Java EE Programming

COMP 303

Lecture 10: Spring Micro Services and spring cloud

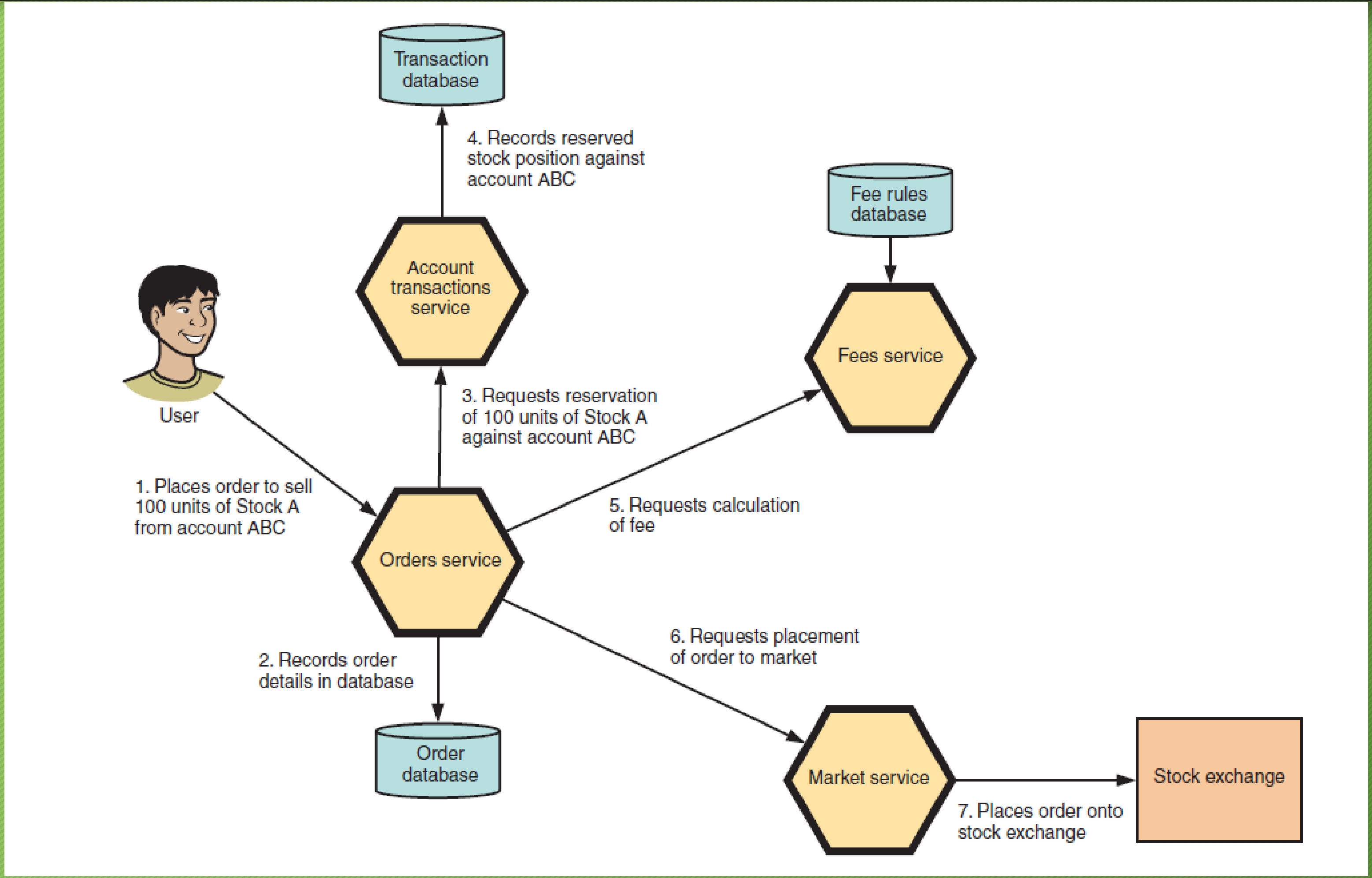
Micro Services

- Microservices are both an architectural style which works with five key principles: autonomy, resilience, transparency, automation, and alignment.
- Microservices reduce friction in development, enabling autonomy, technical flexibility, and loose coupling.
- Designing microservices can be challenging because of the need for adequate domain knowledge and balancing priorities across teams.
- Complexity in long-running software systems is unavoidable, but you can delivervalue sustainably in these systems if you make choices that minimize friction and risk.
- Reliably incident-free (“boring”) deployment reduces the risk of microservices by making releases automated and provable.

