UNIT-IV

Micro services for Java Developers: What is a Micro service Architecture? Challenges, Technology Solutions.

What is a Micro service Architecture?

Micro services are a service-oriented architecture patterns wherein applications are built as a collection of various smallest independent service units.

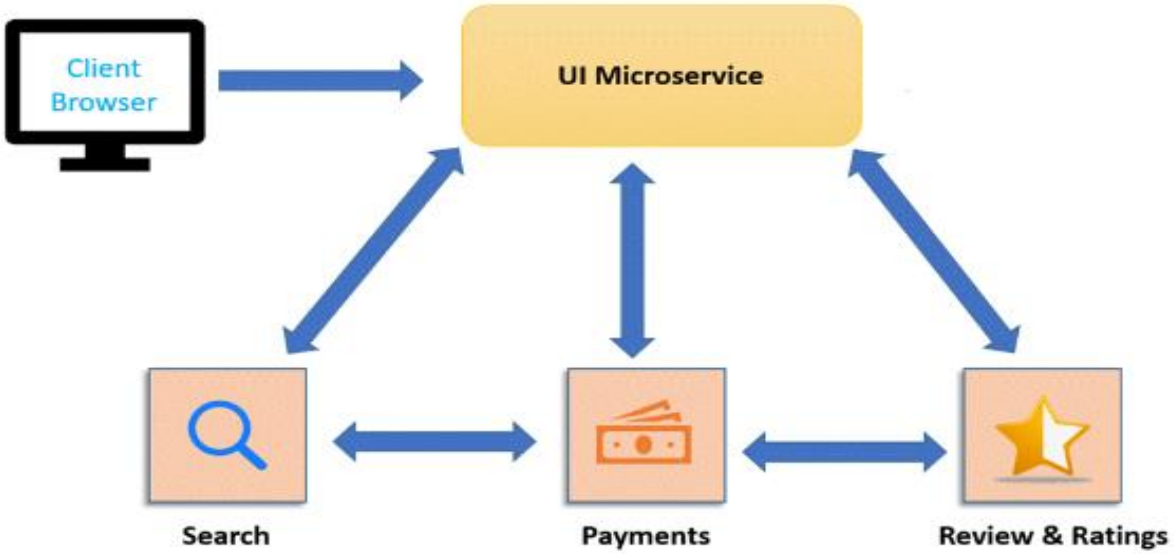
Micro services are a modern approach to software whereby application code is delivered in small, manageable pieces, independent of others.

It is a software engineering approach that focuses on decomposing an application into single-function modules with well-defined interfaces. These modules can be independently deployed and operated by small teams that own the entire lifecycle of the service.

The term "micro" refers to the sizing of a micro service which must be manageable by a single development team (5 to 10 developers). In this methodology, big applications are divided into smallest independent units. Micro services, also called micro service architecture, are a style of architecture that designs an application as a set of loosely coupled, highly maintainable and testable and independently deployable services. The micro service architecture provides the frequent, reliable, and fast delivery of complex applications. Micro services help to develop a large and complex software solution by creating and integrating multiple small development components together.

Micro service Architecture

- Micro service Architecture is an architectural development style that allows building applications as a collection of small autonomous services developed for a business domain.
- The micro service defines an approach to the architecture that divides an application into a pool of loosely coupled services that implements business requirements.



An example of e-commerce application developed with micro service architecture, each micro service is focused on single business capability. Search, Rating & Review and Payment each has their instance (server) and communicates with each other.

In Micro services Architecture they are spread into individual modules (micro service) which communicate with each other as shown in the Micro services example.

The communication between micro services is a stateless communication where each pair of request and response is independent. Hence, Micro services can communicate effortlessly. In the Micro service Architecture, the Data is federated. Each Micro service has its separate data store; it is next to Service-Oriented Architecture (SOA).

The most important feature of the micro service-based architecture is that it can perform continuous delivery of a large and complex application.

