# 1.Microservices

Microservices is a development design model. Microservices is especially important for big and complicated project development process. It provides the project the seperation of small and independenlty running parts. Every micro service serves for a specific purpose, and they all have generally their internal data management and business logic. Microservices interact with APIs or communication protocols.

## 1.1.Microservices Archtiecture:

Microservices provides to break a monolithic application into smaller, loosely coupled(gevşek bağlı, proje içinde birbirine bağımlılıkların azalması durumu) services. Each service handles a specific business capability (e.g. customer management, address management).

Scalibility, maintainability, independent deployment, and flexibility in technology choices are the key benefits of using microservices.

## 1.2.Eureka Server:

Provides microservices can discover each other, acts as a centre for microservices. It is developed by Netflix as a service discovery tool. Generally in microservices architecture, when a service is started it has its own IP and port information. However this kind of information may be dynamic, the services may run on different servers, IP addresses may change in time, or the count of the services may increase or decrease.

Eureka is a system that helps microservices to recognize each others, meaning, one microservice can discover other micro service via Eureka Server. Eureka Server is the main server of this discovering process. All related microservices register to Eureka Server and inform Eureka about which service is working on which IP and port. Other services access to Eureka to discover all other service informations.

### 1.2.1.Flow of Eureka Server:

**1. Service Registration:** Every microservice registers to the Eureka Server when they are started. This registration include the name of the service, the IP address, and the Port.

**2. Service Discovery:** All other microservices applied to Eureka Server when they need to access any other microservices. Eureka Server replies them about which microservice is running on which address.

**3. Heartbeat (for service updates):** All microservices send Eureka Server heartbeat with specific period informals that configured before. This proves the service is active and running still. If Eureka cannot receive the heartbeat of a microservice, it defines the service as offline and it block other microservices to access this microservice.

### 1.2.2.Benefits of Eureka Server:

1. Dynamic Service Discovery

2. Load Balancing

3. High availablility

4. Management of independent microservices

5. Easy to configure and track

### 1.2.3.Build and Package the Eureka Server:

Use ***mvn clean package*** to package the Eureka Server code. And later on you can directly ran the generated jar on a system by running the command:

|  |
| --- |
| java -jar target/eureka-server-0.0.1-SNAPSHOT.jar |

Since there is a incompatibility between Spring Boot version(3.4.2) and Spring Cloud version it gives below error during starting the jar:

|  |
| --- |
| 2025-01-23T19:20:43.416+03:00 INFO 19864 --- [eureka-server] [ main] o.s.v.b.OptionalValidatorFactoryBean : Failed to set up a Bean Validation provider: jakarta.validation.NoProviderFoundException: Unable to create a Configuration, because no Jakarta Bean Validation provider could be found. Add a provider like Hibernate Validator (RI) to your classpath.  2025-01-23T19:20:44.439+03:00 WARN 19864 --- [eureka-server] [ main] ConfigServletWebServerApplicationContext : Exception encountered during context initialization - cancelling refresh attempt: org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'compositeCompatibilityVerifier' defined in class path resource [org/springframework/cloud/configuration/CompatibilityVerifierAutoConfiguration.class]: Failed to instantiate [org.springframework.cloud.configuration.CompositeCompatibilityVerifier]: Factory method 'compositeCompatibilityVerifier' threw exception with message: Spring Cloud/ Spring Boot version compatibility checks have failed: [[VerificationResult@69ed96e1 description = 'Spring Boot [3.4.2] is not compatible with this Spring Cloud release train', action = 'Change Spring Boot version to one of the following versions [3.2.x] .  You can find the latest Spring Boot versions here [https://spring.io/projects/spring-boot#learn].  If you want to learn more about the Spring Cloud Release train compatibility, you can visit this page [https://spring.io/projects/spring-cloud#overview] and check the [Release Trains] section.  If you want to disable this check, just set the property [spring.cloud.compatibility-verifier.enabled=false]']]  2025-01-23T19:20:44.450+03:00 INFO 19864 --- [eureka-server] [ main] o.apache.catalina.core.StandardService : Stopping service [Tomcat]  2025-01-23T19:20:44.493+03:00 INFO 19864 --- [eureka-server] [ main] .s.b.a.l.ConditionEvaluationReportLogger :  Error starting ApplicationContext. To display the condition evaluation report re-run your application with 'debug' enabled.  2025-01-23T19:20:44.542+03:00 ERROR 19864 --- [eureka-server] [ main] o.s.b.d.LoggingFailureAnalysisReporter :  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  APPLICATION FAILED TO START  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Description:  Your project setup is incompatible with our requirements due to following reasons:  - Spring Boot [3.4.2] is not compatible with this Spring Cloud release train  Action:  Consider applying the following actions:  - Change Spring Boot version to one of the following versions [3.2.x] .  You can find the latest Spring Boot versions here [https://spring.io/projects/spring-boot#learn].  If you want to learn more about the Spring Cloud Release train compatibility, you can visit this page [https://spring.io/projects/spring-cloud#overview] and check the [Release Trains] section.  If you want to disable this check, just set the property [spring.cloud.compatibility-verifier.enabled=false] |

