**Non-Functional Requirements –**

1. Performance/Latency –
   1. Product search results should be available within 3 seconds.
   2. No page should take more than 5 seconds to load.
   3. Payment gateway should have seamless integration and redirecting to & from payment gateway should not take more than 60 seconds.
2. Security –
   1. Data in transit should be encrypted, https should be used for frontend to backend communication.
   2. Access should be restricted i.e. least access should be allowed.
   3. The vulnerability security testing should be done on the system and the system should not have any high or medium vulnerabilities.
3. Availability –
   1. The system should have an availability of 99.99%.
4. Privacy –
   1. PI should be stored only after taking consent from user and should be kept secure and private.
5. Usability-
   1. The product buying flow should be consistent across all categories and offers.
   2. The interface should be user-friendly, with all the key components easily accessible. Users should easily be able to navigate to different components.
6. Scalability –
   1. The system should be able to handle millions of concurrent users.

**Functional Requirements –**

1. The system must show a list of products on the home page.
2. The system must support searching for a specific product & show relevant products based on search.
3. The system must support filtering of the products based on price, brand, color etc.
4. The system must support user registration, user login and user profile.
5. The system must maintain an inventory of products and display availability/unavailability of products based on inventory.
6. The system must keep the inventory up to date with product availability.
7. The system must maintain the price of the product and show any sale discount on the product
8. The system must allow users to add products to Cart.
9. The Cart must be easily accessible and display all the products added along with their details, price, discounts and total cart value.
10. The system must allow a discount coupon to be applied and provide a discount on total price if applicable.
11. The system must allow carts to be checked out and orders to be created.
12. The system must have integration with third party gateways for payment.
13. The system must display the correct order status in case of payment success/failure.
14. The system must notify users of the correct order status via email.
15. The system must have a way to track orders.
16. The system must have a place to display order history.
17. The system must monitor and alert any failures.
18. The system must have a way to add/update/remove products.

**Services -**

1. UI – ecommerce site
2. Search Service – Search for the items
3. Product Service – gets the items information for homepage based on user preferences, sales etc.
4. Order Service – Keeps track of Orders & history
5. Inventory Service – keeps items and their inventory information.
6. Cart Service – maintains user cart
7. Discount Service- product discounts
8. Payment Service - order payment
9. Shipping Service – order shipment status
10. User Service – user login, registration, user information.
11. Notification Service – email order status
12. Billing Service – generate invoice

**Why Microservices -**

1. **Individual Scalability** – I want my services to be able to scale independently. Statistically 53% of users leave the site if the search takes more than 3 seconds. So, my search service must be more scalable and provide minimum latency compared to let’s say my order service.
2. **Targeted DB for targeted problem** – I want to be able to use my data storage as effectively as possible. I am using KV stores for my Cart, SQL for my orders, and Document stores for my products.

**Component Details –**

1. **Ecom Site –**

This will be the user interface. It will contain a sign in/registration, a category navigation bar, a search bar, a product overview, a product detail, a user profile, a cart, an order history and an admin section.

1. **Product Service –**

This service will have all product-related operations. Get product list with basic product details, product details for specific product, add product, update product, remove product. The product service will also handle product pricing, discounts etc. It will communicate with Inventory Service to check and update product stock availability/ unavailability. It will also communicate with search services to ingest product details and keep products updated in search service.

1. **Search Service –**

This service will facilitate product search. It will communicate with the product service, get product information and keep it updated. It will handle user searches, create indexes to provide quick search results.

1. **Inventory Service –**

Inventory service will keep the product stock information and communicate with product service to update the inventory of any product added or removed. It will communicate with the Order service to reduce the stock if a product is ordered. It will provide information about the product’s availability/unavailability and when it is expected to be back in stock if the product is unavailable.

1. **Cart Service –**

This will have user cart information and keep it updated to date if any product is added, updated, removed from the cart or if the cart is checked out and order placed.

1. **Discount Service –**

This service will be applicable for handling and providing discounts if applicable.

1. **Order Service –**

This will create /cancel/ update the order details, maintain & provide order history and other order related information.

1. **Payment Service –**

This service will handle all payment related stuff, payment options available, transaction details, payment status, refund initiation and will keep all sensitive information if any secure.

