – Eureka, Zuul
Building Microservices – Deep dive with an example	

Introduction

Introduction

Key Objectives

01 **Monolithic Architecture**

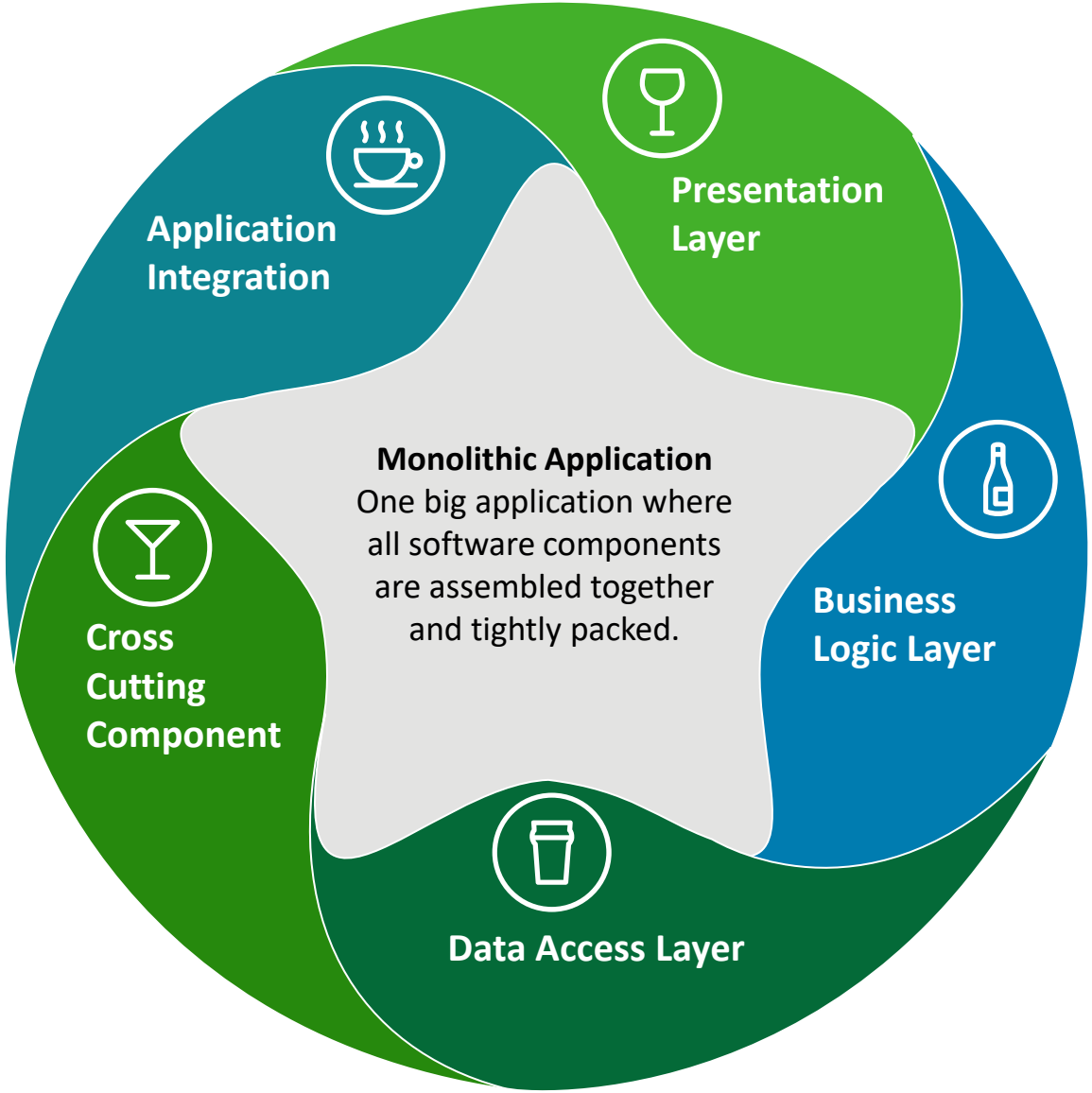
02 **Drawbacks of Monolithic Architecture**

03 **Service Oriented Architecture(SOA)**

04 **Issues with SOA**

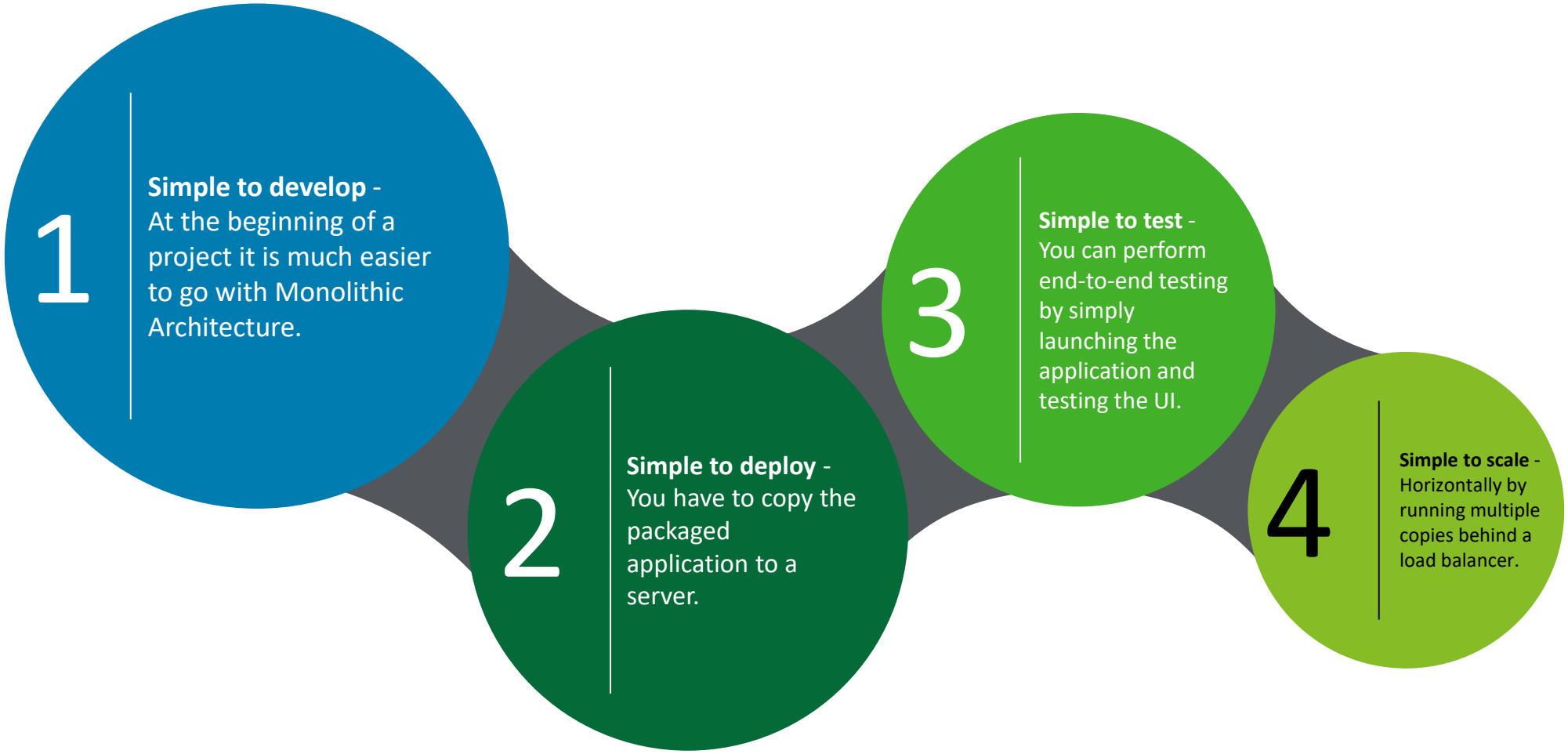
Monolithic Architecture

Introduction



Monolithic Architecture

Benefits

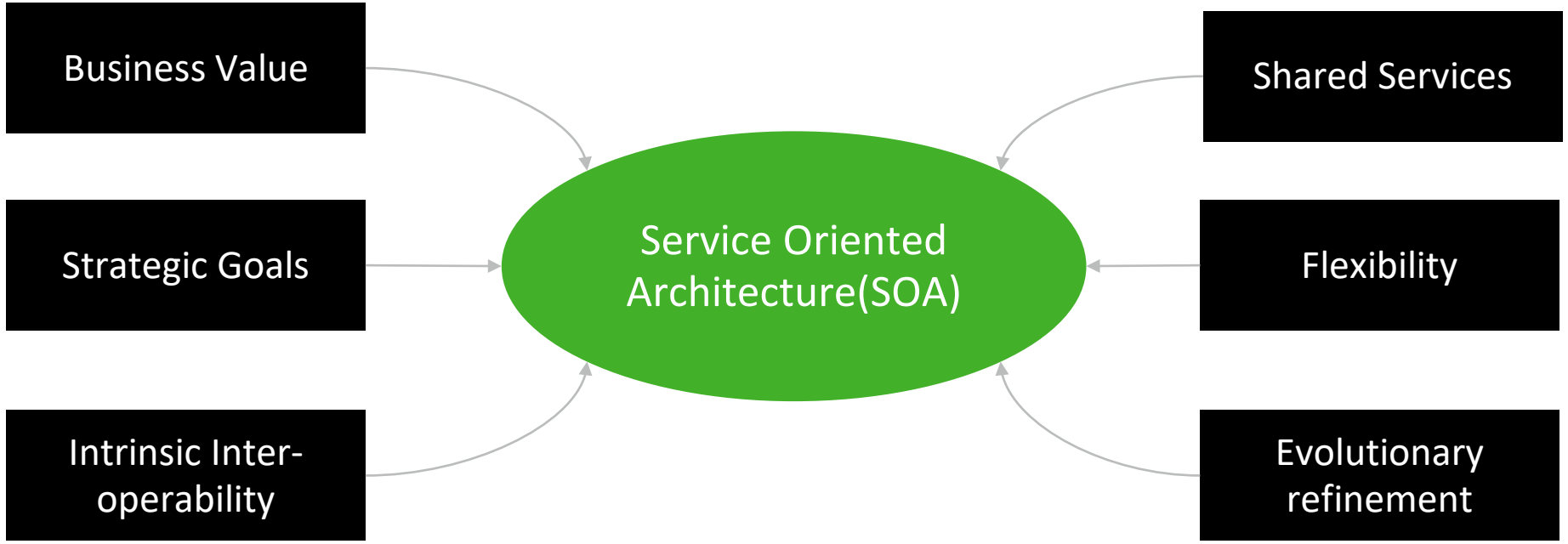


Monolithic Architecture

Drawbacks

Service Oriented Architecture

Characteristics

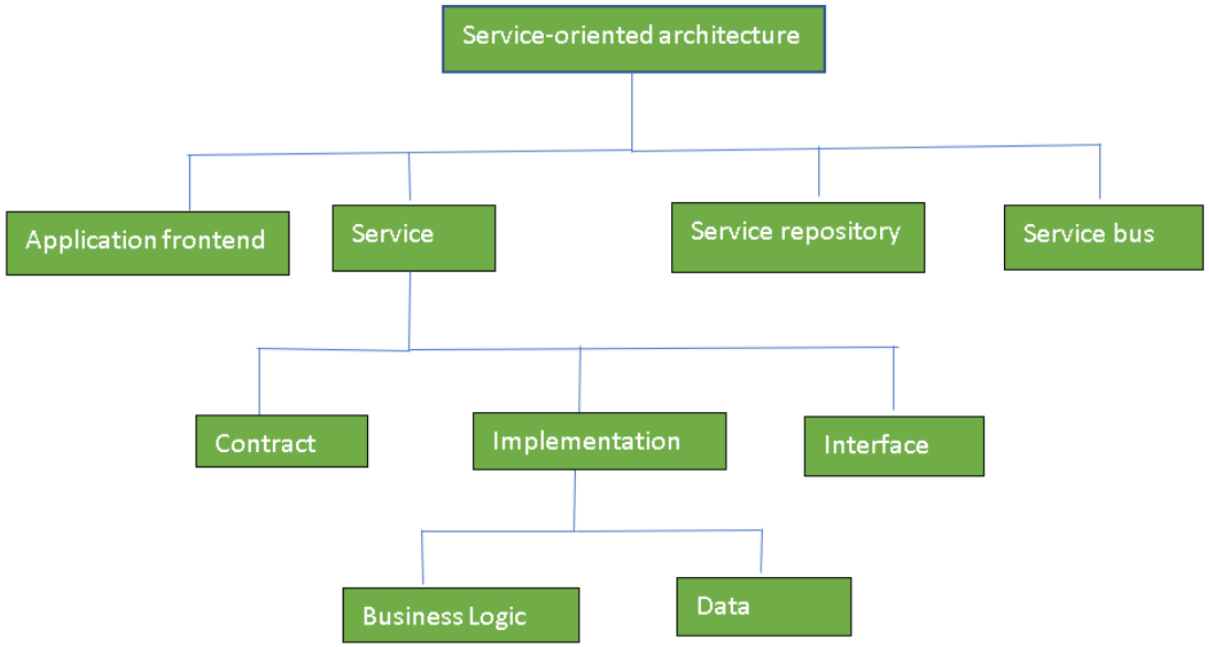


Service Oriented Architecture

Principles

- ✓ **Standardized service contract**: Specified through one or more service description documents.
- ✓ **Loose coupling**: Services are designed as self-contained components, maintain relationships that minimize dependencies on other services.
- ✓ **Abstraction**: A service is completely defined by service contracts and description documents. They hide their logic, which is encapsulated within their implementation.
- ✓ **Reusability**: Designed as components, services can be reused more effectively, thus reducing development time and the associated costs.
- ✓ **Autonomy**: Services have control over the logic they encapsulate and, from a service consumer point of view, there is no need to know about their implementation.
- ✓ **Discoverability**: Services are defined by description documents that constitute supplemental metadata through which they can be effectively discovered. Service discovery provides an effective means for utilizing third-party resources.
- ✓ **Composability**: Using services as building blocks, sophisticated and complex operations can be implemented. Service orchestration and choreography provide a solid support for composing services and achieving business goals.

