Eventful: A Comprehensive Peer-to-Peer Event Management Platform

Leo Zhang
zz2830
Columbia University

Allen Na
an3091
Columbia University

Joshua Segovia
js5872
Columbia University

Lawrence Leung
lsl2162
Columbia University

Abstract—**In this paper, we present a comprehensive analysis of "Eventful," a web app designed to elevate the process of creating, sharing, and attending events within social and professional circles. Unlike similar services, Eventful focuses on peer to peer event creation, simultaneously enabling seamless and efficient management while fostering interpersonal connections. We discuss the core technical components, including its structure, key features, and architecture. Through this paper, we will demonstrate Eventful's transformative potential in reshaping the landscape of event organization and networking. Additionally, we explore the broader implications of Eventful's success, highlighting its contributions to the ever-evolving field of computer science and its potential impact on future digital innovations.**

Keywords—Eventful, decentralized event management, real-time event updates, event categorization, attendance management

I. Introduction

The existing landscape of event-focused applications is missing a robust peer-to-peer component. While platforms like Eventbrite allow users to explore events hosted by businesses and organizations, they don't adequately cater to individual, personal gatherings. Applications like Partiful offer an avenue for individual users to create events, but lack in its capability to manage event specifics like itinerary, inventory, and information. Our application, Eventful, fills this void by offering powerful tools for individual users to create, share, and manage personal events in a convenient manner with a minimal barrier of entry.

In this paper, we present Eventful, a comprehensive event management application designed to address the shortcomings of current platforms in the context of personal gatherings. By integrating advanced features such as itinerary planning, inventory tracking, and seamless information sharing, Eventful fosters efficient collaboration between event hosts and attendees. Built using a robust technical architecture leveraging cloud technologies and serverless functions, Eventful is designed to scale seamlessly as its user base grows. Furthermore, the application's user-friendly interface and seamless integrations make it an ideal choice for individuals seeking a comprehensive solution to plan and manage personal events. We discuss the technical details, key features, and benefits of Eventful, as well as its potential for future enhancements to cater to an even broader range of event management needs.

II. Code Structure

In order to facilitate seamless communication between the front-end and the back-end services, Eventful leverages AWS Lambda functions. These serverless functions enable efficient, scalable, and cost-effective processing of user requests, ensuring high performance and responsiveness.

A. LF1: get-event and LF3: put-event

To start, our first lambda function, get-event, uses the AWS SDK's DynamoDBClient and DynamoDB Document Client to interact with the DynamoDB tables. This allows the function to efficiently query the "Eventful-Events" and "Eventful-Users" tables, ensuring that event and user data are easily accessible and up to date.

Upon receiving a request, the function first checks if the provided event identifier (ID) is valid. If not, it responds with a 400 status code and an error message. If the ID is valid, the function proceeds to fetch event data from the Eventful-Events table using the GetCommand. If the event is not found, the function returns a 404 status code and an error message.

Once the event data is successfully retrieved, the function proceeds to fetch user data from the "Eventful-Users" table by using the event data's user ID (uid) and another GetCommand operation. The user ID serves as the primary key for the Eventful-Users table.

If the host user is found, the function adds the host's name to the event data, combining the two pieces of information to provide a more comprehensive view of the event. The function then returns the combined event data with a 200 status code, allowing the front-end application to display the event details to the user.

On the other hand, the third lambda function, put-event, is tasked with updating event details in the back-end database. Similar to our first lambda function, it uses AWS SDK's DynamoDBClient and DynamoDB DocumentClient to interact with the DynamoDB tables. MarshallOptions are set to removeUndefinedValues for the purpose of preventing the storage of null or undefined values in the database.

Upon receiving a request, the function will validate the provided user ID (uid) and event ID (eid) in the path parameters. The function then checks if the event exists and belongs to the user by querying the Eventful-Events table using a GetCommand operation. If the event is not found or the user does not have permission to edit the event, the function returns a 404 or 403 status code, respectively, and an error message.

Before updating the event in the database, the function validates the input data for each attribute provided in the request body. The validateEventInput function checks the data type and structure of each attribute, ensuring that only valid data is stored in the database. The function also checks the location data using the validateLocation function to ensure that it is a valid location object.