1. **Shipping Service –**

This service will be responsible for shipping, getting delivery cost and tracking of the shipment.

1. **User Service –**

This service will have all the user-related operations. The user login, user registration, user profile related operations.

1. **Notification Service –**

This service will be responsible for sending order notification(email) to users.

1. **Billing Service –**

This service is responsible for generating invoices & storing it to blob storage for user access.

**Cross cutting concerns-**

1. **Authentication & Authorization –**

* User management using Cognito.
* User login & JWT generation by Cognito
* JWT in the API call
* API Gateway validate JWT using Cognito
* Microservices validate JWT using Cognito, also checks relevant permission.
* An auth package which interacts with Cognito, validates JWT, performs permission validation.
* The package is used in all microservice require auth.

1. **Logging –**

* Filebeat as a Deamon set
* Will be deployed on each node
* Can be configured to collect container logs
* From services, write just standard output and it will be written to container logs
* Configure Filebeat to send container log to OpenSearch
* Create OpenSearch Dashboard to visualize

1. **Retries & Circuit Breakers –**

* This will be addressed in each service by using some libraries and providing the required parameters & thresholds.

1. **Sensitive Configurations –**

* Sensitive information like DB connection strings etc. should be kept at some centralized safe location (like parameter store for AWS) and used dynamically from there.

1. **Inter Service Communication –**

* Asynchronous message based; each service involved in inter service communication has its own dedicated Queue which it polls. Other services which want to communicate post the message in queue, the service reads & processes it. If the event is meant to be used by multiple services, it will be published in a Topic, all other service which needs to perform some action on the event will subscribe to that event via their queues and process the message accordingly.
* GRPC for checking & Locking inventory.

1. **Caching –**

* CDN from edge location for static content
* API gateway caching for products, search, cart
* TTL for cache refresh
* AWS Elastic Cache for microservice caching.
* Write through Cache, data updated in DB, updated in cache.
* Cache-aside - Data loaded in cache on demand, when cache miss
* Multi AZ deployment

1. **Rate limiting –**

* AWS WAF at CloudFront, API Gateway, ALB

**No SQL –**

1. Your data is unstructured or semi-structured.
2. You need to handle massive amounts of data with horizontal scalability.
3. Your application has high write throughput and requires low-latency access.
4. You need flexibility in schema design and data models.
5. Your application can tolerate eventual consistency.
6. You need distributed architecture with high availability and fault tolerance.
7. Your app processes real-time analytics or big data.

**Service Details –**

1. Product Service –

* Responsibilities –
* Product catalogue will be fetched.
* Product CRED operations
* Publish product related events to product queue
* DB Choice - Document NoSQL – MongoDB Atlas – NoSQL 1, 2, 3, 4, 5, 6
* I need a flexible schema for my products as a variety of products can have different information. I am designing my product DB to provide AP (high availability & partition tolerance) which is offered by NoSQL dbs like MongoDB, High read/writes, no complex queries.

1. Search Service ––

* Responsibilities –
  + Product search results.
  + Subscribe to product related events from product queue.
  + Sync search DB with product DB.
* DB Choice - Elasticsearch – AWS OpenSearch
  + Indexes, fast search, complex query, scaling

1. Order Service –

* Responsibilities –
  + Check & lock inventory via grpc call to inventory service
  + Create order
  + Publish order events to Order Topic
  + Listens for payment events in Order Queue.
  + Returns Payment URL to frontend
  + Update order based on payment events.
* DB Choice - SQL - PostgreSQL
  + Fixed Schema, ACID, relations, consistency

1. Inventory Service – SQL

* Responsibilities –
  + CRED stock
  + Lock stock, handle grpc request from order service, check & lock stock
  + Listens to Inventory Queue which subscribes to Order topic.
  + Update inventory based on Order status.
* DB Choice - SQL - PostgreSQL
  + Fixed Schema, ACID, relations, consistency

1. Cart Service – KV store – Amazon ElastiCache (Redis)

* Responsibilities –
  + CRED Cart
  + Listens to Cart Queue which subscribes to Order topic.
  + Clear cart once order is created
* DB Choice - NoSQL (KV store – In memory) – Amazon ElastiCache
  + High Speed due to in memory, flexibility, key value pair, temporary data