Components of SOA:



Service Oriented Architecture

Benefits and Drawbacks

Benefits

Drawbacks

Knowledge Check

1. Which are the advantages of Monolithic Architecture?

- ○ Easy to develop and deploy
- ○ Easy to maintain
- ○ Highly Scalable
- ○ Reduced deployment time

2. SOA services can be developed in different languages and OS'es as long as they follow the SOA principles?

- ○ True
- ○ False

3. Point out the correct statement.

- ○ Service Oriented Architecture (SOA) describes a standard method for requesting services from distributed components and managing the results
- ○ SOA provides the translation and management layer in an architecture that removes the barrier for a client obtaining desired services
- ○ Both

Knowledge Check

4. Which of the following is not an advantage of SOA ?

A. Availability

B. Reliability

C. Scalability

D. None of the above

5. What can have the largest impact on the performance of an SOA ?

A. Service granularity

B. Use of open standards

C. Service version management

D. Business monitoring of KPIs

6. What is an SOA repository ?

A. Storage of service metadata

B. Storage for versions of service components

C. The means by which services are loosely coupled

7. Which of the following is false w.r.t to Monolithic Architecture ?

A. The size of the application can slow down the start-up time.

B. You must redeploy the entire application on each update.

C. Continuous deployment is simple.

Microservices

Microservices

Key Objectives

01 **Evolution of Microservice architecture**

02 **Microservices characteristics**

03 **Managing the Complexity**

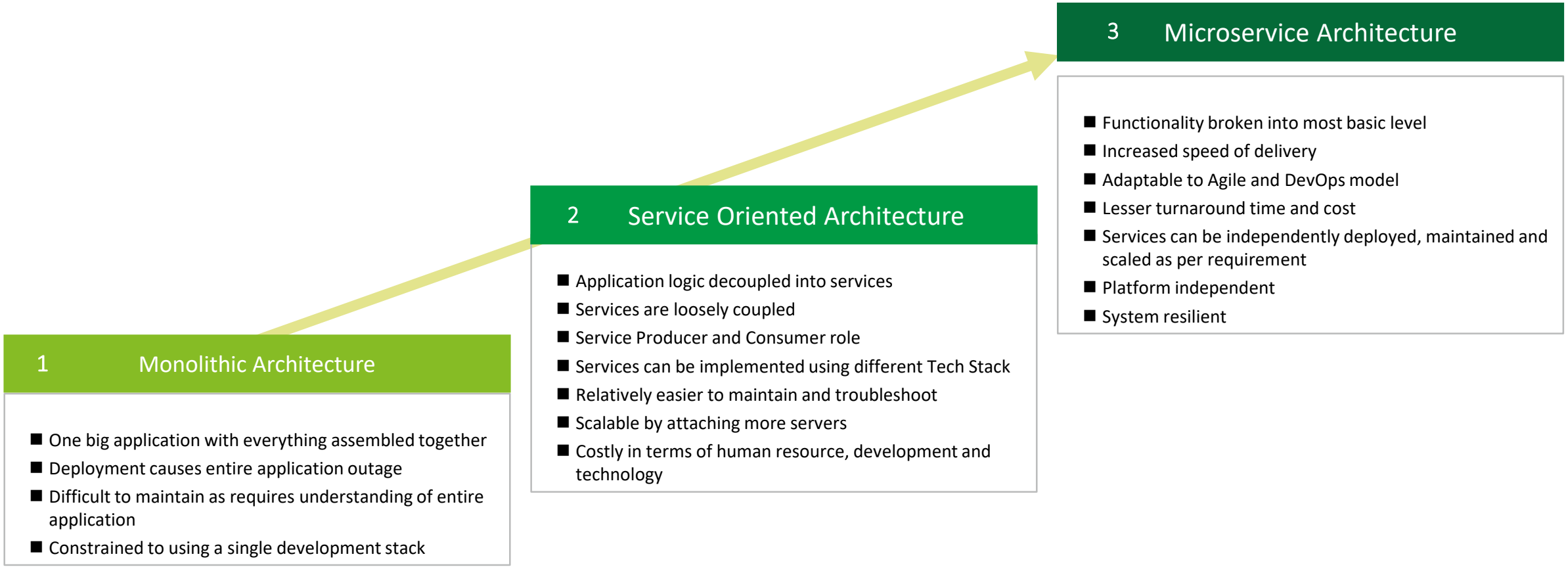
04 **Microservices - Benefits & Challenges**

05 **Monolithic vs Microservice architecture**

06 **Developing the Microservice way**

Microservices

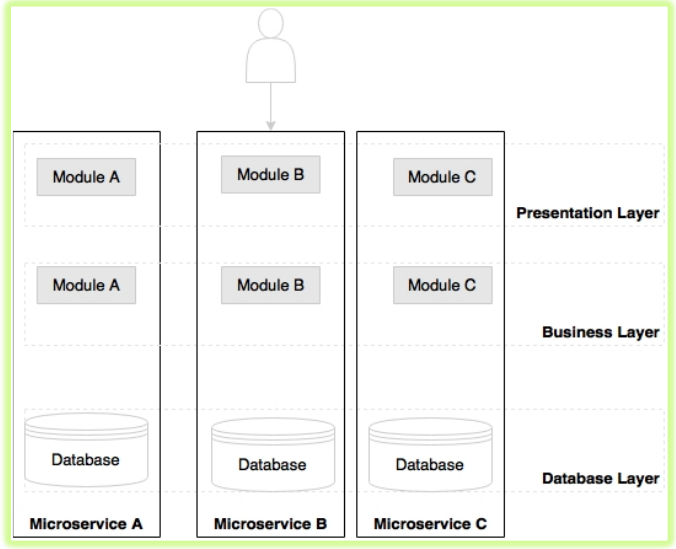
Evolution



Microservices

Characteristics

Lightweight	**Polyglot architecture**	**Distributed and Dynamic**
Loose Coupling	**Service Abstraction**	**Service Reuse**
Statelessness	**Discoverable**	**Fault Tolerant**



Microservice Architecture

Microservices

Benefits

Evolutionary Design

- No need to rewrite your whole application.
- Add new features as Microservices, and plug them into your existing application.

Small codebase

- Each Microservice deals with one concern(SoC) only – this result in a small codebase, which means easier maintainability.

Productivity

- Part of functionality associated with microservice architecture is the ability to easily understand when compared to an entire monolithic app.
- If you plan to expand your development team, microservices are a better pick.

System Resilience

- If some of the services go down, only some features will be affected, not the entire application.

Easy to Deploy

- Deploy only the needed codebase, instead of redeploying the entire application

Auto Scale

- Freedom to scale only the loaded service, as that service will handle the bigger load

Microservices

Real Time Use Cases

1 Netflix

- ✓ Netflix is one of the best examples of implementing microservice architecture.
- ✓ Back in 2009, Netflix moved from a monolithic architecture to microservices due to increasing demand for its services. But as no microservices existed back then, Netflix engineers created an open-source technology that offered the best Internet television network.
- ✓ By moving to microservices, the company's developers were able to deploy thousands of code sections every day to support its 193 million subscribers and 10 billion hours of movies and TV series.

2 Uber

- ✓ Similar to other startups, Uber began with a monolithic architecture.
- ✓ It was simpler for the company's founders when they provided clients only with the UberBLACK service. But, as the startup rapidly grew, developers decided to switch to microservices to use several languages and frameworks.
- ✓ Now, Uber has 1,300+ microservices focusing on improving app's scalability.

3 Spotify

- ✓ With over 75m active users, Spotify's founders decided to build a system with independently scalable components to make synchronization easier.
- ✓ For Spotify, the main benefit of microservices is the ability to prevent massive failures.
- ✓ Even if multiple services fail simultaneously, users won't be affected.

Microservices

Complexity

- ✓ **Project teams need to easily discover services as potential reuse candidates.**
- ✓ **These services should provide documentation, test consoles, etc. so re-using is significantly easier than building from scratch.**

✓ "With great power comes greater responsibility"

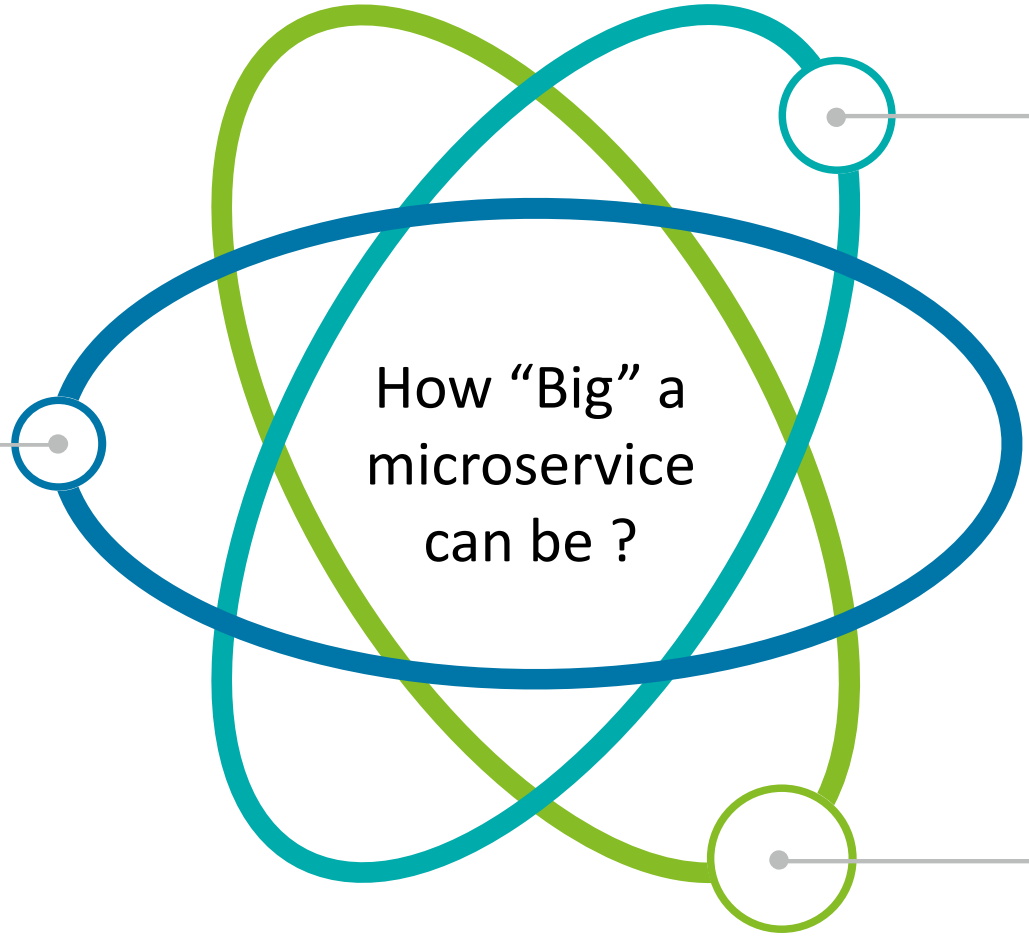
— Peter Parker, Spiderman

- ✓ Interdependencies between services need to be closely monitored.
- ✓ Downtime of services, service outages, service upgrades, etc. can all have cascading downstream effects and such impact should be proactively analyzed.

Microservices

Complexity – Contd.

- The guideline - Amazon’s Two Pizza Team - A microservice should be able to be supported be a team you can fed with two pizzas(around 12 people)



- A microservice can be as small as a single API endpoint
- (eg: 'getOrders’). Or it can be several or even dozens of API endpoints

- Scalability - Higher the scalability(size), the more specialized the service should be

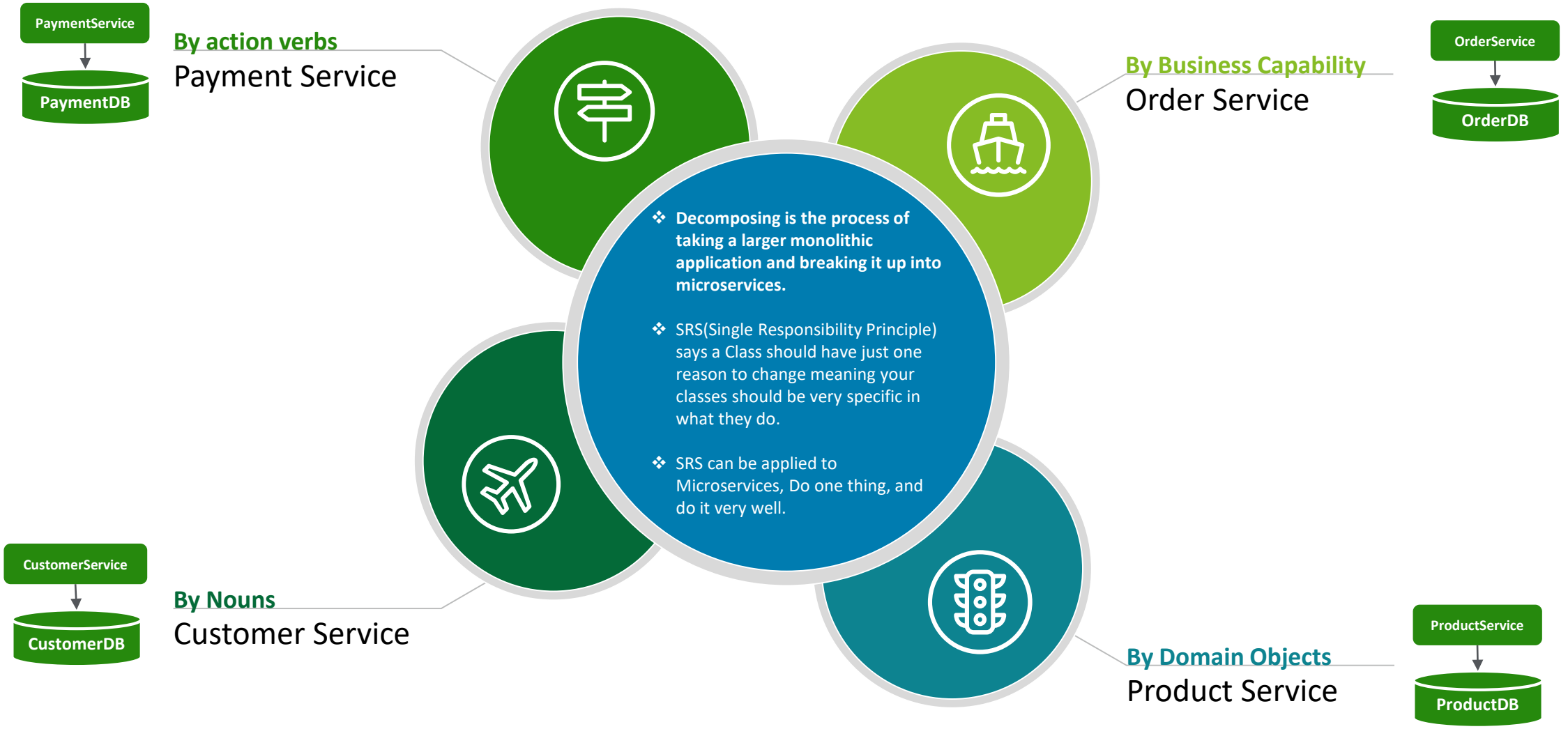
“So there is no hard answer on how big a microservice can be or should be”

