Assignment

Course Code : CSE-316

Course Title : Software project III

Assignment Topic : Designing a scalable web Application architecture

Submitted to :

Fahim Faisal
Lecturer
Department of CSE
Daffodil International University

Submitted by:

Tanha Tul Tasnim
ID: 221-15-5626
Sec: 61_V
Department of CSE
Daffodil International University

Date of submission : 14 – 05 – 2024.

Contents

1.0 Objectives

- Understand the importance of software architecture in building scalable web applications.
- Explore various architectural patterns and their suitability for different scalability requirements. Design a scalable web application architecture considering factors such as performance, reliability, and maintainability.
- Select appropriate technologies and tools to implement the designed architecture.
- Demonstrate the ability to document and present architectural designs effectively.

2.0 Research and Analysis

2.1 What is scalable web architecture?

"Scalable web architecture" refers to the design and building of a website or service that can grow to handle increasing loads and user traffic without affecting its dependability, speed, or performance. Scalability is an essential feature of a modern online application because it allows to grow and accept more customers and information over time.

2.2 Key components of scalable web architecture :

- **Vertical Scaling:** Vertical scaling is the process of making current servers more capable of handling increased traffic. For example - adding additional memory, storage, or CPU power.

- **Horizontal Scaling:** Increasing the number of servers or variants to disperse the load across several machines as opposed to depending only on a single, strong server.

- **Load Balancing:** By dividing up incoming traffic among several servers, load balancing keeps any one server from becoming overloaded.

- **Microservices:** Break down a large application into smaller, independent services that can be developed, deployed, and scaled independently.

- **Content Delivery Networks (CDNs):** Distributing static content, such as images and videos, across a network of servers around the world to deliver content more efficiently to users.

- **Auto-Scaling:** Automatically adjusting resources based on current demand to ensure optimal performance.

- **Caching:** Reducing stress on databases and application servers by caching regularly used data.

- **Asynchronous Processing:** To increase speed and flexibility, separate tasks and processes.

- **Monitoring and logging:** Constantly monitoring tracks on the application's performance and gathering information to identify and fix problems.

2.3 Research on Scalable Web Application Architectures:

1. Microservices Architecture:
- Describe the advantages of microservices architecture.
- Examine case studies related to effective deployments.
- Determine the issues and factors to take into account when using microservices.

2. Monolithic Architecture:
- List the features of this type of architecture.
- Discuss about circumstances that are fit for monolithic architecture.
- Assess the problems and shortcomings of monolithic architectures.

3. Serverless Architecture:
- Explain the concept of serverless architecture.
- Investigate scenarios where serverless technology offers benefits.
- Consider limitations and potential challenges with serverless approaches.

4. Scalability Requirements Analysis:
- Examine traffic patterns, particularly in rush hour.
- Determine the predicted user base and the number of active users.
- Evaluate data volume and growth projections.
- Identify potential bottlenecks and performance considerations.

2.4 Analysis of Scalability Requirements:

- **Expected User Base:** Determine the project number of users accessing the application.

- **Traffic Patterns:** Analyzing the variability and predictability of traffic, including peak usage times and traffic points.

- **Data Volume:** Consider the volume of data processed and stored by the application. Impacts scalability, especially in terms of database performance and storage requirements.

- **Future Growth Projection:** Anticipate future growth in user base, traffic, and data volume to design a scalable architecture capable of accommodating increased demand over time.

3.0 Architecture Design

Designing a Scalable Web Application Architecture:

The microservices architecture appears to be the ideal choice for the web application considering the scalability guidelines identified by the research. This decision serves as the foundation for the scalable architecture design that appears as follows.

3.1 Components/Modules:

- **API Gateway:** Serves as the client's point of entry, relaying inquiries to the relevant microservices. It manages request or response transformations, authorization, and security.
- **User Service:** Manages user-related functionalities such as authentication and profile management.
- **Content Service:** Handles content-related operations such as creation, retrieval, and management.
- **Analytics Service:** Collects and analyzes user behavior data for generating insights and recommendations.
- **Notification Service:** Manages communication with users through emails, push notifications, etc.
- **Database Service:** Provides persistence for microservices, utilizing a database per service approach for data isolation and scalability.

