



**FACULTY
OF ELECTRICAL
ENGINEERING
CTU IN PRAGUE**

Project documentation TIER SERVICE APP

As part of the project, the backend part of the tier service was built in SpringBoot Java and deployed to an AWS environment.

Final version: 2.7.2024

Location: Prague

Team

Artem Aivazian

Ilia Zozuliak

Tatiana Denisova

1. AS-IS	1
2. TO-BE	1
3. 5F analysis	2
1. Threat of New Entrants	2
2. Threat of Substitute Products or Services	2
3. Bargaining Power of Customers	2
4. Bargaining Power of Suppliers	2
5. Competitive Rivalry	2
4. PESTEL analysis	2
1. Political	2
2. Economic	3
3. Social	3
4. Technological	3
5. Legal	3
6. Environmental	3
5. Requirements	3
1. Functional	3
2. Non-Functional	4
6. Architecture	4
7. Analysis and selection of solution alternatives	4
1. Own Development	4
2. Ordering from Another Company	5
3. Open-source Solution	5
8. WBS – breakdown of the project into sub-processes	5
9. Resources	5
1. Human Resources: Team Roles	6
2. Tools needed for development	6
10. Responsibility Matrix	6
11. GANTT Schedule	6
12. FMEA Risk Analysis	7
1. Website Downtime	7
2. Payment Gateway Issues	7
3. Data Security Breach	7
4. Inaccurate Inventory Management	7
5. Poor User Experience (UX)	8
13. Metrics	8
1. Performance Metrics	8
2. User Engagement Metrics	8
3. Customer Satisfaction Metrics	8
4. Other Metrics	8
14. UML diagrams	9
1. Class Diagram	9
2. Use Case Diagram	10

3. Component Diagram	11
4. Deployment Diagram	12
5. Sequence Diagram	13
15. Deployment process	14
1. Development environment	14
2. Production environment	14

1. AS-IS

The current state of operations at the tire service shop involves a predominantly manual and direct interaction-based approach to customer service and appointment scheduling. The shop, managed by the owner and supported by a dedicated team of three employees, specializes in tire mounting services and customer support. Customers looking to utilize the shop's services or purchase items must initiate contact through a phone call to the owner. Following this initial contact, the process requires customers to physically visit the shop's reception area, where transactions are completed using either cash or card payments at the register.

This traditional service model, while straightforward, presents several limitations in terms of accessibility, convenience, and efficiency, both for the shop's management and its clientele. The lack of digital infrastructure for booking and sales has rendered the shop less competitive in an increasingly digital marketplace. The shop's reliance on in-person interactions and cash/card payments at a physical location may deter potential customers seeking more flexible and contactless options. Consequently, this approach to business operations has impacted the shop's popularity and ability to thrive amidst competitive pressures in the tire service industry. Recognizing these challenges, the owner is now considering the development of a website aimed at digitalizing the booking and purchasing processes, with the goal of enhancing customer convenience, expanding service accessibility, and ultimately improving the shop's market position.

2. TO-BE

The envisioned future state of the tire service shop introduces a hybrid model, seamlessly integrating digital and physical customer service channels. The core of this transformation is the adoption of a website that facilitates online booking for services and ordering of items. This digital platform aims to enrich the customer experience by offering a comprehensive catalog of items available in stock, enabling customers to place orders directly through the website. Upon ordering, customers will receive SMS updates regarding the status of their order, from confirmation through to delivery.

Despite the shift towards digitalization, the tire service shop will retain its traditional, in-person service at the reception for those customers who prefer or require physical interaction. When services or items are booked or ordered in person at the shop, customers will be provided with an order ID. This ID can then be used to track the order status or service booking details directly on the website, offering a blend of digital convenience with the assurance of physical customer service.

To support the new online ordering system, the shop will engage a delivery service, ensuring that the digitalization of sales extends beyond the point of purchase to include the delivery of items directly to the customer. This addition aims to streamline the purchasing process, making it more efficient and customer-friendly.

This future state represents a strategic move towards digital transformation, aiming to enhance operational efficiency, improve customer satisfaction, and strengthen the shop's competitive position in the market. By offering both online and in-person services, the tire service shop aims to cater to a broader customer base, accommodating varying preferences for how services and purchases are made and managed.

3. 5F analysis

1. Threat of New Entrants

- Open-source technologies
- Requirements for operating in the automotive service industry

2. Threat of Substitute Products or Services

- The presence of other vehicle maintenance services
- The extent to which customers value specialized tire services over general automotive services.

3. Bargaining Power of Customers

- Customers' sensitivity to pricing changes, especially if comparable services are offered at lower prices by competitors.

4. Bargaining Power of Suppliers

- Dependence on a limited number of tire manufacturers or suppliers could increase their bargaining power.
- The ease with which the shop can switch to different brands or suppliers affects their bargaining position.

5. Competitive Rivalry

- The intensity of competition among existing tire service shops and automotive service providers.
- The shop's ability to differentiate its services and online experience from those of competitors.

4. PESTEL analysis

1. Political

- Regulatory Compliance: Adherence to online business regulations, data protection laws, and consumer rights when selling online.
- Trade Policies: Impact of local and international trade policies on the procurement of tire stock and parts, which could affect pricing.

2. Economic

- Market Competition: The economic landscape of the tire service industry, including competition and market demand.
- Economic Trends: Influence of local and global economic trends on consumer spending habits, especially in automotive services and products.
- Access to Funding: Availability of investments or loans for small businesses to invest in digital infrastructure.

3. Social

- Customer Expectations: Growing consumer preference for online shopping and digital service bookings.
- Demographic Trends: Identifying target demographics that are most likely to use digital platforms for automotive services.
- Digital Literacy: The level of digital literacy among the target customer base, influencing the adoption and effective use of the new website.

4. Technological

- E-commerce Platforms: The availability and selection of e-commerce platforms suitable for automotive services and product sales.
- Cybersecurity Measures: Ensuring the website is secure and protects customer data, building trust in the digital platform.
- Technological Integration: Ability to integrate new digital services with existing in-shop processes and systems for seamless operations.

5. Legal

- Online Business Laws: Compliance with laws governing online businesses, including sales tax, digital marketing, and consumer rights.
- Data Protection Regulations: Adherence to data protection laws like GDPR (for European customers) or local data protection laws.

6. Environmental

- Sustainability Concerns: Incorporating environmentally friendly practices in the delivery and packaging of products sold online.
- Waste Management: Ensuring proper disposal and recycling of old tires and automotive parts, aligning with environmental regulations and customer expectations for sustainability.

5. Requirements

1. Functional

- The system should notify the user upon changing an order state.
- The platform should support the addition of new pages and content sections to accommodate business growth and user interest.
- Users must register, if they do not have an account, otherwise they can log in.

- Administrators do not need to register - they can log in without registration.
- Administrators CRUD info about a product.
- Administrators can also perform all operations as a Registered user.
- Unregistered user can browse the catalog of available products.
- Registered user can order a product and manipulate its user's info.

2. Non-Functional

- It must be capable of handling a high volume of page requests and user actions without degradation in performance.
- The website should be designed with a user-friendly interface, making it easy for users of all technical abilities to navigate and find information.
- Content should be presented in a clear, concise manner, with a consistent layout across all pages.
- The platform must be scalable to accommodate an increasing number of users and data over time without requiring a complete redesign.
- It should support the easy addition of new features and services in response to customer feedback and business needs.
- The system must ensure the security of user data, including personal information and transaction details, adhering to best practices and compliance standards.
- It should include secure access controls for administrators and users.
- The system should guarantee an uptime of 99.9%.

6. Architecture

Project has only the backend part.

Backend is separated into microservices:

- User Service - CRUD operations on user
- Auth Service - issuing JWT token, login
- Api Gateway - request routing
- Discovery Service - microservices registration
- Order Service - placing order, getting user orders
- Product Service - CRUD operations on stock and product
- Notification Service - sending notifications (just logging)

Microservices communication is implemented by using Event-Driven Architecture and sending sync requests using Feign Client and async using Kafka broker.

