**Project with Microservices Architecture**

***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...*

Created angular project

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

*In progress...*

Added basic APIs

Added validation before initiating order

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

Configured Kafka with bitnami docker containers - zookeeper and 1 broker *(2 brokers causing some never ending error logs -solve later)*

Using Conduktor to administer Kafka

*Kafka event notifications for*

*Order Service: ORDER\_INITITATED -> payment, product inventory, email, shipping*

*Payment Service: ORDER\_PAYMENT\_FAILED -> order, product inventory, email, shipping*

*Product Service: ORDER\_ITEM\_OUTOFSTOCK -> order, payment, email, shipping*

*Order Service: ORDER\_CONFIRMED -> email, shipping*

*TODO – Revisit and fine tune Kafka settings - commit strategy, etc..*

Sprint 5 - Add **Authentication Server** and **Resource Server** forAPI security

*In progress...*

Sprint 6 - Add API gateway service – *Spring Cloud Gateway*

Added service with spring cloud gateway and configured routes

*(To Do: Need to see how swagger api docs can be exposed through gateway routes)*

Sprint 7 – Enable CI-CD

(\*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(1GB) and hangs frequently)

Configured it on GCP Instance – with 3.75GB RAM 10G HDD for now – will increase capacity if required

*Not keeping Instance always ON – Saving on free credits – will keep it on once UI is ready*

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

Tried Declarative and Scripted pipeline –

Using Scripted pipeline for now

*Need to refine Jenkins files further – Find services with changes and build-deploy only those – will do it later*

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

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

*Register a domain and expose application with domain name – will do when UI is ready*

Sprint 8 – Add monitoring

Configured Micrometer, Prometheus, Grafana

Some actuator exposed metrics can be monitored with imported JVM dashboard 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...*

*Configuring it using docker containers from elastic stack*

Sprint 11 – Configure Distributed Request Tracing

Spring 12 – Configure Circuit Breaker for Fault Tolerance

Use Cases –

Sprint 13 – Add and Integrate Relational Database - Postgres mostly

Sprint 14 – Add Unit and Integration tests for Order, Payment service (remaining later)–

*Should have added them alongside development (TDD) but first concentrating on delivering basic working skeleton asap*

Revisit each use case with tests and improve code alongside

Sprint 15 – Add few performance tests

Sprint 16 – Configure Kubernetes and deploy all containers

Sprint 17 – Configure scaling in Kubernetes and test with existing performance tests

Sprint 18 – Improve product search – MongoDB, ElasticSearch

Sprint 19 – Try improving it further with WebFlux

**Initial Features Scope:**

User profile (details, addresses) management

Product search, inventory management

Cart management

Order management

Payment management (only shopping credits mode for now)

**GOAL:**

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

Architecture: Microservices

Try: *domain-driven-design, TDD, event storming, event-driven architecture, (event sourcing – Kafka is covering this), 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/) **+** [**Grafana**](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,

Logging –

ELK - ElasticSearch, LogStash, and Kibana

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 – manage using docker-compose

=> Deploy containers on Kubernetes

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

SCM: GitHub

Other Editor Tools: Git Bash, Eclipse/IntelliJ Idea, Visual Studio Code, Postman, SourceTree, Conduktor



User Service

Order Service

Email Service

Payment Service

TO DO

Eureka Service

UI Client

Config Repo On GitHub

Config Service

Resource Server

Product *Search* Service

Auth Server

Cart Service

Gateway Service

Kafka – ORDER\_TOPIC

Product Inventory Service