3.2 Responsibilities and Interactions:

- **API Gateway:** Routes incoming requests to the corresponding microservices based on the endpoint. It implements load balancing and can cache frequently accessed data for improved performance.
- **User Service:** Handles user authentication and authorization, storing user credentials securely. Interacts with the Database Service to retrieve and update user information.
- **Material Service:** Manages the production, recovery, and editing of material. Makes use of storage approaches to deliver content that is often accessed efficiently. Interacts with the database service to store data.
- **Analytics Service:** Collects user behavior data from various sources and performs analysis to generate insights. May utilize a distributed cache for storing intermediate results to improve performance.

- **Notification Service:** Sends notifications to users based on triggers such as new content availability or personalized recommendations. Utilizes asynchronous communication to avoid blocking user requests.

- **Database Service:** Provides data storage for microservices. Utilizes scalable database solutions such as NoSQL or distributed databases to handle increasing data volume.

3.3 Load Balancing & Caching Mechanisms:

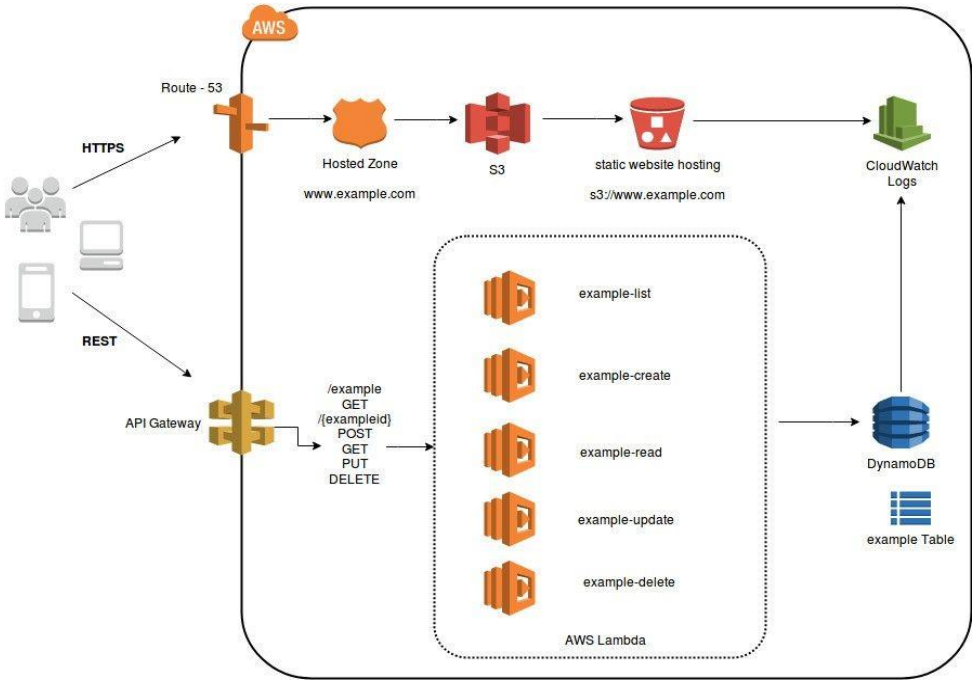
- **Load Balancer:** Distributes incoming traffic evenly across multiple instances of microservices to prevent overload on any single instance.
-
- **Caching:** Utilizes caching mechanisms such as Redis or Memcached to store frequently accessed data, reducing the load on the database and improving response times for read heavy operations.

3.4 Data Storage:

- **Database:** Utilizes a scalable database solution capable of handling large volumes of data and providing horizontal scalability through partitioning.

3.5 Communication Between Components:

- **RESTful APIs:** Microservices communicate with each other using RESTful APIs over HTTP or messaging queues for communication, ensuring scalability.