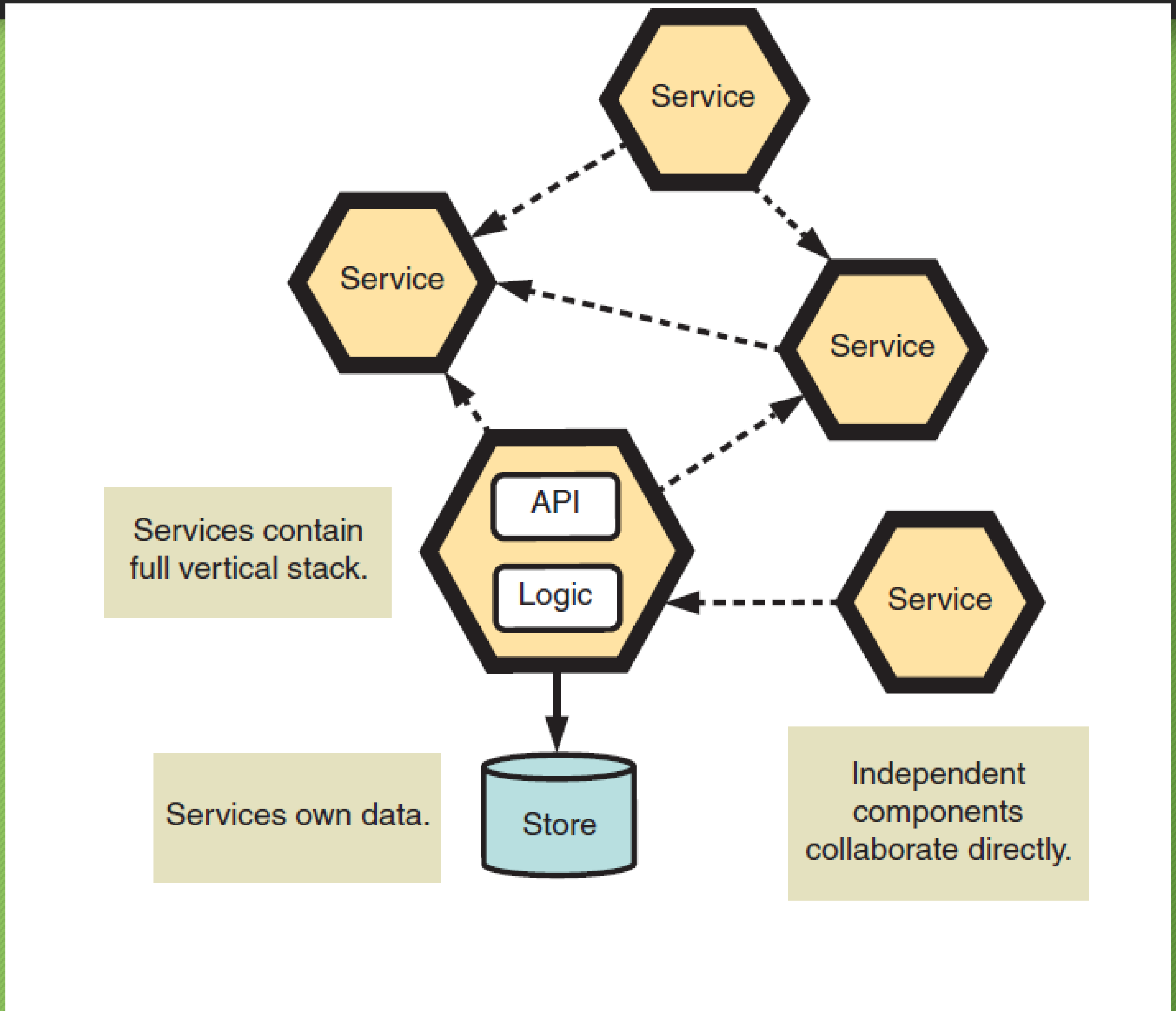
Micro Services Characteristics

- Microservice is *responsible for a single capability. This might be business related or represent a shared technical capability, such as integration with a third party (the stock exchange).*
- A microservice *owns its data store*, if it has one. This reduces coupling between services because other services can only access data they don’t own through the interface that a service provides.
- Microservices themselves, not the messaging mechanism that connects them nor another piece of software, are *responsible for choreography and collaboration*

Understanding a microservice


Transaction database
4. Records reserved stock position against account ABC
Fee rules database
Account transactions service
User
Fees service
3. Requests reservation of 100 units of Stock A against account ABC
1. Places order to sell 100 units of Stock A from account ABC
5. Requests calculation of fee
Orders service
6. Requests placement of order to market
2. Records order details in database
Order database
Market service
Stock exchange
7. Places order onto stock exchange