7. Analysis and selection of solution alternatives

1. Own Development

Of all the alternatives, this seems to be the least expensive. The company is engaged in creating similar systems, so it has programmers and equipment for the realization of such a project. The problem is only the programmers' workload on other projects, so they would work on the solution between projects if they had nothing else to do.

However, this solution will extend the development time and it is not possible to

determine exactly when it will be completed. Another advantage is that we can further define the application specification without any financial penalties.

2. Ordering from Another Company

This option is better than the first one in that we do not have to worry about the development process. Here, it is necessary to specify the requirements more precisely. It is certainly a more expensive development option. The time of completion will probably be shorter, which can be an advantage when the company has other projects to take care of and needs this project completed. In case of any problems, having IT support is an advantage.

3. Open-source Solution

The open-source solution is the cheapest option of all three. Here, it is most important to choose the right product or products (1 product probably won't be enough). The disadvantages include factors such as the lack of IT support in case of problems. Another problem could be that the product does not meet the specifications or that the solution will be clumsy and will rather lead to delays in work.

8. WBS – breakdown of the project into sub-processes

In table 1 is shown WBS.

From - To	Activity
27.03 - 01.04	Planning
02.04 - 05.04	Requirement Analysis
06.04 - 10.04	Design
11.04 - 01.05	Implementation
02.05 - 03.05	Testing
04.03 - 10.05	Deployment
11.05 - inf	Maintenance

Table 1: WBS

9. Resources

1. Human Resources: Team Roles

- Developer
- Project Manager
- Analyst
- Tester

2. Tools needed for development

- Spring Boot - backend
- PostgreSQL - database
- Kafka - message broker
- JUnit - testing
- Docker - containerisation
- ELK - Collection logs, transforming, visualization
- AWS EC2 - deployment to the cloud

10. Responsibility Matrix

In table 2 is shown the RACI Matrix.

Milestones	Manager	Analyst	Developer	Tester
Planning	R	C	C	C
Requirements Analysis	A	R	C	I
Design	A	C	R	C
Implementation	A	I	R	I
Testing	A	I	R	R
Deployment	A	C	R	I
Maintenance	A	I	R	R

Table 2: RACI Matrix

11. GANTT Schedule

In figure 1 is shown the GANTT Schedule.

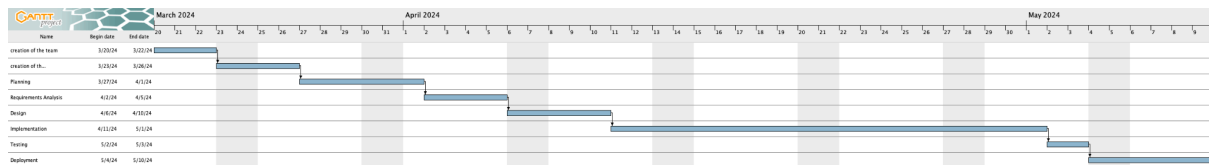


Figure 1: GANTT Schedule

12. FMEA Risk Analysis

1. Website Downtime

- Failure Mode: The website goes offline or becomes unresponsive.
- Effects: Customers can't book services or order products, leading to lost sales and dissatisfaction.
- Severity: High, as it directly affects sales and customer experience.
- Occurrence: Medium, depending on the reliability of the hosting service.
- Detection: High, likely to be noticed quickly.
- Risk Priority: High, critical to address with robust hosting and monitoring.

2. Payment Gateway Issues

- Failure Mode: Problems with the payment system, such as transaction failures.
- Effects: Frustration for customers, incomplete transactions, and potential loss of revenue.
- Severity: High, impacts customer trust and financial flow.
- Occurrence: Low to medium, depends on the gateway provider's reliability.
- Detection: Medium, might not be immediate if issues are intermittent.
- Risk Priority: High, requires choosing reliable payment solutions and having support ready.

3. Data Security Breach

- Failure Mode: Unauthorized access to customer data.
- Effects: Loss of customer trust, legal consequences, and potential financial liabilities.
- Severity: Very high, due to the impact on reputation and compliance issues.
- Occurrence: Low, if proper security measures are in place.
- Detection: Medium to high, can be complex to detect sophisticated breaches.
- Risk Priority: Very high, necessitates investing in strong cybersecurity measures.

4. Inaccurate Inventory Management

- Failure Mode: The website shows incorrect stock levels for products.
- Effects: Orders cannot be fulfilled as promised, leading to customer dissatisfaction and operational inefficiencies.
- Severity: Medium to high, affects customer satisfaction and inventory control.
- Occurrence: Medium, possible if not integrating real-time inventory tracking.

- Detection: Medium, might take time to realize discrepancies.
- Risk Priority: High, critical to integrate accurate, real-time inventory management systems.

5. Poor User Experience (UX)

- Failure Mode: The website is difficult to navigate or use.
- Effects: Potential customers leave the site without booking or buying, reducing conversion rates.
- Severity: Medium, impacts brand perception and sales.
- Occurrence: High, if not designed with user feedback.
- Detection: Low to medium, may require specific feedback or analytics to identify.
- Risk Priority: Medium to high, emphasizes the need for user testing and iterative design improvements.

13. Metrics

1. Performance Metrics

- Website Load Time: Measures the time taken for the site to become fully interactive. A critical factor for user satisfaction and SEO.
- System Uptime: Percentage of time the digital platform is operational and accessible to users. High uptime ensures reliability.
- Response Time for Service Requests: The average time the system takes to respond to user actions, affecting user experience.

2. User Engagement Metrics

- Page Views: The total number of pages viewed. Reflects the level of interest and exploration by users.
- Session Duration: Average time spent by a user per visit. Longer sessions can indicate more engaging content or features.

3. Customer Satisfaction Metrics

- Net Promoter Score (NPS): Measures customer willingness to recommend your service to others. A high NPS indicates strong customer loyalty.
- Customer Satisfaction Score (CSAT): Direct feedback on customer satisfaction with a specific service or transaction.

4. Other Metrics

- Number of User Cases
- Number of Microservices
- Number of the Roles
- Number of the JUnit tests
- Number of the Environments

14. UML diagrams

1. Class Diagram

In figure 2 is shown the Class Diagram.

