Microservices-based
Digital eCommerce Platform

Developed Java Spring Boot microservices hosted on Kubernetes cluster as an orchestration layer for the web and mobile app.

QBurst

Project Overview

The digital eCommerce platform comprises a Single Page Application and a Mobile App with microservices architecture forming the backend. Microservices, based on Java Spring Boot were implemented for merchandising, costing, and inventory platforms. This provided the information for the frontend. REST API calls were used to communicate with each microservice and this was handled by the BFF layer (Backend for Frontend).

One of the main advantages of our code is that it can be easily configured for different regions and brands. Using the same codebase, we deployed the system in eight countries for three different brands. The code can also be customized with features specific to one country.

Performance testing revealed that our solution was able to handle twice the load when compared to the previous year. Being a global retail player, implementing Microservices resolved their biggest problems related to expansion and scaling of their IT systems, while bringing in flexibility and agility.

Client

Our client is one of Asia's largest clothing retailers with over 2500 stores across the globe.

Business Requirement

Our client used multiple third party systems for order management with individual deployments in regions such as Canada, China, and the United States. They faced several challenges with the existing system.

- Lack of scalability
- High dependency on vendors for implementation across geographies
- Costly outages during festive seasons

The client wanted to replace the existing system with an in-house, global order management system.

Solution

The microservices-based architecture provides the foundation for the digital eCommerce solution, comprising over 20 platforms that cater to all the functions of the EC store. Following is an overview of the main ones developed:

Order Fulfillment: Core, Basket, Payment, Inventory, Order History, Cost, and Sales

Frontend: CMS, SPA, Web and Mobile BFF

Customer: Account, Coupon, Newsletter, Analytics, Catalog

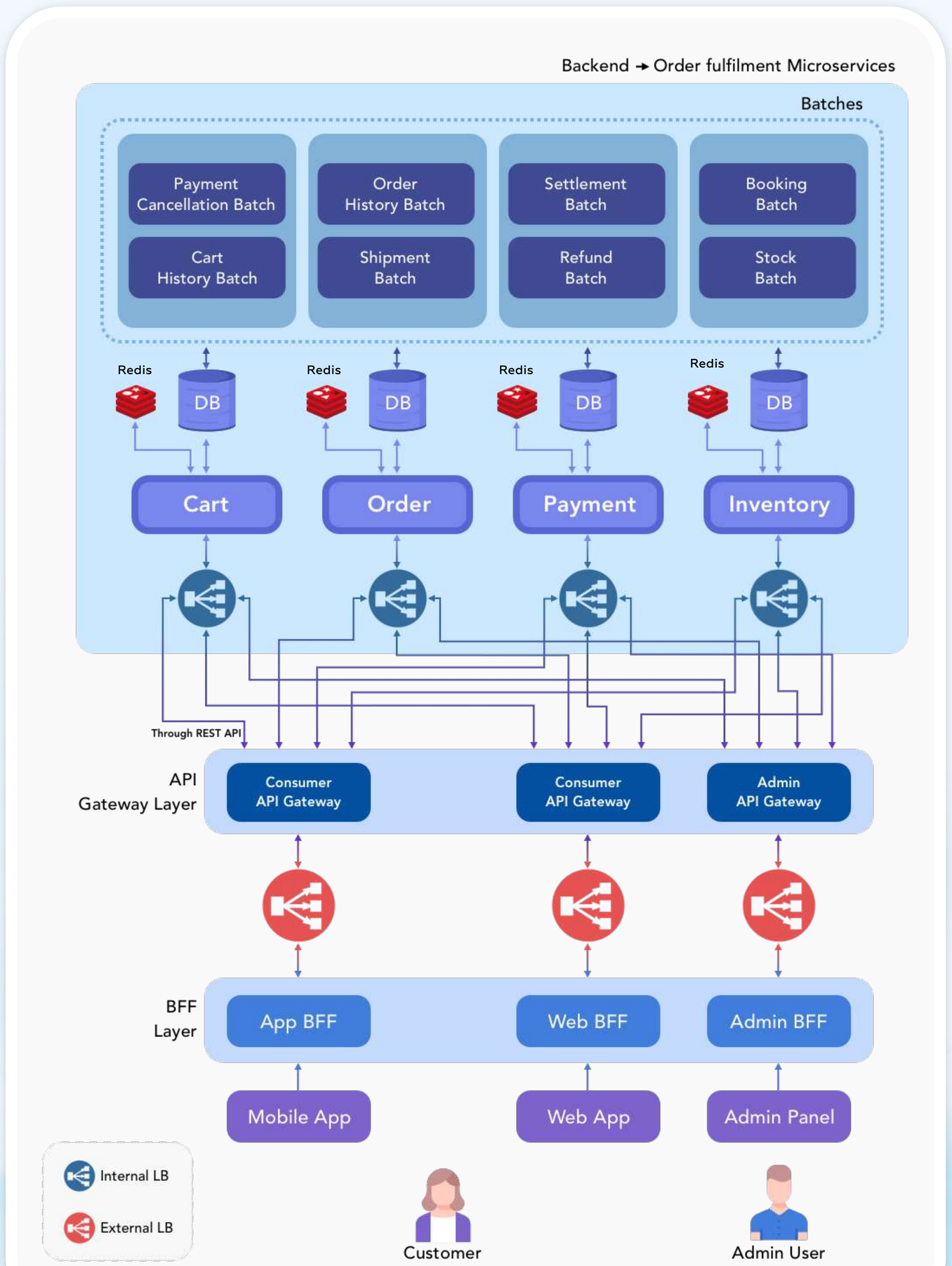
Store: Customer Support, Order Management, POS, and Reporting