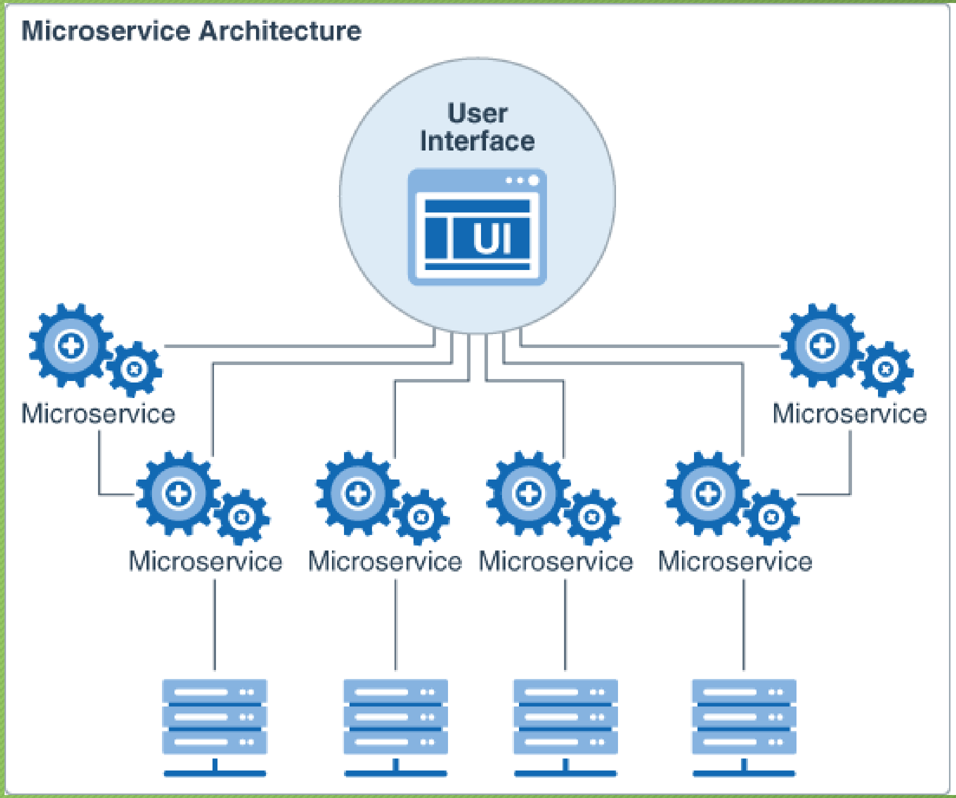
Micro Service Architecture

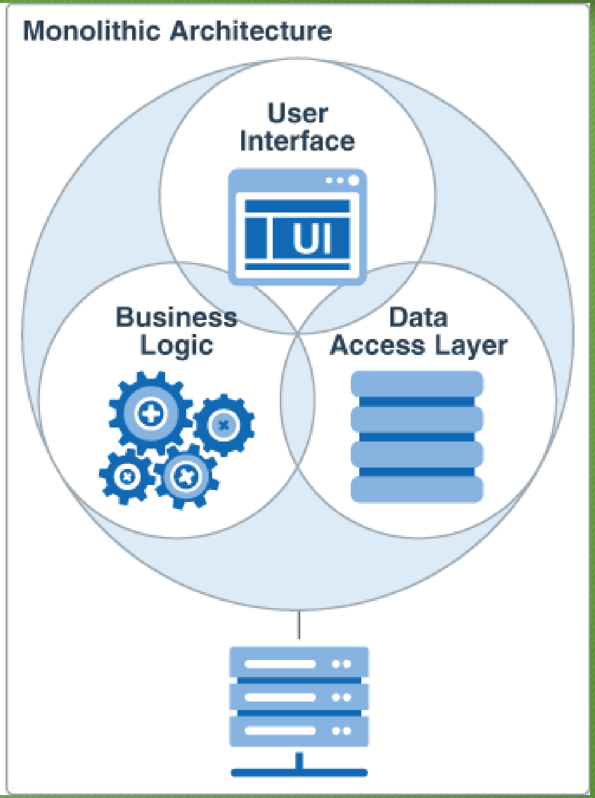

Service
Service
Service
API
Logic
Service
Services contain full vertical stack.
Services own data.
Store
Independent components collaborate directly.

Why Microservices Architecture?

- Provides solution for monolithic system issues.
- Allows to build small logical independent systems.
- Supports cloud and builds cloud app with minimum effort.
- Uses lightweight protocols HTTP, REST and JMS.
- Adopts new technologies and changes in technology.
- Works well with agile approach.

Monolithic System vs. Microservices





https://docs.oracle.com/en/solutions/learn-architect-microservice/index.html#GUID-BDCEFE30-C883-45D5-B2E6-325C241388A5

Microservices vs. Monolithic Architecture

Characteristic	Microservices Architecture	Monolithic Architecture
Unit design	The application consists of loosely coupled services. Each service supports a single business task.	The entire application is designed, developed, and deployed as a single unit.
Functionality reuse	Microservices define APIs that expose their functionality to any client. The clients could even be other applications.	The opportunity for reusing functionality across applications is limited.
Communication within the application	To communicate with each other, the microservices of an application use the request-response communication model. The typical implementation uses REST API calls based on the HTTP protocol.	Internal procedures (function calls) facilitate communication between the components of the application. There is no need to limit the number of internal procedure calls.
Technological flexibility	Each microservice can be developed using a programming language and framework that best suits the problem that the microservice is designed to solve.	Usually, the entire application is written in a single programming language.
Data management	Decentralized: Each microservice may use its own database.	Centralized: The entire application uses one or more databases.
Deployment	Each microservice is deployed independently, without affecting the other microservices in the application.	Any change, however small, requires redeploying and restarting the entire application.
Maintainability	Microservices are simple, focused, and independent. So the application is easier to maintain.	As the application scope increases, maintaining the code becomes more complex.
Resiliency	The application functionality is distributed across multiple services. If a microservice fails, the functionality offered by the other microservices continues to be available.	A failure in any component could affect the availability of the entire application.
Scalability	Each microservice can be scaled independently of the other services.	The entire application must be scaled, even when the business requirement is for scaling only certain parts of the application.

https://docs.oracle.com/en/solutions/learn-architect-microservice/index.html#GUID-BDCEFE30-C883-45D5-B2E6-325C241388A5

Who uses Micro Services?

- *The Guardian* - content distribution
- SoundCloud - Netflix
- Transport and logistics - Hailo, Uber
- e-commerce - Amazon, Gilt, Zalando
- Banking - Monzo
- Social media –Twitter
- Amazon AWS

Java Microservices Framework

- Spring Boot based on IoC and AOP
- Swagger - https://swagger.io/
- Jersey – RESTful framework with JAX-RS APIs -

