QBurst

Building a

Centralized Microservices-based E-commerce Platform

for a Global Fashion Retail Chain

Overview

The project involved a comprehensive transformation of a leading global fashion retail company's E-commerce (EC) platform, which previously operated on disparate third-party systems across various countries. The key challenges included poor vendor control, high licensing and change request costs, performance issues, and lack of timely rollouts. QBurst partnered with the client to build a centralized, microservices-based platform, streamlining processes, improving performance, and significantly reducing costs.

Our solution focused on transitioning from third-party systems such as Demandware and Magento to an in-house platform, optimizing the catalog, Order Management Systems (OMS), and ensuring robust integrations with other critical business systems like Inventory Management System (IMS) and Warehouse Management Systems (WMS). The microservices-based architecture allowed the platform to scale effortlessly as business demands grew, supporting multiple countries from a single, centralized system.

Client Profile

A leading fashion retail company that owns several brands and operates across diverse markets.

Business Challenges

The client relied on different e-commerce solutions in each country. This led to inconsistent customer experiences, increased maintenance efforts, and operational challenges.

- **Limited vendor control:** The client lacked control over third-party vendors managing the applications, limiting flexibility and responsiveness to business needs.
- **High license and change costs:** The cost of maintaining third-party platforms was significant. Even small change requests (CRs) were expensive and time-consuming.
- **Delayed rollouts:** New features and bug fixes were not rolled out on time, leading to lost revenue opportunities.
- **Data ownership:** The client did not have full ownership of their e-commerce data, particularly customer and order data, which was managed by third-party vendors.
- **Performance and scalability:** The existing systems struggled to handle peak traffic during major events, causing frequent downtimes and poor customer experiences.

QBurst Solution

We designed and implemented a microservices-based architecture that enabled the client to take control of their e-commerce platform. We helped transition from third-party systems to an in-house solution, moving critical operations such as account management, catalog, and order management to microservices that could scale seamlessly across multiple countries.

Solution Highlights

1. Microservices architecture

We built a robust microservices architecture to replace the disparate systems. Key microservices included:

- **Account microservice:** Centralized account management for multiple countries.
- **Catalog microservice:** A critical API for retrieving product data and inventory information.

- **OMS microservice:** Handled orders and integrated with the client's inventory and warehouse management systems.

These microservices were designed to be modular, scalable, and easily integrated with both internal and third-party systems.

2. Data migration

- **Account data migration:** We migrated account data from third party systems using a streamlined process:
  - Data was exported in XML format, compressed into .gz files, and stored in an AWS S3 bucket.
  - The migration process involved converting the XML files to a database format and performing integrity checks before moving the data to the new account system.
- **Order data migration:** We implemented a two-tier data migration process for order history:
  - Recent data was migrated to a PostgreSQL database (hot storage) for quick access, while older data remained in a legacy datastore (cold storage).
  - A Kafka migrator continuously transferred legacy data to the PostgreSQL database, ensuring real-time access to historical order data.

3. OMS integration

The OMS was integrated with the client's IMS and WMS using APIs and ETL (Extract, Transform, Load) processes for real-time data synchronization. We also implemented asynchronous processing for OMS batch jobs to enhance performance.

4. Catalog performance enhancements

The catalog platform was heavily optimized for performance, given its high volume of API requests:

- **In-memory caching:** Master data, which changes infrequently, was stored in in-memory caches (Redis), avoiding unnecessary database joins.
- **API caching:** API responses were cached to minimize database queries. During peak traffic hours, the caching time was dynamically adjusted to handle large volumes of requests.
- **Concurrent queries:** API queries were executed concurrently to speed up response times, and complex calculations were handled by batch workers to avoid delays in real-time API responses.
- **Denormalization:** Key data in the database was denormalized to improve access speed, further reducing query time.

5. Order data access optimization

We implemented a streamlined process for retrieving user order histories:

- When a user requests order history, the system first queries the PostgreSQL database for recent records.
- If data is insufficient, it queries the legacy datastore and merges the results.
- A Kafka pipeline continuously migrates older order history into PostgreSQL, progressively improving performance over time.

6. Performance enhancements in OMS

To improve the performance of the OMS during peak sales hours, several optimizations were made:

- **In-memory basket data store:** We migrated basket data from PostgreSQL to an in-memory solution, reducing data retrieval time from 2 seconds to under 10 ms during cart updates.
- **ElasticSearch integration:** Price data was migrated from PostgreSQL to ElasticSearch to optimize frequent price lookups, drastically improving response time and reducing database load.

- **Database query optimization:** Indexes were added to frequently queried fields, reducing query execution time from 200 ms to 20 ms, improving batch processing performance by 70%, and lowering peak CPU usage by 40%.
- **Database sharding and partitioning:** Implemented sharding to distribute data across multiple database nodes, improving system scalability and performance during high transaction volumes.
- **Asynchronous logging:** We refactored the logging system to be asynchronous, with failover support, reducing main process overhead and improving API response time by 30%.

7. Front-end optimization

- **Server-Side Rendering (SSR):** Pre-rendered pages on the server to improve load time, enhance SEO, and reduce delays for users on slow connections.
- **Lazy loading:** Implemented lazy loading for non-essential resources like images and videos, improving initial page load times.
- **Resource compression:** Utilized modern formats such as WebP to compress images and reduce file sizes, further enhancing performance.

Technologies

Business Benefits

- Reduced licensing costs and eliminated expensive change request fees by transitioning to an in-house platform, providing significant cost savings and minimizing vendor dependency.
- Gained full control over the e-commerce platform, enabling faster decision-making and real-time access to critical business data, improving operational efficiency and responsiveness.
- Achieved flexibility to quickly introduce new features and improvements, reducing delays in rolling out updates and capturing revenue opportunities more effectively.
- Delivered a consistent and superior shopping experience across all countries through enhanced platform optimizations like SSR, lazy loading, and resource compression, resulting in improved load time and smoother user experiences.
- Optimized system architecture resulted in faster response time, better handling during peak traffic, and a reduction in downtime, ensuring seamless scalability and improved performance during high-demand events.