The Order Fulfillment module consists of multiple microservices (built-in Spring Boot running on EKS) each having its own narrow and focused responsibilities:

- Cart microservice for eCart related functions
- Order microservice for order placement and returns
- Payment microservice for payment-related functions
- Inventory microservice for inventory and related functions
- Cost microservice for storing product pricing and related details
- Sales microservice manages the sales data and provides information that can be used for various analysis and accounting classification

Each microservice has its own database and specific job processing batches. Additionally, all microservices are interconnected with/dependent on many other microservices in the system and are connected through Rest APIs. Microservices associated with the previous platform had to be deprecated and integrated with the new system.

The following diagram portrays a high-level structure of the microservices and the communication flow with different user interfaces (mobile/web app, admin panel).



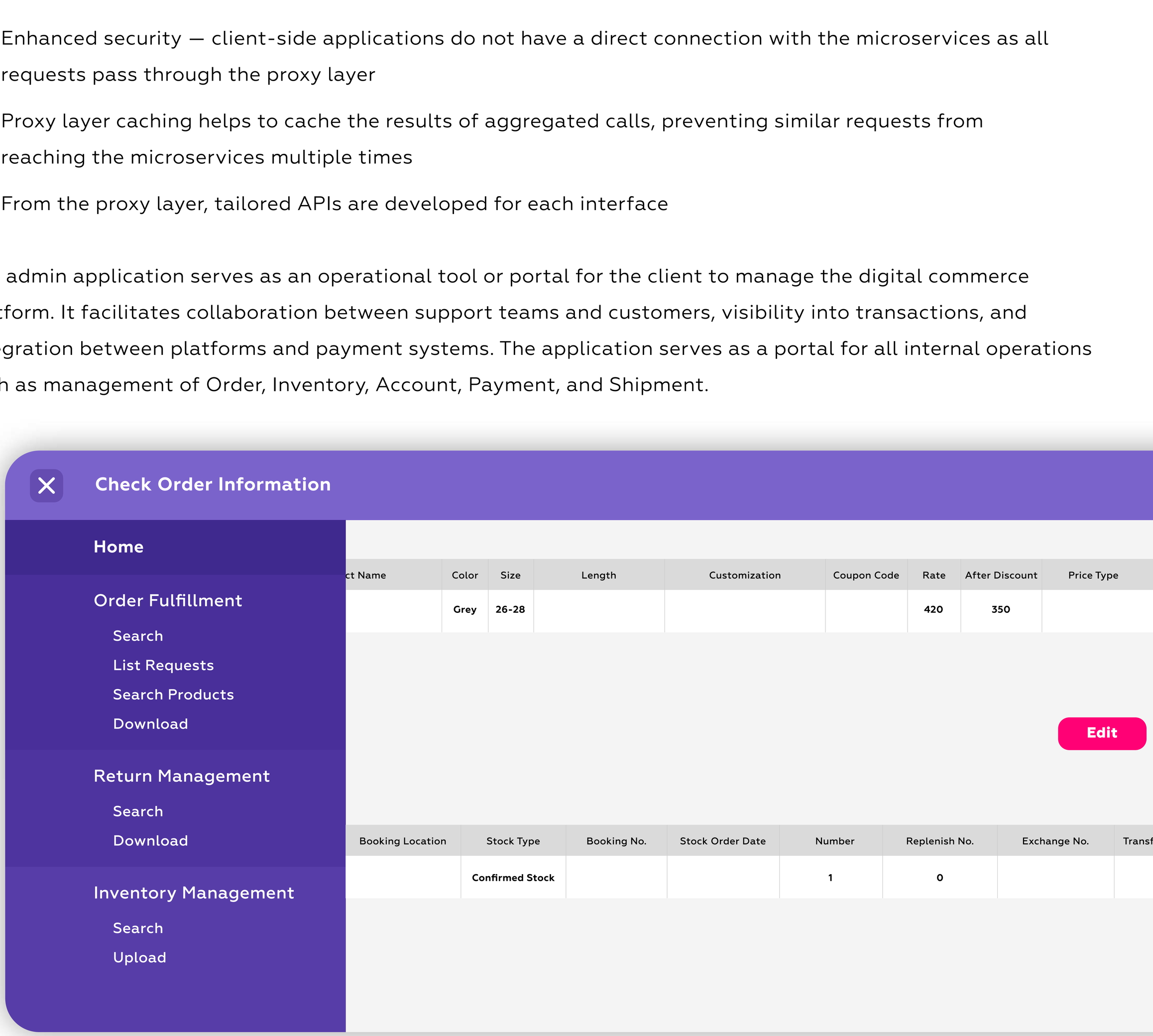
Load Balancers: There are internal and external load balancers available between the BFFs and microservices for efficient distribution of traffic. For external load balancing, separate load balancers are used for consumer and admin traffic.