Spring Cloud

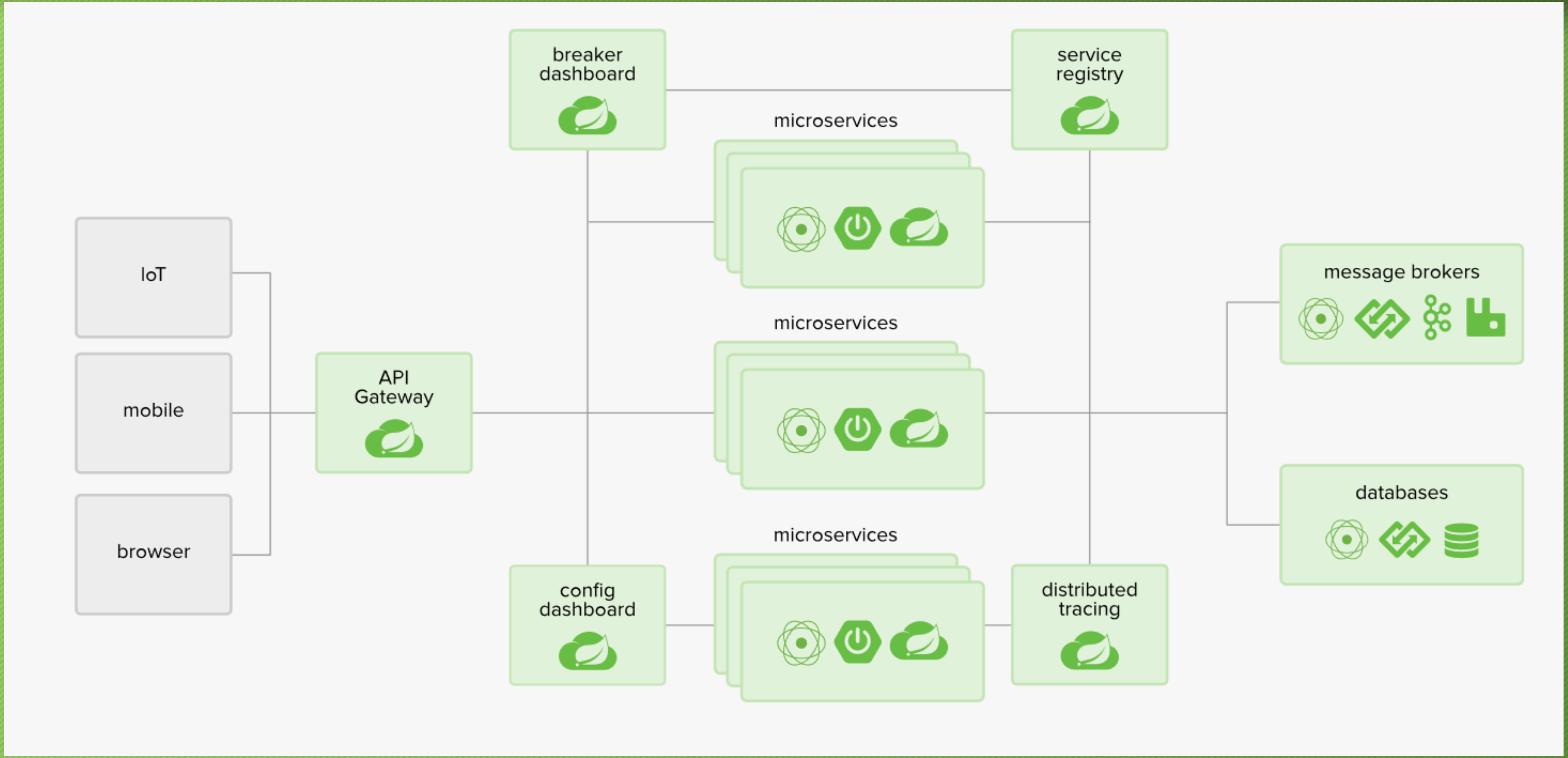
- Spring Cloud provides tools to perform the following tasks:
  - Service registration and discovery
  - Distributed/versioned configuration
  - Routing
  - Service-to-service calls
  - Load balancing
  - Circuit Breakers
  - Global locks
  - Leadership election and cluster state
  - Distributed messaging

Spring Cloud Projects

- Spring Cloud Config
- **Spring Cloud Netflix**
- Spring Cloud Security
- Spring Cloud Data Flow
- Spring Cloud Zookeeper
- Spring Cloud Gateway
- Spring Cloud OpenFeign
- Spring Cloud Function

https://spring.io/projects/spring-cloud

Spring Cloud Framework



Service registration and discovery

- **Service Discovery**
  A dynamic directory that enables client side load balancing and smart routing
- **Circuit Breaker**
  Microservice fault tolerance with a monitoring dashboard
- **Configuration Server**
  Dynamic, centralized configuration management for your decentralized applications
-

Spring Cloud Eureka Framework

- Netflix offers eureka cloud
- building event-driven Spring Boot microservices for real-time stream processing
- provides a mechanism for service discovery
- service discovery is important for Microservices Architecture
- built-in framework for spring boot

Creating a Microservice with Spring Boot

- When we create a microservice with Spring Boot, we can follow the following steps:
  - Creating a spring boot project to setup new service with eureka server.
  - Setting application.properties
  - Run and Test the project to confirm eureka server is started.
  - Create another project as a microservice to register in the eureka server.
    - Explore and practice SpringBootMicroservice example.

Eureka Server configuration properties

- spring.application.name - provides unique service identifier (Your project name) and it registers with discovery server)
- server.port - this port number used to listen the service.
- eureka.client.register-with-eureka = false - confirms that the server will not attempt to register itself.
- eureka.client.fetch-registry = false -
- client is fetching registry information form eureka server or not.

