Microservices Architecture for Online Marketplace Platform Design

Architecture Overview

1. **User Service**
    - **Responsibilities:** Manages user accounts, authentication and authorization with paseto, and user profiles.
    - **Authentication & Authorization:**
        - **Token-based Authentication:** Implement PASETO (Platform-Agnostic Security Tokens) for secure authentication and authorization.
            - **PASETO Tokens:** Use PASETO for generating and validating authentication tokens, ensuring secure and tamper-proof tokens that are easy to validate.
            - **Implementation:** Generate a PASETO token upon user login, include relevant user information and roles, and validate the token for each request to ensure the user is authorized.
    - **Database:** MongoDB
        - **Justification:** MongoDB's schema-less design allows flexibility for user profiles and can handle varying user data structures efficiently. Its scalability(horizontal scaling like sharding) is ideal for handling high read and write loads, great for ecommerce
    - **Scaling Considerations:**
        - **Horizontal Scaling:** Use sharding to distribute user data across multiple servers.
        - **Load Balancing:** Deploy multiple instances behind a load balancer to distribute incoming requests.
        - **Service Discovery:** Implement service discovery with a tool like Kubernetes' built-in service discovery.
2. **Product Service**
    - **Responsibilities:** Manages the product catalog, including product listings, categories, and search functionality.
    - **Database:** MongoDB
        - **Justification:** MongoDB is well-suited for handling large volumes of product data with complex queries and indexing. It supports fast searches and can manage unstructured product attributes.
    - **Scaling Considerations:**
        - **Horizontal Scaling:** Use sharding and replica sets to handle large datasets and ensure high availability.
        - **Load Balancing:** Deploy multiple instances behind a load balancer.
        - **Service Discovery:** Use Kubernetes' service discovery.
3. **Order Service**
    - **Responsibilities:** Handles order creation, updates, tracking, and history.
    - **Database:** MongoDB

        - **Justification:** MongoDB provides high write throughput and is efficient in managing transaction logs and order histories. It supports flexible schemas, which is beneficial for evolving order data.
    - **Scaling Considerations:**
        - **Horizontal Scaling:** Use sharding for distributing orders and replica sets for redundancy.
        - **Load Balancing:** Utilize load balancers to manage traffic to multiple service instances.
        - **Service Discovery:** Implement service discovery with Kubernetes.

CI/CD Pipeline

1. **Version Control:**
    - **Tool:** GitHub
    - **Setup:** Repositories for each microservice with branch protection rules and pull request workflows.
2. **Continuous Integration (CI):**
    - **Tool:** GitHub Actions
    - **Configuration:** Set up workflows to build and test each microservice on every push or pull request. Include linting, unit tests, and integration tests. Best part they are all automated
3. **Continuous Deployment (CD):**
    - **Tool:** GitHub Actions
    - **Configuration:** Define workflows to build Docker images, push them to a container registry like Dockerhub while tagging them appropriately , and deploy to Kubernetes clusters.
4. **Infrastructure Management:**
    - **Tool:** Kubernetes
    - **Configuration:** Use Helm charts for managing deployments and services. Implement autoscaling and rolling updates for handling scaling and deployments.
5. **Cloud Platform:**
    - **Recommendation:** AWS
    - **Services:** Use managed Kubernetes services (EKS on AWS) for deploying microservices, managed MongoDB services (Atlas on AWS or GCP) for databases, and cloud-based load balancers for distributing traffic.most cloud platforms have developed means to handle kubernetes elegantly.

Summary

This microservices architecture incorporates PASETO for secure authentication and authorization in the User Service, ensuring robust and scalable security measures. MongoDB is used across services for its performance and flexibility. The CI/CD pipeline with GitHub Actions and Kubernetes provides an automated and reliable deployment process, with cloud services enhancing infrastructure management.