Microservices

Challenges

Microservices

Decomposing



Microservices

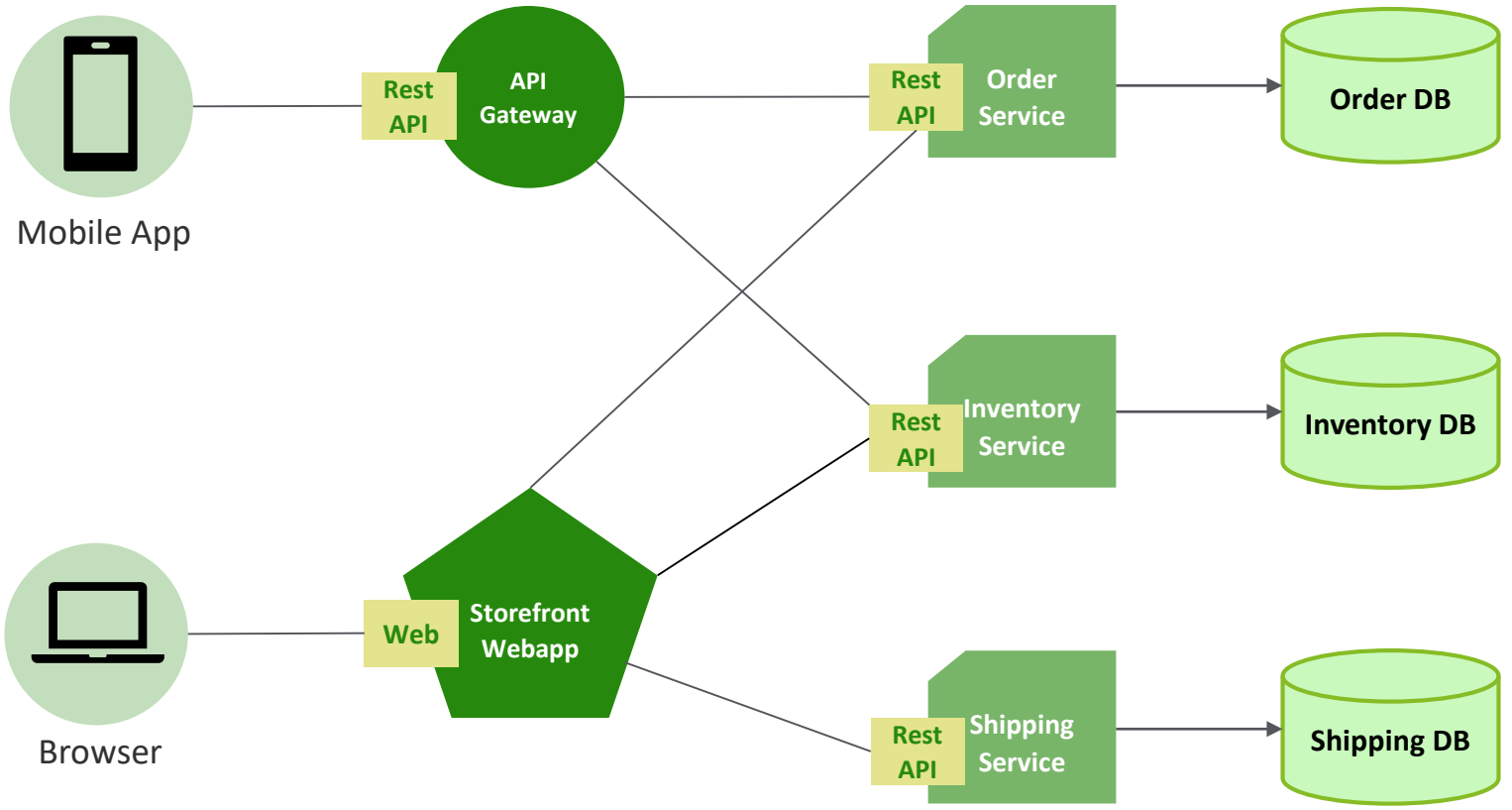
A real time example

Fictitious e-commerce application

- Let's imagine that you are building an e-commerce application that takes orders from customers, verifies inventory and available credit, and ships them.
- The application consists of several components including the StoreFrontUI, which implements the user interface, along with some backend services for checking credit, maintaining inventory and shipping orders.

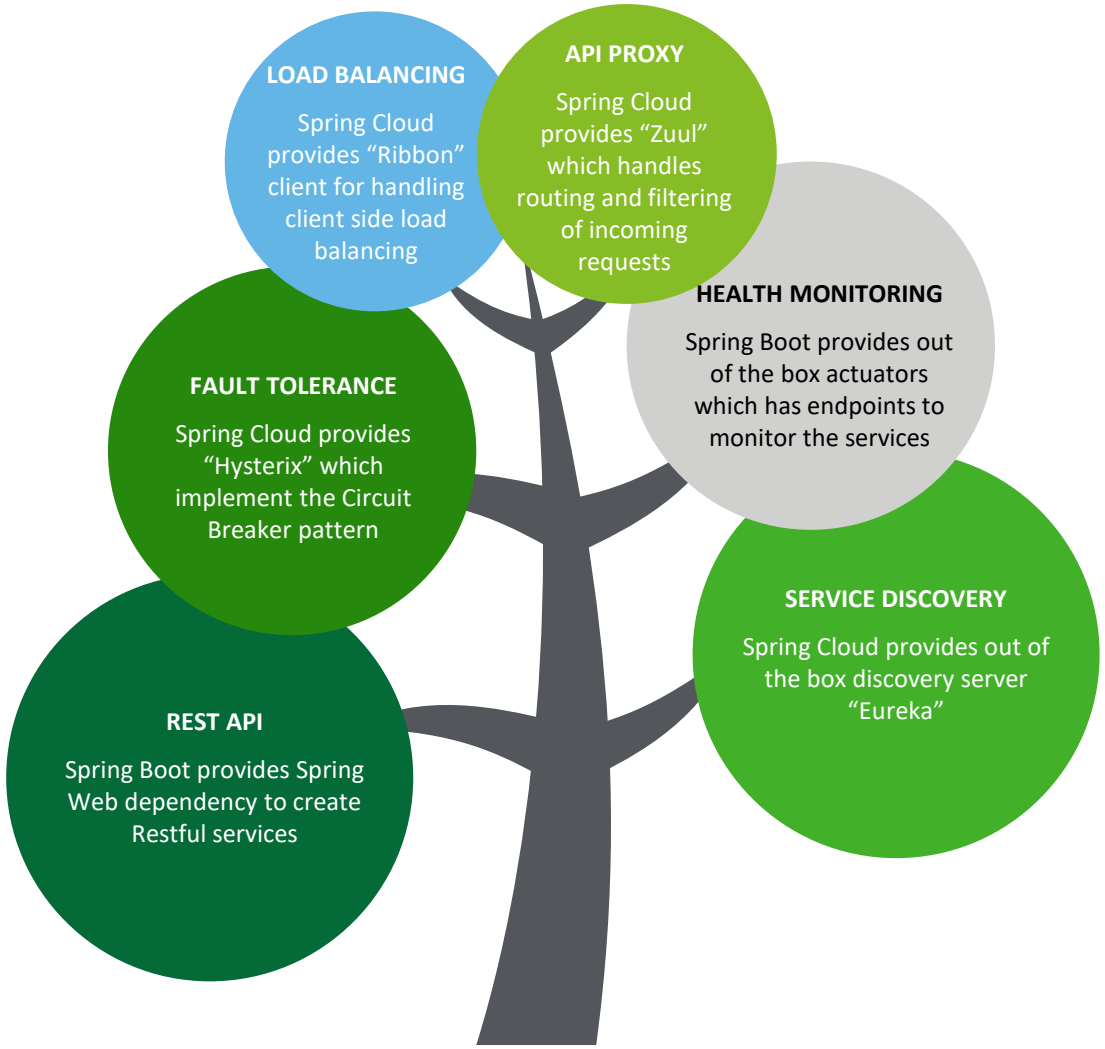
The application consists of a set of microservices.

- Account Service encapsulates the customer orders business logic.
- Inventory Service encapsulates the logic for maintaining inventory items.
- Shipping Service encapsulates the logic for shipping the customer order once confirmed.



Microservices

Spring Boot and Spring Cloud as the first choice



Microservices

Case Study

Employee Service: (Microservice 1 running on Port 8080)

```
@RestController
@RequestMapping("/employeeservice")
public class EmployeeController {

    Employees employees = new Employees();

    @GetMapping("/employees/{employeeid}")
    public Employee getEmployeeById(@PathVariable("employeeid")
    Integer employeeId) {
        return employees.getEmployees().stream()
                .filter(emp -> emp.getEmployeeId() == employeeId)
                .findFirst().get();
    }

    @GetMapping("/employees")
    public Employees getAllEmployees() {
        return this.employees;
    }

}
```

Department Service: (Microservice 2 running on Port 8082)

```
@RestController
@RequestMapping("/departmentservice")
public class DepartmentController {

    List<Department> departmentList = Arrays.asList(
            new Department(1, "Department 1"),
            new Department(2, "Department 2"),
            new Department(3, "Department 3"));

    @GetMapping("/departments/{departmentid}")
    public Department getDepartmentById(@PathVariable("departmentid")
    Integer departmentId) {
        return departmentList.stream()
                .filter(dept -> dept.getDeptId() == departmentId)
                .findFirst().get();
    }

}
```

Microservices

Case Study – Communication between Microservices

Employee Data : (Microservice 3 running on Port 8083)

```
@RestController
@RequestMapping("/employeedataservice")
public class EmpDeptController {

    @Autowired
    EmpDeptService empDeptService;

    @GetMapping("/employeeinfo/{empId}")
    public EmployeeInfoResponse getEmployeeInfo(@PathVariable("empId")
    Integer empId) {
        Employee employee = empDeptService.getEmployeeById(empId);

        return new EmployeeInfoResponse(
                employee.getEmployeeName(),
                employee.getSalary(),
                empDeptService.getDeparmentById(employee.getDeptId())
                .getDeptName());
    }

}
```

Employee Data Service: - Communication with MS1 and MS2

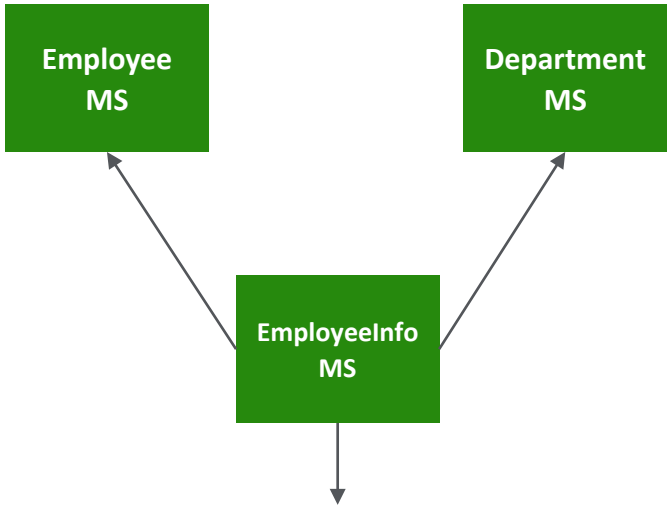
```
@Service
public class EmpDeptService {

    @Autowired
    RestTemplate restTemplate;


    public Employee getEmployeeById(Integer empId) {
        return restTemplate.getForObject(
                "http://localhost:8080/employeeservice/employees/"+empId,
                Employee.class);
    }

    public Department getDeparmentById(Integer deptId) {
        return restTemplate.getForObject(
                "http://localhost:8082/departmentservice/departments/"+deptId,
                Department.class);
    }

}
```

Microservices

Case Study – Communication between Microservices

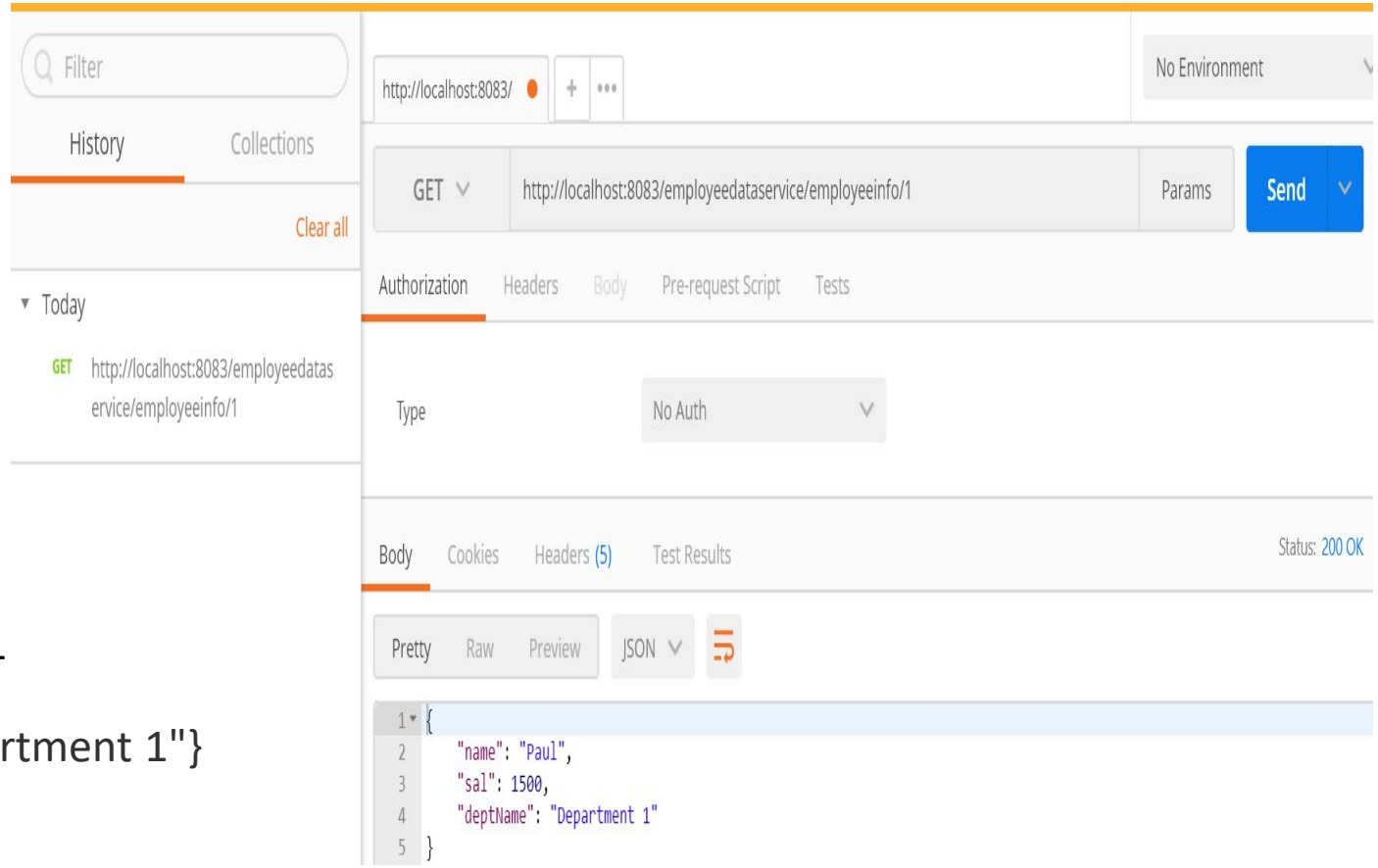
Contd..

Employee Data : (In Action)



REST API: /employeedataservice/employeeinfo/1

{"name":"Paul","sal":1500.0,"deptName":"Department 1"}



Knowledge Check

1. Which are the benefits of Microservice Architecture?

- ○ Evolutionary design
- ○ Easy to maintain
- ○ Easy to deploy
- ○ Supports Agile and DevOps model

2. Hysterix is used for?

- ○ Service Discovery
- ○ Health Monitoring
- ○ Fault Tolerance
- ○ Load Balancing

3. Microservices focuses on products, not projects.

- ○ True
- ○ False

4. Who was the pioneer on microservice architecture?

- ○ Netflix
- ○ Ola
- ○ Flipkart
- ○ Voot

Knowledge Check

5. Springboot actuators are used for ?

A. Microservice communication

B. Health monitoring

C. Fault tolerance

D. Deployment

6. Following is not an embedded container supported by Springboot?

A. JBoss

B. Tomcat

C. Jetty

D. UnderTow

7. Your environment will be more agile if each microservice needs to access only one type of resource.

A. True

B. False

8. Which of these personnel should be involved in managing APIs for microservices?

A. Business leaders

B. Enterprise architects

C. Developers

D. All of the above

Break – 15 min.

