Cloud-powered Agility: A Case Study on event-driven Microservices Architecture for an emerging online Retail Business

Siwei Xie

University of Stuttgart
st165900@stud.uni-stuttgart.com

Keywords: Microservices · Containerization · Cloud Computing · AWS · Serverless · Event-driven · CQRS · Saga · Customer 360

This case study starts with a brief overview of the software products and services necessary for its establishment. Subsequently. In **Sec.2**, according to relevant industry standards, specifically the 6 Pillars of the AWS Well-Architected Framework[1], an analysis of the technical requirements crucial for developing the well-functioning software and achieving business goals is conducted. Based on these demands, **Sec.3** then involves assessing and proposing architectural frameworks and software patterns to determine the most suitable option. Following the analysis, a high-level deployment plan on the AWS cloud is outlined in next section, ensuring alignment with the adopted design in **Sec.3** and validating its feasibility. Through this structured approach, a clear pathway towards realizing business goals while ensuring conformity with industry standards is established.

1 Software Products and Services

- A blog that provides engaging information(a static website)
- A website that allows a client to browse a catalog, order goods and return goods
- Several services necessary for maintaining business workflow and ensuring the customer-facing application operational. These include inventory, payment, and customer relationship(contact/help/complaints) services. While most services are developed in-house, only a few are sourced from third parties.
- Data integration and analytics platform for understanding customer behavior and derive valuable insights for future business decisions.

2 Technical Requirements Analysis

According to the 6 Pillars of the AWS Well-Architected Framework[1], summarized as operational excellence, security, reliability, performance efficiency, cost optimization and sustainability, we propose more context-involved technical requirements to guide the software design process in particular facets.

- **Availability**: Ensuring uninterrupted access to the website and all functional backend services is vital for seamless user experience and efficient business operations. This reliability ensures that customers can browse products, make purchases, and access support services without any hindrance, thereby maximizing sales opportunities.
- **Information Security**: Running an online retail business involves gathering and retaining sensitive user data, such as personal information and payment details, which is unavoidable. Safeguarding data storage and access is crucial for fostering customer trust and adhering to data protection regulation. Moreover, internal departments, like analytics teams, must responsibly and legally utilize the collected data with granted access. Therefore, it is imperative to implement robust identification, authentication, and authorization mechanisms to protect sensitive information and ensure data privacy and security.
- **Scalability**: As business grows, the backend services are expected to serve a larger user base, efficiently process more transactions, handle greater data capacity without experiencing slowdowns or interruptions. It's essential for the software infrastructure to be robust and flexible enough to accommodate these changes seamlessly, ensuring that the business can scale its operations effectively to meet the evolving needs of its customers.
- **Continuous Deployment and Delivery**: CI/CD streamlines the software development process by automating testing, integration, and deployment tasks, reducing manual errors and enabling faster time-to-market for new features and improvements. This enables prompt responses to customer feedback, swift issue resolution, and ensures the software remains competitive in the landscape of online retail business.
- **Flexibility**: The retail business can effectively accommodate the varying demands of off-seasons and peak seasons by adjusting its computing resources accordingly. During peak seasons, such as holidays or promotional events, there is typically a spike in website traffic and transaction volumes. In contrast, off-seasons may experience reduced activity. With flexible computing resources, the business can scale up or down its infrastructure accordingly to meet these fluctuations in demand. This means being able to provision additional server capacity during peak periods to ensure optimal performance and customer experience, and then scale back down to avoid unnecessary costs. Additionally, flexibility enables the business to experiment with different resource configurations and adjust its infrastructure based on real-time insights and changing market conditions, ensuring cost-effective operations and agility in responding to dynamic retail environments.

3 Software Design, Integration and Architecture

3.1 Event-driven Microservices Architecture

Event-driven architecture (EDA)[2]provides a framework where software components respond to events within the system, such as user interactions or data updates. This flexibility allows the system to adapt dynamically to changing conditions, optimizing resource utilization and minimizing unnecessary computing overhead. EDA enables the efficient allocation of resources based on real-time demand, helping save costs by avoiding the provision of excessive computing resources during periods of low activity. By leveraging EDA, an online retail business can achieve greater agility and cost efficiency while maintaining a high level of responsiveness to customer needs and market dynamics.

