Microservices with Spring Boot

MOHAMMED LUQMAN SHAREEF

Agenda

1. What is Monolithic Architecture
2. What is Microservice Architecture
3. Monolithic vs Microservice Architecture
4. Spring Boot Rest Application - Get/Put/Post/Delete
5. Spring Data JPA/H2 Database
6. Consume Application using Postman
7. Global Exception
8. Swagger - API documentation
9. Service-Discovery
10. Zuul Api Gateway Server
11. Inter-micorservice Communication
12. Spring Cloud Configuration
13. Spring Security with Microservice - JWT Authentication
14. Application Scalability in Microservice
15. Project - Microservice application development
16. Introduction to Docker file implementation in microservice.

Day 1

- Monolithic Architecture
- Microservice Architecture
- Monolithic vs Microservice Architecture
- Spring Boot Rest
- Swagger - API documentation
- Consuming Rest App using Postman
- Hands-On

Day 2

- Spring Data JPA
  - H2 Database
- Global Exception Handling
- Inter-micorservice Communication
- Hands-On

Day 3

- Introduction to Spring Cloud
- Service Discovery
- Zuul Api Gateway Server
- Spring Cloud Configuration
- Hands-On

Day 4

- Spring Security with Microservice - JWT Authentication
- Application Scalability in Microservice
- Project - Microservice application development
- Introduction to Docker file implementation in microservice.
- Hands-On

What are Microservices?

The microservice ***architectural style*** is an approach to developing a single application as a ***suite of small services***, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and ***independently deployable*** by fully automated deployment machinery.

-- Martin Fowler

What is software Architecture?

Architecting a hut

Can be built by one person

Requires

- Minimal modeling
- Simple process
- Simple tools

Architecting a house

Built most efficiently and timely by a team

Requires

- Modeling
- Well-defined process
- Power tools

Architecting a high rise

What is the difference?

Scale

Process

Cost

Schedule

Skills and development teams

Materials and technologies

Stakeholders

Risks

Architectural patterns

The **architectural style**, also called as **architectural pattern**, is a set of principles which shapes an application. It defines an abstract framework for a family of system in terms of the pattern of structural organization.

What is Software Architecture?

The software architecture discipline is centered on the idea of reducing complexity through abstraction and separation of concerns.

Software architecture is the fundamental **organization** of a system embodied in its components, their **relationships** to each other, and to the environment, and the **principles** guiding its design and evolution.

Software Architecture Patterns

Monolithic

Layered

Pipes and Filters

Blackboard Architecture

Event Driven

Microkernel

Service Oriented

Microservices

What is a Monolith Application?

Have you ever worked in a project

- Which is released (taken to production) once every few months
- Which has a wide range of features and functionality
- Which has a team of more than 30 working for it
- Where debugging problems is a big challenge
- Where bringing in new technology and new process is almost impossible

These are typical characteristics of a Monolith applications.