1. Discount Service – KV store - DynamoDB

* Responsibilities –
  + Manage discount coupon
  + Apply discounts
* DB Choice - NoSQL (KV store - (persistent)) – Amazon DynamoDB
  + Scalable & Fast, millisecond latency
  + Flexible data model
  + Highly Available
  + Low cost

1. Payment Service – SQL - Amazon RDS - PostgreSQL

ACID, security, encryption, transaction, keeping sensitive information secure, fixed information

1. Shipping Service – SQL -- Amazon RDS - PostgreSQL
2. User Service – SQL - -- Amazon RDS - PostgreSQL

Fixed schema, security, encryption

**Design Thoughts–**

1. Architecture –

Microservice

1. Deployment –

* Frontend – AWS S3 with AWS Cloud Front Distribution
* Backend – AWS EKS
* Frontend -Backend Communication Entry Point– AWS API Gateway
* API Gateway- Microservice Communication Entry Point – Ingress Controller which creates a single ALB and routes request to appropriate service based on rules
* Inter Service Communication Synchronous – https/rest/grpc using Kubernetes ClusterIP Service
* Inter Service Communication Asynchronous – event driven microservices publishing event to Topics & Queues and microservices polling queues for new events/messages🡪microservices reading queues & processing messages

1. Communication Flow –
   * External –
     + Website 🡪 AWS API Gateway 🡪 Ingress Controller ALB🡪 AWS EKS ClusterIP Service🡪 Microservice
   * Internal Synchronous –
     + Microservice A🡪 ClusterIP Microservice B🡪 Microservice B
   * Internal Asynchronous –
     + Microservice A🡪Publish message to Topic 🡪 Microservice B Queue subscribe to topic🡪Microservice B poll the B Queue🡪 Microservice B Read Queue and Process message
     + Microservice A🡪Publish message to B Queue 🡪 Microservice B poll the B Queue🡪 Microservice B Read Queue and Process message

**NFRs Implementation –**

1. Security –

* Website (S3, CloudFront) –
  + Encryption for data in transit using Https/TLS enforcement in CloudFront
  + Encryption for data in rest at blob storage (S3 has encryption using SSE /KMS)
  + Restrict access by IAM policies
  + Restrict S3 access by only allowing via using CloudFront origin access identity
  + Signed URL for private objects like invoices
  + Attach AWS WAF to CloudFront – create web ACL with DDOS, XSS rules
* API Gateway-
  + Encryption using https
  + Authenticate request
  + AWS WAF - Rate limiting, SQL injection
* AWS ALB(Ingress)-
  + Private within VPC.
  + Access allowed only from API gateway via Private link
  + AWS WAF
* Microservices-
  + Deployed in private subnet
  + Exposed only within cluster using ClusterIP
  + Auth & Authz – validate JWT and permissions
  + Sensitive info in AWS SSM
  + AWS WAF – rate limit, SQL injection rules

1. Availability –
   1. Website (S3, CloudFront) –
      1. Cloud managed, highly available
      2. Cross region replication for DR
   2. API Gateway-
      1. Cloud Managed, highly available
      2. Cross region replication for DR
   3. AWS ALB(Ingress)-
      1. Spans multiple AZ.
      2. Cloud Managed, highly available
   4. EKS Microservices-
      1. Multi AZ deployment for microservices
      2. HPA
   5. Database-??
2. Disaster Recovery –
   1. Website, API Gateway - Multi Region deployment, Active – Passive, for static content & API gateway automatic switch using Route 53 health check.
   2. EKS cluster -API gateway in replication region will become active and redirect traffic.
   3. Database--?
3. Scalability –
   1. CloudFront–
      1. Cloud managed, auto scale
      2. Cross region replication for DR
   2. API Gateway-
      1. Cloud Managed, Auto Scale
      2. Rate limit to prevent overload
   3. EKS Microservices-
      1. Cluster auto scaler to add/remove nodes based on load
      2. HPA for pods
   4. Databases – ?