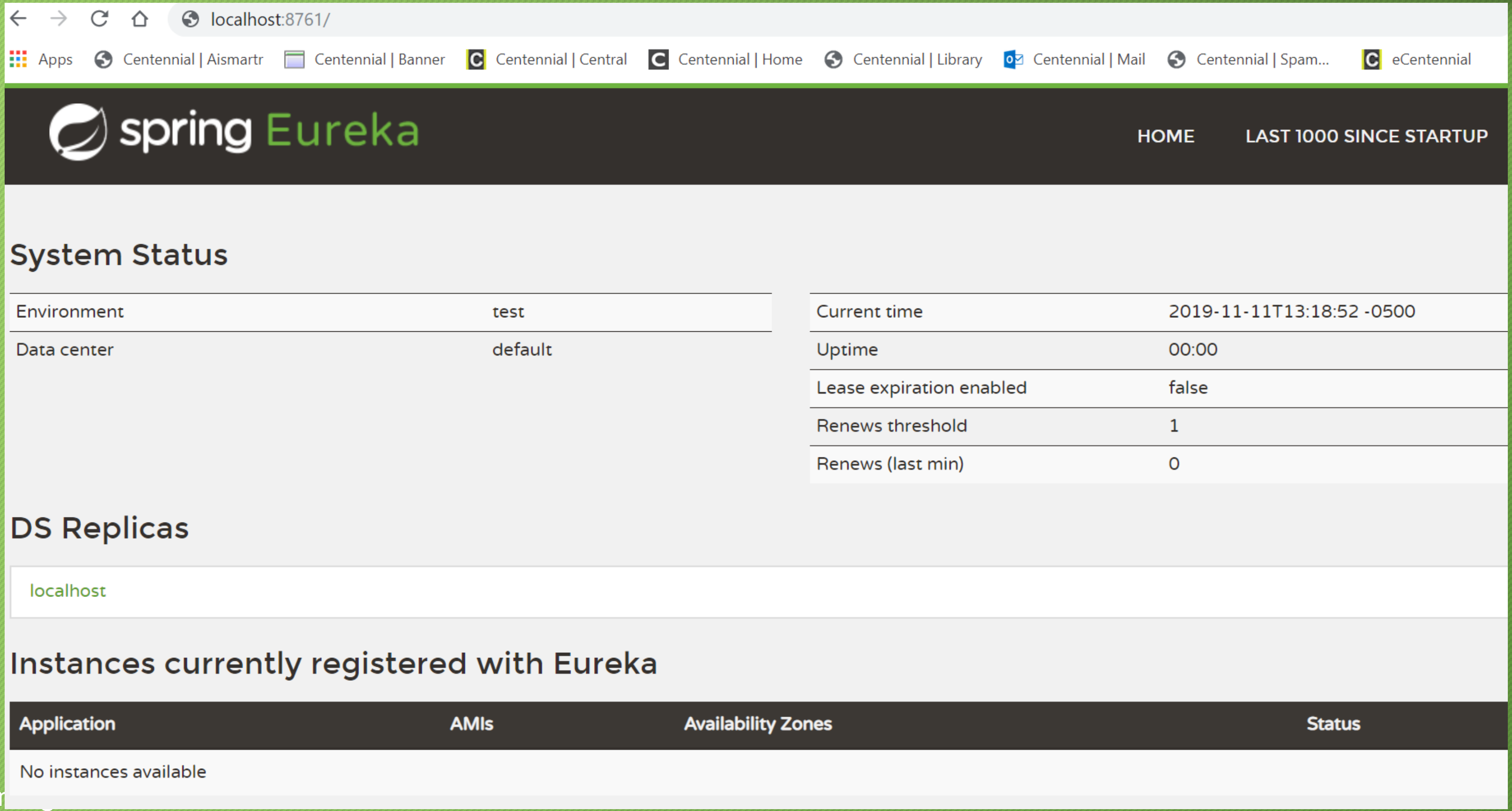
Application.properties configuration

- server.port=8761
- spring.application.name=SpringBootMicros

- eureka.client.register-with-eureka=false
- eureka.client.fetch-registry=false

- logging.level.com.netflix.eureka=OFF
- logging.level.com.netflix.discovery=OFF

Spring Eureka Server



System Status

Environment	test
Data center	default

Current time	2019-11-11T13:18:52 -0500
Uptime	00:00
Lease expiration enabled	false
Renews threshold	1
Renews (last min)	0

DS Replicas

localhost

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
No instances available			

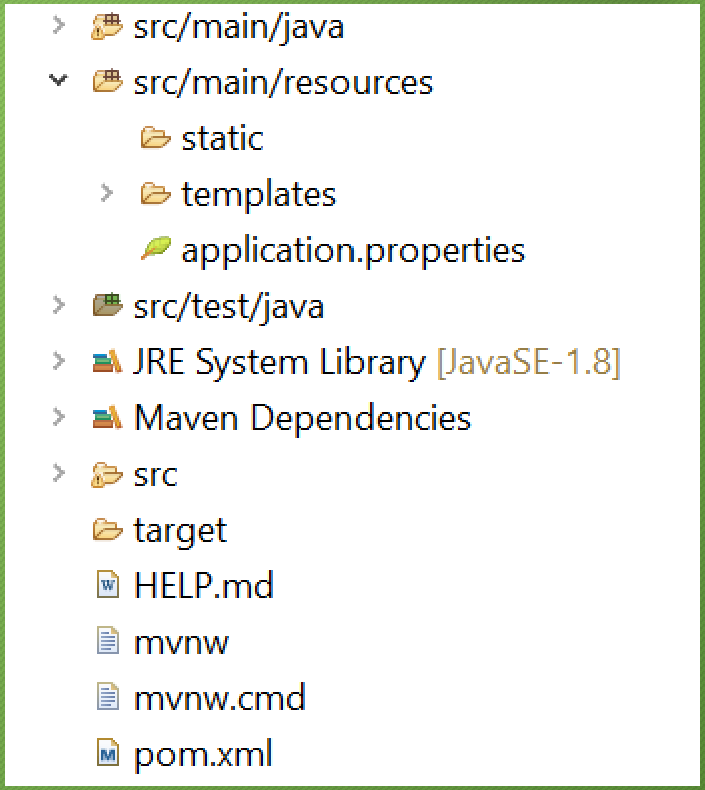
POM.xml - Dependency for Eureka server

```
<dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
</dependency>
```

Example 1: SpringBootMicroService

Notable Folders of Spring Boot App

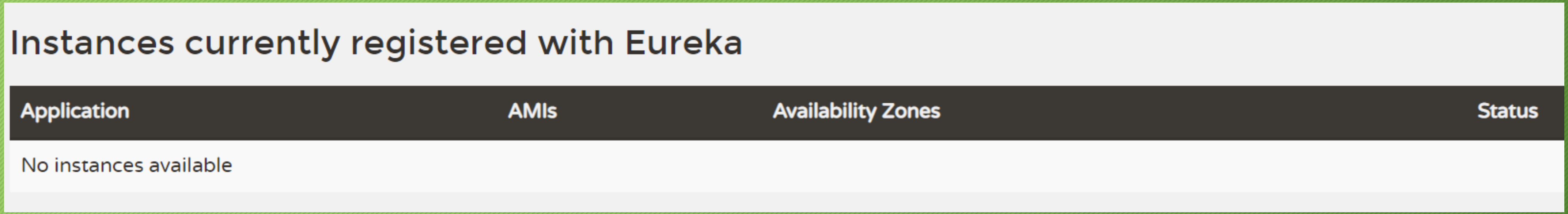
- src/main/java -
- src/main/resources/static - is used to have css, js and images files.
- ../resources/templates - is used to have server-side templates like thymeleaf.
- resources/application.properties - is used to configure application-wide properties like server's default port, server's context path, database credentials and other configurations.Run



```
> src/main/java
v src/main/resources
    static
  > templates
    application.properties
> src/test/java
> JRE System Library [JavaSE-1.8]
> Maven Dependencies
> src
  target
  HELP.md
  mvnw
  mvnw.cmd
  pom.xml
```

Example 1: SpringBootMicroService

- Run this SpringBootMicroService on Run As -> Spring Boot App
- Now type http://localhost:8761/ and get the eureka server page
- See there is no instances available – which means no microservices registered yet.



Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
No instances available			

Netflix Eureka - Service Discovery Server

- This allows microservices to register themselves at runtime as they appear in the system.
- Eureka Server implementation involves the followings:
  - You must add spring-cloud-starter-netflix-eureka-server to the POM.xml (or select Eureka Server dependency when you create a spring project).
  - Use this @EnableEurekaServer annotation in Spring Boot Application with @SpringBootApplication
  - You should configure the Applicaiton.properties file..

REST Template

- To send HTTP requests to a RESTful server and fetch data in a number of formats (JSON and XML).
- The RestTemplate bean will be intercepted and auto-configured by Spring Cloud (due to the @LoadBalanced annotation).
- A microservice (discovery) client can use a RestTemplate and Spring will automatically configure it.
- A RestTemplate instance is thread-safe and can be used to access any number of services in different parts of your application.
-

REST Template code

```
String baseUrl = "http://localhost:8761/employee";
RestTemplate restTemplate = new RestTemplate();
ResponseEntity<String> response=null;
    try{
        response=restTemplate.exchange(baseUrl,
        HttpMethod.GET, getHeaders(),String.class);
    }catch (Exception ex)
    {
        System.out.println(ex);
    }
System.out.println(response.getBody());
```

Example 2

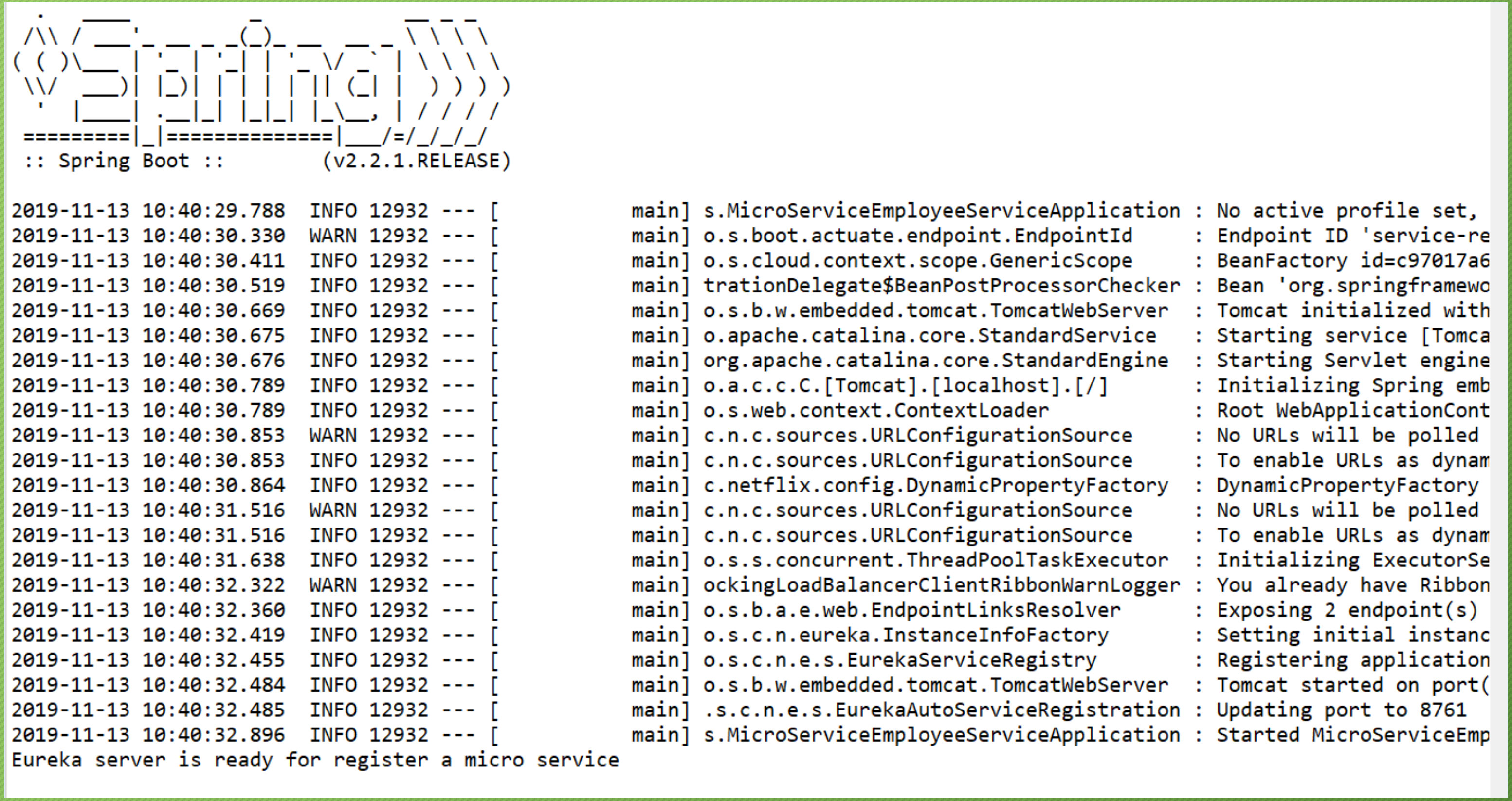
Practice example 2:

- **MicroServiceEurekaServer** – This app will setup the eureka server

- **MicroServiceEmployeeService** – This app provides a micro-service to register with the eureka server and run on the server.