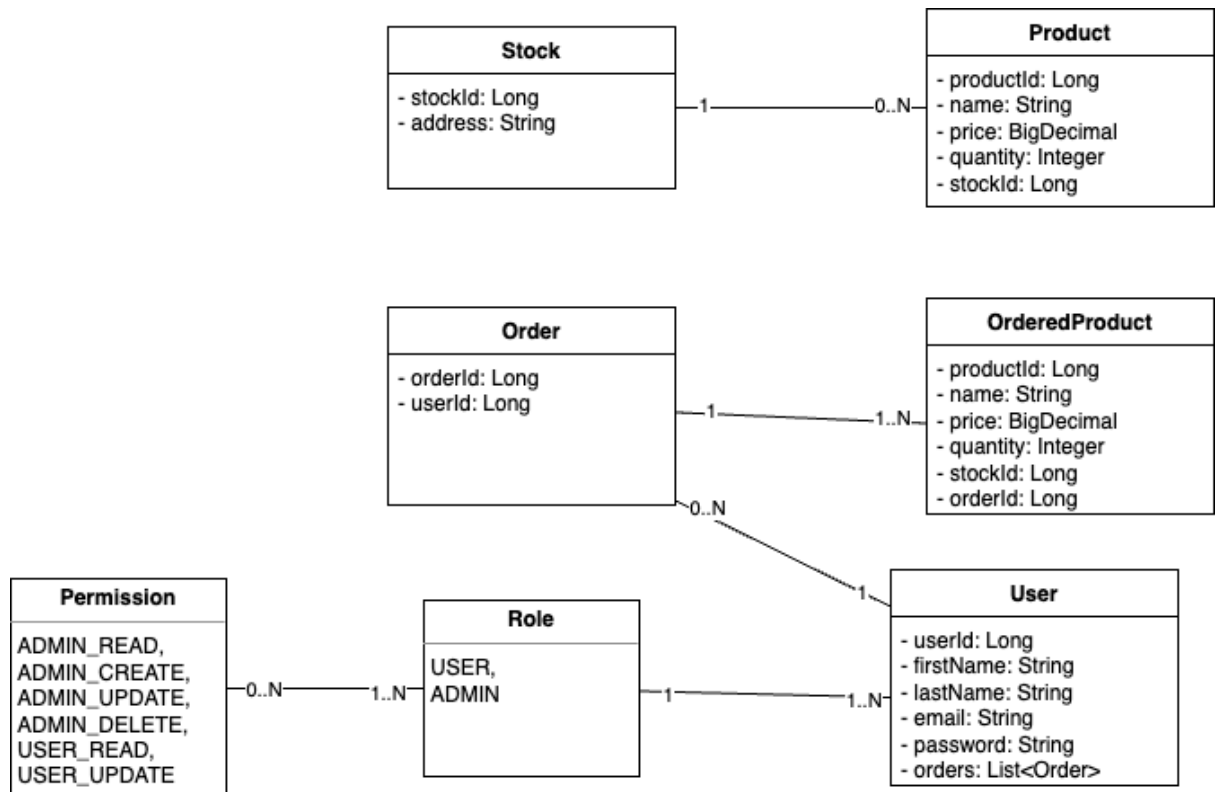


Figure 2: Class Diagram

2. Use Case Diagram

In figure 3 is shown the Use Case Diagram.

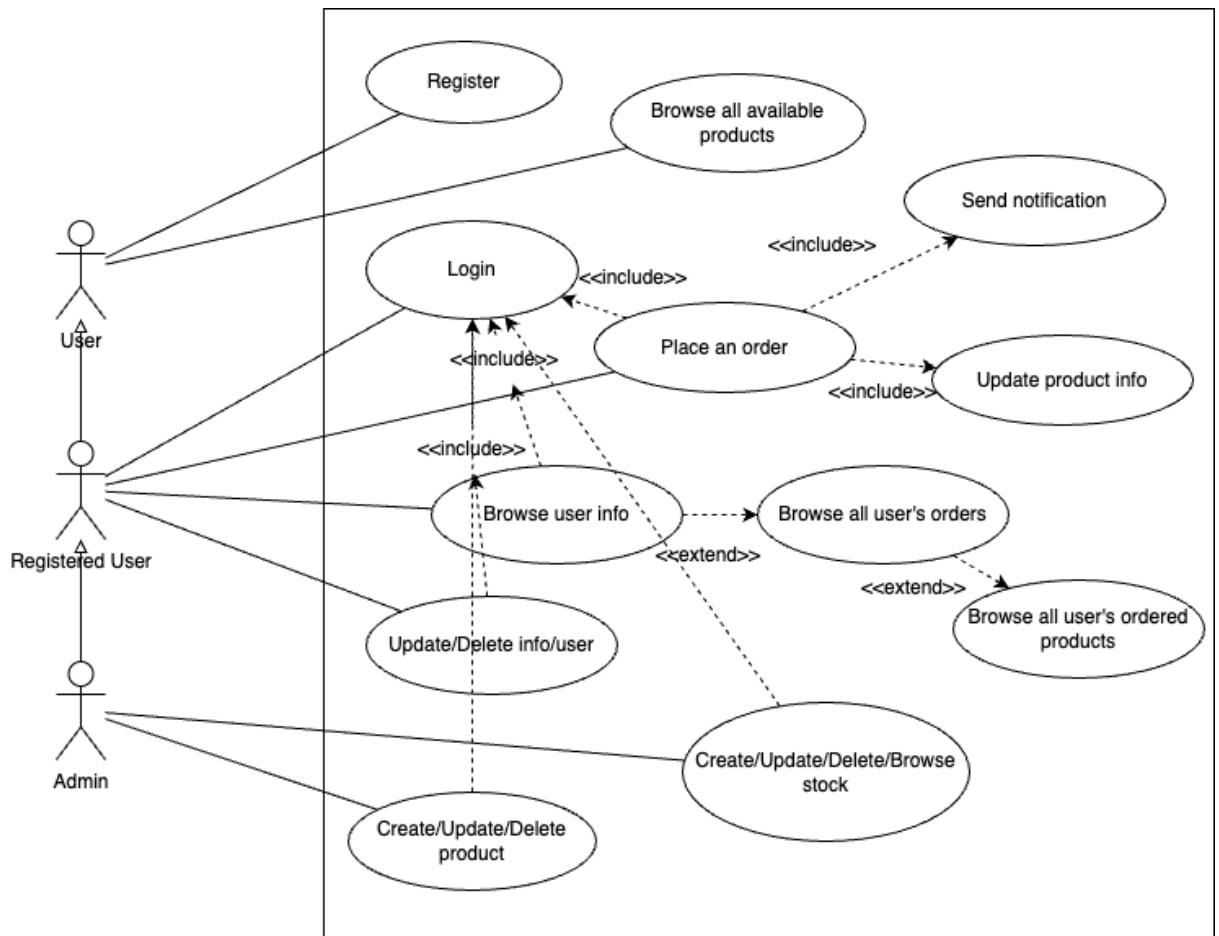


Figure 3: Use Case Diagram

3. Component Diagram

In figure 4 is shown the Use Case Diagram.

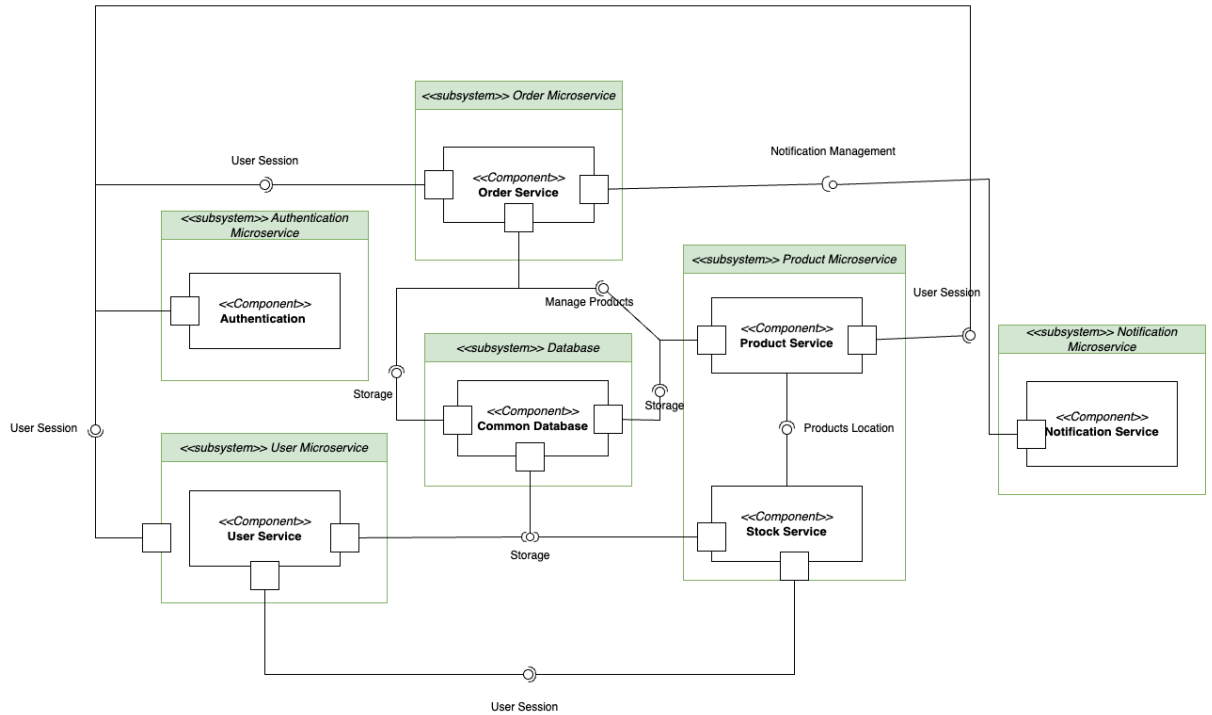


Figure 4: Component Diagram

4. Deployment Diagram

In figure 5 is shown the Deployment Diagram.