Spring Cloud

Spring Cloud

Key Objectives

01 **Postman and its usage**

02 **Spring Cloud**

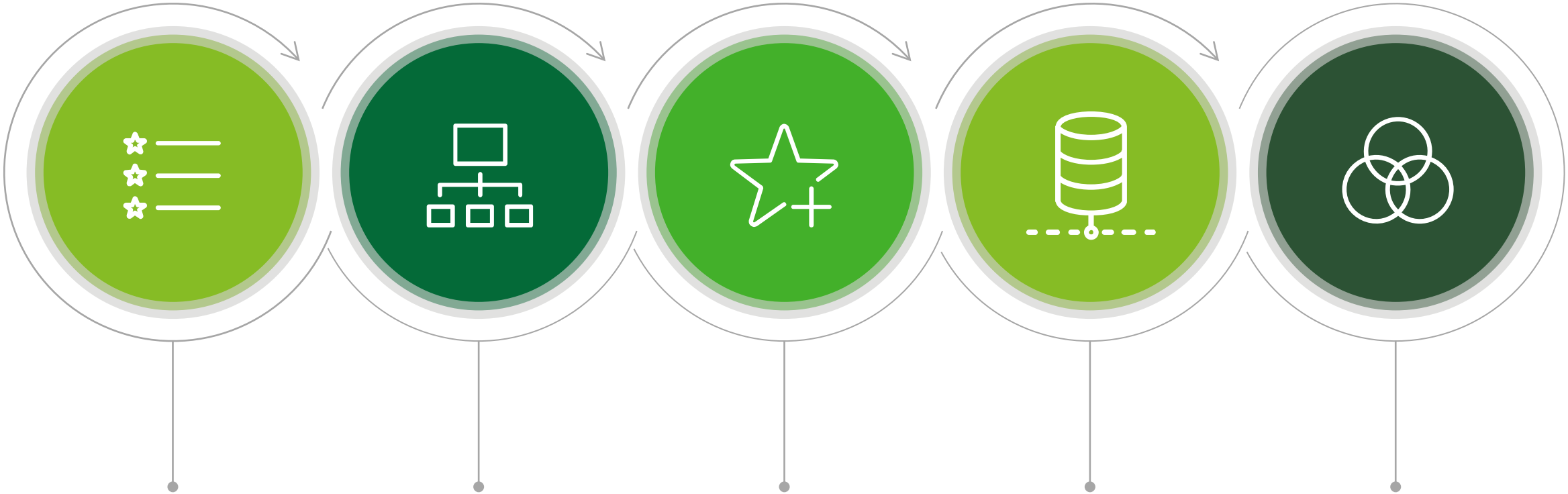
03 **Service Discovery - Eureka**

04 **API Gateway - Zuul**

Spring Cloud

What is Postman?

- Postman is a software development tool.



DEBUG

Test APIs, examine responses, add tests and scripts

DESIGN & MOCK

Design in Postman & use Postman's mock service

DOCUMENT

Create beautiful web-viewable documentation

MONITOR

Create automated tests to monitor API responsiveness

AUTOMATED TESTING

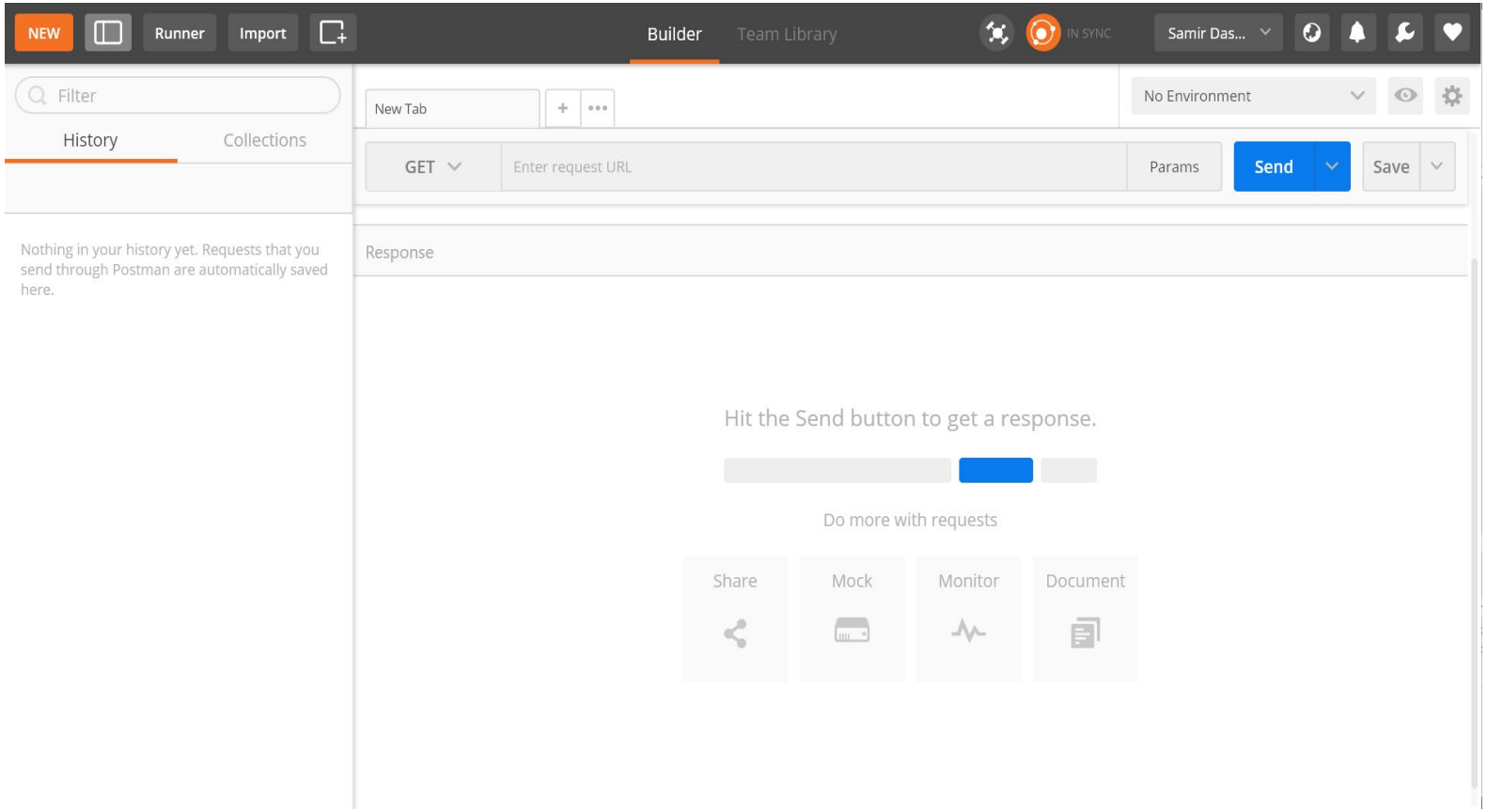
Run Automated tests using Postman collection runner

Spring Cloud

Postman installation steps

Installation steps

- Download the Postman API Client from the Postman website *(https://www.getpostman.com/product/api-client).*
- Double-click the downloaded installer and follow prompts.
- Once the installer finishes, launch the application by clicking the Postman icon.
- Once the application starts for the first time, users will be presented with a login window.
- Have a Postman account? Enter your Postman username and password.
- Have a Google account for university use? Click the "Sign in with Google" button and follow prompts. Do not use a personal Google account.



Spring Cloud

Postman in Action

Employee Microservice

We have defined two REST APIs in Employee microservice in the earlier example.

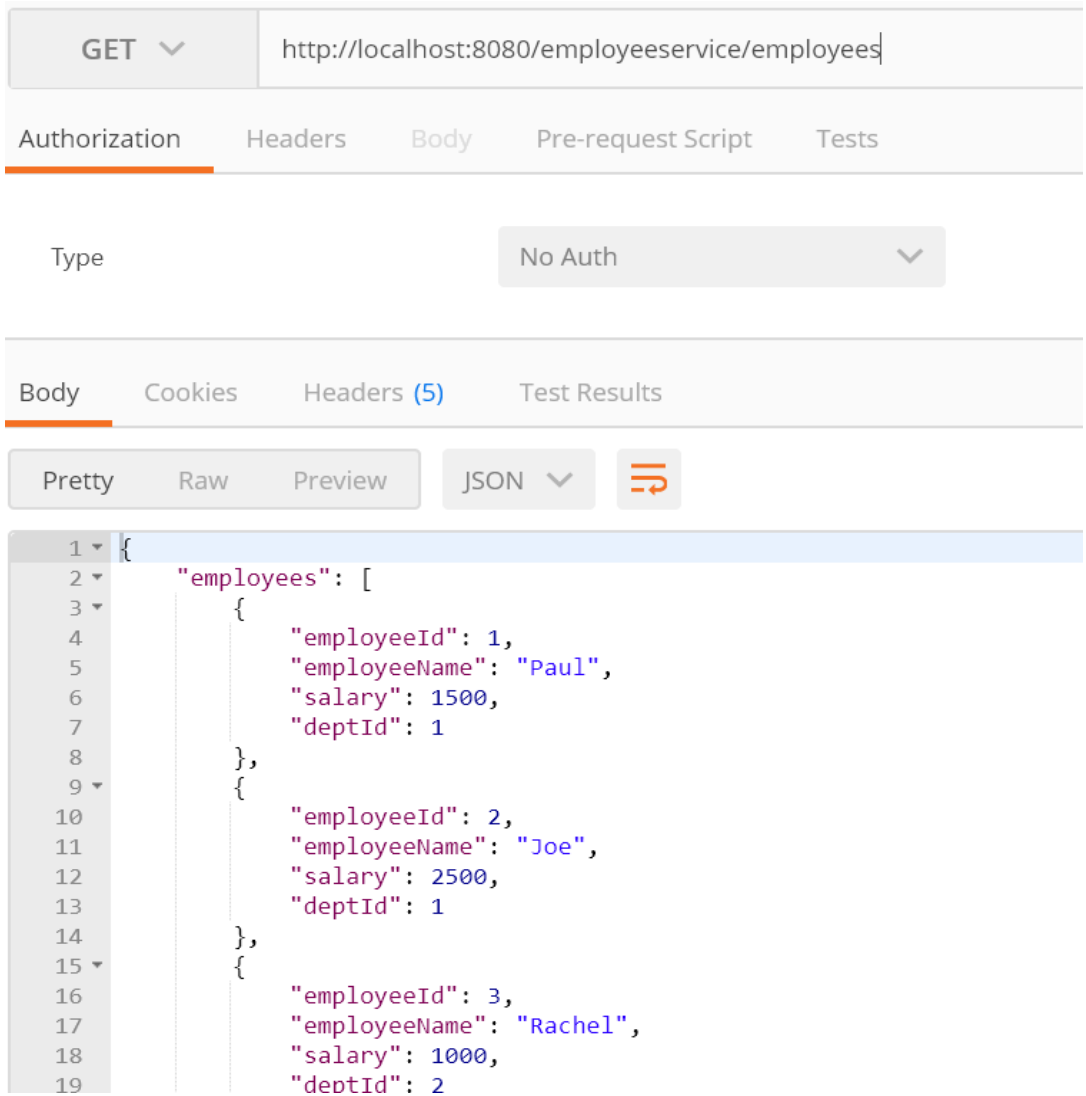
- For fetching the list of all employees

  /employeeservice/employees

- For fetching employee by employee Id

  /employeeservice/employees/{employeeid}

Here we see Postman in action where user has involved the **/employees** API and the employee list is returned by the API in the form of a JSON array containing 3 employees.



Spring Cloud

Features

Distributed/versioned configuration	Service registration and discovery	Routing
Service-to-service calls	Load balancing	Circuit Breakers
Global locks	Leadership election and cluster state	Distributed messaging

- The **Spring Cloud project** is an umbrella project from the Spring team that implements a set of common patterns required by distributed systems, as a set of easy-to-use Java Spring libraries.
- Despite its name, Spring Cloud by itself is **not a cloud solution**. Rather, it provides a number of capabilities that are essential when developing applications targeting cloud deployments.
- By using Spring Cloud, developers just need to focus on building business capabilities using Spring Boot, and leverage the distributed, fault-tolerant, and self-healing capabilities available out of the box from Spring Cloud.

Spring Cloud

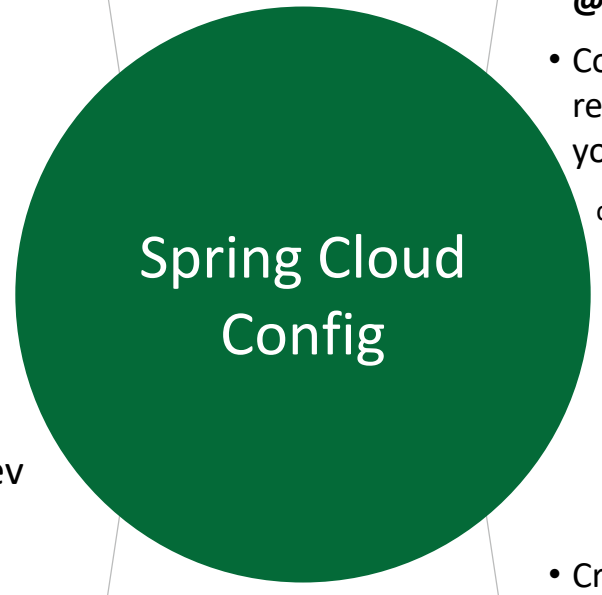
Main Components

1. **Spring Cloud Config**
   - Centralized external configuration management backed by a git repository.
   - The configuration resources map directly to Spring Environment but could be used by non-Spring applications if desired.
2. **Spring Cloud Netflix**
   Integration with various Netflix OSS components (Eureka, Hystrix, Zuul, Archaius, etc.).
3. **Spring Cloud Foundry**
   - Integrates your application with Pivotal Cloud Foundry (PCF)
   - Provides a service discovery implementation and also makes it easy to implement SSO and OAuth2 protected resources.
4. **Spring Cloud Security**
   Provides support for load-balanced OAuth2 rest client and authentication header relays in a Zuul proxy.
5. **Spring Cloud Sleuth**
   Distributed tracing for Spring Cloud applications, compatible with Zipkin, HTrace and log-based (e.g. ELK) tracing.

Spring Cloud

Spring Cloud Config

- Externalized configuration server in which applications and services can deposit, access, and manage all runtime configuration properties
- Also supports version control of the configuration properties
- Manage the configuration between different environments and be certain that applications have everything they need to run when they migrate from Dev to Test and finally to Production



Setting up Config Server

- Add config server as a dependency to the project.
- Add the **@EnableConfigServer** and **@SpringBootApplication** annotations.
- Configure your application.yml file in your resources folder. This file will be where you set up your cloud config server's access to Github.

```
cloud:
  config:
    server:
      git:
        uri: <Credentials URI>
        username:
        Password:
```

- Create a bootstrap.yml file which will live alongside your application.yml file in your src/main/resources folder, and it will contain information pointing to the config server's location wherever it's being hosted.