To resolve this, it’s better to downgrade Spring Boot version to a compatible version (3.2.2) which aligns with the Spring Cloud release train. So pom is updated:

|  |  |
| --- | --- |
| Current | Updated |
| <**parent**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-starter-parent</**artifactId**>  <**version**>3.4.2</**version**>  <**relativePath**/> *<!-- lookup parent from repository -->* </**parent**> | <**parent**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-starter-parent</**artifactId**>  <**version**>3.2.2</**version**>  <**relativePath**/> *<!-- lookup parent from repository -->* </**parent**> |

#### 1.2.3.1.Possible error during packing:

If you have below error during “mvn clean package” step, you can skip testing:

|  |
| --- |
| [ERROR] Errors:  [ERROR] **EurekaServerAppApplicationTests**.contextLoads » IllegalState Failed to load ApplicationContext for [WebMergedContextConfiguration@1de30c31 testClass = com.example.eurkaserver.EurekaServerAppApplicationTests, locations = [], classes = [com.example.eurkaserver.EurekaServerAppApplication], contextInitializerClasses = [], activeProfiles = [], propertySourceDescriptors = [], propertySourceProperties = ["org.springframework.boot.test.context.SpringBootTestContextBootstrapper=true"], contextCustomizers = [org.springframework.boot.test.context.filter.ExcludeFilterContextCustomizer@6ca18a14, org.springframework.boot.test.json.DuplicateJsonObjectContextCustomizerFactory$DuplicateJsonObjectContextCustomizer@2462cb01, org.springframework.boot.test.mock.mockito.MockitoContextCustomizer@0, org.springframework.boot.test.web.client.TestRestTemplateContextCustomizer@102cec62, org.springframework.boot.test.web.reactor.netty.DisableReactorResourceFactoryGlobalResourcesContextCustomizerFactory$DisableReactorResourceFactoryGlobalResourcesContextCustomizerCustomizer@7c3fdb62, org.springframework.boot.test.autoconfigure.OnFailureConditionReportContextCustomizerFactory$OnFailureConditionReportContextCustomizer@6dd7b5a3, org.springframework.boot.test.autoconfigure.actuate.observability.ObservabilityContextCustomizerFactory$DisableObservabilityContextCustomizer@1f, org.springframework.boot.test.autoconfigure.properties.PropertyMappingContextCustomizer@0, org.springframework.boot.test.autoconfigure.web.servlet.WebDriverContextCustomizer@295cf707, org.springframework.test.context.support.DynamicPropertiesContextCustomizer@0, org.springframework.boot.test.context.SpringBootTestAnnotation@1d019d24], resourceBasePath = "src/main/webapp", contextLoader = org.springframework.boot.test.context.SpringBootContextLoader, parent = null]  [INFO]  [ERROR] Tests run: 1, Failures: 0, Errors: 1, Skipped: 0  [INFO]  [INFO] ------------------------------------------------------------------------  [INFO] BUILD FAILURE  [INFO] ------------------------------------------------------------------------  [INFO] Total time: 26.876 s  [INFO] Finished at: 2025-01-23T18:44:40+03:00  [INFO] ------------------------------------------------------------------------  [ERROR] Failed to execute goal org.apache.maven.plugins:maven-surefire-plugin:3.5.2:test (default-test) on project eurka-server:  [ERROR]  [ERROR] See D:\IdeaProjects\EurekaServerApp\target\surefire-reports for the individual test results.  [ERROR] See dump files (if any exist) [date].dump, [date]-jvmRun[N].dump and [date].dumpstream. |

This error is actually occured during **EurekaServerAppApplicationTests** (it is failing to load Spring ApplicationContext)

You can directly skip test process with mvn command:

|  |
| --- |
| mvn clean package -DskipTests |

Or as an alternative solution you can update your pom file to skip tests automatically during “mvn clean package” process:

|  |
| --- |
| <build>  <plugins>  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-surefire-plugin</artifactId>  <version>3.5.2</version>  <configuration>  <skipTests>true</skipTests>  </configuration>  </plugin>  </plugins>  </build> |

Or you can fix the issue.

## 1.3.Eureka Client:

The microservices is needed to define as Eureka Client in the system. Eureka Client is a development that all microservices can identify themselves to the Eureka Server, and so they can access to the other microservices. Generally in Java based applications Spring Cloud is used to create Eureka Clients. Spring Cloud is a framework that helps to integration of Eureka in an easy way.

### 1.3.1.Registering Clients:

Once your Eureka server is running, you can configure microservices (clients) to register with the server by adding:

|  |
| --- |
| **eureka.client.service-url.defaultZone**=http://localhost:8761/eureka/ |

1.3.2. Enabling Client on Service Side:

For enabling it you can use both @EnableEurekaClient or @EnableDiscoveryClient on Application.class:

|  |
| --- |
| @SpringBootApplication @EnableAspectJAutoProxy @EnableDiscoveryClient  ORRRRR  @ EnableEurekaClient **public class** CustomerWebServicerApplication {  **public static void** main(String[] args) {  SpringApplication.*run*(CustomerWebServicerApplication.**class**, args);  } } |

Here is the difference between EurekaClient and DiscoveryClient:

|  |  |
| --- | --- |
| **Annotation** | **Description** |
| **@EnableEurekaClient** | Specifically used when registering with Eureka service discovery. It tightly couples your app to Eureka |
| **@EnableDiscoveryClient** | A more generalized annotation that allows registration with multiple service discovery providers (e.g., Eureka, Consul, Zookeeper). It works with Eureka as well. More **flexible** and allows you to switch service discovery mechanisms in the future if needed. |