Microservices[3]is a design approach where a large software application is broken down into smaller, independent services, each responsible for specific business functions. These services communicate and cooperate synchronously or asynchronously, promoting modularity, agility, and scalability. In the context of an online retail business, microservices architecture enables easier development, deployment, and maintenance of software components, aligning well with the ever-changing demands of the industry. Additionally, the loosely coupled architecture facilitates scalability and fault tolerance, enabling the system to handle increased user traffic and data volume efficiently. However, it should not be overlooked that a decentralized design pattern may result in data inconsistencies and it is critical to apply appropriate models to ensure eventual consistency.

By analyzing the specific technical requirements, **event-driven microservices** architecture combining the strengths of both EDA and microservices emerges as the most suitable choice. It offers the flexibility to handle fluctuating demand, the resilience to withstand failures, and the scalability to support growth. Leveraging cloud computing technologies further enhances these capabilities, ensuring that the online retail system can deliver a seamless and responsive shopping experience to customers while optimizing operational efficiency. The fourth section will focus on illustrating the deployment of the software using suitable AWS computing resources and serverless stacks.

3.2 Other applicable Software Patterns

In the pursuit of implementing a robust microservices architecture, additional software patterns play pivotal roles in guaranteeing data persistence, efficient inter-service communication, and effective state management.

CQRS [4] Command Query Responsibility Segregation (CQRS) is a design pattern that advocates for separating the responsibility of handling read and write operations on data. By decoupling these operations, CQRS utilizes distinct models optimized for reading and writing data. The write model focuses on executing commands to modify data, ensuring consistency and transactional integrity, while the read model prioritizes efficient data retrieval to support user interactions. Online retail platforms typically experience much heavier read than write operations, with users frequently browsing product catalogs, viewing product details, and accessing order history. By separating read and write operations, CQRS encourages the optimization of each model to handle the specific demands of these operations. This segregation enhances the scalability and performance of the system, enabling the online retail business to efficiently handle the high volume of read operations while ensuring data consistency and reliability.

Saga [5] The saga pattern stands as a cornerstone in distributed systems, offering an approach for state management and failure management. By orchestrating sequences of local transactions, it navigates through complex workflows, handling both successful and fail transaction gracefully. In the realm of online retail, where services engage in complex interactions, the saga pattern is indispensable. With services triggered by orders and dependencies among them, a failure in one service can disrupt the entire process. The saga pattern addresses this challenge by facilitating the sequential execution of transactions, enabling swift reactions to any unforeseen issues while maintaining overall consistency across the system.

Asynchronous messaging and event passing [6] Asynchronous messaging and event passing are fundamental integral components within microservices architecture. Asynchronous messaging involves communication between services where the sender and receiver do not need to interact in real-time. Messages are sent and received via a message broker, allowing services to operate independently and asynchronously. Event passing involves the broadcasting of events or messages to multiple services without direct interaction. These mechanisms facilitate loose coupling between services, allowing them to handle tasks independently and efficiently without being tightly coupled to one another, thereby ensuring smooth and uninterrupted operations of the online retail platform.

4 Cloud Services Selection and Deployment

The entire software system can be segmented into three distinct components: the blog, the operational website for retail business activities, and the data integration and analytics platform. This section will delineate the selection and deployment of cloud services for each of them.

4.1 Blog

To deploy a static blog on AWS, we leverage various services to ensure efficient access and prompt update notifications. Firstly, Route 53 is employed for domain management and DNS routing, enabling convenient access to the blog. CloudFront is utilized as a content delivery network (CDN) to enhance the blog's performance and global reach. Static webpages are stored securely in Amazon S3, offering reliable storage and scalability. Additionally, SNS (Simple Notification Service) and SES (Simple Email Service) are included for notifying clients about updates to emerging retail information, ensuring timely engagement.