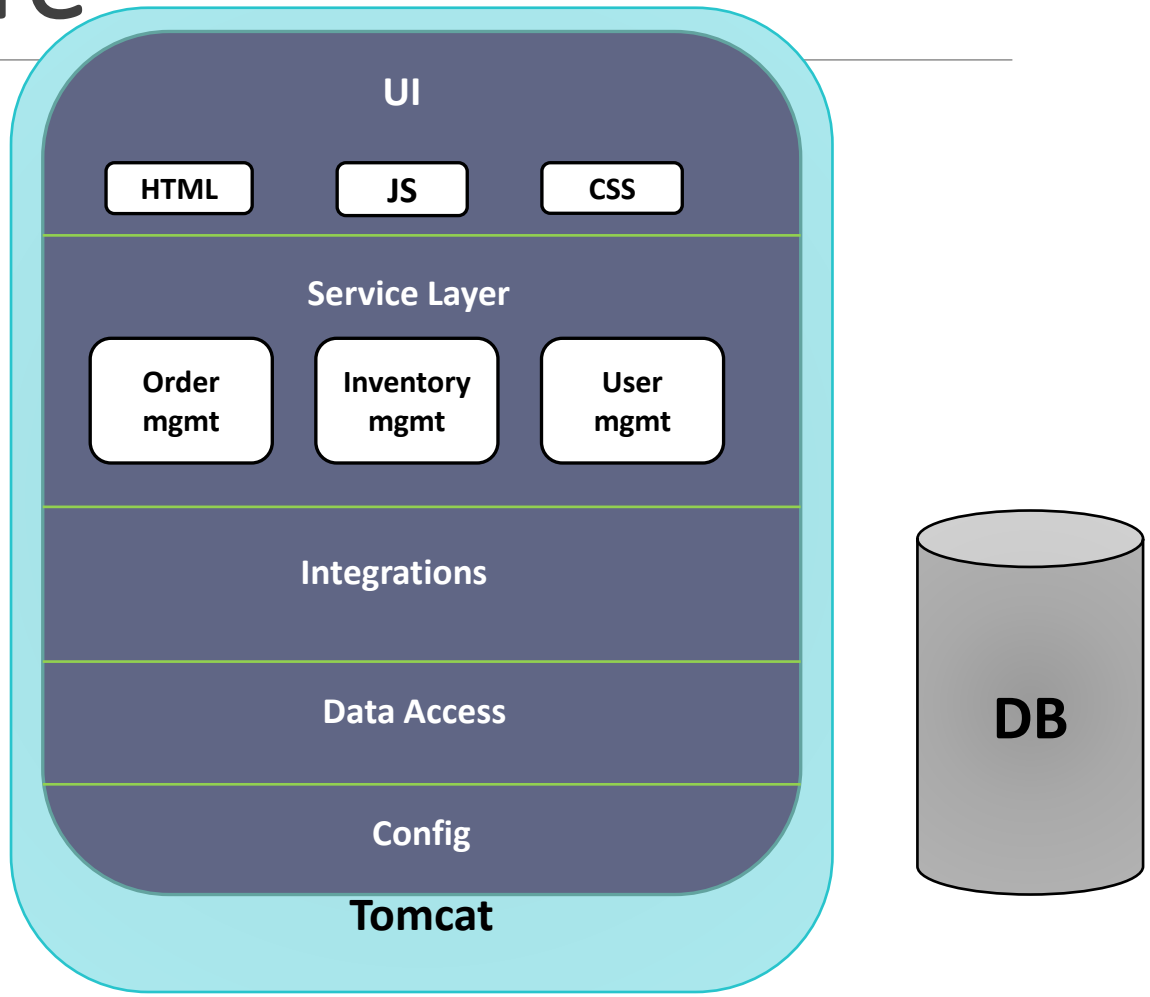
Monolith applications are typically huge - more 100,000 line of code.

Monolithic Architecture

A typical e-commerce application

The application consists of several components including the user interface, along with some backend services for checking credit, maintaining inventory and shipping orders.

The application is deployed as a single **monolithic** application.



Microservices

What is a Service ?

Self-contained module that perform a predetermined task

Software component that have published contracts/interfaces

Black-box to the consumers

Platform-Independent

Interoperable

Service Oriented Architecture (SOA)

Service Oriented Architecture (SOA) is an architectural style of building software applications that promotes loose coupling between components for their reuse.

A software system is divided into highly flexible and re-usable components called services.

SOA is just an architecture blue print

It is NOT a technical standard

It is NOT a technology

SOA Architectural Principles

Loose coupling

- Minimize dependencies.

Service contract

- Services adhere to a communications agreement.

Service abstraction

- Hide the service execution logic from the outside world.

Service reusability

- Logic is divided into services for reuse.

SOA Architectural Principles

Service composability

- Services can be assembled to form composite service.

Service autonomy

- Services have control over the logic they encapsulate.

