Experiment - 2

Student Name: Tanay Manish Nesari **UID:** 23BCS13761
Branch: BE-CSE **Section/Group:** KRG_2B
Semester: 6th **Date of Performance:** 14/1/26
Subject Name: System Design
Subject Code: 23CSH-314

Aim -

Design an Online shopping platform similar to Amazon / Flipkart that will allow users to purchase mobiles, laptops, cameras, clothes etc.

Objectives -

- To study cloud-based system design architecture
- To understand the use of VPC, subnets, and network isolation
- To design a highly available system using load balancing
- To implement scalability using auto scaling mechanisms
- To apply security, monitoring, and storage best practices in system design

Procedure -

1. Analyzed system requirements and identified architectural components.
2. Created a Virtual Private Cloud (VPC) for network isolation.
3. Configured public and private subnets within the VPC.
4. Deployed an Application Load Balancer in the public subnet.
5. Launched EC2 instances in private subnets for application hosting.
6. Enabled Auto Scaling to handle variable traffic loads.
7. Deployed a relational database in private subnets for secure data storage.
8. Configured object storage for static content and backups.
9. Enabled monitoring and logging services for system performance tracking.
10. Documented the complete architecture using a system design diagram.

Functional Requirements -

- User should be able to search and find the products based on product title or names.
- User should be able to view the details of the product llike description, image, available quantity, review.
- User should be able to select the quantity and move the product/item into the cart.
- User should be able to make the payment and should be able to perform the check out.
- User should be able to check the status of the order.

- System should be able to manage purchase of items having limited stocks.
- Race condition happens during a flash sale when inventory has limited stock, and the system must handle multiple transactions occurring at the same time.

Non-Functional Requirements -

- The system is designed for 100 million daily active users with around 10 orders handled per second.
- Consistency & Availability: Based on the target scale, both are required, but at different system levels. :

  i. According to functional requirements, users must search products smoothly, so product search needs high availability.

  ii. Strong consistency is required for critical components such as payment processing, order placement, and inventory management.

- Expected response time is approximately 200 ms.
- Scaling will be done either Horizontally or vertically wherever applicable.