Upon successfully validating the input data, the function prepares the update parameters for the DynamoDB UpdateCommand operation. It dynamically generates the UpdateExpression, ExpressionAttributeNames, and ExpressionAttributeValues based on the provided input data.

The UpdateExpression is a string that specifies the attributes to be updated and the corresponding new values. The ExpressionAttributeNames and ExpressionAttributeValues are objects that map attribute names and values to placeholders in the UpdateExpression. The function sends the UpdateCommand with the prepared parameters to the Eventful-Events table, updating the specified attributes of the event.

B. LF2: post-event

The second lambda function, post-event, defines two validation functions, validateLocation and validateEventInput, to ensure that the provided location and event data are valid and correctly formatted. The validateLocation function checks the integrity of the location object and its properties, such as the latitude and longitude coordinates.

The validateEventInput function checks the provided event data, including the event's unique identifier (ID), user ID (uid), name, date and time, description, detail, capacity, public/private status, creation timestamp, location, and image URL. The function receives an event object containing the event data. It then calls the validateEventInput function with the event data to check its validity. If the input data is valid, the function proceeds to construct a params object containing the TableName Eventful-Events and the Item data to be inserted.

The PutCommand is executed using the DocClient to store the new event in the database. If the event is created successfully, the function returns a 200 status code with a "Event created successfully" message. In case of an error during the event creation process, a 500 status code is returned with an "Error creating event" error message.

C. LF4: get-user and LF6: post-user

The get-user function is responsible for retrieving user profile information, which is integral for displaying user profiles, populating attendee lists, and personalizing event recommendations. When the front-end requests user data, it sends an event object containing the user's unique identifier (ID) in the pathParameters property to the Lambda function. The function validates the provided ID and, if it is valid, constructs a query object targeting the Eventful-Users table, specifying the Key (ID) to search for.

The GetCommand is executed using the DynamoDB DocumentClient to fetch the user data from the database. If the user is found, the function returns a 200 status code with the user data in a JSON object, whereas a 404 status code is returned if the user is not found. Any errors encountered during the data fetching process will result in a 500 status code with an "Error fetching user" error message.

On the other hand, the post-user function manages the registration of new users, which is critical for expanding the user base of the Eventful app. The function imports the necessary AWS SDK modules and initializes the DynamoDBClient and DynamoDBDocumentClient instances, similar to the get-user function. The function receives an event object containing the user's ID, name, username, email, and biography, if the user chose to fill it out. A validateInput function checks the format and data type of the provided user data, by using regex to validate the email address format.

After the user data passes validation, the function constructs an insertion object targeting the Eventful-Users table, specifying the user data to be inserted. The PutCommand is executed, using the DynamoDB DocumentClient to insert the new user data into the database.

C. LF7: post-invitation

post-invitation is responsible for handling the process of inviting users to events. This function interacts with other parts of the system, including DynamoDB for storing invitation data, SQS for sending invitation notifications, and SES for sending invitation emails.

Upon receiving an event object containing the event ID (eid), recipient's email, and an optional message, the function validates the input data. If the input is valid, it proceeds to check whether an invitation for the same event and email already exists in the Eventful-Invitations table. If the invitation exists, a 400 status code and the user is sent a message that the invitation has already been sent. If the invitation does not exist, the function retrieves the event and host user information from the Eventful-Events and Eventful-Users tables, respectively. Using this information, it constructs and sends an email to the recipient using the SES service. If the email is sent successfully, the function creates a new entry in the Eventful-Invitations table with an invitation status of pending.

Additionally, an SQS message containing the invitation details is sent to the Eventful-Invitations queue. This message includes the event name, event ID, host's name, host ID, recipient's email, and the optional message. The recipient's clients can poll this queue for new invitations.

Upon receiving the invitation, the recipient can respond by either accepting or declining it. The frontend then calls the Update Invitation API to update the invitation's status in the Eventful-Invitations table. If the invitation is accepted, the Eventful-Events table is updated with the new attendee.

D.LF9: get-user-events

The get-user-events function is responsible for fetching events related to a specific user in the Eventful application. It retrieves events that a user is hosting or events that the user has received an invite.