MicroServiceEurekaServer

- Run this project on Run As - Spring Boot App

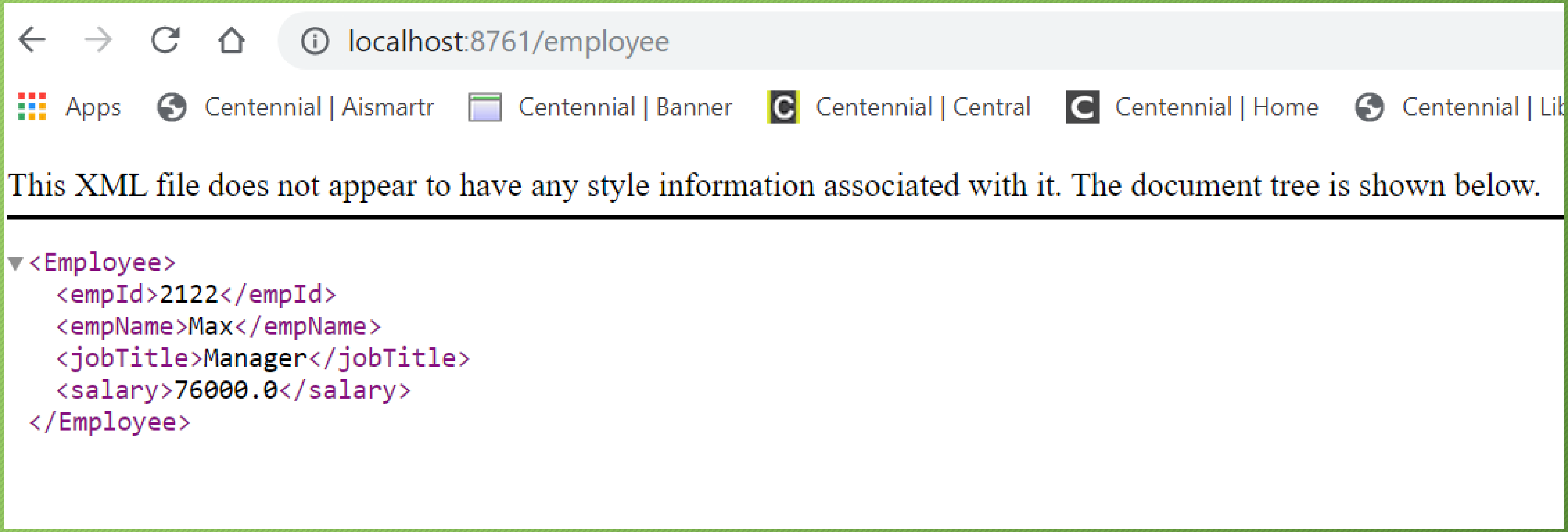


```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.2.1.RELEASE)

2019-11-13 10:40:29.788  INFO 12932 --- [           main] s.MicroServiceEmployeeServiceApplication : No active profile set,
2019-11-13 10:40:30.330  WARN 12932 --- [           main] o.s.boot.actuate.endpoint.EndpointId     : Endpoint ID 'service-re
2019-11-13 10:40:30.411  INFO 12932 --- [           main] o.s.cloud.context.scope.GenericScope     : BeanFactory id=c97017a6
2019-11-13 10:40:30.519  INFO 12932 --- [           main] trationDelegate$BeanPostProcessorChecker : Bean 'org.springframewo
2019-11-13 10:40:30.669  INFO 12932 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with
2019-11-13 10:40:30.675  INFO 12932 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomca
2019-11-13 10:40:30.676  INFO 12932 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet engine
2019-11-13 10:40:30.789  INFO 12932 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring emb
2019-11-13 10:40:30.789  INFO 12932 --- [           main] o.s.web.context.ContextLoader            : Root WebApplicationCont
2019-11-13 10:40:30.853  WARN 12932 --- [           main] c.n.c.sources.URLConfigurationSource     : No URLs will be polled
2019-11-13 10:40:30.853  INFO 12932 --- [           main] c.n.c.sources.URLConfigurationSource     : To enable URLs as dynam
2019-11-13 10:40:30.864  INFO 12932 --- [           main] c.netflix.config.DynamicPropertyFactory  : DynamicPropertyFactory
2019-11-13 10:40:31.516  WARN 12932 --- [           main] c.n.c.sources.URLConfigurationSource     : No URLs will be polled
2019-11-13 10:40:31.516  INFO 12932 --- [           main] c.n.c.sources.URLConfigurationSource     : To enable URLs as dynam
2019-11-13 10:40:31.638  INFO 12932 --- [           main] o.s.s.concurrent.ThreadPoolTaskExecutor  : Initializing ExecutorSe
2019-11-13 10:40:32.322  WARN 12932 --- [           main] ockingLoadBalancerClientRibbonWarnLogger : You already have Ribbon
2019-11-13 10:40:32.360  INFO 12932 --- [           main] o.s.b.a.e.web.EndpointLinksResolver      : Exposing 2 endpoint(s)
2019-11-13 10:40:32.419  INFO 12932 --- [           main] o.s.c.n.eureka.InstanceInfoFactory       : Setting initial instanc
2019-11-13 10:40:32.455  INFO 12932 --- [           main] o.s.c.n.e.s.EurekaServiceRegistry        : Registering application
2019-11-13 10:40:32.484  INFO 12932 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(
2019-11-13 10:40:32.485  INFO 12932 --- [           main] .s.c.n.e.s.EurekaAutoServiceRegistration : Updating port to 8761
2019-11-13 10:40:32.896  INFO 12932 --- [           main] s.MicroServiceEmployeeServiceApplication : Started MicroServiceEmp
Eureka server is ready for register a micro service
```