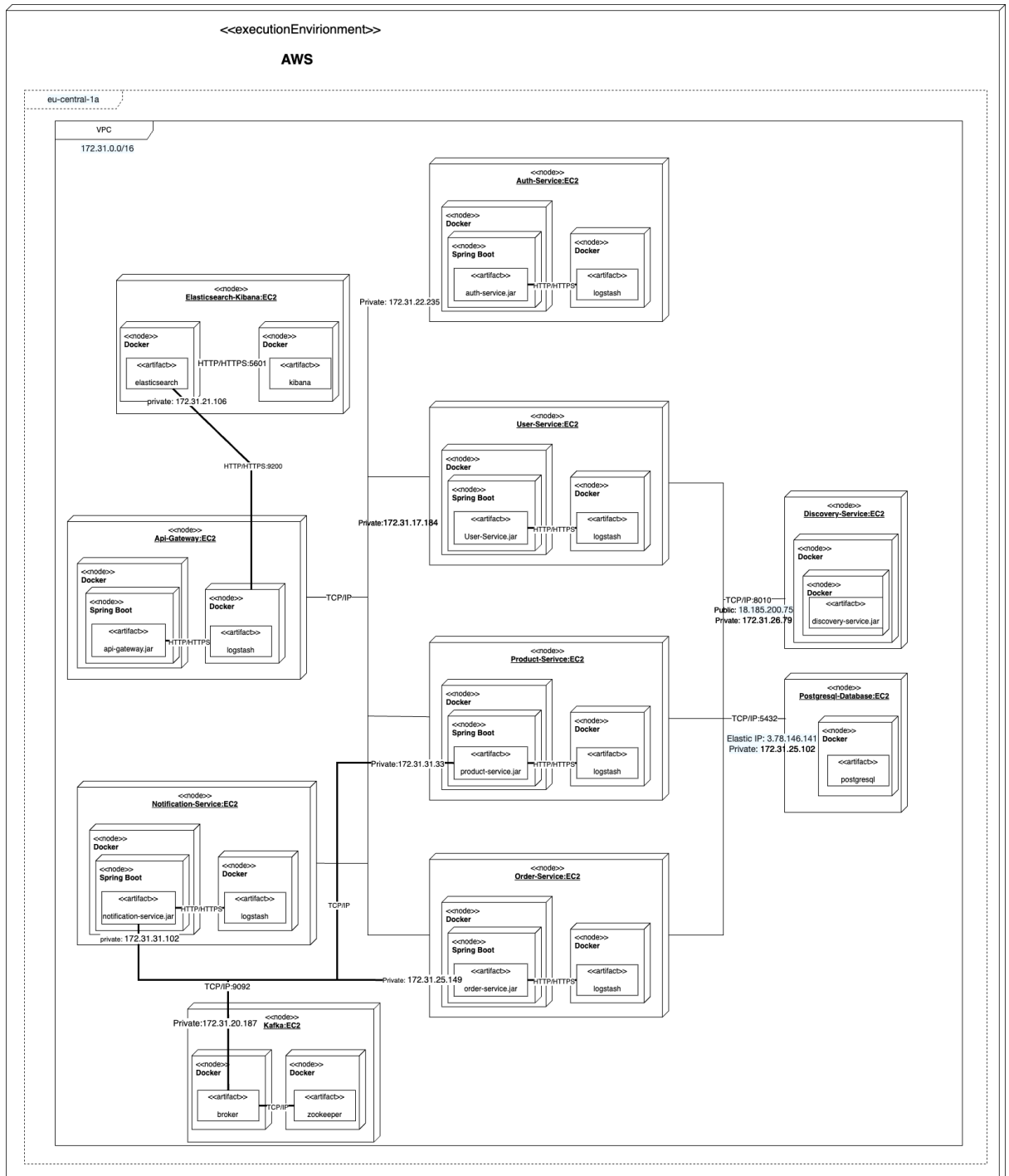


Figure 5: Deployment Diagram

5. Sequence Diagram

In figure 6 is shown the Sequence Diagram of placing an order.

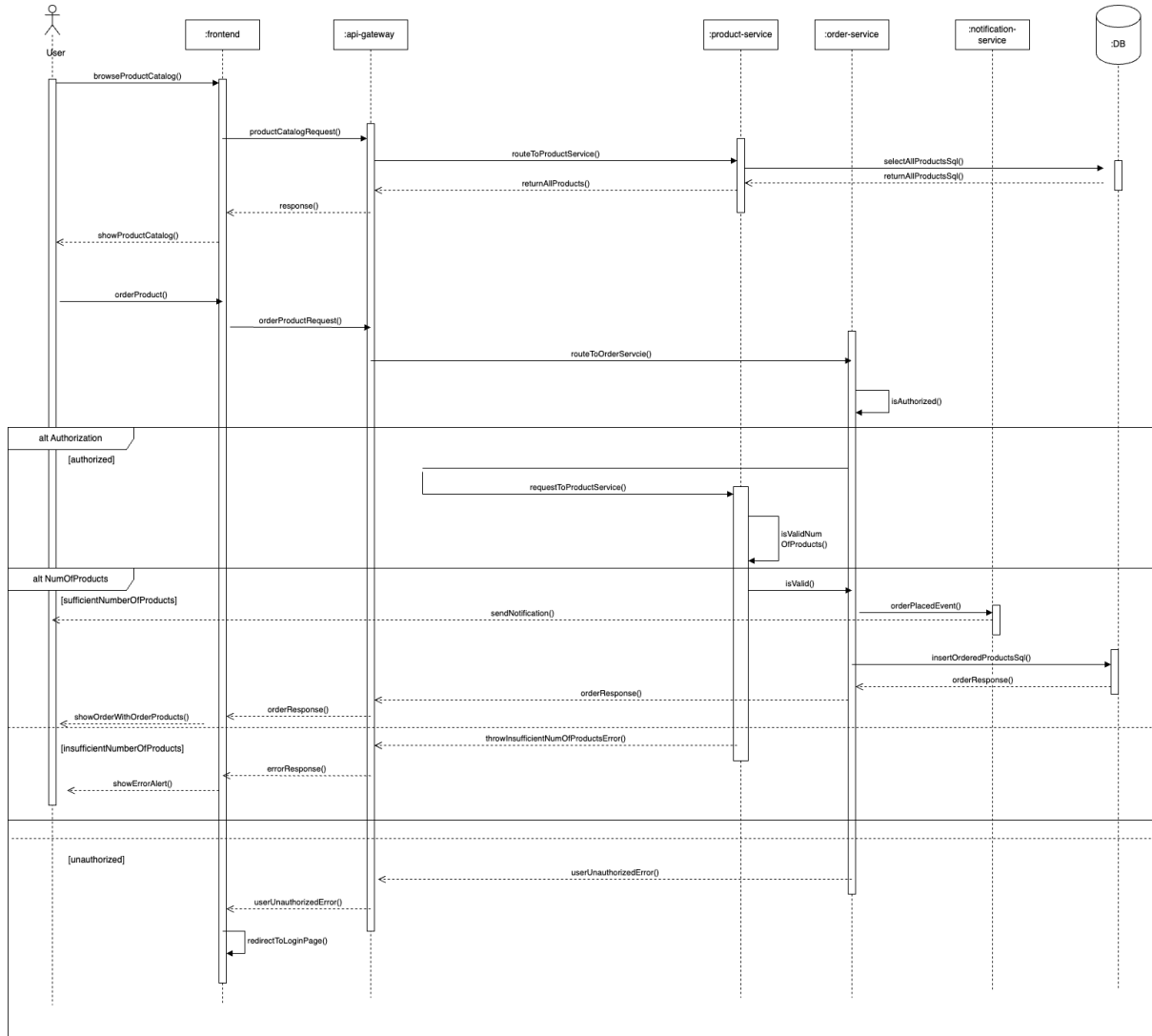


Figure 6: Sequence Diagram of placing an order

15. Deployment process

1. Development environment

Run the following command to run the application in the development environment:

```
docker compose up -d
```