Spring Cloud

Illustration

Add dependency in pom.xml

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-dependencies</artifactId>
            <version>Hoxton.SR6</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-config-server</artifactId>
    </dependency>
```

Add @EnableConfigServer annotation and application.yml

```
package com.example.employee;

import org.springframework.boot.SpringApplication;

@SpringBootApplication
@EnableConfigServer
public class EmployeeServiceApplication {

    public static void main(String[] args) {
        SpringApplication.run(EmployeeServiceApplication.cla
    }

}
```

```
server:
      port: 8080

spring:
  application:
    name: employee-config-server


  cloud:
    config:
      server:
        git:
          uri: ${} --URI key goes here
          username: ${} -- username key goes here
          password: ${} -- password key goes here
```

Spring Cloud

Service Discovery

Services registration and discovery

There are two types of Service Discovery Pattern:

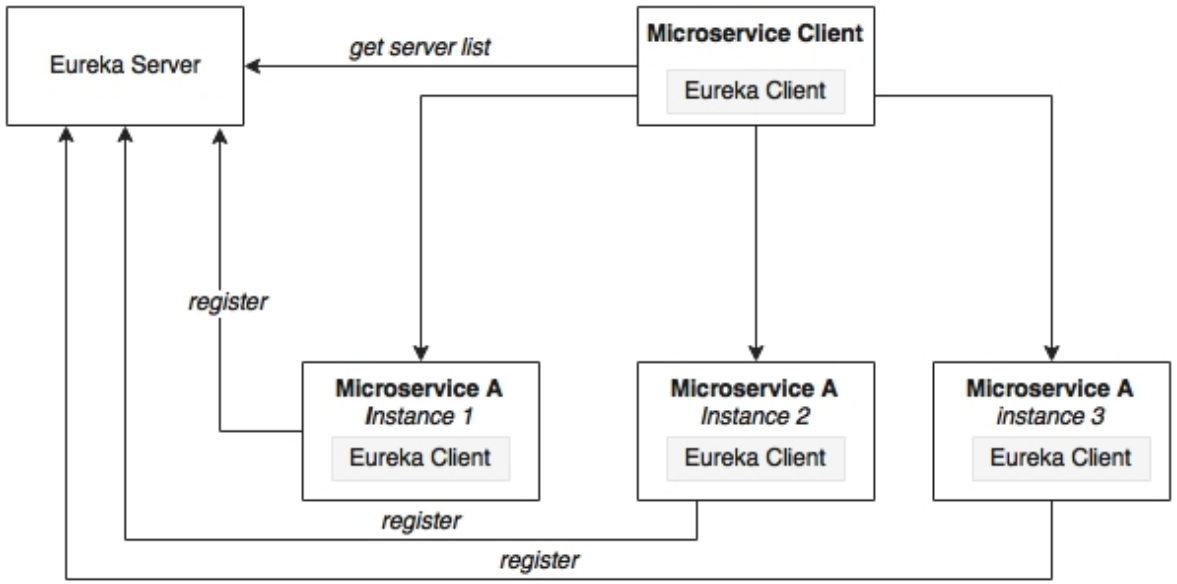
- ✓ **Client-side discovery**: The client is responsible for determining the network locations of available service instances and load balancing requests across them.
- ✓ **Server-side discovery**: The client makes a request to a service via a load balancer. The load balancer queries the service registry and routes each request to an available service instance.

Eureka

- ✓ Eureka Server is an application that holds the information about all client-service applications
- ✓ Micro service will register into the Eureka server and Eureka server knows all the client applications running on each port and IP address
- ✓ Eureka Server is also known as Discovery Server
- ✓ Eureka uses Ribbon for load balancing internally and client-side service discovery pattern

Illustration:

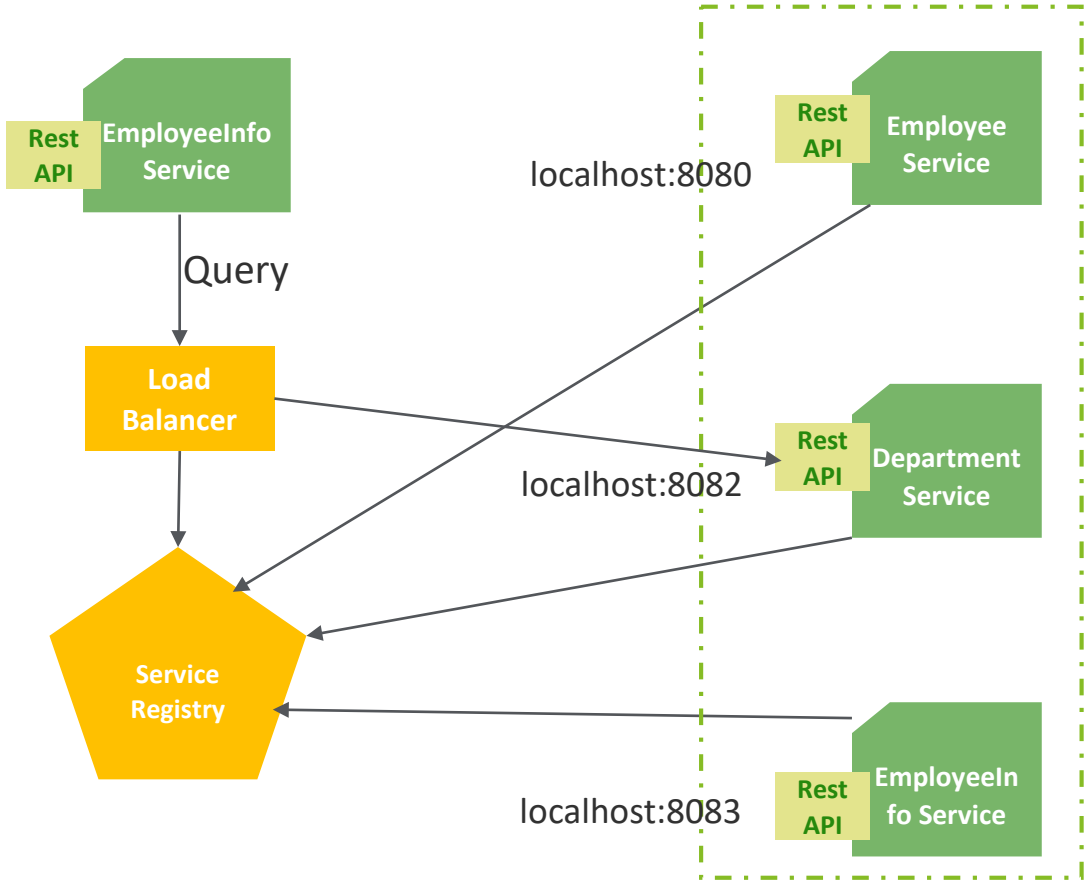
@EnableEurekaServer
@EnableDiscoveryClient



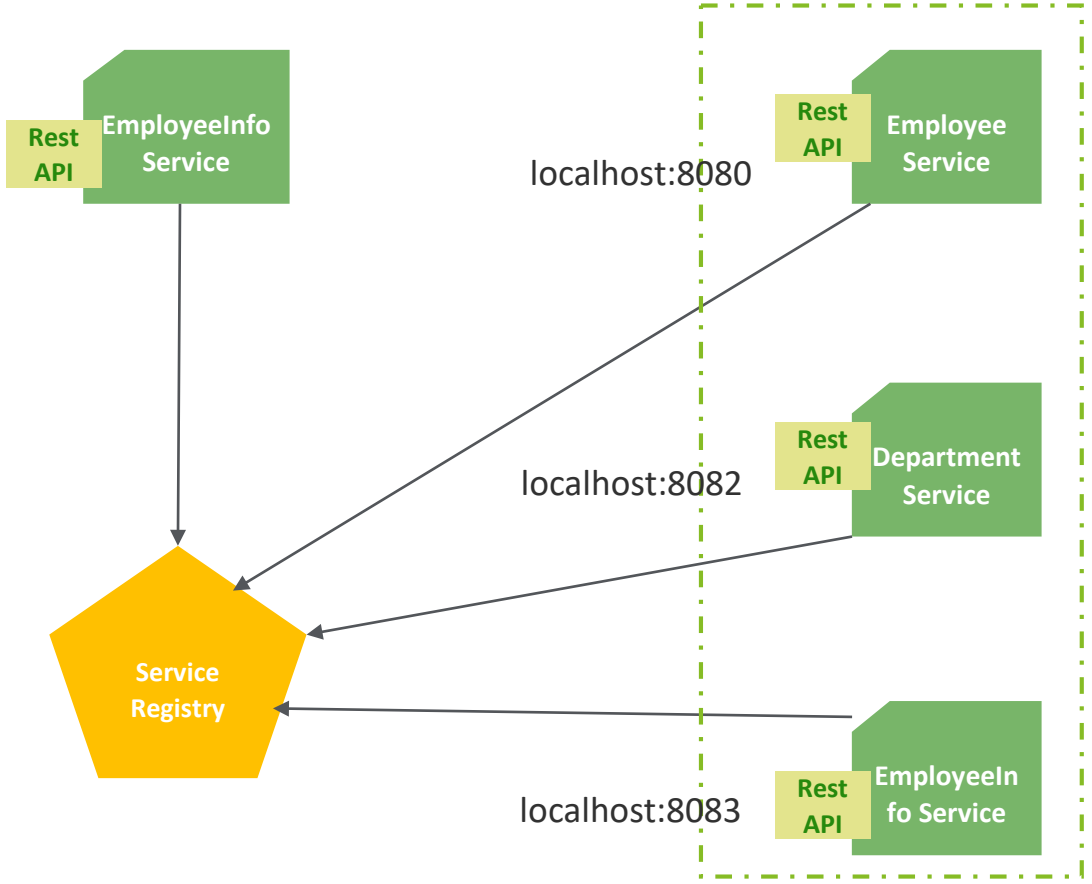
Spring Cloud

Server-side service discovery vs Client-side service discovery

Server-side service discovery



Client-side service discovery



Spring Cloud

API Gateway

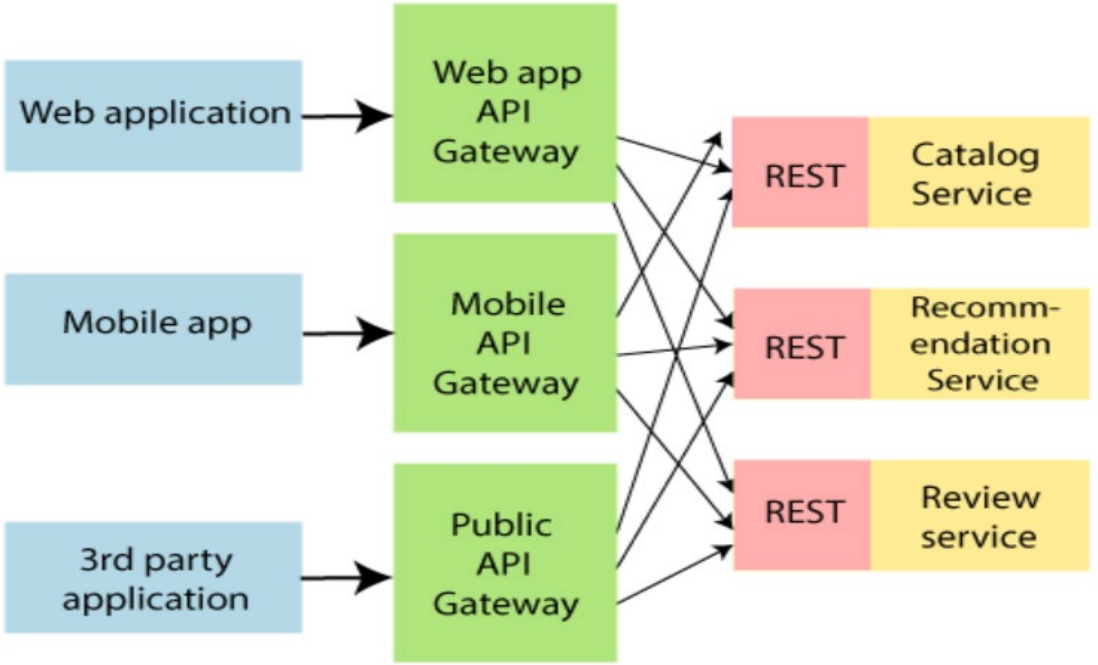
API Gateway

- ✓ The API Gateway is a server.
- ✓ It is a single entry point into a system.
- ✓ API Gateway encapsulates the internal system architecture. It provides an API that is tailored to each client.
- ✓ All the requests made by the client go through the API Gateway. After that, the API Gateway routes requests to the appropriate microservice.

API Gateway Responsibilities

- ➢ Security
- ➢ Caching
- ➢ API composition and processing
- ➢ Managing access quotas
- ➢ API health monitoring
- ➢ Versioning
- ➢ Routing

Illustration:



Knowledge Check

1. Eureka is a -

- Client-side service discovery tool
- Server-side service discovery tool

2. Which is the annotation used for denoting Eureka client?

- @EnableConfigServer
- @EnableEurekaServer
- @EnableEurekaClient
- @EurekaClient

3. Which is not a component of Spring Cloud?

- Spring Cloud Security
- Spring Cloud Docker
- Spring Cloud Netflix
- Spring Cloud Configuration

4. Postman is a -

- Deployment tool
- Health monitoring tool
- API Testing tool
- Log stashing tool

Knowledge Check

5. Load balancing is not a feature of Spring cloud?

A. True

B. False

6. Which annotation is used to setup the Config server?

A. @EnableConfigServer

B. @EnableEurekaServer

C. @EnableEurekaClient

D. @EurekaClient

7. Spring Cloud by itself is a cloud solution

A. True

B. False

8. Spring Cloud config supports version control of properties?

A. True

B. False

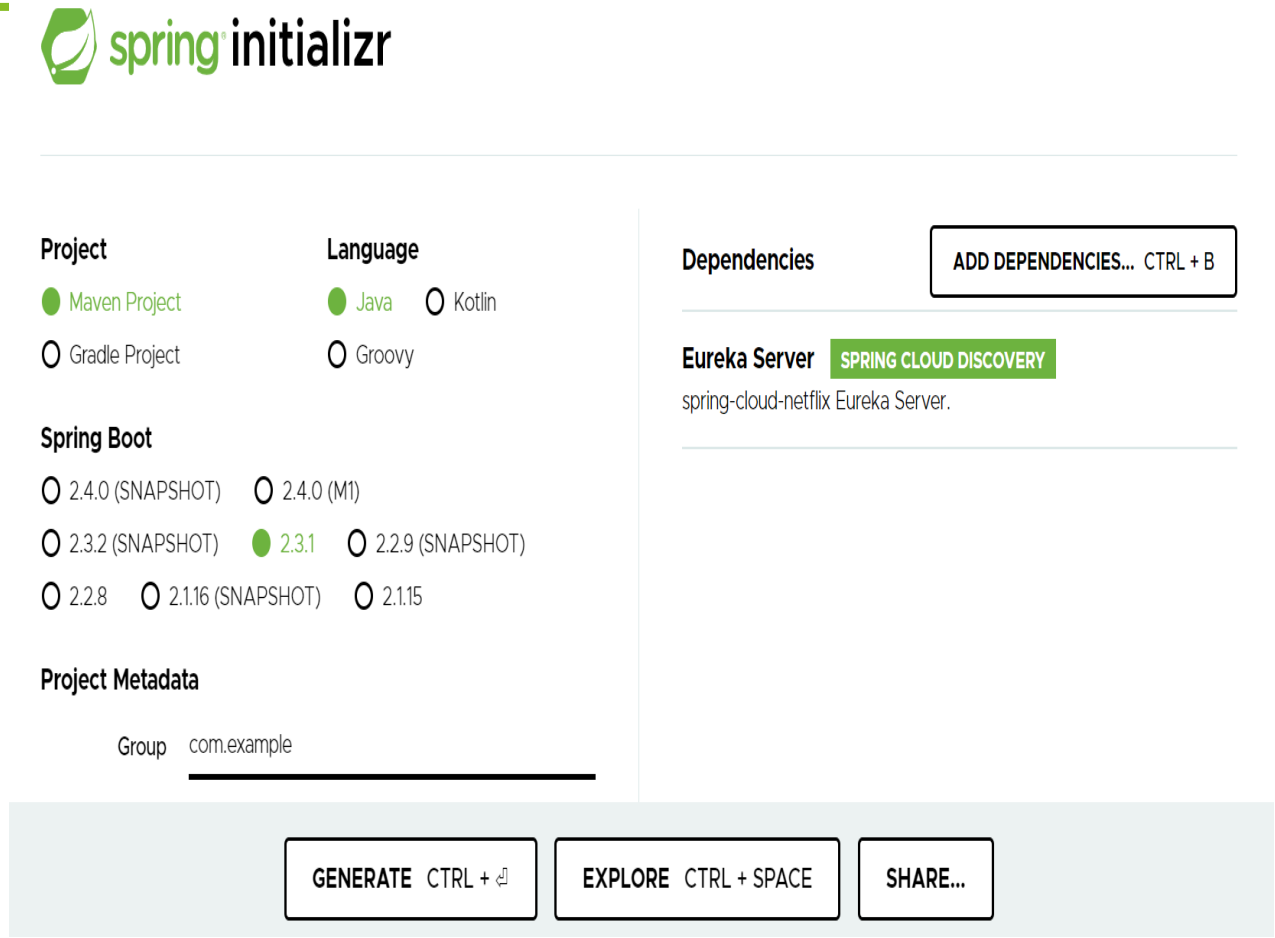
Lunch Break – 45 min.

Microservices Ecosystem & Tools

Eureka Illustration

Creation of Discovery server

- ✓ Go to start.spring.io
- ✓ Select the Spring Boot version
- ✓ Click on Add Dependencies and search for Eureka server
- ✓ Click on **GENERATE** to create the Maven project
- ✓ Import the project to your workspace as Maven project
- ✓ Eureka server is nothing but a normal Spring boot application
- ✓ Change the port number to 8761 in application.properties/application.yml if applicable
- ✓ Add @EnableEurekaServer to the ---Application.java
- ✓ Start the application



Microservices Ecosystem & Tools

Eureka Illustration

Eureka server dashboard

i. When a microservice is bootstrapped, it reaches out to the Eureka server, and advertises its existence with the binding information.

ii. Once registered, the service endpoint sends ping requests to the registry every 30 seconds to renew its lease.

iii. If a service endpoint cannot renew its lease in a few attempts, that service endpoint will be taken out of the service registry.

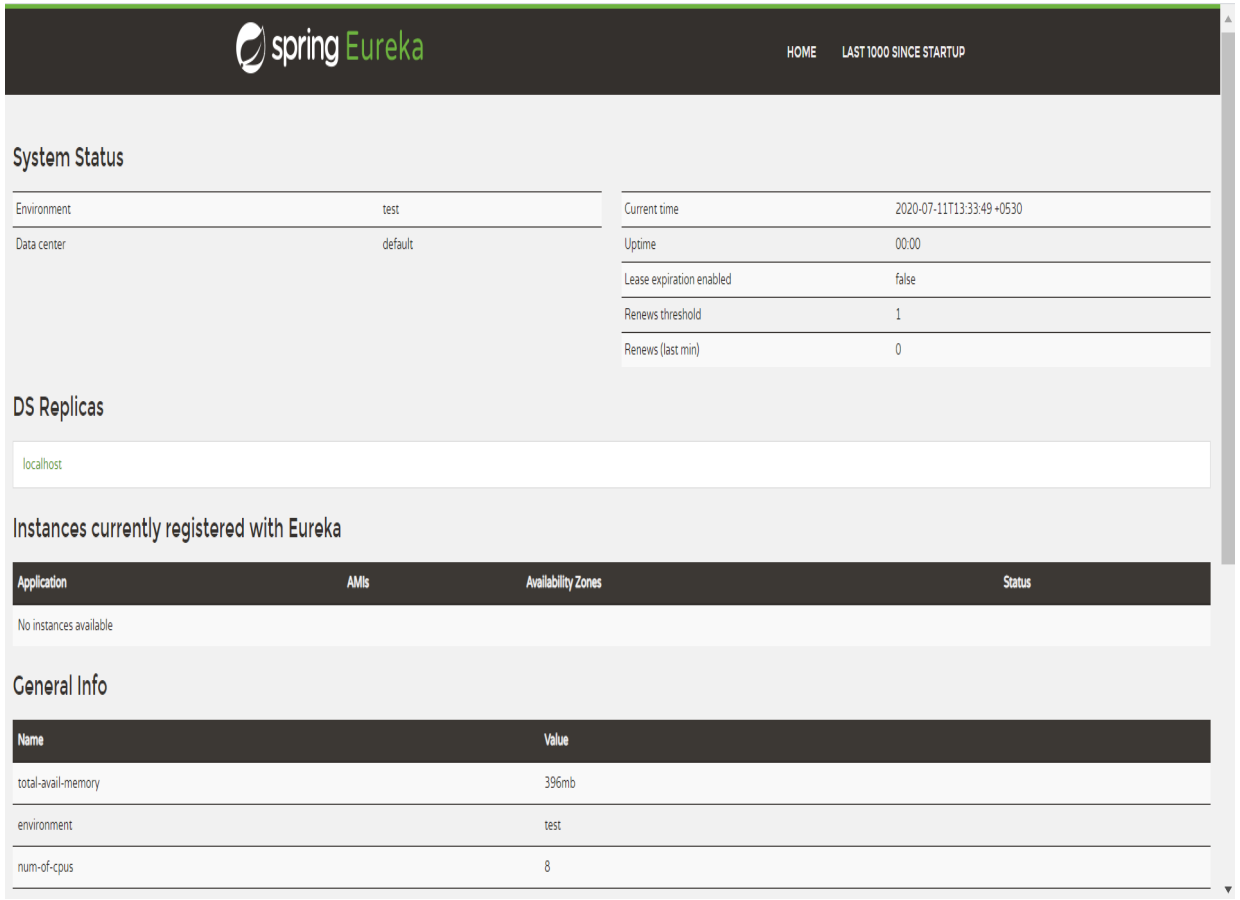
iv. When a client wants to contact a microservice endpoint, the Eureka client provides a list of currently available services based on the requested service ID.

v. The Eureka server is zone aware. Zone information can also be supplied when registering a service.

vi. When a client requests for a services instance, the Eureka service tries to find the service running in the same zone.

vii. The Ribbon client then load balances across these available service instances supplied by the Eureka client.

viii. The communication between the Eureka client and the server is done using REST and JSON.



Microservices Ecosystem & Tools

Registering Eureka Clients

Add dependency in pom.xml