4.0 Technology Selection

The community support, scalability, efficiency, and ease of creation are some of the factors that must be considered while selecting the technologies and tools to implement the intended microservices architecture. An analysis of various choices for databases, cloud services, programming languages, and frameworks is provided below:

4.1 Programming Languages:

- **Java:** Java remains a solid choice for its maturity and extensive ecosystem.
- **JavaScript**: Offers non-blocking I/O and lightweight architecture suitable for real-time applications.
- **Python:** Python is a versatile programming language often used in web development. It offers various web frameworks and libraries that simplify the process of building and maintaining web applications. Python's readability, ease of use, and rich ecosystem make it a popular choice for web development.
- **C#:** When it comes to web development, C# is a key language in the NET ecosystem, which includes a variety of powerful frameworks and libraries for building modern, high-performance web applications.

-

4.2 Frameworks:

- **Spring Boot (Java):** A popular choice for building microservices in Java, offering features like auto-configuration and dependency injection.
- **Express.js (Node.js):** A minimalist web framework for Node.js, well-suited for building RESTful APIs and lightweight microservices.
- **Django (Python):** Provides a high-level framework for rapid development, including built-in features like ORM, admin interface, and authentication.
- **ASP.NET Core (C#):** Offers a powerful framework for building microservices with features like MVC pattern, middleware pipeline, and built-in support for dependency injection.

4.3 Database:

- **MongoDB:** Offers flexibility and scalability for handling diverse data types and high volumes of data.
- **Amazon DynamoDB:** A fully managed NoSQL database service provided by AWS, offering seamless scalability and high availability.
- **PostgreSQL:** A robust relational database with support for ACID transactions and advanced features like JSONB data type.

4.4 Cloud Services:

- **AWS (Amazon Web Services**): Offers a comprehensive suite of cloud services, including compute (EC2, Lambda), storage (S3, DynamoDB), and database (RDS, DynamoDB).
- **Google Cloud Platform (GCP):** Provides a variety of managed services for building microservices, including Compute Engine, Cloud Functions, and Cloud Spanner.
- **Microsoft Azure:** Offers services like Azure App Service, Azure Functions, Azure Cosmos DB, and Azure Kubernetes Service (AKS).

4.5 Justification:

- Python with Django Provides a high-level, rapid development framework suitable for building microservices quickly and efficiently.
- ASP.NET Core Offers a mature and powerful framework for building microservices with robust features and strong support for the C# language.
- The range of possibilities for developers is increased by including Python Django and ASP.NET Core, which accommodate various preferences, skill levels, and project requirements.

5.0 Documentation

5.1 Microservices Architecture Documentation

The range of possibilities for developers is increased by including Python Django and ASP.NET Core, which atone various preferences, skill levels, and project requirements.

5.2 Architecture Components

1. API Gateway.

- Its job is to direct client requests to the required services.
- Executes transformations for requests and responses, permission, and identification.
- Employs load balancing methods to allocate incoming traffic among services

2. User Service.

- Oversees user-related features like profile management, permission, and verification.
- Stores user data in MongoDB.
- Provides access to RESTful APIs for managing profiles, login in, and user registration.

3. Content Service.

- Manages tasks associated with material, including generation, retrieval, and administration.
- Utilizes MongoDB for storing content data.
- Implements caching mechanisms to improve performance for read-heavy operations.

