**Microservices Architecture Project**

***Legend****: COMPLETE, IN-PROGRESS*

**Progress:**

Spring 1 - Implement Basic Services using H2 database with basic REST endpoints

Added user, product, cart service *(**HATEOAS not for now)*

Sprint 2 - Configure service discovery, config server

Configured

Spring 3 - Implement basic UI using angular 8

*In progress...*

Sprint 4 - Add order service and required communication with other services

Added basic APIs

Added validation before initiating order

*(\*Configured Jenkins pipeline script for CI-CD for user-service and tried deployment using docker container, docker-hub.*

*Tried it on an AWS free instance, but free instances are too small and frequently hanged. Try GCP)*

*In progress...*

*Configured Kafka on local machine and tried pub-sub with it*

*Configure Kafka docker image*

*Kafka event notifications for payment, order confirmation, shipping & notification*

Sprint 5 - Add auth service with JWT tokens

Added *(Integrate other services to use it – not for now)*

Sprint 6 - Add API gateway service

Added service with spring cloud gateway and configured routes

Sprint 7 – Enable CI-CD

Reserved a Google Cloud VM instance

Installed and configured Jenkins to link with GitHub repos

Created pipeline to build, test and create docker image and then push them to docker-hub

Tried mono-repo as well as repo-per-microservice – will continue with mono-repo for now

Tried Declarative and Scripted pipeline –

Using Scripted pipeline for now

*Need to refine Jenkins files further – will do it later*

*(faced issues with deployment steps in Declarative - will try again later)*

Using docker-compose within Jenkins to start/stop all docker containers for now *(will try orchestration later)*

Sprint 8 – Add monitoring

Configured Micrometer, Prometheus, Grafana

Application metrics can be monitored on (Grafana)<http://35.244.121.244:3000/>, (Prometheus) <http://35.244.121.244:9090/>

Currently only actuator exposed metrics are recorded *(Will add custom metrics later)*

Sprint 9 – Enable logging and configure ELK - ElasticSearch, LogStash, and Kibana

*In progress...*

Sprint 10 – Configure MongoDB for product-service

Sprint 11 – Configure Distributed Request Tracing

Spring 12 -

**Initial Features Scope:**

User profile (details, addresses) management

Product search, inventory management

Cart management

Order management

*Maybe - Payment management (only shopping credits mode for now)*

**GOAL:**

Host it on: ~~AWS~~ or Google Cloud

Architecture: Microservice based

Try: *domain-driven-design, TDD, event storming, event-driven architecture, event sourcing, CQRS*

Service communication: REST APIs, send events over Kafka ~~(try gRPC LATER)~~

Database: In-memory H2 for now, integrate Postgres & NoSQL(MongoDB/~~DynamoDB~~) later

UI: Angular 8, TypeScript, Bootstrap

Event Streaming: Kafka

Caching: Redis

Also (*try* *later*): Reactive stream - [WebFlux](https://docs.spring.io/spring/docs/current/spring-framework-reference/web-reactive.html)

Microservice tools -

Service Discovery: [**Spring Cloud Netflix Eureka**](https://spring.io/projects/spring-cloud-netflix)

Configuration Server: [**Spring Cloud Config**](https://spring.io/projects/spring-cloud-config)

Security: [**Spring Security**](https://spring.io/projects/spring-security) – **JWT** (Explore [Spring Cloud Security](https://spring.io/projects/spring-cloud-security) later)

Monitoring: [**Micrometer**](https://micrometer.io/) **+** [**Prometheus**](https://prometheus.io/) **+** [**Graphana**](https://grafana.com/)

Request [Tracing](https://medium.com/swlh/distributed-tracing-in-micoservices-using-spring-zipkin-sleuth-and-elk-stack-5665c5fbecf): [**Spring Cloud Sleuth**](https://spring.io/projects/spring-cloud-sleuth) **+** [**Zipkin**](https://zipkin.io/)

Fault Tolerance: [**Spring Cloud Circuit Breaker**](https://spring.io/projects/spring-cloud-circuitbreaker) **+** [**Resilience4J**](https://github.com/resilience4j/resilience4j)

Load Balancing: [**Spring Cloud LoadBalancer**](https://spring.io/guides/gs/spring-cloud-loadbalancer/)

For microservices to discover each other with service name and communicate internally(not through gateway)

API Gateway: [**Spring Cloud Gateway**](https://spring.io/projects/spring-cloud-gateway)

An api gateway – single point of entry (port 8000) for all client requests which will then redirect it to respective to service

Other –

JPA, Swagger2 (for API docs), DevTools, Actuator,

Testing:

Unit: Junit, Mockito, Hamcrest - @DataJpaTest, @WebMvcTest, @SpringBootTest

Deployment:

CI/CD: Do with Jenkins configured on GCP, ~~Bamboo~~ and then maybe Jenkins X for kubernetes

Path: => Bare OS Spring Boot services deployment

=> Bare OS docker containers deployment

=> Deploy containers on Kubernetes

=> *If possible, try on OpenStack/CloudFoundry*

SCM: GitHub

Other Editor Tools: Git Bash, Eclipse, Visual Studio Code, Postman, SourceTree