It is a variant of structural style architecture that helps arrange applications as a loosely coupled service collection. The Micro service Architecture contains fine-grained services and lightweight protocols

Challenges

- *Micro Services rely on each other, and they will have to communicate with each other.*
- *Compared to monolithic systems, there are more services to monitor which are developed using different programming languages.*
- *As it is a distributed system, it is an inherently complex model.*
- *Different services will have its separate mechanism, resulting in a large amount of memory for an unstructured data.*
- *Effective management and teamwork required to prevent cascading issues.*
- *Reproducing a problem will be a difficult task when it's gone in one version, and comes back in the latest version.*
- *Independent Deployment is complicated with Micro services.*
- *Micro service architecture brings plenty of operations overhead; it is difficult to manage application when new services are added to the system.*
- *Micro service is costly, as you need to maintain different server space for different business tasks.*
- *A wide array of skilled professionals is required to support heterogeneously distributed micro services.*

Micro services Tools

Wiremock: Testing Micro services: WireMock is a flexible library for stubbing and mocking web services. It can configure the response returned by the HTTP API when it receives a specific request. It is also why used for testing Micro services.

Docker: Docker is open source project that allows us to create, deploy, and run applications by using containers. By using these containers, developers can run an application as a single package. It allows you to ship libraries and other dependencies in one package**.**

Hystrix: Hystrix is a fault tolerance java library. This tool is designed to separate points of access to remote services, systems, and 3rd-party libraries in a distributed environment like Micro services. It improves overall system by isolating the failing services and preventing the cascading effect of failures.

Spring Boot: with Spring Boot, your micro services can start small and iterate fast. That's why it has become the de facto standard for Java™ micro services. Spring Boot provides a good platform for Java developers to develop a stand-alone and production-grade spring application that you can just run.

Spring Boot for Micro services: Getting Started, Hello World, and Calling another Service.

Spring Boot for Micro services

Spring Boot is an opinionated Java framework for building micro services based on the spring dependency injection framework.

Spring Boot allows developers to create micro services through reduced boilerplate, configuration, and developer friction.

Spring Boot does this by:

- *Favoring automatic, conventional configuration by default*
- *Curating sets of popular starter dependencies for easier consumption*
- *Simplifying application packaging*
- *Baking in application insight (e.g., metrics and environment info*

Spring Boot Hello World Example

We need the following tools and technologies to develop the application.

- Spring Boot 2.2.2.RELEASE
- JavaSE 1.8
- Maven 3.3.9
- STS IDE

Step 1:
Open Spring Initializr https://start.spring.io/.

Step 2:
Provide the Group name. We have provided com.Ramesh.

Step 3:
Provide the Artifact Id. We have provided the spring-boot-hello-world-example.

Step 4:
Add the dependency Spring Web.

Step 5:
Click on the **Generate** button. When we click on the Generate button, it wraps all the specifications into a jar file and downloads it to our local system.

Step 6:
Extract the RAR file.

Step 7:
Import the project folder by using the following steps:
File -> Import -> Existing Maven Project -> Next -> Browse -> Select the Project Folder -> Finish
When the project imports successfully, it shows the following project directory in the Package Explorer section of the IDE.

Step 8:
Create a package with the name *com.Ramesh.controller inside the folder src/main/java.*

Step 9:
Create a Controller class with the name *HelloWorldController.*

Step 10:
Create a method named **hello()** that returns a String.

HelloWorldController.java

```
package com.Ramesh.controller;
import org.springframework.web.bind.annotation.RequestMap
ping;
import org.springframework.web.bind.annotation.RestControll
er;
@RestController
public class HelloWorldController
{
@RequestMapping("/")
public String hello()
{
return "Hello Ramesh";
}
}
```

Step 11:
Run the **SpringBootHelloWorldExampleApplication.java** file.

SpringBootHelloWorldExampleApplication.java

```
package com.Ramesh;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApp
lication;
@SpringBootApplication
public class SpringBootHelloWorldExampleApplication
{
public static void main(String[] args)
{
SpringApplication.run(SpringBootHelloWorldExampleApplicati
on.class, args);
}
}
```

When the application runs successfully, it shows a message in the console, as shown in the following figure.

```
: Initializing ExecutorService 'applicationTaskExecutor'
: Tomcat started on port(s): 8080 (http) with context path ''
: Started SpringBootHelloWorldExampleApplication in 4.526 seconds (JVM running for 5
: Initializing Spring DispatcherServlet 'dispatcherServlet'
: Initializing Servlet 'dispatcherServlet'
: Completed initialization in 44 ms
```

Step 12:

Open the browser and invoke the URL **https://localhost:8080**.
It returns a String that we have specified in the Controller.