The function first extracts the user ID from the path parameters, as well as the event type: all, hosting, or invited. “All” will display both the user’s created events, as well as events that the user has been invited to. “Hosting” and “Invited” will query only the events that the user has created or been invited to, respectively. Furthermore, the function also handles the pagination parameters from the query string. It then retrieves the user's email by calling the get_user_by_id function.

Depending on the event type, the function calls one of the following functions to fetch the relevant events: fetch_all_events, fetch_hosting_events, and fetch_invited_events. fetch_all_events fetches both hosting and invited events for the user and returns a merged and sorted list of events, while fetch_hosting_events returns the events hosted by the user, and fetch_invited_events retrieves the events the user is invited to. These functions use pagination to limit the number of events returned in the response.

E. LF10: put-invitation and LF11: get-invitation

The put-invitation function is responsible for updating the invitation status for a user in the Eventful application. It also sends an SQS message to notify the host of the updated invitation status.

The function validates the input data, which includes the event ID (eid), host ID (hostId), host name (hostName), user ID (uid), user name (name), user email (email), event name (eventName), and invitation status (invitationStatus). If any of the required data is missing, it returns an error response with a 400 status code.

If the input data is valid, the function proceeds to fetch the host user's email from the Eventful-Users table. It then constructs the SQS message body, which contains the event and user-related data, along with the updated invitation status. The function sends the SQS message to the Eventful-Invitations queue using the SendMessageCommand.

After successfully sending the SQS message, the function updates the Eventful-Invitations table with the new invitation status. If any errors occur during this process, it returns an error response with a 500 status code. Finally, if the invitation status update and SQS message sending are both successful, the function returns a success response with a 200 status code.

Next, get-invitation extracts the event ID (eid) from the path parameters. If the event ID is missing, it returns an error response with a 400 status code. If the event ID is valid, the function proceeds to query the Eventful-Invitations table for all invitations associated with the given event ID. It then extracts the email addresses from the returned invitation records. The function queries the Eventful-Users table for each email address to retrieve the user ID and name. It stores the user data in a usersByEmail object, with the email address as the key.

Finally, the function constructs an invitationList array, which contains the user ID, name, email, and invitation status for each invitation record. It maps the invitation records from the "Eventful-Invitations" table to the corresponding user data from the Eventful-Users table using the usersByEmail object.

If there are any errors while fetching the invitation data, the function returns an error response with a 500 status code.If the invitation data is fetched successfully, the function returns a success response and the invitationList as the response body.

E. LF12: post-comment

The twelfth lambda function, post-comment, extracts the comment ID (id), event ID (eid), user ID (uid), and comment content from the input event. If the input data is valid, the function creates a timestamp and constructs a new comment item with the ID, event ID, user ID, content, and timestamp. It then uses a PutCommand to insert the new comment item into the "Eventful-Comments" table. If there are any errors while inserting the comment, the function returns with an error response with a 500 status code. When a comment is added successfully, the function returns a success response with a 201 status code.

E. LF13: get-comments and LF14: put-comment

The next lambda function, get-comments, retrieves all comments related to a specific event in the Eventful application. It extracts the event ID (id) from the input event's path parameters. If the input data is missing or of an incorrect type, the function returns an error response with a 400 status code.

If the input data is valid, the function constructs a queryParams object to fetch the comments related to the event ID using eid-timestamp-index. It then sends a QueryCommand with these queryParams to retrieve the comments from the Eventful-Comments table. Next, it extracts the unique user

IDs from the comments and fetches the corresponding user data from the Eventful-Users table using a BatchGetCommand. It then creates a usersById object to map the user IDs to their respective user data.

The function then creates a new array called commentsWithUserNames by mapping over the comments data and adding the user's name to each comment object. Finally, it returns a success response with a 200 status code and the commentsWithUserNames array in the response body.

Next, put-comment updates a comment related to a specific event in the Eventful application. The function starts by extracting the eventId, id, content, and timestamp from the input event. If any of these properties are missing, it returns an error response with a 400 status code.

If the input data is valid, the function constructs a params object to update the content of the comment using the eventId and timestamp as the primary key. It then sends an UpdateCommand with these params to modify the content of the comment in the Eventful-Comments table. If the update is successful, the function returns a success response with a 200 status code and a message indicating that the comment has been updated successfully.