Source Code 1: Starting application in development mode

2. Production environment

Configure execution environment in AWS:

1. Select the nearest Region - Frankfurt (eu-central-1a)
2. Select VPC - default is ok (172.31.0.0/16 mask)
3. Create .pem file (key pair)

Launch EC2:

1. Give a name
2. Select Amazon Machine Image (AMI) - Amazon Linux 2023 AMI (free tier)
3. Instance Type column (Figure 7)
4. Key Pair - I selected the same for all instances
5. Security Groups (Figure 8 - 17)

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone
kafka	i-0bc2e5bf095ff84bf	Running	t2.small	2/2 checks passed	View alarms	eu-central-1a
auth-service	i-0be88c01dd8f401f8	Running	t2.small	2/2 checks passed	View alarms	eu-central-1a
product-service	i-06a7d1f37afb40864	Running	t2.small	2/2 checks passed	View alarms	eu-central-1a
order-service	i-05e67238e07e7eeff	Running	t2.small	2/2 checks passed	View alarms	eu-central-1a
discovery-serv...	i-09ae84d4cb1947eb9	Running	t2.micro	2/2 checks passed	View alarms	eu-central-1a
es-kibana	i-0c7b0ba886bb6cc98	Running	t2.medium	2/2 checks passed	View alarms	eu-central-1a
postgresql	i-0426ba36f77d4cca3	Running	t2.micro	2/2 checks passed	View alarms	eu-central-1a
user-service	i-0b32d00858c069b39	Running	t2.small	2/2 checks passed	View alarms	eu-central-1a
api-gateway	i-0603e6a5ca1c97e81	Running	t2.small	2/2 checks passed	View alarms	eu-central-1a
notification-se...	i-0a04b8bd5394089cd	Running	t2.small	2/2 checks passed	View alarms	eu-central-1a

Figure 7: Running EC2's

Type	Protocol	Port range	Source	Description
Custom TCP	TCP	8010	172.31.0.0/16	Only VPC
Custom TCP	TCP	8010	147.32.104.56/32	My IP
SSH	TCP	22	0.0.0.0/0	-

Figure 8: Security group set up for discovery service EC2

Inbound rules (2)

Manage tags

Edit inbound rules

Q Search

< 1 >

▼	Type	▼	Protocol	▼	Port range	▼	Source	▼	Description	▼
	Custom TCP		TCP		8080		0.0.0.0/0		anywhere ipv4	
	SSH		TCP		22		0.0.0.0/0		–	

Figure 9: Security group set up for api gateway EC2

Inbound rules (3)

Manage tags

Edit inbound rules

Q Search

< 1 >

▼	Type	▼	Protocol	▼	Port range	▼	Source	▼	Description	▼
	PostgreSQL		TCP		5432		172.31.0.0/16		Only in VPC	
	SSH		TCP		22		0.0.0.0/0		–	
	PostgreSQL		TCP		5432		147.32.104.56/32		My IP	

Figure 10: Security group set up for postgresql database EC2

Inbound rules (2)

Manage tags

Edit inbound rules

Search

<

1

>

<div>▼</div>	Type	<div>▼</div>	Protocol	<div>▼</div>	Port range	<div>▼</div>	Source	<div>▼</div>	Description	<div>▼</div>
	SSH		TCP		22		0.0.0.0/0		–	
	All TCP		TCP		0 - 65535		172.31.0.0/16		All in VPC	

Figure 11: Security group set up for user service EC2

Inbound rules (2)

Manage tags

Edit inbound rules

Q Search

< 1 >

▼	Type	▼	Protocol	▼	Port range	▼	Source	▼	Description	▼
	SSH		TCP		22		0.0.0.0/0		–	
	All TCP		TCP		0 - 65535		172.31.0.0/16		All in VPC	

Figure 12: Security group set up for auth service EC2

Inbound rules (2)

Manage tags

Edit inbound rules

Q Search

< 1 >

▼	Type	▼	Protocol	▼	Port range	▼	Source	▼	Description	▼
	SSH		TCP		22		0.0.0.0/0		-	
	Custom TCP		TCP		9092		172.31.0.0/16		only VPC instances acc...	

Figure 13: Security group set up for kafka EC2

Inbound rules (2)

Manage tags

Edit inbound rules

Q Search

< 1 >

▼	Type	▼	Protocol	▼	Port range	▼	Source	▼	Description	▼
	SSH		TCP		22		0.0.0.0/0		-	
	All TCP		TCP		0 - 65535		172.31.0.0/16		All in VPC	

Figure 14: Security group set up for product service EC2

Inbound rules (2)

Manage tags

Edit inbound rules

Q Search

< 1 >

▼	Type	▼	Protocol	▼	Port range	▼	Source	▼	Description	▼
	SSH		TCP		22		0.0.0.0/0		–	
	All TCP		TCP		0 - 65535		172.31.0.0/16		All in VPC	

Figure 15: Security group set up for order service EC2

Inbound rules (2)

Manage tags

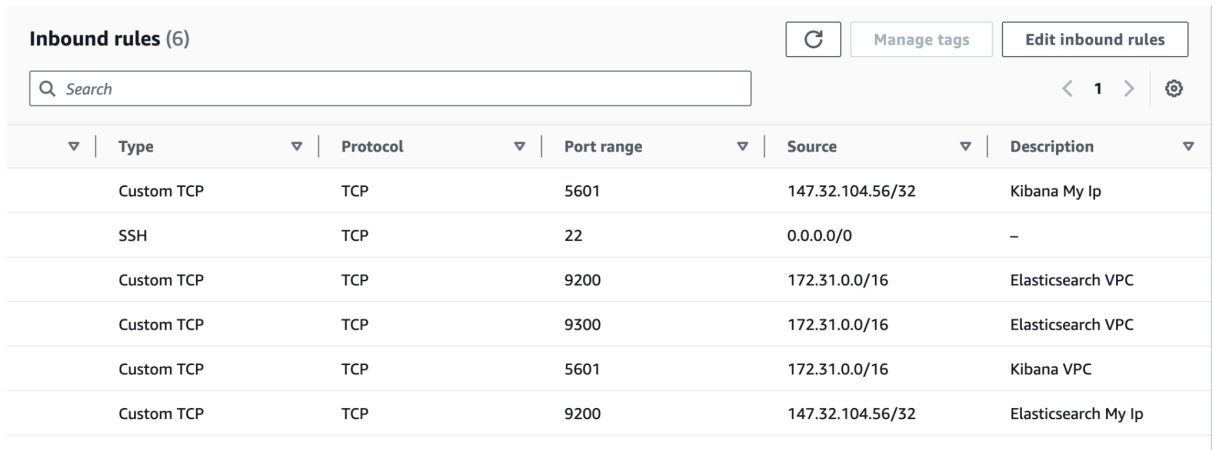
Edit inbound rules

Q Search

< 1 >

▼	Type	▼	Protocol	▼	Port range	▼	Source	▼	Description	▼
	SSH		TCP		22		0.0.0.0/0		–	
	All TCP		TCP		0 - 65535		172.31.0.0/16		All in VPC	