Service discoverability

- Services can be found and assessed via available discovery mechanisms.

Service relevance

- Service presented at a granularity recognized by user a meaningful service.

What are Microservices?

Microservices is an **architectural style**.

It is an approach to architecting an application by breaking down a large software application into multiple smaller services.

Each service performs a specific task or business goal and uses a simple, well-defined interface, such as an API, to communicate with other sets of services.

> The microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery.
>
> -- Martin Fowler

Quiz

New few Architecture patterns.

Do monolithic applications have better performance than microservices?

Which ones have faster release cycles? Monolith / Microservices?

Are SOA and Microservices different architecture styles?

Microservice Vs. SOA

Microservices to be a more granular approach to SOA.

Microservice architecture is the natural evolution of SOA needed in order to accommodate cloud computing and meet increasing demands for faster software development cycles.

Microservices are a more platform-agnostic approach to application development and, therefore, should have a unique name.

Advantages

Easier to scale each individual microservice

Easier to maintain and evolve the system

Rapid build/test/release cycles

Clear ownership and accountability

Increased agility

Faster innovation

Faster time to market

New Technology & Process Adaption becomes easier. You can try new technologies with the newer microservices that we create.

Challenges

Potentially too much granularity

Extra effort designing for communication between services

Latency during heavy use

Complex testing

Challenges with Microservice Architectures

Quick Setup needed: You cannot spend a month setting up each microservice. You should be able to create microservices quickly.

Automation: Because there are a number of smaller components instead of a monolith, you need to automate everything - Builds, Deployment, Monitoring etc.

Visibility: You now have a number of smaller components to deploy and maintain. You should be able to monitor and identify problems automatically. You need great visibility around all the components.

Bounded Context: Deciding the boundaries of a microservice is not an easy task. Bounded Contexts from Domain Driven Design is a good starting point. Your understanding of the domain evolves over a period of time. You need to ensure that the microservice boundaries evolve.

Configuration Management: You need to maintain configurations for hundreds of components across environments. You would need a Configuration Management solution

Challenges with Microservice Architectures

Dynamic Scale Up and Scale Down: The advantages of microservices will only be realized if your applications can scaled up and down easily in the cloud.

Pack of Cards: If a microservice at the bottom of the call chain fails, it can have knock on effects on all other microservices. Microservices should be fault tolerant by Design.

Debugging: When there is a problem that needs investigation, you might need to look into multiple services across different components. Centralized Logging and Dashboards are essential to make it easy to debug problems.

Consistency: You cannot have a wide range of tools solving the same problem. While it is important to foster innovation, it is also important to have some decentralized governance around the languages, platforms, technology and tools used for implementing/deploying/monitoring microservices.

How about Security in Microservices?

Microservice architecture can alleviate some security issues that arise with monolithic applications.

Microservices simplify security monitoring because the various parts of an application are isolated.

A security breach could happen in one section without affecting other areas of the project.

Microservices provide resistance against distributed denial-of-service (DDoS) attacks when used with containers by minimizing an infrastructure takeover with too many server requests.

Microservice security

However, there are still challenges when securing microservices applications, including:

- More network areas are open to vulnerabilities.
- Less overall consistency between app updates allows for more security breaches.
- There's a greater area of attack, through multiple ports and APIs.
- There's a lack of control of third-party software.
- Security needs to be maintained for each service.

Quiz

Are microservices more secure compared to Monolithic apps?

What are the advantages of Microservice Architecture?

Name few challenges with Microservices Architecure.

Implementation

Spring boot

- embedded servers (easy deployment with containers)
- metrics (monitoring)
- health checks (monitoring)
- externalized configuration

Spring Cloud

- Dynamic Scale Up and Down.

RESTful Services

REST - REpresentational State Transfer

It is a simple stateless architecture that generally runs over HTTP.

The focus is on a resource in the system

- Ex: an Employee, an Account, a Product etc.