**Cost Effectiveness –**

* S3, CloudFront – Managed service, only storage and access cost, CloudFront serves data from edge locations reducing data transfer costs. No infra cost.
* API Gateway – Managed Service, no infrastructure cost, pay per request
* API Gateway – Cost per active users
* Caching – reduces expensive database reads
* Logging – Open source Filebeat, no license cost
* OpenSearch – No infra, only storage & compute cost
* EKS Cluster- auto scalers for node & pod based on load
* SNS/SQS – cost only for message pushed, received, event bridge also cost for filtering routing etc.
* DR – Active Passive, ready to switch in minimum downtime but idle so incurring minimum cost.

CAP –

Document DB for high availability & partition tolerance as eventual consistency is acceptable.

**Disaster Recovery Plan for Databases –**

**1. Product Service (MongoDB Atlas)**

* **Primary Region**: MongoDB Atlas is deployed in your primary AWS region with cross-region replication enabled. MongoDB Atlas provides automated backups and point-in-time recovery.
* **DR Region**: In case of a disaster, the replica region will automatically promote the secondary replica set to the primary. This ensures zero data loss as MongoDB Atlas uses synchronous replication.
* **Private Connectivity**: MongoDB Atlas can be accessed from your private VPC through VPC peering. The communication between your Product Service (running on EKS) and MongoDB Atlas will happen over private IPs using VPC peering.
* **Failover Process**:
  + If the primary region is down, the traffic is routed to the secondary region.
  + MongoDB Atlas will automatically failover to the replica in the secondary region without any manual intervention.

**2. Search Service (AWS OpenSearch)**

* **Primary Region**: AWS OpenSearch is deployed in the primary region with cross-region replication enabled.
* **DR Region**: In the event of a region failure, AWS OpenSearch will promote the secondary domain in the DR region to be the primary. OpenSearch’s cross-cluster replication (CCR) ensures data consistency across regions.
* **Private Connectivity**: OpenSearch should be deployed in the same VPC as your EKS cluster, and security groups will allow communication from only the Search Service to OpenSearch.
* **Failover Process**:
  + In case of a region failure, the Route 53 health checks will detect the failure and reroute traffic to the secondary region where OpenSearch is running.
  + Ensure your application points to a Route 53 private hosted zone with the OpenSearch endpoint, so that in a failover scenario, traffic is redirected to the healthy region.

**3. Inventory Service (Amazon RDS - PostgreSQL)**

* **Primary Region**: Amazon RDS (PostgreSQL) is deployed with Multi-AZ replication for high availability.
* **DR Region**: Cross-region read replicas are configured in the DR region. AWS provides a managed failover process for RDS, and read replicas can be promoted to primary in the event of a failover.
* **Private Connectivity**: The RDS instance will be in the same VPC as your EKS cluster, with private subnets used for RDS connectivity. Security groups will be configured to allow communication only from your Inventory Service.
* **Failover Process**:
  + In case of a disaster, RDS will handle failover automatically if the primary instance goes down. If you're using cross-region replication, traffic is redirected to the replica in the DR region.
  + Route 53 health checks will monitor the RDS primary endpoint, and in case of failure, it will reroute traffic to the DR region.

**4. Cart Service (Amazon ElastiCache - Redis)**

* **Primary Region**: Amazon ElastiCache is deployed with replication and multi-AZ support in the primary region.
* **DR Region**: Using **Global Datastore for Redis**, cross-region replication is enabled. This allows automatic replication of data to the DR region.
* **Private Connectivity**: ElastiCache is deployed in the VPC, and your Cart Service running on EKS can access it using VPC endpoints.
* **Failover Process**:
  + ElastiCache automatically detects failures and promotes a replica to the primary in the event of a failover.
  + Route 53 health checks will monitor the ElastiCache endpoint, and traffic will be routed to the DR region if the primary region is unavailable.
  + In case of a failure, the Cart Service will point to the new endpoint provided by Route 53 for the DR region.