Figure 16: Security group set up for notification service EC2



Type	Protocol	Port range	Source	Description
Custom TCP	TCP	5601	147.32.104.56/32	Kibana My Ip
SSH	TCP	22	0.0.0.0/0	–
Custom TCP	TCP	9200	172.31.0.0/16	Elasticsearch VPC
Custom TCP	TCP	9300	172.31.0.0/16	Elasticsearch VPC
Custom TCP	TCP	5601	172.31.0.0/16	Kibana VPC
Custom TCP	TCP	9200	147.32.104.56/32	Elasticsearch My Ip

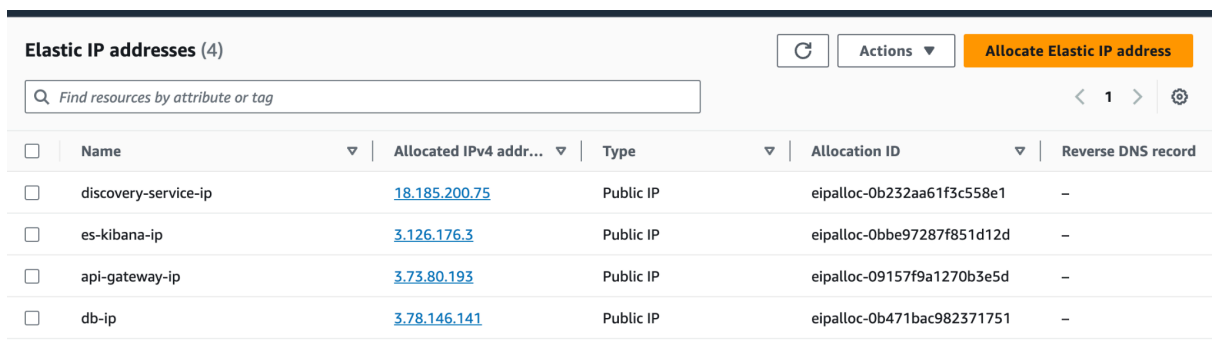
Figure 17: Security group set up for elasticsearch kibana EC2

6. Install docker on EC2 instance (Source Code 2).

```
sudo yum update && \
sudo yum install docker -y && \
sudo service docker start && \
sudo usermod -a -G docker ec2-user && \
exit
```

Source Code 2: Sequence of the commands for docker installation on EC2

7. Allocate Static IPs for curtain EC2's (Figure 18)



Name	Allocated IPv4 address	Type	Allocation ID	Reverse DNS record
discovery-service-ip	18.185.200.75	Public IP	eipalloc-0b232aa61f3c558e1	–
es-kibana-ip	3.126.176.3	Public IP	eipalloc-0bbe97287f851d12d	–
api-gateway-ip	3.73.80.193	Public IP	eipalloc-09157f9a1270b3e5d	–
db-ip	3.78.146.141	Public IP	eipalloc-0b471bac982371751	–

Figure 18: Allocated Elastic IPs for discovery service, es-kibana, api-gateway and postgresql database

8. Build docker image and docker container from it on EC2 for

- discovery service image build (Source Code 3)

```
docker buildx build --platform linux/amd64 -t
aivazart/discovery-service:2.1 --force-rm=true . && docker push
aivazart/discovery-service:2.1
```

Source Code 3: discovery service image build

discovery service container running (Source Code 4)

```
docker run -d --name discovery-service -p 8010:8010 -e
"SPRING_PROFILES_ACTIVE=production" aivazart/discovery-service:2.1
```

Source Code 4: discovery service container running

- api-gateway image build (Source Code 5)

```
docker buildx build --platform linux/amd64 -t aivazart/api-gateway:2.0
--force-rm=true . && docker push aivazart/api-gateway:2.0
```

Source Code 5: api-gateway image build

api-gateway container running (Source Code 6)

```
docker run -d --name api-gateway -p 8080:8080 -e
"SPRING_PROFILES_ACTIVE=production" -v /api-logs:/api-logs
aivazart/api-gateway:2.0
```

Source Code 6: api-gateway container running

- postgresql database
 1. Choose a directory on the EC2 host to store the PostgreSQL data. For example, let's use /var/lib/postgresql/data.
 2. Create dir on host to persist data (Source Code 7):

```
sudo mkdir -p /var/lib/postgresql/data
sudo chown 999:999 /var/lib/postgresql/data
sudo chmod -R 700 /var/lib/postgresql/data
```

Source Code 7: Directory creation for data persisting

3. Create .sql script for database creation (Source Code 8):

```
touch /scripts/init-db.sql
sudo chmod 644 /scripts/init-db.sql
sudo chown 999:999 /scripts/init-db.sql
```

Source Code 8: Directory creation for data persisting

4. postgresql database container running (Source Code 9)

```
docker run -d \
  --name eshop-db \
  -e POSTGRES_DB=eshop \
  -e POSTGRES_USER=admin \
  -e POSTGRES_PASSWORD=strong-production-password \
  -e PGDATA=/var/lib/postgresql/data/pgdata \
  -p 5432:5432 \
  -v /var/lib/postgresql/data:/var/lib/postgresql/data \
  -v /scripts:/docker-entrypoint-initdb.d \
  postgres:15
```

Source Code 9: postgresql database container running

- auth-service build image (Source Code 10):

```
docker buildx build --platform linux/amd64 -t
aivazart/auth-service:2.0 --force-rm=true . && docker push
aivazart/auth-service:2.0
```

Source Code 10: auth-service build image

auth-service run container (Source Code 11):

```
docker run -d \
  --name auth-service \
  --env SPRING_PROFILES_ACTIVE=production \
  -v /api-logs:/api-logs \
  aivazart/auth-service:2.0
```

Source Code 11:auth-service run container

- user-service build image (Source Code 12)

```
docker buildx build --platform linux/amd64 -t
aivazart/user-service:2.4 --force-rm=true -f Dockerfile-production . &&
docker push aivazart/user-service:2.4
```

Source Code 12: user-service build image

user-service un container in production mode and execute script for admin insertion (inside container logic, vz Dockerfile-production in user-service) (Source Code 13)

```
docker buildx build --platform linux/amd64 -t
aivazart/user-service:2.4 --force-rm=true -f Dockerfile-production . &&
docker push aivazart/user-service:2.4
```

Source Code 13: user-service run container

- Kafka

1. Create docker network kafka-network for container communication by their names (Source Code 14)

```
docker network create kafka-network
```

Source Code 14: Create docker network for kafka

2. Run Zookeeper container (Source Code 15)

```
docker run -d \
  --name zookeeper \
  --network kafka-network \
  -p 2181:2181 \
  -e ZOOKEEPER_CLIENT_PORT=2181 \
  -e ZOOKEEPER_TICK_TIME=2000 \
  confluentinc/cp-zookeeper:7.0.1
```

Source Code 15: Run Zookeeper container

3. Run Kafka Broker container (Source Code 16)

```
docker run -d \
  --name broker \
  --network kafka-network \
  -p 9092:9092 \
  -e KAFKA_BROKER_ID=1 \
  -e KAFKA_ZOOKEEPER_CONNECT=zookeeper:2181 \
  -e
  KAFKA_LISTENER_SECURITY_PROTOCOL_MAP=PLAINTEXT:PLAINTEXT,PLAINTEXT_IN
  TERNAL:PLAINTEXT \
  -e
  KAFKA_ADVERTISED_LISTENERS=PLAINTEXT://172.31.20.187:9092,PLAINTEXT_IN
  TERNAL://broker:29092 \
  -e KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR=1 \
  -e KAFKA_TRANSACTION_STATE_LOG_MIN_ISR=1 \
  -e KAFKA_TRANSACTION_STATE_LOG_REPLICATION_FACTOR=1 \
  confluentinc/cp-kafka:7.0.1
```

Source Code 16: Run Kafka Broker container

- product-service build image (Source Code 17)

```
docker buildx build --platform linux/amd64 -t  
aivazart/product-service:2.1 --force-rm=true . && docker push  
aivazart/product-service:2.1
```