API Gateway: APIs are channeled through a designated API gateway to ensure API authentication, monitoring, security, and sessions/logs. Separate API gateways for consumers and admin are used to channel requests from consumers and administrators.

All user interfaces are connected with these microservices through a proxy layer — BFF specific to each interface. The BFF layer orchestrates/merges internal APIs from multiple microservices. This layer ensures:

- Enhanced security — client-side applications do not have a direct connection with the microservices as all requests pass through the proxy layer
- Proxy layer caching helps to cache the results of aggregated calls, preventing similar requests from reaching the microservices multiple times
- From the proxy layer, tailored APIs are developed for each interface

The admin application serves as an operational tool or portal for the client to manage the digital commerce platform. It facilitates collaboration between support teams and customers, visibility into transactions, and integration between platforms and payment systems. The application serves as a portal for all internal operations such as management of Order, Inventory, Account, Payment, and Shipment.



The stock system is designed to manage eCommerce platforms, distribution centers, and stores. The system supports inventory inquiry, provisional inventory booking, real booking, and transfer operations. It interacts with the catalog system for item information and with warehouses to import stock information.

The core module caters to multiple warehouses while handling communications and emails to customers.

- Management of orders, settlement, shipment/receipt
- Return control
- Linkage with inventory, payment, sales, and accounting platforms

The progress of operations for each order received is centrally managed by the order fulfillment module. Order status can be checked by the client using the admin app and by customers through the eCommerce site.

The cost system enables calculation of the product prices by applying predefined business rules. It includes a data adapter, merchandise service for product definition, and tax lookup service.

Sales microservice manages the sales data that can be used for various analysis and accounting classification. It manages who, when, where, which brand, what, sales/returns/cancellations in the smallest unit that can be analyzed.

We developed the customer frontend as a Single Page Application using React as the JavaScript framework because of its flexibility and speed. SPA comprises the main application where we combine components such as buttons, links, and images as well as CSS together with business logic. Packages employed include Webpack, TypeScript, React-Router, Redux, and Redux-Saga. Different platforms, served by microservices, maintain their own application individually and provide the information to be displayed in the frontend interface. For example, the cost system stores prices of products, while merchandising system stores product details such as name, size, and color of the product.

Within the system, we implemented a rolling deployment strategy for our ECS clusters that scale to 200%. This means, for each new change, the system scales up 200%. Newly created containers are exposed to custom health checks, after which the old containers are removed. Along with this, we run the entire cluster on two sets of ASGs (Auto Scaling Groups). The ECS cluster (underlying hardware) is scaled out based on CPU requirements, after which, each ECS service (the software) is scaled out based on the performance of the system. This facilitates quick and automatic scaling during peak times while running at a bare minimum during low traffic.

Highlights

- Distributed caching layer (Redis) stores a copy of data fetched from the database and other microservices, reducing the number of expensive API/database calls, and improving performance
- Query optimization and replacement of certain libraries with those having lower processing costs improved response time
- Optimal use of in-memory data to avoid repetitive calls to database
- Asynchronous processing of orders ensures stability during spikes in demand
- Server-side rendering for a few pages helped cache data and display results faster

Technologies

- Java 11
- Maven
- PHP
- PostgreSQL
- Flyway
- Spring Boot
- Node
- Kubernetes
- Docker
- React/Redux-Saga
- Spring Batch
- Golang
- Redis
- Kafka

Benefits

- Performance testing revealed that the eCommerce system was able to handle twice the load compared to the previous year
- Ability to handle up to a million requests per hour
- Solution deployed across geographies by enabling features location-wise
- Sales of a seasonal product increased six times in a single day
- Increased control over inventory
- Significant cost savings with in-house solution when compared to the older system

QBurst

USA | UK | UAE | INDIA | SINGAPORE | AUSTRALIA | JAPAN

www.qburst.com | info@qburst.com