MicroServiceEurekaServer

- Use this url: http://localhost:8761/employee

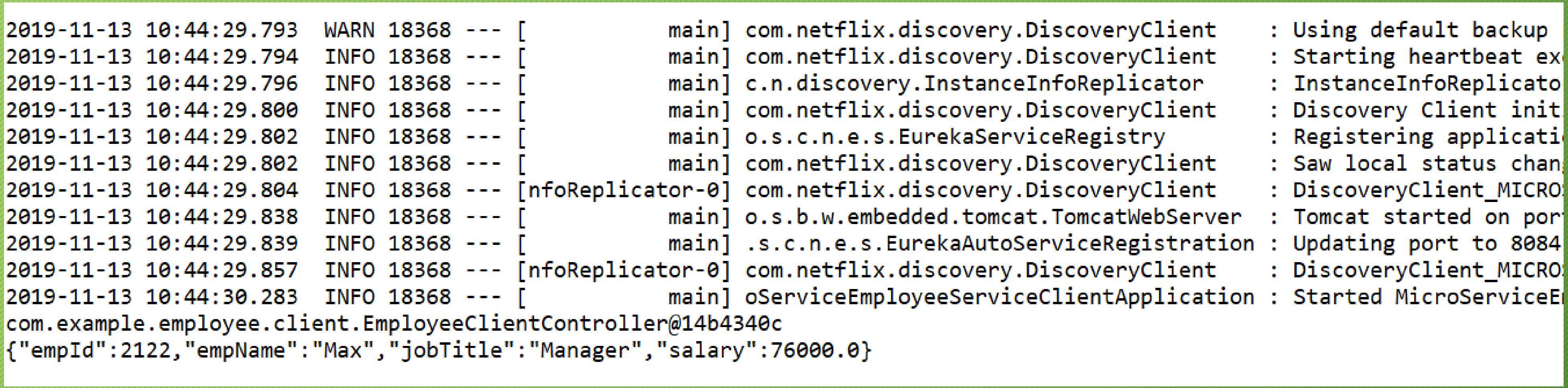


This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<Employee>
   <empId>2122</empId>
   <empName>Max</empName>
   <jobTitle>Manager</jobTitle>
   <salary>76000.0</salary>
 </Employee>
```

MicroServiceEmployeeService

- Run this project on Run As – Spring Boot App



```
2019-11-13 10:44:29.793  WARN 18368 --- [           main] com.netflix.discovery.DiscoveryClient    : Using default backup
2019-11-13 10:44:29.794  INFO 18368 --- [           main] com.netflix.discovery.DiscoveryClient    : Starting heartbeat ex
2019-11-13 10:44:29.796  INFO 18368 --- [           main] c.n.discovery.InstanceInfoReplicator     : InstanceInfoReplicato
2019-11-13 10:44:29.800  INFO 18368 --- [           main] com.netflix.discovery.DiscoveryClient    : Discovery Client init
2019-11-13 10:44:29.802  INFO 18368 --- [           main] o.s.c.n.e.s.EurekaServiceRegistry        : Registering applicati
2019-11-13 10:44:29.802  INFO 18368 --- [           main] com.netflix.discovery.DiscoveryClient    : Saw local status chan
2019-11-13 10:44:29.804  INFO 18368 --- [nfoReplicator-0] com.netflix.discovery.DiscoveryClient    : DiscoveryClient_MICRO
2019-11-13 10:44:29.838  INFO 18368 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on por
2019-11-13 10:44:29.839  INFO 18368 --- [           main] .s.c.n.e.s.EurekaAutoServiceRegistration : Updating port to 8084
2019-11-13 10:44:29.857  INFO 18368 --- [nfoReplicator-0] com.netflix.discovery.DiscoveryClient    : DiscoveryClient_MICRO
2019-11-13 10:44:30.283  INFO 18368 --- [           main] oServiceEmployeeServiceClientApplication : Started MicroServiceE
com.example.employee.client.EmployeeClientController@14b4340c
{"empId":2122,"empName":"Max","jobTitle":"Manager","salary":76000.0}
```

Annotations used in Microservices

- **@EnableEurekaServer-** to enable the eureka discovery service registry.
- **@EnableDiscoveryClient-** to activate the Netflix Eureka DiscoveryClient implementation.
- **@Configuration-** is bootstrapped and considered as a source of other bean definition.
- **@EnableAutoConfiguration-** will automatically configure the spring project based on the pom.xml file of the available dependencies.

JPA Implementation

- As we learned earlier, JPA is a specification and it supports the implementations like EclipseLink
- In this lesson, we will explore and practice how JPA specification is used by Hibernate implementation.

Spring Hibernate

- Hibernate is an implementation of JPA.
- Hibernate is a POJO-based approach, ease of development, and support of sophisticated
- relationship definitions.
- JPA has a lot of concepts that were influenced by popular ORM libraries such as Hibernate, TopLink, and JDO.
- The relationship between Hibernate and JPA is also very close.
- You can choose to use either Hibernate's own API or the JPA API with Hibernate as the persistence service provider.

Spring JPA Hibernate

spring.jpa.hibernate.ddl-auto

- property values are create, update, create-drop, validate and none

- **create** – first drops existing table and then creates new table.
- **update** – updates the schema and never deletes the existing tables or columns
- **create-drop** – similar to create, and will drop the database after all operations are completed (suitable for unit testing)
- **validate** – validates whether the tables and columns exist or not.
- **none** – turns off the DDL generation

Logging Configurations

- Logging level values are : ERROR, WARN, INFO, DEBUG, or TRACE

Example configuration code

- logging.level.root=warn
- logging.level.org.springframework.web=debug
- logging.level.org.hibernate=error
- logging.level.com.netflix.eureka=OFF
- logging.level.com.netflix.discovery=OFF

Exercise: Try this example

https://spring.io/guides/gs/service-registration-and-discovery/

Questions?

References

- Textbooks
- Microservices in Action Morgam Bruce and Paulo A.Pereira 2019
  - Chapters 1, and 3
- Pro Spring 5 by Iuliana Cosmina, Rob Harrop, Chris Schaefer, Clarence Ho, 2019
- Online Resources:
- https://spring.io/blog/2015/07/14/microservices-with-spring
- https://docs.spring.io/spring-boot/docs/2.1.x/reference/html/howto-database-initialization.html
- https://docs.spring.io/spring-boot/docs/current/reference/html/spring-boot-features.html#boot-features-logging