Source Code 17: Product-service build image

product-service run container (Source Code 18)

```
docker run -d \  
--name product-service \  
--env SPRING_PROFILES_ACTIVE=production \  
-v /api-logs:/api-logs \  
aivazart/product-service:2.1
```

Source Code 18: Product-service run container

- order-service build image (Source Code 19)

```
docker buildx build --platform linux/amd64 -t  
aivazart/order-service:2.0 --force-rm=true . && docker push  
aivazart/order-service:2.1
```

Source Code 19: Order-service build image

order-service run container (Source Code 20)

```
docker run -d \  
--name order-service \  
--env SPRING_PROFILES_ACTIVE=production \  
-v /api-logs:/api-logs \  
aivazart/order-service:2.1
```

Source Code 20: Order-service run container

- notification-service build image (Source Code 21)

```
docker buildx build --platform linux/amd64 -t  
aivazart/notification-service:2.0 --force-rm=true . && docker push  
aivazart/notification-service:2.0
```

Source Code 21: Notification-service build image

notification-service run container (Source Code 22)

```
docker run -d \  
--name notification-service \  
--env SPRING_PROFILES_ACTIVE=production \  
-v /api-logs:/api-logs \  
aivazart/notification-service:2.0
```

Source Code 22: Notification-service run container

- Setting up Elasticsearch, Kibana and Logstash on EC2.

The main purpose of usage ELK stack in this project is to aggregate logs, transform them and visualize them in Kibana using data views.

1. Create docker network for communication between containers by their names (Source Code 23)

```
docker network create elastic
```

Source Code 22: Create docker network for ELK stack

2. Docker command sets up a temporary container named `setup` (Source Code 23) to generate SSL certificates for an Elasticsearch cluster and sets initial passwords using variables like in `.env` file for development environment (Source Code 25)

```
# Password for the 'elastic' user (at least 6 characters)
ELASTIC_PASSWORD=elasticsearch1

# Password for the 'kibana_system' user (at least 6 characters)
KIBANA_PASSWORD=kibana1

# Version of Elastic products
STACK_VERSION=8.7.1

# Set the cluster name
CLUSTER_NAME=docker-cluster

# Set to 'basic' or 'trial' to automatically start the 30-day trial
LICENSE=basic

# Port to expose Elasticsearch HTTP API to the host
ES_PORT=9200

# Port to expose Kibana to the host
KIBANA_PORT=5601

# Increase or decrease based on the available host memory
ES_MEM_LIMIT=1g
KB_MEM_LIMIT=1g
LS_MEM_LIMIT=1g

# SAMPLE Predefined Key only to be used in POC environments
ENCRYPTION_KEY=e55337180158e77d16a5a881281584c74beec94ad7dd973c18dfd2eec75913f6
```

Source Code 25: `.env` file for development environment

```

docker run --name setup --net elastic --rm -v
certs:/usr/share/elasticsearch/config/certs -e "ELASTIC_PASSWORD=elasticsearch1" -e
"KIBANA_PASSWORD=kibana1" -e "STACK_VERSION=8.7.1" -e "CLUSTER_NAME=docker-cluster"
--user 0 docker.elastic.co/elasticsearch/elasticsearch:8.7.1 bash -c '
if [ ! -f config/certs/ca.zip ]; then
    echo "Creating CA";
    bin/elasticsearch-certutil ca --silent --pem -out config/certs/ca.zip;
    unzip config/certs/ca.zip -d config/certs;
fi;
if [ ! -f config/certs/certs.zip ]; then
    echo "Creating certs";
    echo -ne \
    "instances:\n"\
    "  - name: es01\n"\
    "    dns:\n"\
    "      - es01\n"\
    "      - localhost\n"\
    "      - 172.31.21.106\n"\
    "    ip:\n"\
    "      - 127.0.0.1\n"\
    "      - 172.31.21.106\n"\
    "  - name: kibana\n"\
    "    dns:\n"\
    "      - kibana\n"\
    "      - localhost\n"\
    "      - 172.31.21.106\n"\
    "    ip:\n"\
    "      - 127.0.0.1\n"\
    "      - 172.31.21.106\n"
    > config/certs/instances.yml;
    bin/elasticsearch-certutil cert --silent --pem -out config/certs/certs.zip --in
config/certs/instances.yml --ca-cert config/certs/ca/ca.crt --ca-key
config/certs/ca/ca.key;
    unzip config/certs/certs.zip -d config/certs;
fi;
echo "Setting file permissions"
chown -R root:root config/certs;
find . -type d -exec chmod 750 {\} \;;
find . -type f -exec chmod 640 {\} \;;
echo "Waiting for Elasticsearch availability";
until curl -s --cacert config/certs/ca/ca.crt https://es01:9200 | grep -q "missing
authentication credentials"; do sleep 5; done;
echo "Setting kibana_system password";

```

```
until curl -s -X POST --cacert config/certs/ca/ca.crt -u
"elastic:${ELASTIC_PASSWORD}" -H "Content-Type: application/json"
https://es01:9200/_security/user/kibana_system/_password -d
"{\"password\": \"${KIBANA_PASSWORD}\"}" | grep -q "^{}"; do sleep 5; done;
echo "All done!";
,
```

Source Code 23: Run Setup container

3. Run elasticsearch container (Source Code 24):

```
docker run -d --name es01 --net elastic -v
certs:/usr/share/elasticsearch/config/certs -v
esdata01:/usr/share/elasticsearch/data -e "node.name=es01" -e
"cluster.name=docker-cluster" -e "discovery.type=single-node"
-e "ELASTIC_PASSWORD=elasticsearch1" -e
"bootstrap.memory_lock=true" -e "xpack.security.enabled=true"
-e "xpack.security.http.ssl.enabled=true" -e
"xpack.security.http.ssl.key=certs/es01/es01.key" -e
"xpack.security.http.ssl.certificate=certs/es01/es01.crt" -e
"xpack.security.http.ssl.certificate_authorities=certs/ca/ca.c
rt" -e "xpack.security.transport.ssl.enabled=true" -e
"xpack.security.transport.ssl.key=certs/es01/es01.key" -e
"xpack.security.transport.ssl.certificate=certs/es01/es01.crt"
-e
"xpack.security.transport.ssl.certificate_authorities=certs/ca/
ca.crt" -e
"xpack.security.transport.ssl.verification_mode=certificate" -e
"xpack.license.self_generated.type=basic" -p 9200:9200 --ulimit
memlock=-1:-1
docker.elastic.co/elasticsearch/elasticsearch:8.7.1
```

Source Code 24: Run Elasticsearch container

4. Run Kibana container (Source Code 26)

```
docker run -d --name kibana --net elastic -v
certs:/usr/share/kibana/config/certs -v
kibanadata:/usr/share/kibana/data -e "SERVERNAME=kibana" -e
"ELASTICSEARCH_HOSTS=https://es01:9200" -e
"ELASTICSEARCH_USERNAME=kibana_system" -e
"ELASTICSEARCH_PASSWORD=kibana1" -e
"ELASTICSEARCH_SSL_CERTIFICATEAUTHORITIES=config/certs/ca/ca.cer
t" -e
"XPACK_SECURITY_ENCRYPTIONKEY=e55337180158e77d16a5a881281584c74
beec94ad7dd973c18dfd2eec75913f6" -e
"XPACK_ENCRYPTEDSAVEDOBJECTS_ENCRYPTIONKEY=e55337180158e77d16a5
a881281584c74beec94ad7dd973c18dfd2eec75913f6" -e
"XPACK_REPORTING_ENCRYPTIONKEY=e55337180158e77d16a5a881281584c7
4beec94ad7dd973c18dfd2eec75913f6" -p 5601:5601
docker.elastic.co/kibana/kibana:8.7.1
```