Each unique URL is a representation of some object

- Ex : http://www.vendorx.com/products
- http://www.vendorx.com/products/x123

HTTP methods (GET/POST/…) operates on the resource.

Object state is transferred and stored at client.

Representation of the resource state in XML, JSON etc.

HTTP Protocol

Lets use standard HTTP protocol.

It can be accessed via a URL (Uniform Resource Locator) like

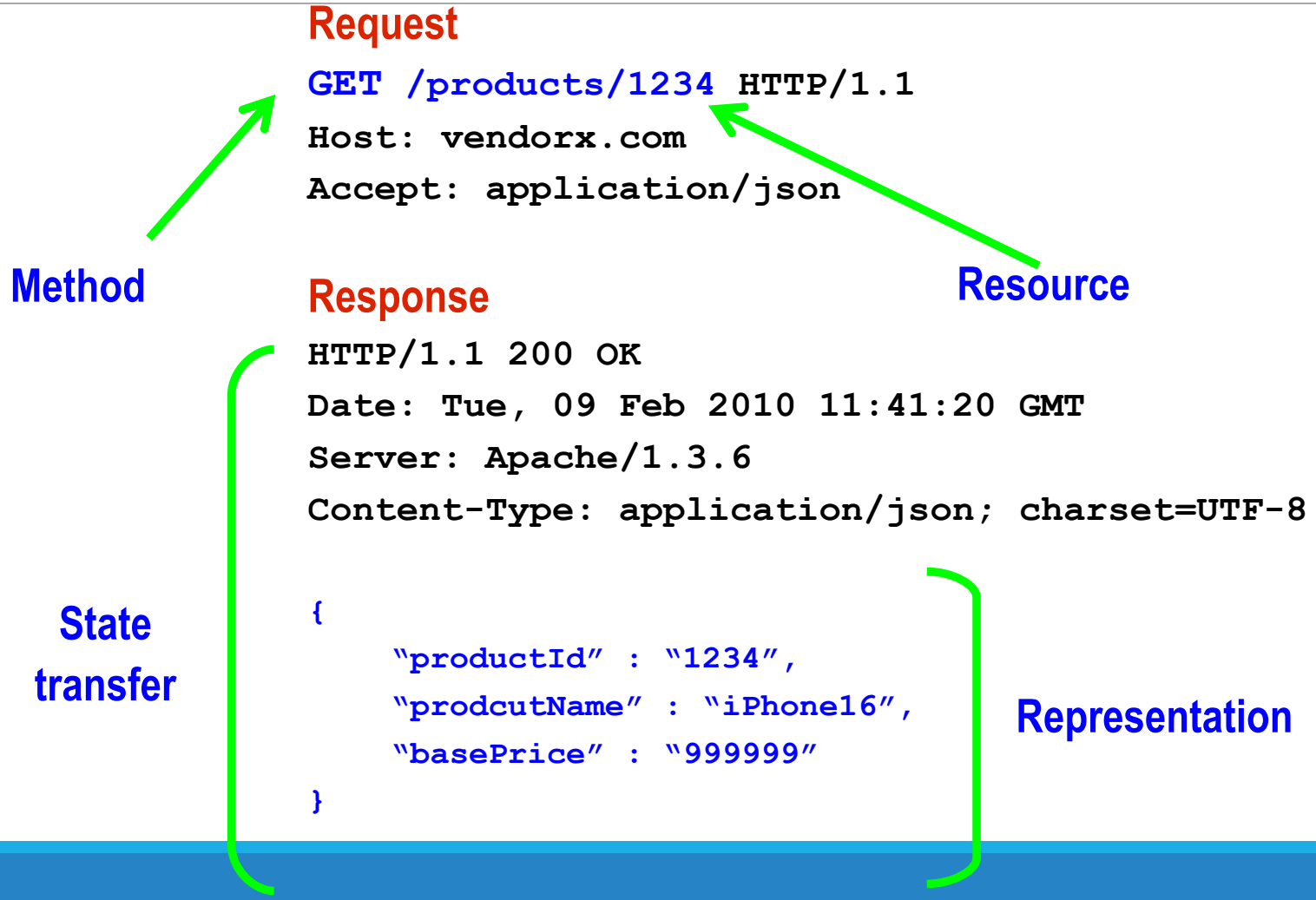
- http://www.vendorx.com/products/x123

It has standard methods to

- Retrieve (GET)
- Create (POST)
- Update (PUT)
- Delete (DELETE)

Payload can be directly added to http message. No need to wrap it in another protocol.

Product service as RESTful



HTTP Methods

Http Method	Description
GET	Requests data from a specified resource
POST	Submits data to be processed to a specified resource
HEAD	Same as GET but returns only HTTP headers and no document body
PUT	Uploads a representation of the specified URI
DELETE	Deletes the specified resource
OPTIONS	Returns the HTTP methods that the server supports
CONNECT	Converts the request connection to a transparent TCP/IP tunnel

REST Annotations

@RestController
- Combines @Controller and @ResponseBody, indicating that the class handles HTTP requests and the return values of its methods are bound to the web response body.

@PathVariable
- method argument is bound to a URI template variable.

@RequestParam
- HTTP request parameters (Query String)

@RequestBody
- Payload in POST / PUT methods

@ResponseBody
- Spring treats the result of the method as the response itself:

@RequestHeader

@ResponseHeader

@CrossOrigin

Documenting APIs

Enhances Developer Experience

Facilitates Team Collaboration

Simplifies API Integration

Introduction to Swagger and OpenAPI Specification

Swagger:

- Swagger is a set of open-source tools built around the OpenAPI Specification (OAS), which is a standard, language-agnostic interface description for RESTful APIs.

Swagger to OpenAPI:

- Originally developed by SmartBear Software, Swagger was donated to the OpenAPI Initiative in 2015 and has since been renamed the OpenAPI Specification.
