I'm splitting the online marketplace platform into four microservices. Every microservice concentrates on one business function. To manage microservices I propose using Kubernetes because it is an effective tool for managing, scaling, and testing. Kubernetes has automatic scaling and load balancing which can improve the functioning of the production environment.

Services

- User service - authorization users, handling account operations.
- Product service - managing product catalog and getting product list,
- Order service - storing information about order
- Delivery service - storing the history of delivery

Databases:

- SQL databases for user service, order service, and product service because they have a relational structure. Every table has defined schemas and every row has the same structure. This way delivers data consistency on account operations and orders.
- NoSQL databases for product service and delivery service. First, most users spend time looking for items in the marketplace. NoSQL server is easy to horizontal scaling (add new server) and works with multiple instances, without replication. Delivery service tracks every detail about the package and the status of delivery.

Scaling:

Horizontal scaling every service can be scaled by creating new instances.
When we create new microservices instances, we need more databases to work properly.
We can use database sharding:

- Splitting data into smaller databases by the value range or key
- Independence, every shard works independently and requests can be processed at the same time

Replications, we can create a replicated database, the main database saves new data, and replicas support read queries.
Every microservice is run in a container created using Docker. Containers are light and portable units, it's working using virtualization - system kernel is shared between containers but containers are independent. We can use this to run applications in an isolated environment. For clarity and management of containers, we can use Kubernetes. It's a tool for automation orchestration containers. We have a few main improvements:

- Automatically scaling - The application is scaled in real time when the load is changing
- state management - When the application crashes, is automatically restored
- Load balancing - is automatically distributed by containers
- Continuous deployment - use Kubernetes support to implement new versions of applications and minimize downtime.

Pipeline CI/CD

Pipeline CI/CD automates integration and implementation of microservices providing fast and reliable delivering new functions and updates.

1. Code repository - Use Github or Gitlab for code management
2. CI:
    a. Building in every commit for microservice using Docker, every microservice has a defined Dockerfile.
    b. Automatically running unit tests and integration tests
3. Containerization when the Build is passed Docker Image is generating
4. Integration
    a. Automatic delivery image to staging to next tests
5. Continuous Delivery:
    a. Implementation in production: If every test is passed solution is automatically delivery in production
    b. Rollback - Preparing rollback strategy to come back to the latest version, if problems occur after implementation.