Source Code 26: Run Kibana container

5. **Example of running a logstash container for api-gateway. There is the same procedure for others EC2's:**

Copy(create) logstash.conf to \$(pwd)/logstash-production/api-gateway
(Source Code 27) (Source Code 28)

```
mkdir -p ~/logstash-production/api-gateway/
touch ~/logstash-production/api-gateway/logstash.conf
```

Source Code 27: Create logstash.conf file

```

input {
  file {
    path => "/api-logs/api-gateway.log"
    tags => ["api-gateway-log"]
  }
}

output {
  if "api-gateway-log" in [tags] {
    elasticsearch {
      index => "api-gateway-%{+YYYY.MM.dd}"
      hosts => ["https://172.31.21.106:9200"]
      user=> "elastic"
      password=> "elasticsearch1"
      cacert=> "/usr/share/logstash/certs/ca/ca.crt"
    }
  }
}

stdout { codec => rubydebug }
}

```

Source Code 28: logstash.conf file for api-gateway EC2

6. Because Elasticsearch with Kibana and Logstash run on different EC2 instances we need to copy SSL certificates from es-kibana EC2 to api-gateway EC2 with a logstash container.
7. Copying certificates folder from es-kibana EC2 to Api Gateway EC2 (Source Code 29)(Source Code 30)(Source Code 31)

```

ssh -i "tier-service.pem"
ec2-user@ec2-3-126-176-3.eu-central-1.compute.amazonaws.com

```

Source Code 29: Connect to Elasticsearch Kibana EC2

```

scp -i ~/Downloads/tier-service.pem
~/Downloads/tier-service.pem
ec2-user@ec2-3-126-176-3.eu-central-1.compute.amazonaws.com:/home/ec2-user/.ssh/

```

```

chmod 400 /home/ec2-user/.ssh/tier-service.pem

```

Source Code 30: Add .pem file to the es-kibana EC2 using scp and ensure that correct permissions are set

```
sudo scp -i /home/ec2-user/.ssh/tier-service.pem -r
/var/lib/docker/volumes/certs/_data
ec2-user@ec2-3-120-247-140.eu-central-1.compute.amazonaws.com:/
home/ec2-user/certs
```

Source Code 31: Copy the certs directory to the Api-gateway EC2 instance from the es-kibana EC2 instance

8. Run logstash container (Source Code 32)

```
docker run -d --name logstash
-v /home/ec2-user/certs:/usr/share/logstash/certs
-v /api-logs:/api-logs
-v
/home/ec2-user/logstash-production/api-gateway/logstash.conf:/usr/share/logstash/pipeline/logstash.conf:ro
-e "xpack.monitoring.enabled=false"
-e "ELASTIC_USER=elastic"
-e "ELASTIC_PASSWORD=elasticsearch1"
-e "ELASTIC_HOSTS=https://172.31.21.106:9200"
docker.elastic.co/logstash/logstash:8.7.1
```

Source Code 32: Run logstash container on Api Gateway EC2 instance

9. Check if it works (Figure 19) (Figure 20) (Figure 21)



Figure 19: Check in browser if index for api-gateway has been created in elasticsearch

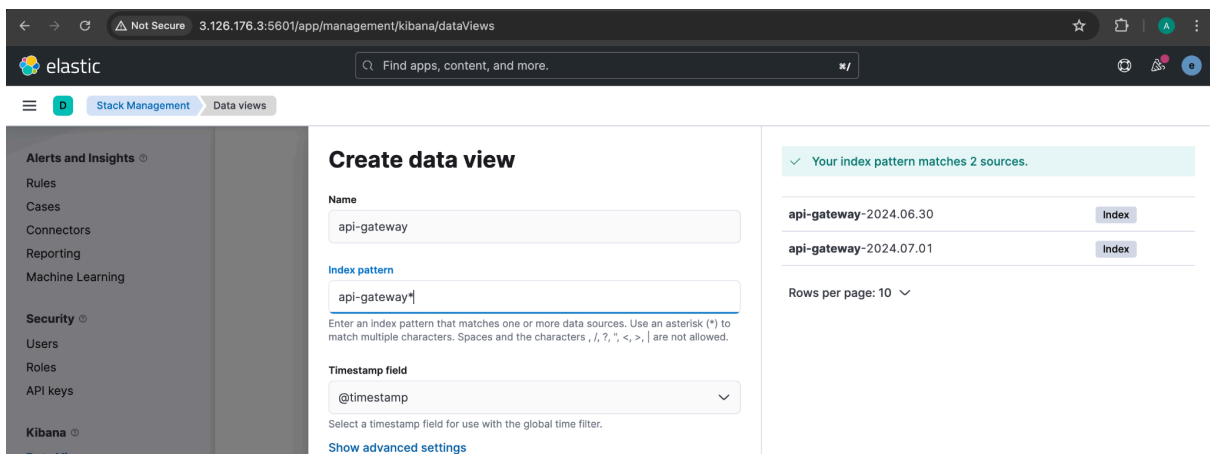


Figure 20: Create Data View for api-gateway

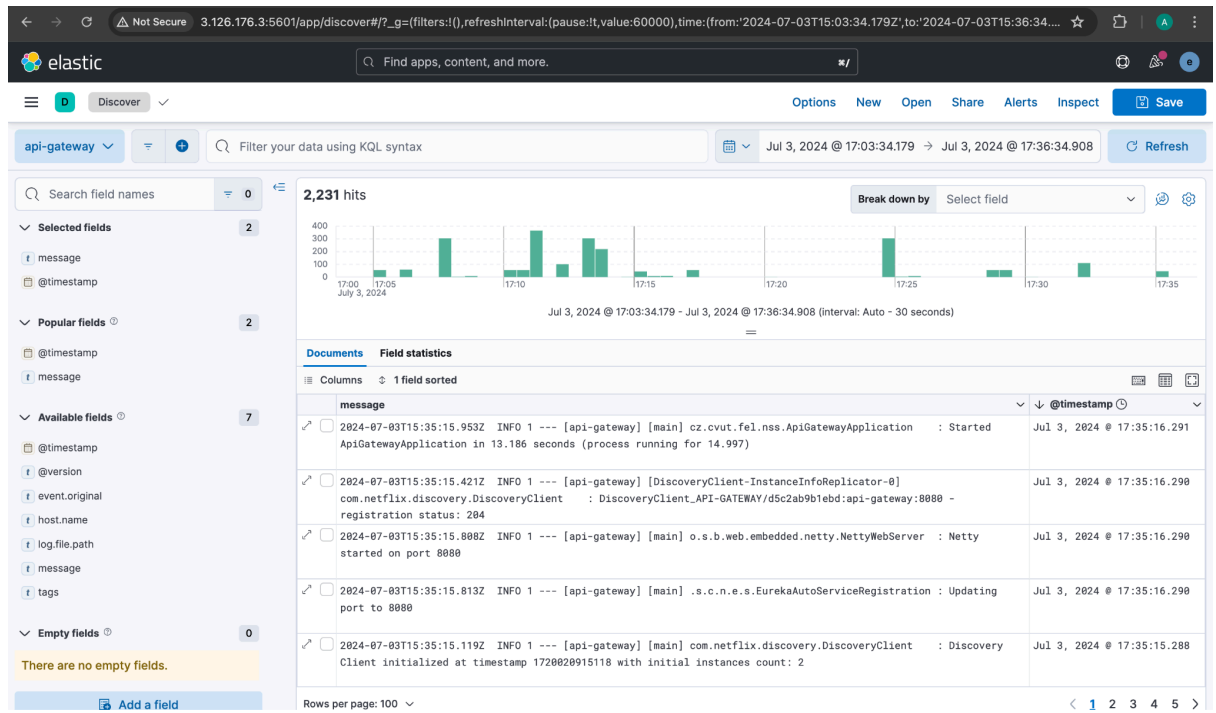


Figure 20: Visualize logs for api-gateway EC2 in Kibana