**5. Order Service (Amazon RDS - PostgreSQL)**

* **Primary Region**: Amazon RDS (PostgreSQL) is deployed with Multi-AZ replication for high availability.
* **DR Region**: Cross-region read replicas are set up, similar to the Inventory Service.
* **Private Connectivity**: The RDS instance is in the same VPC as your EKS cluster, with private subnets ensuring all traffic remains private. Security groups will restrict access to only authorized services.
* **Failover Process**:
  + Like the Inventory Service, RDS will failover automatically, promoting the read replica to primary in the event of failure.
  + Route 53 health checks will monitor the RDS instance, and traffic will be rerouted to the DR region in case of failure.

**Key Considerations:**

* **Cross-region replication** for databases like MongoDB Atlas and RDS (PostgreSQL) ensures that data is available in the event of a regional failure.
* **Private connectivity** using VPC peering and security groups ensures that no database is publicly exposed.
* **Failover** is automatic for databases with multi-AZ support, and cross-region replication ensures that your DR region has an up-to-date copy of your data.
* **Route 53 health checks** will redirect traffic to the secondary region in case of a regional failure.

**Summary of DR for Databases:**

1. **MongoDB Atlas**: Automatic failover with cross-region replication. No public exposure.
2. **OpenSearch**: Cross-cluster replication ensures DR. Route 53 health checks will reroute traffic.
3. **PostgreSQL (RDS)**: Multi-AZ for primary region and cross-region replication for DR. Automatic failover and Route 53 health checks.
4. **ElastiCache (Redis)**: Global Datastore for Redis enables cross-region replication. Failover managed by ElastiCache with Route 53 health checks.

Here’s the complete list of your microservices along with their **finalized databases** and a brief justification for each choice.

**📝 Finalized Services and Databases**

| **Service** | **Database** | **Justification** |
| --- | --- | --- |
| **1. Product Service** | **MongoDB Atlas** | Flexible schema, document-oriented, supports high availability, global clusters, and cross-region replication. |
| **2. Search Service** | **Amazon OpenSearch** | Optimized for search functionality, supports indexing, querying, and real-time search. |
| **3. Inventory Service** | **Amazon RDS (PostgreSQL)** | Relational data, requires ACID compliance, supports consistency and complex queries. |
| **4. Cart Service** | **Amazon ElastiCache (Redis)** | In-memory key-value store for fast access, supports high availability and low-latency data retrieval. |
| **5. Order Service** | **Amazon RDS (PostgreSQL)** | ACID compliance for order consistency, relational model for complex transactions. |
| **6. Payment Service** | **Amazon RDS (PostgreSQL)** | Consistent, durable, and secure storage for transactions and payment records. |
| **7. Shipping Service** | **Amazon RDS (PostgreSQL)** | Structured shipping data, supports ACID transactions, and relational queries for tracking. |
| **8. User Service** | **Amazon RDS (PostgreSQL)** | Structured user data, ACID compliance for profile and authentication operations. |
| **9. Discount Service** | **Amazon DynamoDB** | Simple key-value operations, highly scalable, low-latency access for discount validation. |

**🚀 Key Points**

1. **MongoDB Atlas**:
   * Chosen for **Product Service** due to its flexibility, high availability, and global distribution capabilities.
2. **Amazon OpenSearch**:
   * Selected for **Search Service** for efficient indexing and real-time search performance.
3. **Amazon RDS (PostgreSQL)**:
   * Preferred for services requiring **structured data, ACID transactions, and relational queries**:
     + **Inventory Service**
     + **Order Service**
     + **Payment Service**
     + **Shipping Service**
     + **User Service**
4. **Amazon ElastiCache (Redis)**:
   * Ideal for **Cart Service** due to its need for **low-latency, in-memory storage**.
5. **Amazon DynamoDB**:
   * Chosen for **Discount Service** for its **high scalability and simple key-value access patterns**.

**🌐 Cross-Region Replication and DR Considerations**

* **Cross-Region Replication** is enabled where available for all databases to ensure **disaster recovery (DR)** readiness.
* **Private Access**: All databases are deployed within **private subnets** of your VPC, ensuring no public exposure.
* **Automatic Failover**:
  + **MongoDB Atlas** and **Amazon ElastiCache Global Datastore** support automatic failover.
  + **RDS and OpenSearch** require **manual promotion** of replicas during cross-region failover.

This setup ensures your e-commerce architecture is **highly available, scalable, and resilient**.