```
<properties>
    <java.version>1.8</java.version>
    <spring-cloud.version>Greenwich.RELEASE</spring-cloud.version>
</properties>

<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
        <exclusions>
            <exclusion>
                <groupId>org.junit.vintage</groupId>
                <artifactId>junit-vintage-engine</artifactId>
            </exclusion>
        </exclusions>
    </dependency>
</dependencies>
```

Add @EnableEurekaClient annotation

```
package com.example.department;

import org.springframework.boot.SpringApplication;

@SpringBootApplication
@EnableEurekaClient
public class DepartmentServiceApplication {

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

}
```

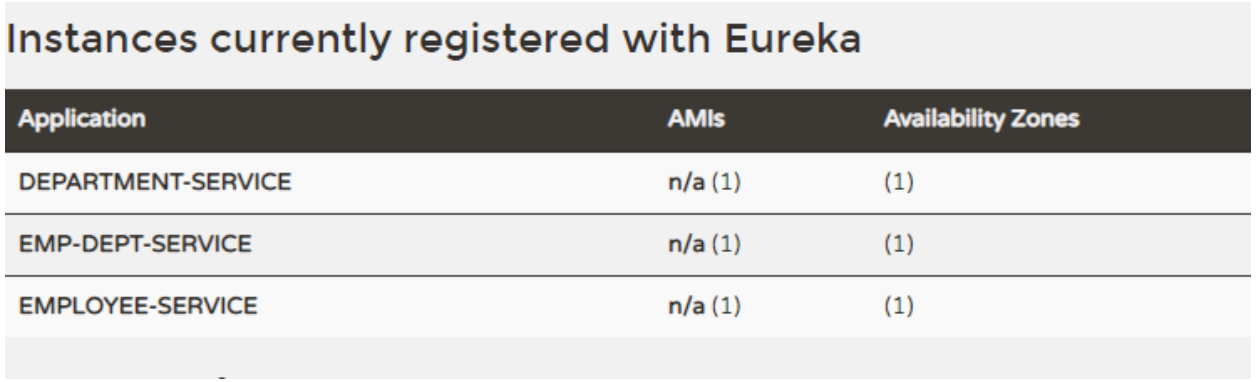
```
server:
   port: 8082

spring:
  application:
    name: department-service
```

Microservices Ecosystem & Tools

Registering Eureka Clients

Change EmpDeptService.java by replacing hard-coded values

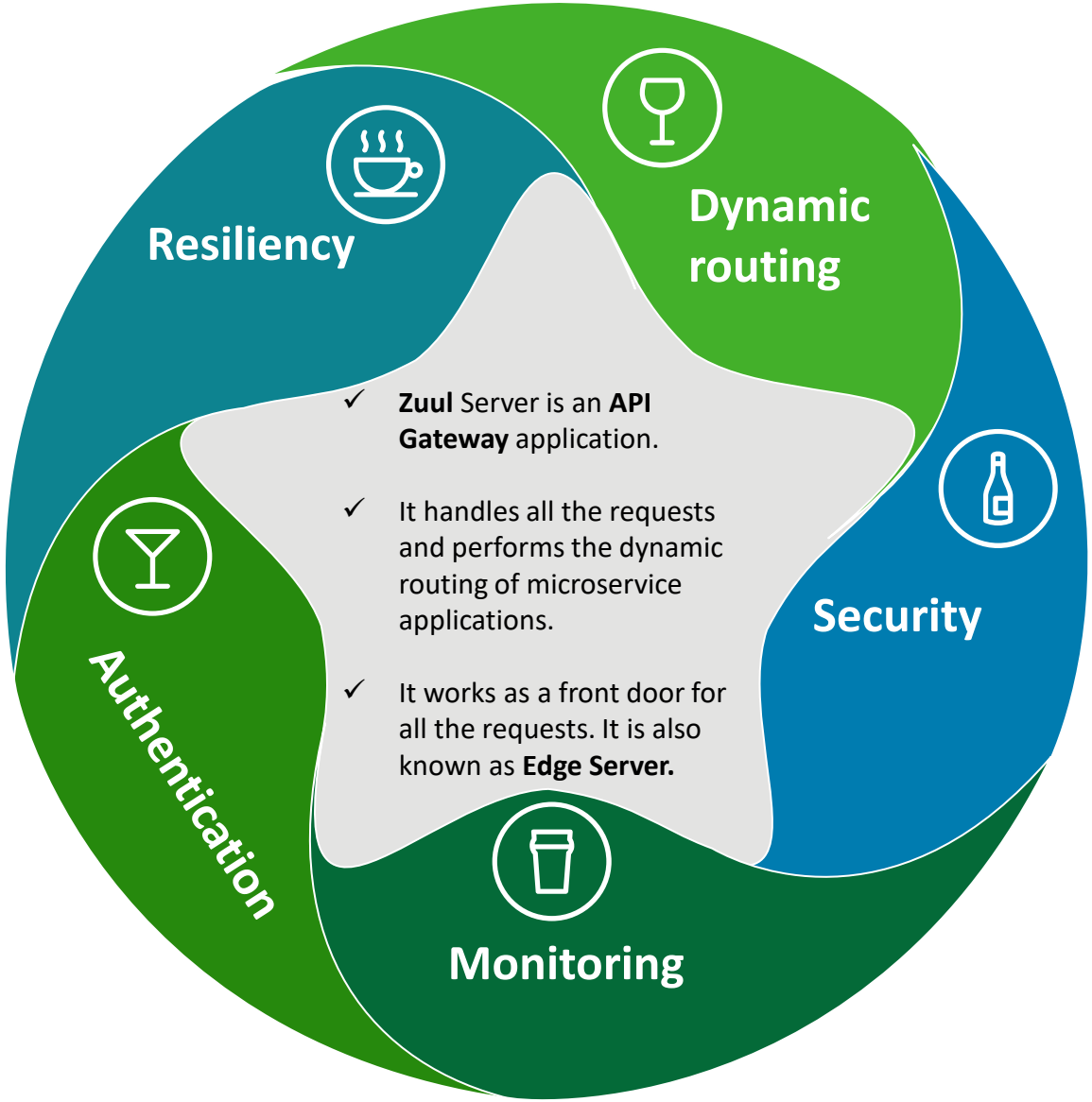
```
@Service
public class EmpDeptService {

    @Autowired
    RestTemplate restTemplate;


    public Employee getEmployeeById(Integer empId) {
        return restTemplate.getForObject(
                "http://employee-service/employeeservice/employees/"+empId,
                Employee.class);
    }

    public Department getDeparmentById(Integer deptId) {
        return restTemplate.getForObject(
                "http://department-service/departmentservice/departments/"+deptId,
                Department.class);
    }