4. Analytics Service.

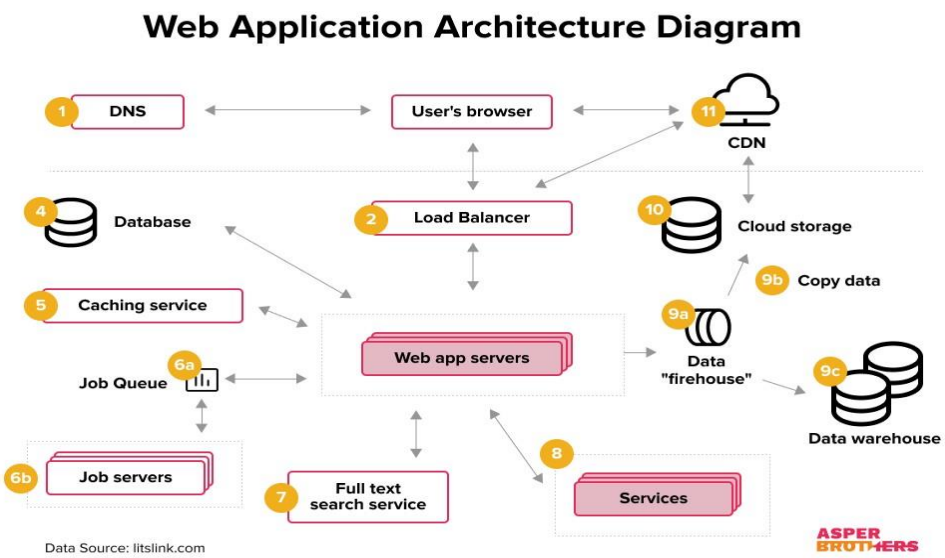
- Collects and analyzes user behavior data for generating insights and recommendations.
- Utilizes MongoDB for storing analytical data.
- Implements event-driven architecture for processing user behavior events asynchronously.

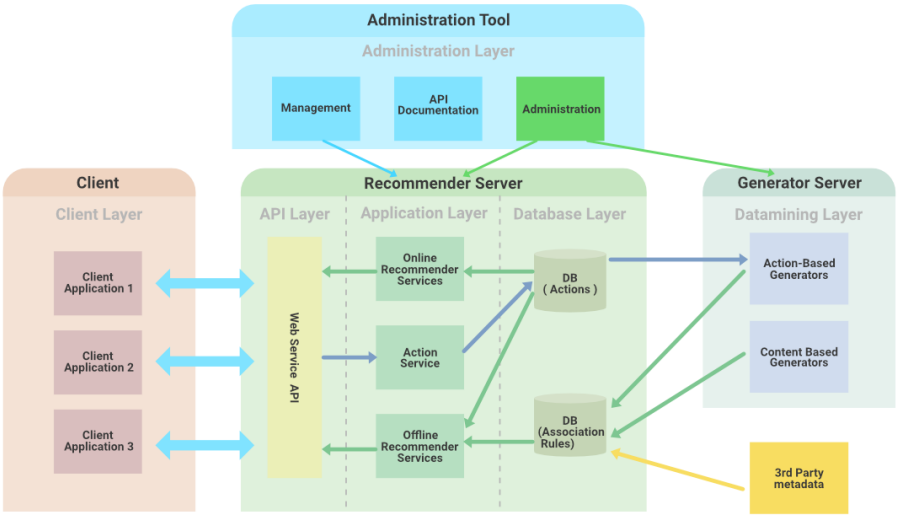
5. Notification Service.

- Manages communication with users through emails, push notifications, etc.
- Integrates with external email and notification services for message delivery.
- Utilizes message queues for asynchronous communication to avoid blocking user requests.

6. Database Service.

- Provides persistence for microservices.
- Utilizes MongoDB for data storage, offering scalability and flexibility.
- Implements data partitioning/sharding for horizontal scalability.





5.3 Interactions

1. Client Interaction:

- Clients send requests to the API Gateway.
- API Gateway routes requests to the appropriate microservices based on the endpoint.

2. Microservices Interaction:

- Microservices exchange messages with one another via message queues or RESTful APIs.
- For user-generated content activities, User Service and Media Service may interact.
- Analytics Service handles events related to user behavior that are received from different platforms.

5.4 Conclusion

This web application has durability, scalability, and flexibility thanks to the microservices design. The application may be developed, deployed, and maintained more efficiently by breaking it up into smaller, independently deployable services. Scalability and dependability are ensured by the use of AWS for cloud services and MongoDB for data storage, which makes the architecture fit for managing a variety of workloads and potential expansion.