In case of any errors during the update process, the function returns an error response with a 500 status code and a message indicating that there was an error updating the comment.

III. Key Elements

Our application, eventful, is designed with the personal user in mind. For this reason, we approached the application with a low barrier of entry in mind. A user only needs an email in order to be invited to an event. Second, our front end is designed with REACT and is simplistic in nature, showcasing only the most necessary information to the user.

A. Low barrier of entry

A key aspect of Eventful's design is its low barrier of entry, which aims to make the platform as accessible and inclusive as possible for a wide range of users. This approach is crucial in order to accommodate individuals who may not be active on social media or have not yet joined the platform. Amazon Simple Email Service (SES) allows us to send event invitations via email to users outside of the Eventful platform. The email contains essential event information and a unique link that directs recipients to the event page within our application. This process allows them to seamlessly join the event and interact with other attendees, regardless of their prior engagement with the app.

Our focus on inclusivity is key in providing an alternative option of social media or the typical “friends list”, as it gives users the assurance that the widest possible audience can participate in events organized through Eventful. Although social media integration will eventually be added as an option to invite attendees to an event, we still give users the option to directly email and text users, eliminating the dependence on an online presence or pre-existing connections within the app. This purposeful design choice ensures that we make our platform more inclusive and attractive to a diverse user base, while empowering users to focus on the essential aspects of event planning and execution.

B. Real time notification and collaboration

Eventful's real-time collaboration and communication feature allows event organizers and attendees to actively engage with one another, making it easier to manage event logistics and share critical information. By integrating real-time chat functionality, we ensure that users can efficiently coordinate and discuss event specifics, all the while being provided with pertinent event information, all on a single page.

The in-app chat feature enables event organizers and attendees to communicate instantly, allowing them to discuss event details, ask questions, and provide updates. This functionality eliminates the need for lengthy email chains or external messaging platforms, streamlining the communication process and ensuring that all users stay up-to-date on event-related information.

For example, the chat feature can be used to mention items attendees plan to bring to the event. This helps avoid duplication, and ensures that all necessary items are accounted for. Users can also discuss special requirements they may have, including allergies or accessibility needs. Organizers can make specific requests or suggestions, guiding attendees in their contributions. In addition, users can share their expected arrival and departure times, enabling the event organizers to better plan and coordinate activities. Furthermore, once the event is over, the chat system can be used to reminisce and joke about events that occurred.

D. Simple and Robust Dashboard

Eventful utilized REACT for the front end, as its component based architecture allows us to build the user interface modularly, so future additions to the app’s functionality can be done stepwise, without compromising previous functionality. Furthermore, since REACT uses DOM, we can render componentes quickly and efficiently so that the user has the most up to date information.

The dashboard itself is designed with the user in mind, presenting all the information the user needs, within a single, easy to navigate page. At the top of the dashboard, users can find the event title, date, time, and location. A quick glance down will display “People”, which is a list of attendees, indicating the names of those who have confirmed attendance, as well as those who have indicated that they are unable to come. Next to people is the chat feature, where users can converse with each other on topics ranging from jokes to the dress code.

When an event is created, the host can input the event’s address, which is then geocoded using AWS Location Service API to obtain the location of the event. On the dashboard, the event’s location is displayed with an embedded interactive map, providing users with a visual representation of the surrounding area and its offerings. Attendees have the ability to obtain personalized directions to the event, directly from their current location. All it takes is a single click on the events location, and the google maps app or website with the destination pre-filled appears. This helps give attendees real time information on the length of their journey to the event location, allowing them to update hosts and other attendees of their estimated time of arrival through the chat box.

In addition, the events page utilizes pagination and infinite scrolling to manage the presentation of a large number of events, As the user reaches the end of the displayed content, the frontend sends a request to the backend to retrieve the subsequent set of events. The backend subsequently provides the relevant batch of events, taking into account the requested page number and the number of events per page. Finally, the frontend seamlessly appends the newly fetched events to the existing list, allowing users to continue scrolling without interruption.