}
```



Three services are currently registered in Eureka server which can be seen in the Eureka dashboard.

Microservices Ecosystem & Tools

Zuul API Gateway

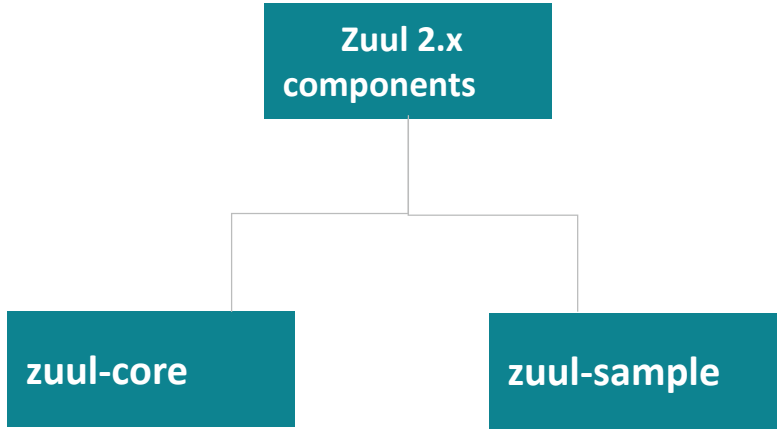


Microservices Ecosystem & Tools

Zuul API Gateway - Filters

- ✓ **Authentication and Security**: It provides authentication requirements for each resource.
- ✓ **Insights and Monitoring**: It tracks meaningful data and statistics that give us an accurate view of production.
- ✓ **Dynamic Routing**: It dynamically routes the requests to different backed clusters as needed.
- ✓ **Stress Testing**: It increases the traffic to a cluster in order to test performance.
- ✓ **Load Shedding**: It allocates capacity for each type of request and drops a request that goes over the limit.
- ✓ **Static Response Handling**: It builds some responses directly at the edge instead of forwarding them to an internal cluster.
- ✓ **Multi-region Resiliency**: It routes requests across AWS regions in order to diversify our ELB usage.

Components of Zuul:



Microservices Ecosystem & Tools

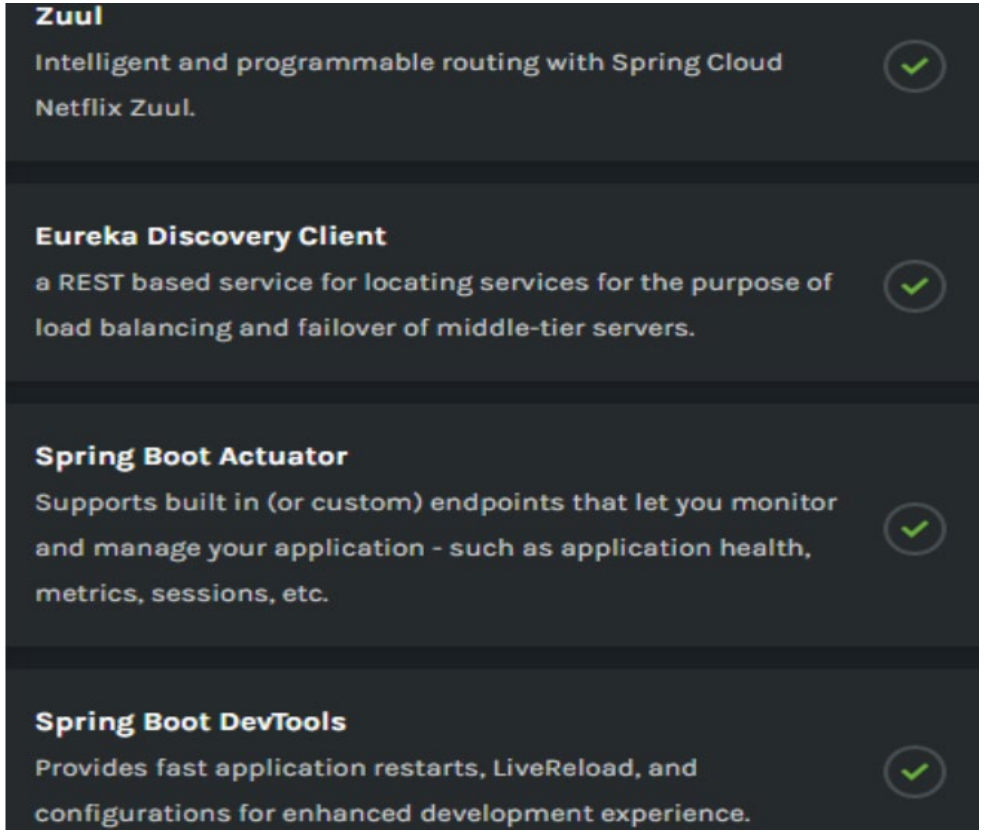
Zuul API Gateway Server

Setting up Zuul API Gateway Server

- ✓ Create a component for the Zuul API Gateway
- ✓ Decide the things that the Zuul API Gateway should do
- ✓ All the important requests are configured to pass through the Zuul API Gateway

Steps to set up the Zuul API Gateway server.

- ➢ Open Spring Initializer https://start.spring.io.
- ➢ Provide the Group name. ex: com.deloitte.microservices.
- ➢ Provide the Artifact. We have provided netflix-zuul-api-gateway-server.
- ➢ Add the dependencies: Zuul, Eureka Discovery, Actuator, and devTools.
- ➢ Click on the Generate button, import into STS or Eclipse.



Microservices Ecosystem & Tools

Zuul API Gateway Server

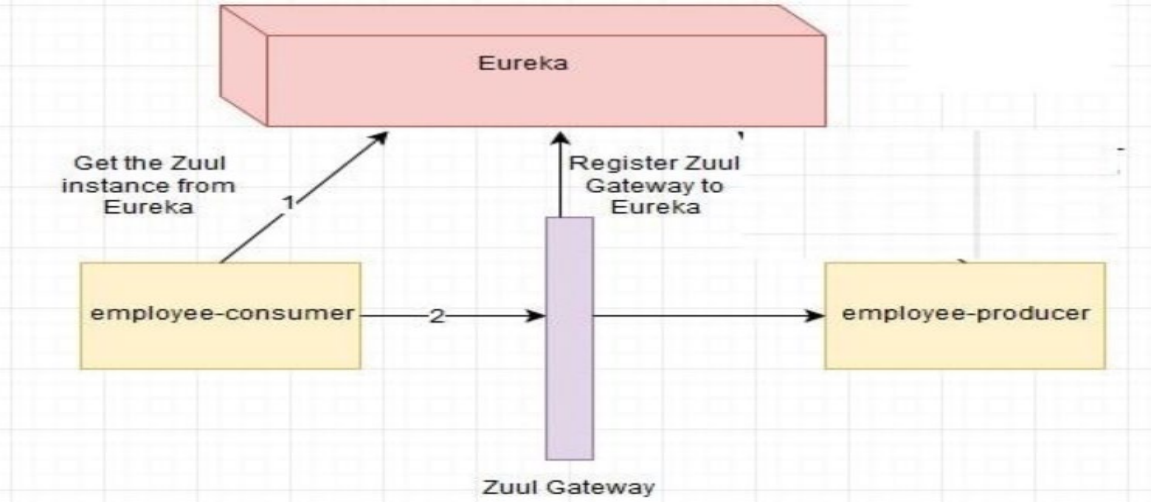
Contd..

Open the **NetflixZuulApiGatewayServerApplication.java** file and enable zuul proxy and discovery client by using the annotations **@EnableZuulProxy** and **@EnableDiscoveryClient**, respectively.

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
import org.springframework.cloud.netflix.zuul.EnableZuulProxy;
@EnableZuulProxy
@EnableDiscoveryClient
@SpringBootApplication
public class NetflixZuulApiGatewayServerApplication
{
public static void main(String[] args)
{
SpringApplication.run(NetflixZuulApiGatewayServerApplication.class, args);
}
}
```

Open **application.properties** file and configure the application **name**, **port**, and **eureka naming server**.

```
spring.application.name=netflix-zuul-api-gateway-server
server.port=8765
eureka.client.service-url.default-zone=http://localhost:8765/eureka
```



Knowledge Check

1. Zuul is a -

- Client-side service discovery tool
- Gateway application

2. Which is the annotation used for denoting Zuul Proxy?

- @EnableConfigServer
- @EnableZuulProxy
- @EnableEurekaClient
- @EurekaClient

3. Dynamic routing is not a responsibility of Zuul API Gateway?

- TRUE
- FALSE

4. API gateway ensures

A. Request routing

B. Protocol translation

C. Composition

Knowledge Check

5.Which of the following works as front door for all the requests?

A. Zuul server

B. Hystrix

C. Eureka

6. External clients communicate with Microservices using?

A. API gateway

B. Config Server

C. Messaging

7. Caching is not a responsibility of Zuul API Gateway?

A. TRUE

B. FALSE

8. Which of the following provides client side load -balancing?

A. Ribbon

B. MQ

C. None of the above

Building Microservices – Deep Dive with an example

Building Microservices

Key Objectives

01 **Pre-requisites to build Microservices**

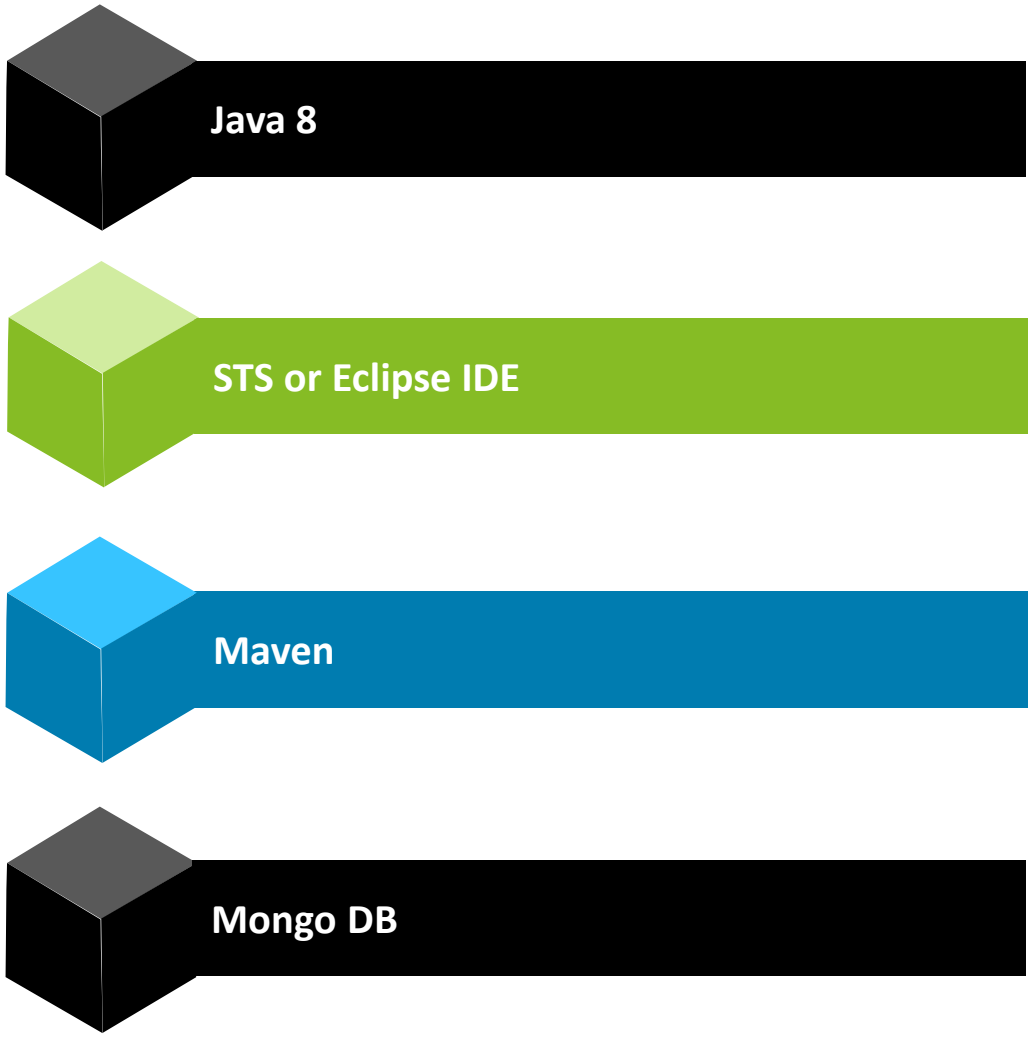
02 **Use Case Details**

03 **Design and Approach**

04 **Build Microservices**

Building Microservices

Pre-requisites



In this example we will build a Spring Boot Microservice using Java and MongoDB.

Please setup the required environment variables related to the tools listed here.

Building Microservices

Use Case Details

Building Microservice to support e-commerce application

❑ PROBLEM STATEMENT

- Need to design an e-commerce application where user can do online shopping for branded clothes.
- User should be able to search for a specific brand of clothes using the search button and filter the items as per choice.
- User should have the ability to add items in the cart and confirm the order in the cart summary page.
- User should also be able to access the in progress and past orders when navigated to my orders page.
- The personal details can be added / edited in the My accounts page and should get saved accordingly.

❑ HIGH LEVEL REQUIREMENT

A Microservice should be build to support the e-commerce application with following functionalities:

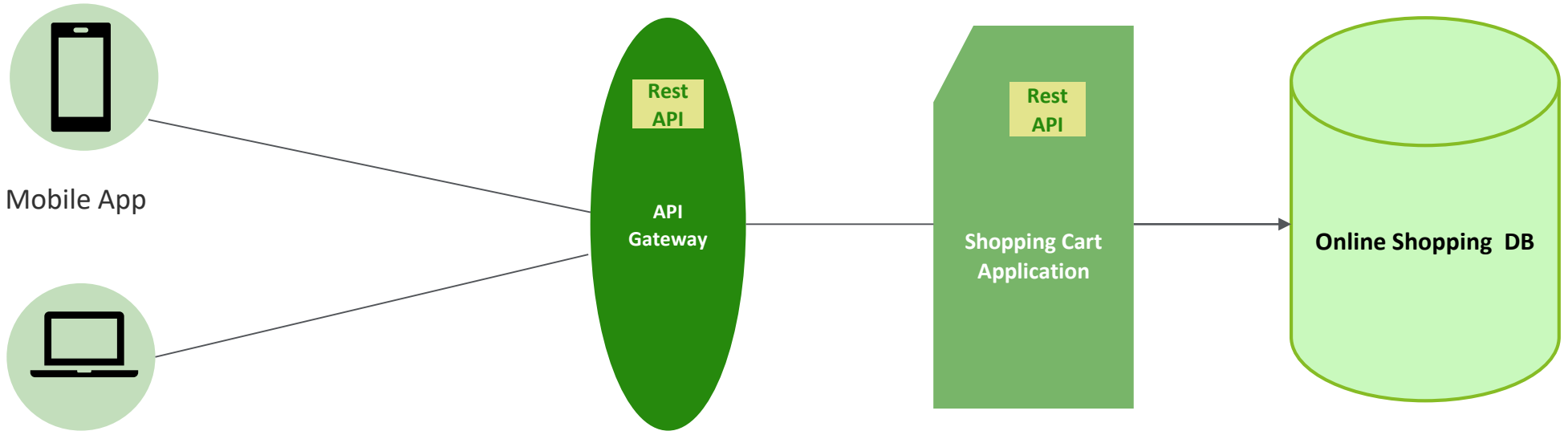
✓ Sign in existing user or register new user
✓ Update existing user information
✓ Get available products
✓ Place Orders
✓ Get Orders History
✓ Cancel / Update Orders

Building Microservices

Design and Approach

Building Microservices

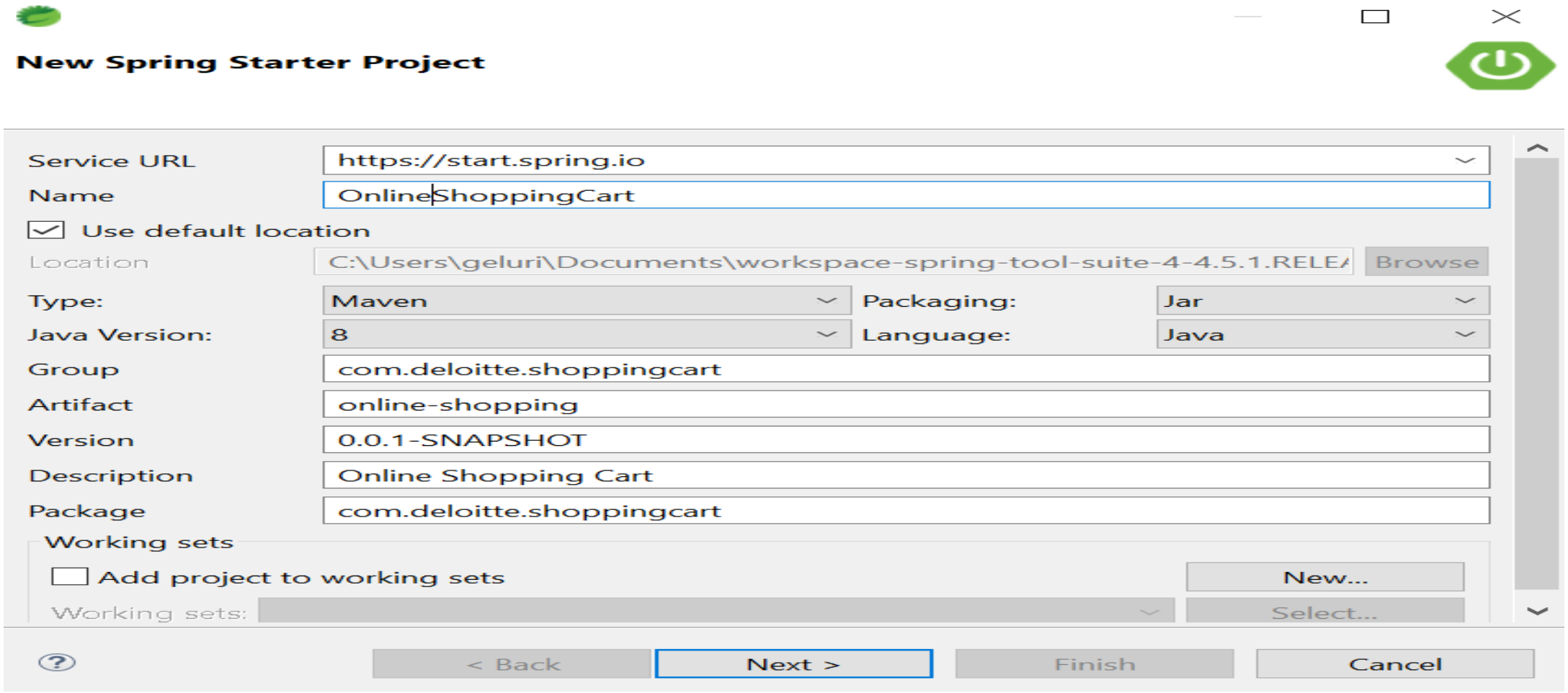
Design and Approach



Building Microservices

Implementation

Open STS -> Click on New Spring Starter Project -> Give Required Details -> Finish



Building Microservices

Implementation Contd..