- However, the tooling around it still retains the Swagger branding.

Why OpenAPI Specification:

- The main goal of OAS is to define a standard, programming language-agnostic interface description for REST APIs, which allows both humans and machines to understand the capabilities of a service without requiring access to the source code.

The Open API Spec

OpenAPI Specification (formerly Swagger Specification) is an API description format for REST APIs.

An OpenAPI file allows you to describe your entire API, including:

- Available endpoints (/users) and operations on each endpoint (GET /users, POST /users)
- Operation parameters Input and output for each operation
- Authentication methods
- Contact information, license, terms of use and other information.

API specifications can be written in YAML or JSON.

The format is easy to learn and readable to both humans and machines.

The complete OpenAPI Specification can be found on GitHub: OpenAPI 3.0 Specification

Generate OAS

Add dependencies

- Springfox-swagger-2 / io.swagger.core.v3. swagger-annotations
- Springfox-swagger-ui / springdoc-openapi-starter-webmvc-ui

Add Swagger Config class

- @Configuration
- @EnableSwagger2

API Spec Configuration Class

```
@OpenAPIDefinition(
    info = @Info(
        title = “Tets Ecommerce API",
        version = "1.0",
        description = "API documentation for My E-Commerce Application",
        contact = @Contact(
            name = “Test Ecommerce App",
            email = “email@example.com",
            url = "https://www.example.com"
        ),
        license = @License(
            name = "Apache 2.0",
            url = "http://www.apache.org/licenses/LICENSE-2.0.html"
        )
    ),
    security = @SecurityRequirement(name = "bearerAuth")
)
@SecurityScheme(
    name = "bearerAuth",
    type = SecuritySchemeType.HTTP,
    scheme = "bearer",
    bearerFormat = "JWT"
)
@Configuration
public class OpenApiConfig {
  // No additional configuration needed
}
```