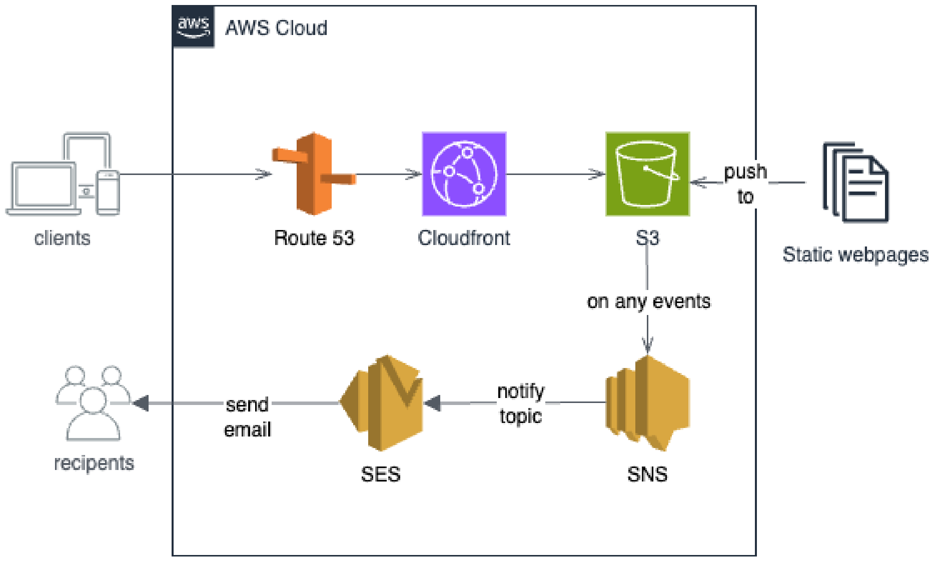

Fig. 1. Static blog for emerging information

4.2 Operational Website

The retail website architecture comprises several key components for essential functionalities. Cognito is employed for robust identification, authentication, and authorization in order to achieve security goals. CloudFront enhances content delivery performance by leveraging a global network of edge locations. Static contents are securely stored in Amazon S3. API Gateway routes requests to various backend services, segregating command and query operations, with subsequent invocation of corresponding Lambda functions and requisite services for database access and specific task completion. Opting for containerized microservices and leveraging ECS, especially with Fargate, offers substantial benefits in terms of scalability, flexibility, and CI/CD. Containerization ensures efficient resource utilization and easy scalability, allowing applications to adapt seamlessly to changing demands. ECS further enhances flexibility by abstracting away infrastructure management complexities, enabling teams to concentrate on software development. Additionally, ECS integrates smoothly with CI/CD pipelines(e.g., AWS CDK), streamlining the deployment process and enabling rapid iteration and delivery of new features and updates.

Serverless computing stacks AWS provides two primary serverless computing options: Lambda and Fargate. Lambda is ideal for event-driven, short-lived tasks, such as processing data or responding to HTTP requests, scaling automatically and billing based on usage. Conversely, Fargate is better for containerized applications requiring longer runtimes and more resource control. It offers managed infrastructure for container execution without server management. There-

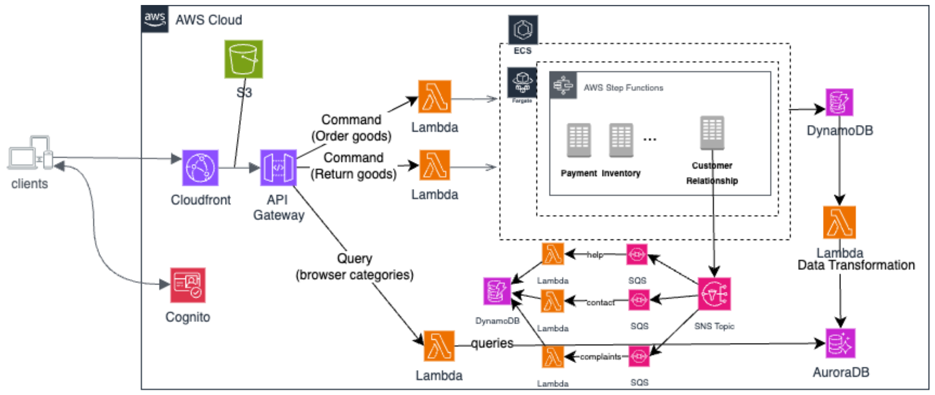

Fig. 2. The website that allows a client to browse a catalog, order goods and return goods

fore, when choosing between Lambda and Fargate, it is necessary to assess the workload and scalability needs. For HTTP requests related to browsing, ordering, and returning, Lambda is a suitable choice due to its event-driven nature and automatic scaling based on request volume. However, for services like payment, inventory and customer relationship management, where longer runtimes or specific resource allocation may be necessary, Fargate could be preferred. Fargate offers more control over resource allocation and can handle services with longer execution times more effectively, ensuring optimal performance for these critical functions within the online retail business architecture.

