http://localhost:8080/SpringWebDemos/login

Dispatcher Servlet

Controller

RequestMapping

Model

MonoLithic App

View

.jsp

.html

Spring

1. Dependency Management –
2. Spring Configuration for startup—xml
3. Web Container like Tomcat management.
4. Not Production Ready
   1. War file
   2. Upload to tomcat webapps

SpringBoot – BootStrapper for Spring app

Book

SOA -ServiceOrientedArchitecture

Book-producer

Books

Advice

RestController

Application

BookDto

BookServiceI

BookService

ORM – Object Relation Mapping

Hibernate

JPA -Java Persistence API JPA Repository

RDBMS/NOSQL

Domain Driven Design

http://localhost:8081/bookstore/api/books

BookRepo

angular

ios

android

ACID -- Transaction

DB

UserService

NS

US 8080



NS 8091

NOSQL – MOngoDB

Around

BeforeAspect

No Structure

JSON Data (“1”,”sam”} {“2”,”john”,[john@email.com](mailto:john@email.com)}

around

getName

Aspect

Service

getXX()

setXX()

AfterAspect

Client

Get()

Get1()

Abc()

Xyz()

Spring Boot Security

User1,user123,USER – In memory – launched

When launched and hit any endpoint

1. Open the login form
2. Enter user1, user123
3. Authenticate in AuthenticationManager
4. Credentials are set to Principal Object (UserNamePasswordAuthenticationToken)
5. loadUserByUserName(String username) in UserDetailsService
6. Retrieves the User Object – if username is right else null
7. Password encoder – read password from the User Object (existing credential)
8. Checks with the password in the Principal Object

UserDetailsService

User Object

Authenticates

Principal Object

Login Form

Reads the rule that is set & validates

Reads the role from User Object

MyUser

UserRepo

MyUserDetailsService

Login Form

Principal Object

User Object

Authenticates

DB Authentication

1. Create credentials

JWT Authentication

1. Authenticate the user
2. Generate the token – JwtUtil .. abcd
3. Validate the token

Filter

token

RestController

/user/hello

/authenticate

Request Object

Filter

1. Read the Authorization Request Header using HttpServletRequest
   1. Bearer abcd
2. Tokenize read abcd
3. Extract username
4. Validate the user
5. Gets the User (UserDetails Object)
6. Validate the token – Util
7. Set the UserNamePasswordAuthenticationToken
8. Set SecurityContext

BatchProcessing – ETL

Extract – Transform – Load

Load

Writer

Process

Processor

Extract

Reader

DB

FileSystem

Topic

DB

FileSystem

Topic

/load

DB

.csv

Sam,1

Process()

SAM,”Sales”

API Gateway

App.prop

Message=xyz

1.

Book-Service

App.prop

Message=abc

Order-service

Inventory-service

Eureka Service

Challenges

1. Externalized Configuration – Cloud Config Server
2. Bounded Context
   1. Centralized Logging -- ELK Stack , Splunk
   2. Distributed Tracing -- Zipkin and Sleuth
   3. Client Load Balancing – Ribbon (maintainence) -- LoadBalancerClient
   4. Call a remote service – RestTemplate , Feign
3. Service Naming and Discovery – Eureka- 8761
4. Fault Tolerance –Hystrix / Resilience4J
5. API Gateway - Zuul / Spring Cloud gateway
6. Transaction management – SAGA Pattern
7. Reactive Web –
   1. Call a remote -- WebClient

book-service

[http://localhost:8081/bookstore/api/book/isbn/{isbn}](http://localhost:8081/bookstore/api/book/isbn/%7bisbn%7d)

order-service

<http://localhost:8084/orders/api/isbn/1234/qty/10>

stub

Order-service

OS

bs 3

bs 2

bs 1

book-order

order-service

Book-service

getbook

Book-service

getbook

Book-service

getbook

client

client

C bs

B os

A ps

Sleuth – TraceId – 1234 (AB C)

Spanid - A (23), B(45), C(89)

Tracing and logging

Zipkin – Centralized service for distributed tracing

E- Elastic Search -- Read the logs from logstash.conf and populating on kibana

L – Logstash -- reads all the logs from different services – consolidates into a single file—Eg: logstash.conf

CheckOut

D

K—Kibana -- GUI --

API Gateway

9001

SpringCloudGateway

Fliter

http://localhost:9001/helloservice2/hello2

http://localhost:9001/helloservice1/hello1

Client

HelloService1 8088

HelloService2 8089

Processor

KStream

CloudStream

consumer

kafkamovie

out

MovieProcessorGenre

WebCrawlerService

movies

KafkaMovie

MovieProcessor1

MovieService

Helloservice1

prefilter

Post filter

S1

BookService

Shared Database --

db

S2

CartService

Db per Sevice

DB1 t1

S1

Movie Data - abcd

CustData

History

abc

CustDb

S3

Movie Data

abcd

Query

DML

command

S2 -- Query

S1 --

10 records

Db2

S2

Kafka 3000

Order\_event

Payment\_event

OrderService

Order –

101,”nuts”,2000

SagaDb – order – 1,nuts,3000

PaymentService

SagaDb 101 5000

1. Kafka - Spring Web -- Blocking
2. KStream and Cloud stream – ETL – Streaming

3 Reactive web with Kafka – Non Blocking

SAGA Pattern

1. Choreography
2. Orchestration

Transaction Management across services

A

Payment-service

Order-service

Book-Service

CircuitBreaker Pattern -- fallback

Hystrix web/ Reslience4J

Orchestration – Kubernetes

Minikube kuberctl

C3

C2

C1