By loading smaller sets of events initially, the application can significantly reduce the load time for the events page and minimize the amount of processing required by the browser. This not only results in a more responsive user interface but also lessens the strain on the backend server, maintaining a consistent performance during periods of high user activity. Moreover, infinite scrolling offers an intuitive and engaging way for users to explore events without the need for manual navigation through pages of results.

All of these offerings are not limited to just the web, as Eventful is also optimized for mobile users. Eventful can detect the device's screen size and adjust the layout, font sizes, and other design elements accordingly. Additionally, navigation elements and buttons are designed to be touch-friendly, accommodating users who are interacting with the app using their fingers on a touchscreen. This ensures that the app maintains its aesthetic appeal and usability across a wide range of devices, from the smallest smartphones to large desktop monitors.

IV. Architecture

As we discussed in the section above, the frontend is developed using React, and the application employs Redux for state management, ensuring a structured and predictable flow of data between components.

The frontend communicates with the backend services through AWS API Gateway, which acts as a secure and scalable point of entry for exposing the application's functionality to users. API Gateway is responsible for handling client requests and routing them to the appropriate Lambda functions, which in turn perform the required operations.

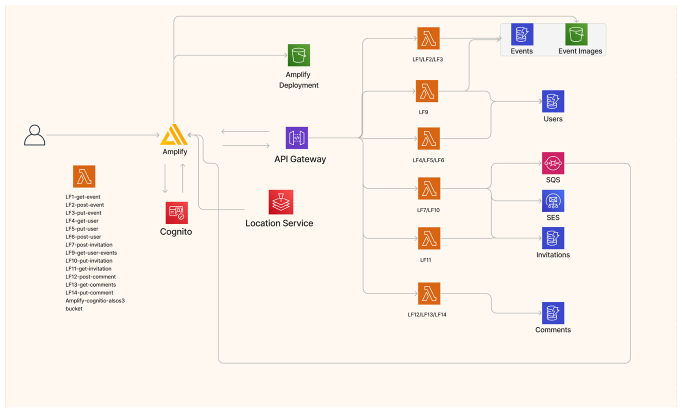
The backend services are powered by Node.js, running on AWS Lambda functions. Each Lambda function is designed to handle a specific responsibility, such as user registration (LF1), user login (LF2), event creation (LF3), event editing (LF4), event deletion (LF5), fetching event details (LF6), fetching user events (LF7), sending invitations (LF8), updating invitation status (LF9), fetching invitations (LF10), posting comments (LF11), fetching comments (LF12), and updating comments (LF13). The Lambda functions are organized in a modular fashion, enabling each one to be developed, updated, or modified independently, promoting maintainability and extensibility.

The Lambda functions interact with Amazon DynamoDB, a NoSQL database service, for data storage. Distinct tables are created for users, events, invitations, and comments. Secondary indexes are utilized to enable efficient querying based on non-primary key attributes, such as email or event timestamp. The DynamoDB DocumentClient library is employed to simplify data retrieval and manipulation, enabling more efficient and readable code.

Some Lambda functions, such as LF8 (invitation sending), are connected to Amazon Simple Email Service (SES) for sending email notifications related to event invitations. Amazon SES provides a reliable and scalable solution for sending emails, ensuring that notifications are delivered promptly and accurately.

The application also makes use of Amazon Simple Queue Service (SQS) to manage asynchronous tasks, like sending email notifications. Lambda functions listen for events from SQS and process tasks accordingly. This decoupling of event processing from the main application flow allows for better scalability and fault tolerance.

Moreover, the architecture employs a serverless approach, leveraging the inherent scalability and cost-efficiency of AWS Lambda. This allows the application to automatically scale to handle varying workloads, ensuring optimal performance without any manual intervention.



V. Results

The application demonstrates impressive response times and low latencies, primarily due to the serverless architecture and the utilization of AWS Lambda functions. The use of Amazon DynamoDB, a fast and flexible NoSQL database, further contributes to the low-latency access to the data. By adopting these technologies, the application maintains consistent performance even under high workloads.

The React-based frontend delivers a smooth and engaging experience, making it easy for users to navigate and interact with the platform. The implementation of infinite scrolling on the events page showcases the application's ability to handle large amounts of data efficiently while maintaining a responsive and seamless interface. Furthermore, the user-centric design approach adopted throughout the application ensures that users find it intuitive and enjoyable to use.