Shopping Cart Application can be generated as shown below.

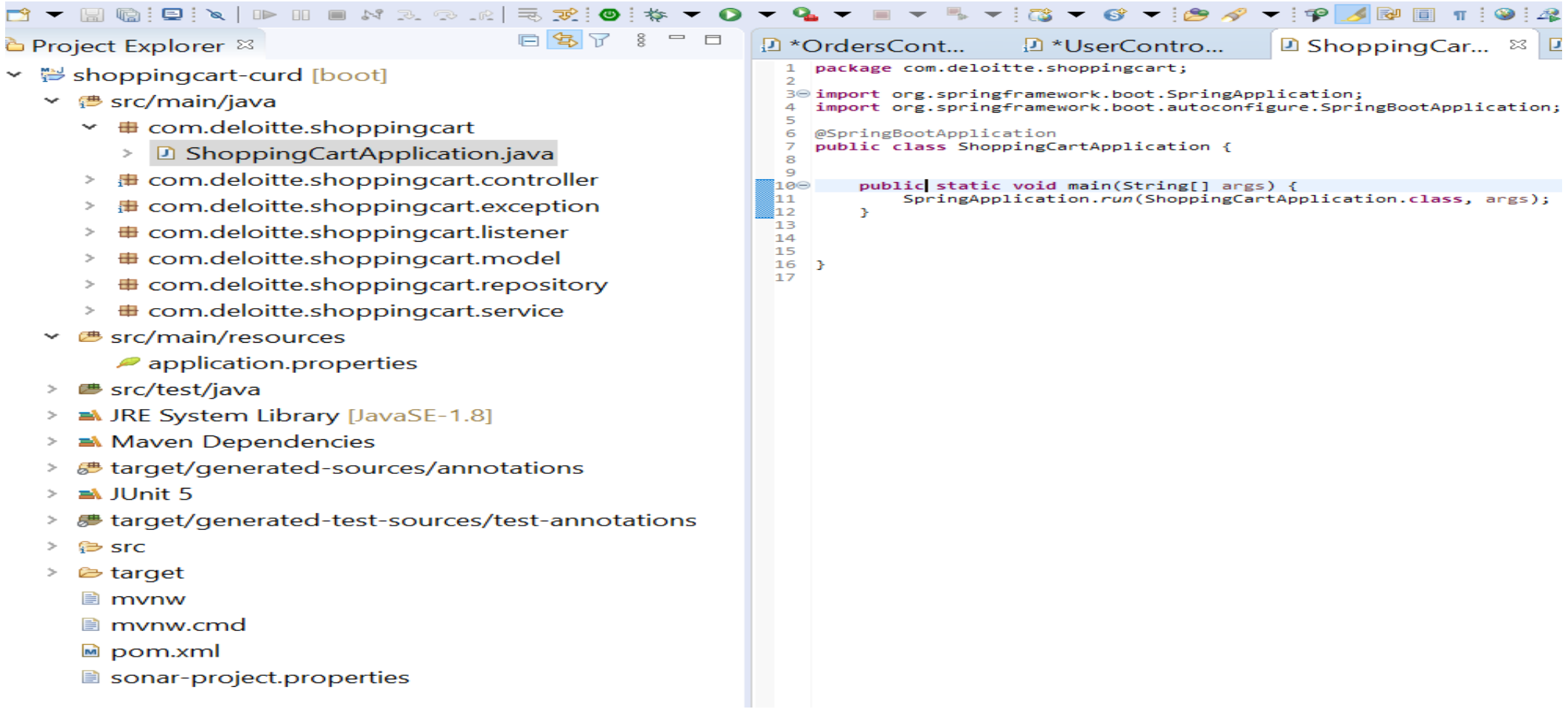
```
1  package com.deloitte.shoppingcart;
2
3  import org.springframework.boot.SpringApplication;
4  import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6  @SpringBootApplication
7  public class ShoppingCartApplication {
8
9
10     public static void main(String[] args) {
11         SpringApplication.run(ShoppingCartApplication.class, args);
12     }
13
14
15
16 }
```

Building Microservices

Implementation

Contd..

Code structure



Building Microservices

Implementation Contd..

POM.xml can be created as shown below.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.1.0.RELEASE</version>
        <relativePath /> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.deloitte.shoppingcart</groupId>
    <artifactId>shoppingcart-curd</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>shoppingcart-springboot-mongodb-web-app</name>
    <description>Shopping Cart Backend - Spring Boot + MongoDB</description>

    <properties>
        <java.version>1.8</java.version>
    </properties>

    <dependencies>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-mongodb</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-engine</artifactId>
            <scope>test</scope>

        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
            <exclusions>
                <exclusion>
                    <groupId>org.junit.vintage</groupId>
                    <artifactId>junit-vintage-engine</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
    </dependencies>

```

Building Microservices

Implementation

Contd..

OrdersController can be created as shown below.

```
import java.util.List;
@RestControll  shoppingcart-curd/sonar-project.properties
@RequestMappi
public class OrdersController {

    private final OrderRepository orderRepository;
    private static final Logger logger = LoggerFactory.getLogger(OrdersController.class);

    @Autowired
    OrdersController(OrderRepository orderRepository) {
        this.orderRepository = orderRepository;
    }
    /**
     * To get Orders using emailId
     * @param emailId
     * @return
     * @throws ResourceNotFoundException
     */
    @GetMapping("/myorders/{emailId}")
    public ResponseEntity<List<Order>> getOrderByEmailId(@PathVariable String emailId)
            throws ResourceNotFoundException {
        logger.debug(" --- Getting all orders for given user ---");
        List<Order> orders = this.orderRepository.findByEmailId(emailId);
        if (null != orders && !orders.isEmpty()) {
            return new ResponseEntity<List<Order>>(orders, HttpStatus.OK);
        } else {
            throw new ResourceNotFoundException(" Orders not found for EmailId :: " + emailId);
        }
    }
    /**
     * To Save orders data
     * @param order
     * @return
     */
    @PostMapping("/placeorder")
    public ResponseEntity<Order> saveOrderDetails(@Valid @RequestBody Order order) {
        logger.debug(" --- Saving Order Details for given EmailId --- ");
        orderRepository.save(order);
        return new ResponseEntity<Order>(order, HttpStatus.OK);
    }

    /**
     * To cancel INPROGRESS orders
     * @param orderId
     * @param order
     * @return
     * @throws ResourceNotFoundException
     */
    @PutMapping("/orders/updatestatus/{id}")
    public ResponseEntity<Order> updateOrderStatus(@PathVariable(value = "id") Long orderId,
            @Valid @RequestBody Order order) throws ResourceNotFoundException {
        logger.debug(" --- Updating Order Status based on OrderId --- ");
        Order dbOrder = orderRepository.findById(orderId)
                .orElseThrow(() -> new ResourceNotFoundException(" Order not found for OrderId :: " + orderId));
        dbOrder.setStatus(order.getStatus());
        dbOrder.setDeliveredDate(order.getDeliveredDate());
        final Order updatedOrder = orderRepository.save(dbOrder);
        return new ResponseEntity<Order>(updatedOrder, HttpStatus.OK);
    }
```

Building Microservices

Implementation Contd..

ProductsController can be created as shown below.

```
1  package com.deloitte.shoppingcart.controller;
2
3  import java.util.List;
4
5  @RestController
6  @RequestMapping("/shoppingcart")
7  public class ProductController {
8
9      private final ProductRepository productRepository;
10     private static final Logger logger = LoggerFactory.getLogger(ProductController.class);
11
12     @Autowired
13     ProductController(ProductRepository productRepository) {
14         this.productRepository = productRepository;
15     }
16
17     /**
18      * To get all the available products.
19      * @return
20      */
21     @GetMapping("/products")
22     public List<Product> getProducts() {
23         logger.debug("---Getting all products---");
24         return this.productRepository.findAll();
25     }
26
27
28 }
29
```

Building Microservices

Implementation Contd..

UserController can be created as shown below.

```
public class UserController {

    private final UserRepository userRepository;
    private static final Logger Logger = LoggerFactory.getLogger(UserController.class);

    @Autowired
    UserController(UserRepository userRepository) {
        this.userRepository = userRepository;
    }
    /**
     * To allow the new user to register into the online shopping application
     * @param user
     * @return
     * @throws ResourceNotFoundException
     */
    @PostMapping("/signup")
    public ResponseEntity<?> login(@Valid @RequestBody User user) throws ResourceNotFoundException {

        User dbUser = userRepository.findByEmailId(user.getEmailId());
        if (null != dbUser) {
            throw new ResourceNotFoundException("User already registered found withi emailId :: " + user.getEmailId());

        } else {
            userRepository.save(user);
            return new ResponseEntity<User>(user, HttpStatus.OK);
        }
    }

    /**
     * To update the user information such as billing address and shipping address.
     * @param user
     * @return
     * @throws ResourceNotFoundException
     */
    @PutMapping("/updateUserDetails")
    public ResponseEntity<?> updateUserData(@Valid @RequestBody User user) throws ResourceNotFoundException {
        User dbUser = userRepository.findByEmailId(user.getEmailId());
        if (null != dbUser) {
            if (null != user.getDefaultBillingAddress()) {
                dbUser.setDefaultBillingAddress(user.getDefaultBillingAddress());
            }
            if (null != user.getDefaultShippingAddress()) {
                dbUser.setDefaultShippingAddress(user.getDefaultShippingAddress());
            }
            userRepository.save(dbUser);
            return new ResponseEntity<User>(user, HttpStatus.OK);
        } else {
            throw new ResourceNotFoundException("User not found for this id :: " + user.getEmailId());
        }
    }

}
```

Building Microservices

Implementation

Contd..

OrderRepository can be created as shown below.

```
1  package com.deloit
2
3  import java.util.List;
8
9  public interface OrderRepository extends MongoRepository<Order, Long> {
10
11     public List<Order> findByEmailId(String emailId);
12
13     public List<Order> findAll();
14
15 }
16
```

shoppingcart-curd/src/main/java/com/deloitte/shoppingcart/cont

ProductRepository can be created as shown below.

```
1  package com.deloitte.shoppingcart.repository;
2
3  import java.util.List;
8
9  public interface ProductRepository extends MongoRepository<Product, String> {
10
11     public List<Product> findAll();
12 }
13
```

Building Microservices

Implementation Contd..

UserRepository can be created as shown below.

```java
package com.deloitte.shoppingcart.repository;


import org.springframework.data.mongodb.repository.MongoRepository;


public interface UserRepository extends MongoRepository<User, String> {

    public User findByEmailId(String emailId);


}

```

Setup application.properties as shown below.

```
# MONGODB (MongoProperties)
spring.data.mongodb.uri=mongodb://localhost:27017/OnlineShopping

# Logging Properties
logging.level.root=WARN
logging.level.org.springframework.web=INFO
logging.level.com.hkb.shoppingcart=INFO

# Logging pattern for the console
logging.pattern.console= "%d{yyyy-MM-dd HH:mm:ss} - %msg%n"

# Logging pattern for file
logging.pattern.file= "%d{yyyy-MM-dd HH:mm:ss} [%thread] %-5level %logger{36} - %msg%n"

logging.file=${java.io.tmpdir}/OnlineShopping.log
```

Building Microservices

Implementation Contd..

To run this application, run the following commands:

- ✓ Run MongoDB - Command to run at command prompt – mongod
- ✓ Insert sequence into mongodb - Run command and execute the command - mongo
- ✓ Switch to DB - OnlineShopping - Command>use OnlineShopping
- ✓ Insert sequence identifier for the uniqueness of the Order

  Command>db.database_sequences.insert({_id:"database_sequences",sequence_value:1});
- ✓ Run Spring Boot Microservice - mvnw spring-boot:run

Check if the microservice is running fine or not: *http://localhost:8080/shoppingcart/products*

This microservice loads the products items into MongoDB and the above URL shows the same entered data.

Assignment

Microservices
Assignment

Music Rating portal

Create the following using microservice architecture:

MUSICALBUM-INFO-SERVICE: Exposes REST APIs to retrieve music album info

API list

GET - /musicalbum/{musicalbumid}

```
{
        musicAlbumId:           Integer,
        musicAlbumName:         String,
        musicAlbumDescription:  String
}
```

RATINGS-DATA-SERVICE: Exposes REST APIS to retrieve the ratings information by music_album_id or user_id

API list

GET - /ratingsdata/musicalbum/{musicalbumid}

```
{
        musicAlbumId:   Integer,
        rating:         Integer
}
```

GET - /ratingsdata/user/{userid}

```
{
        userId:   Integer
        ratings: [{musicAlbumId: Integer, rating: Integer}]
}
```

Microservices
Assignment

MUSICALBUM-CATALOGUE-SERVICE:

- Exposes REST APIs to retrieve the Music Album Details along with the ratings for a given user.
- This service should communicate with MUSICALBUM-INFO-SERVICE and RATINGS-DATA-SERVICE to fetch the data, consolidate it and send back the response.

API list

GET - /musicalbumcatalogue/{userid}

RESPONSE STRUCTURE -

```
[{
        musicAlbumId:          Integer,
        musicAlbumName:        String,
        musicAlbumDescription: String,
        rating:                Integer
}]
```

Key points to note:

1. Use Eureka server to register the services. No hardcoding of local URLs
2. You may enhance the microservices with APIs to add new music album information, enable the user to add the ratings for the music albums
3. Use Zuul API Proxy
4. In case you are using database, please use Spring Cloud Config Server to externalize the database configurations(GitHub Repo is required).
5. Use POSTMAN to test your APIs
6. Use Java 8 features and Springboot
7. Happy Coding ☺

Any Questions ?

Thank You

Deloitte.