CQRS [4] By separating the read(browser different categories) and write(order/ return goods) operations, they can be scaled independently. Given the higher frequency of reads compared to writes, independent scaling of the read transactions can enhance efficiency and cost-effectiveness. Proper implementation ensures that downtime or performance issues on one side do not negatively impact the other, thereby enhancing system availability and resilience. Additionally, decoupling reads from writes enables the adoption of diverse technologies, including multiple databases or frameworks. Event-driven services like AWS Lambda, which can respond to triggers such as commands or database changes, can be utilized to streamline CQRS architectures and maximize cost effectiveness.

Saga [5] Applying saga pattern is essential for ensuring the workflow of our operations, particularly in scenarios where sequential actions are critical, such as inventory operations only after successful payments. By implementing the saga pattern, we can manage states and handle failures effectively, ensuring that each step in the workflow progresses smoothly and consistently. AWS Step Functions offer a powerful solution to realize the saga pattern within microservices system[7]. With Step Functions, we can define and orchestrate complex workflows composed of multiple steps, each of which can be executed independently and rolled back if necessary. This enables us to create robust and resilient workflows that adhere to the principles of the saga pattern, ensuring the integrity and reliability of our system's operations.

Asynchronous messaging and event passing [6] Amazon SNS and Amazon SQS are used as integral components for event-driven asynchronous messaging. Amazon SNS allows publishers to send messages to multiple subscribers asynchronously, making it ideal for broadcasting events across distributed systems. Amazon SQS provides a reliable message queuing service, allowing decoupling of message producers and consumers(e.g., help/contact/complaints in above diagram), ensuring message delivery and processing even in high-traffic scenarios. When combined, Amazon SNS and Amazon SQS enable robust event-driven architectures, facilitating scalable and resilient communication between microservices and applications.

Third-party services integration - AWS PrivateLink[8] AWS PrivateLink is an Amazon Virtual Private Cloud feature that facilitates private connectivity between VPCs and AWS services. It enables secure communication between two parties without traversing the public internet, reducing exposure to external threats. PrivateLink is commonly utilized to integrate third-party services, providing a seamless and secure connection without the need for an internet gateway. Its advantages include enhanced security, simplified network architecture, and support for private communication between VPCs and AWS services.

4.3 Data Integration and Analytics

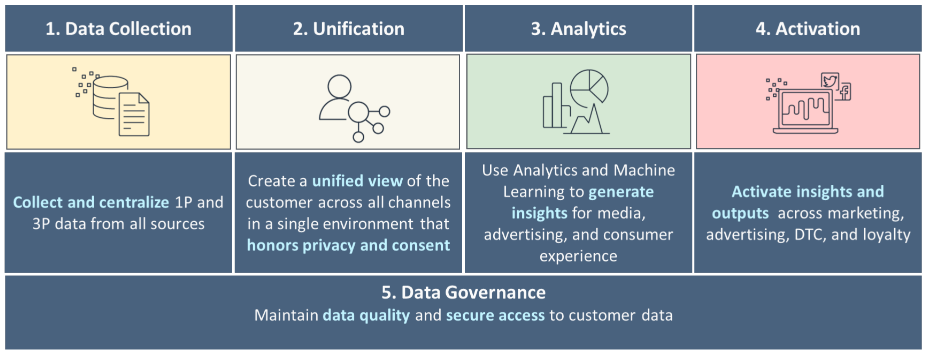
1. Data Collection
2. Unification
3. Analytics
4. Activation
Collect and centralize 1P and 3P data from all sources
Create a unified view of the customer across all channels in a single environment that honors privacy and consent
Use Analytics and Machine Learning to generate insights for media, advertising, and consumer experience
Activate insights and outputs across marketing, advertising, DTC, and loyalty
5. Data Governance
Maintain data quality and secure access to customer data

Fig. 3. The five pillars of a mature C360

Customer 360 (C360)[10] offers a comprehensive and consolidated perspective of customer interactions and behaviors across various touch points and channels, facilitating data-driven decisions to enhance business outcomes. As shown in **Fig.3**, the five pillars of C360 entail basic data collection, unification of data from diverse customer channels, advancing to analytics for decision-making, and personalized engagement across multiple channels. Meanwhile, it is crucial to guarantee effective data governance by continually incorporating new data while upholding its quality and maintaining stringent security measures.