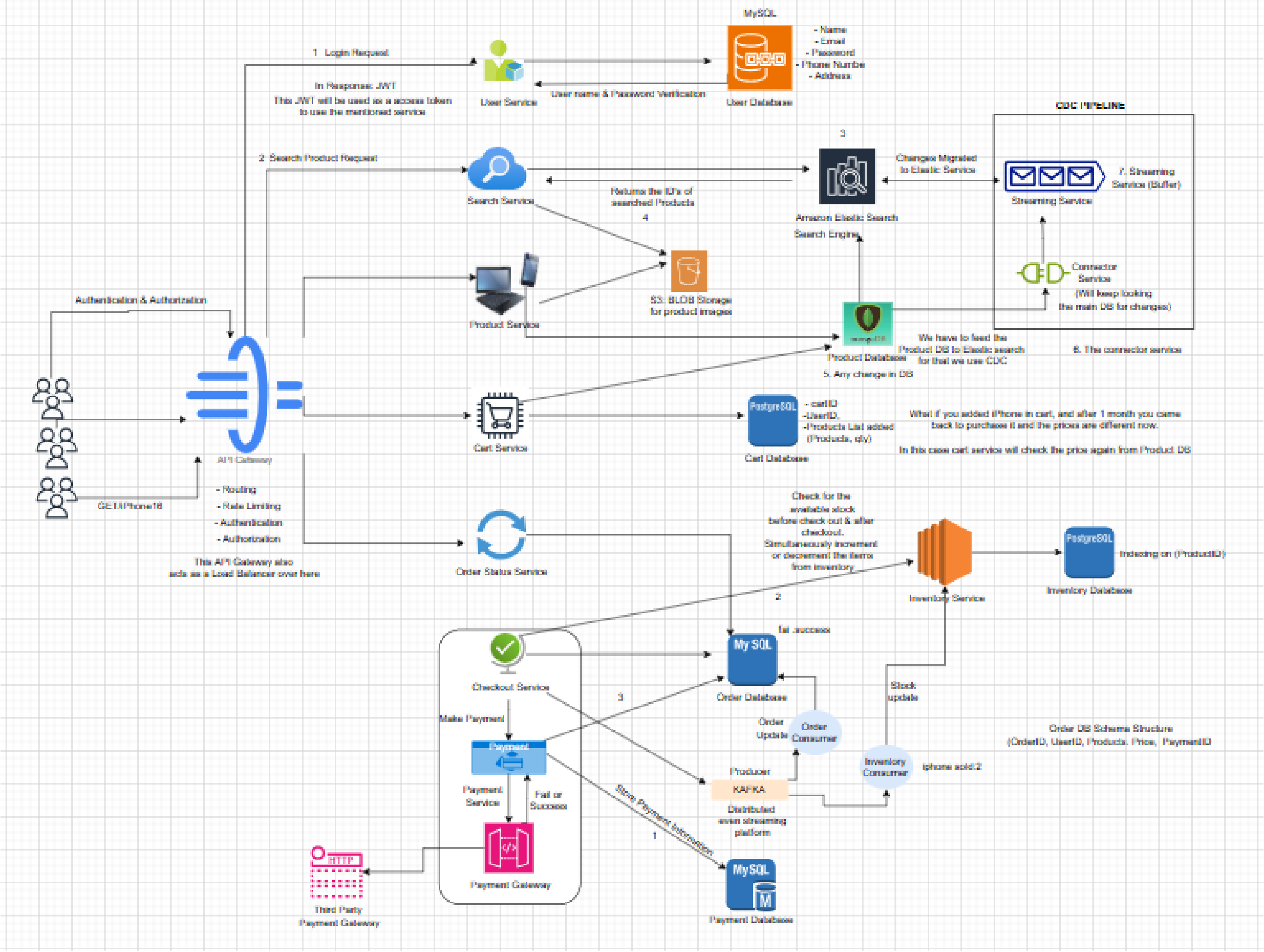
Core-entities of System

- User/Client
- Products
- Cart
- Orders
- Checkout followed by Payment

Outcome / Result -

- A secure and scalable cloud-based system architecture was successfully designed
- High availability was achieved through load balancing and auto scaling
- Network isolation was implemented using public and private subnets
- Sensitive resources were protected by deploying them in private subnets
- Monitoring and logging enabled effective system performance tracking
- The design follows standard cloud and system design best practices.

Required System Design –


MySQL
- Name
- Email
- Password
- Phone Numbe
- Address
1 Login Request
In Response: JWT
This JWT will be used as a access token
to use the mentioned service
User Service
User name & Password Verification
User Database
CDC PIPELINE
3
2 Search Product Request
Search Service
Returns the ID's of
searched Products
4
Amazon Elastic Search
Search Engine
Changes Migrated
to Elastic Service
Streaming Service
7. Streaming
Service (Buffer)
Connector
Service
(Will keep looking
the main DB for changes)
Authentication & Authorization
Product Service
S3: BLOB Storage
for product images
mongoDB
Product Database
We have to feed the
Product DB to Elastic search
for that we use CDC
6. The connector service
5. Any change in DB
API Gateway
Cart Service
PostgreSQL
- cartID
-UserID,
-Products List added
(Products, qty)
Cart Database
What if you added iPhone in cart, and after 1 month you came
back to purchase it and the prices are different now.
In this case cart service will check the price again from Product DB
GET:iPhone16
- Routing
- Rate Limiting
- Authentication
- Authorization
This API Gateway also
acts as a Load Balancer over here
Order Status Service
Check for the
available stock
before check out & after
checkout.
Simultaneously increment
or decrement the items
from inventory
2
Inventory Service
PostgreSQL
Inventory Database
Indexing on (ProductID)
fail ,success
Checkout Service
My SQL
Order Database
3
Make Payment
Payment
Payment
Service
Fail or
Success
Order
Update
Order
Consumer
Stock
update
Order DB Schema Structure
(OrderID, UserID, Products, Price, PaymentID
Producer
KAFKA
Distributed
even streaming
platform
Inventory
Consumer
iphone sold:2
Store Payment Information
1
HTTP
Third Party
Payment Gateway
Payment Gateway
MySQL
Payment Database