By providing a robust set of features, such as creating events, managing invitations, and facilitating real-time discussions through comments, Eventful successfully enables users to plan and coordinate events with ease. The application's capabilities in handling email notifications and asynchronous tasks ensure that users are kept informed of the latest updates, further enhancing their overall experience.

In terms of scalability, the serverless architecture employed by the application allows it to handle an increasing number of users and events effortlessly. This architecture ensures that resources are allocated dynamically based on demand, resulting in optimal cost-efficiency and reduced operational overhead.

VI. Conclusion and Future Updates

Eventful app addresses the existing gaps in the event management landscape by providing a comprehensive and user-friendly solution for organizing and managing personal events. By focusing on key elements such as low barrier of entry, real-time chat functionality, a simple and robust dashboard, seamless integration with Google Maps, and mobile optimization, Eventful offers a powerful and versatile platform that caters to a wide range of users and their event planning needs.

At its current state, Eventful is a fully usable application that is already being used to plan interpersonal events. However, our team has already pinpointed additional features and implementations to give our users more control over their events.

A future update will add Spotify functionality to the Eventful dashboard, providing users with a seamless way to integrate music into their events. This feature will enable event organizers and attendees to collaboratively create and manage playlists tailored to the event's theme and atmosphere. By connecting their Spotify accounts, users will be able to search for songs, add tracks to the event's playlist, and vote on their favorite tunes.

Next, we are also working on introducing an itemization box to the dashboard, creating a dedicated space for users to manage and view the items each attendee is bringing to the event. The days of multiple desserts and no entrees are over, as the itemization box will streamline the organization of resources, and ensure that all event requirements are met. The itemization box will enable users to add, edit, or remove items they plan to bring, along with a brief description and quantity. Users can also mark items as "confirmed" once they have been procured, which will update the list in real-time for all attendees to see. To further functionality, the itemization box will provide smart suggestions based on the event type, size, and duration.

In addition to manual input, the itemization box will be designed to be automatically populated based on the contents of the event's chat. As users engage in the chat, AWS Comprehend will analyze the messages in real-time to identify any mentions of items, quantities, and the user responsible for bringing them.Once the necessary information is extracted, the application will automatically update the itemization box, adding the detected items to the list alongside the corresponding user's name. The box will be designed to recognize when users update or remove items in the chat, ensuring that the itemization box always reflects the most current information saving users on redundant work.

Social media integration is another way that Eventful will eventually expand to users who frequently use social media. To implement social media integration, Eventful will leverage APIs provided by a variety of social media platforms, allowing users to link their social media accounts to their Eventful profile. Once connected, users will have the option to share event details, send invites, and post updates directly to their social media accounts, all from within the Eventful app. When creating a new event, the host can choose to automatically generate a Facebook event and invite their friends from the platform. Similarly, users can share event updates on Twitter, or create an Instagram story with a countdown timer and a link to the Eventful event page. This seamless integration not only simplifies the invitation process but also helps to promote the event and increase attendance.

Users will also eventually be able to use Eventful to not only create and manage interpersonal events among friends, but also by discovering and attending public functions hosted by strangers. Eventful will expand on the concept of public events, which users create, knowing that their event is public and will be suggested to others. To implement this, Eventful will utilize Amazon Personalize and Amazon SageMaker to analyze user interests, preferences, and past event attendance to curate a list of personalized event recommendations. When users create public events, they can tag them with relevant categories and keywords, allowing the recommendation engine to match these events with users who have similar interests. The app will also use Amazon Location Service to display public events happening in the user's vicinity. Users can then browse through the list of recommended events, view event details, and RSVP directly through the Eventful platform.

We are committed to evolving eventful into the premier interpersonal event management platform. By integrating new features and services that foster seamless communication, collaboration, and discovery, Eventful will empower users to create and manage events with ease, connecting people from diverse backgrounds and interests. As we move forward, our vision is to build a platform that not only simplifies event management but also cultivates a sense of community and belonging, ultimately enhancing social interactions and enriching the lives of our users.