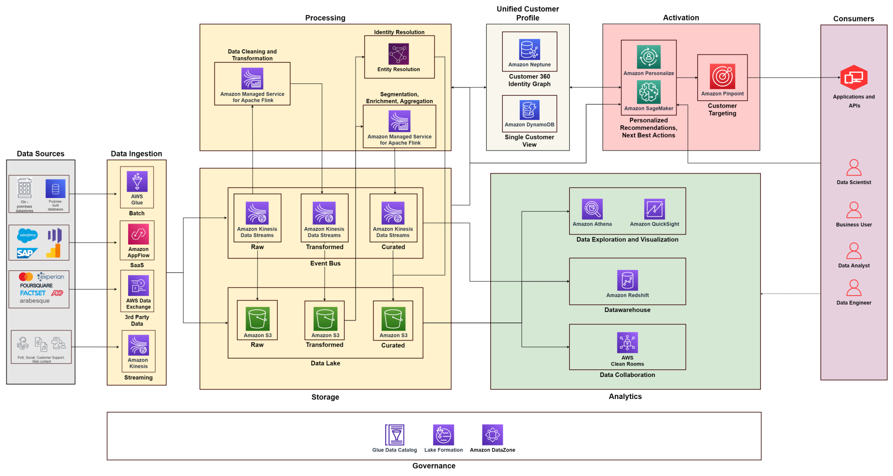

Fig. 4. An end-to-end data strategy for Customer 360 on AWS[9]

To build data platform for the online retail business, we can learn experience from the solution architecture shown in **Fig.4**, which combines the building blocks of a Customer Data Platform on AWS[11] with additional components used to design an end-to-end C360 solution. Utilizing it as a framework with proper AWS services, we can establish projects and teams dedicated to developing and managing various capabilities, adhering closely to the guidance provided by the five pillars. For instance, a data integration team consisting of data architects and engineers can concentrate on the data collection pillar, analytics practices can be developed to address the analytics and activation pillars, and a specialized team can be formed to construct the unified view of the customer. Additionally, a data governance team responsible for balancing control and accessing gives users trust and confidence in data. The governance team has to build processes that validate data quality and take corrective actions, e.g., with AWS Glue Data Quality.

In conclusion, it's crucial for organizations to develop a robust C360 capability to gain insights for future business decisions. Utilizing AWS Databases, Analytics, and AI/ML services can streamline this process by offering scalability and efficiency. Following the five pillars helps create an end-to-end data strategy, ensuring data accuracy, and establishing cross-functional governance for customer data.

References

1. AWS Well-Architected and the Six Pillars, https://aws.amazon.com/architecture/well-architected/?wa-lens-whitepapers.sort-by=item.additionalFields.sortDate&wa-lens-whitepapers.sort-order=desc&wa-guidance-whitepapers.sort-by=item.additionalFields.sortDate&wa-guidance-whitepapers.sort-order=desc.
2. Event-Driven Architecture, https://aws.amazon.com/event-driven-architecture/.
3. Microservices Architecture, https://aws.amazon.com/microservices/.
4. CQRS pattern, https://docs.aws.amazon.com/prescriptive-guidance/latest/modernization-data-persistence/cqrs-pattern.html.
5. Saga pattern, https://docs.aws.amazon.com/prescriptive-guidance/latest/modernization-data-persistence/saga-pattern.html
6. Asynchronous messaging and event passing, https://docs.aws.amazon.com/whitepapers/latest/microservices-on-aws/asynchronous-messaging-and-event-passing.html
7. Implement the serverless saga pattern by using AWS step functions, https://docs.aws.amazon.com/prescriptive-guidance/latest/patterns/implement-the-serverless-saga-pattern-by-using-aws-step-functions.html
8. AWS PrivateLink, https://docs.aws.amazon.com/prescriptive-guidance/latest/integrate-third-party-services/architecture-1.html
9. Create an end-to-end strategy for customer 360 on AWS, https://aws.amazon.com/blogs/big-data/create-an-end-to-end-data-strategy-for-customer-360-on-aws/
10. Elevate customer experiences with a 360-degree view of data, https://aws.amazon.com/data/customer-360/
11. An overview and architecture of building a Customer Data Platform on AWS, https://aws.amazon.com/blogs/architecture/overview-and-architecture-building-customer-data-platform-on-aws/