Sanjeet Vinod Jain

CWID : 20012768

CS-554 Lab 5 Choosing Technical Stacks

Scenario 1: Logging

Solution :

1. Storing Log Entries (Elasticsearch): Elasticsearch is chosen for its robust log storage capabilities. It excels in full-text search, real-time indexing, and distributed architecture, making it an efficient and scalable solution for log management.
2. Allowing Users to Submit Log Entries (Express.js for Node.js): Users submit log entries through a user-friendly RESTful API, powered by Express.js for Node.js.
3. Allowing Users to Query Log Entries (Elasticsearch + GraphQL): Elasticsearch, in tandem with GraphQL, offers flexible log querying. Elasticsearch's search and filtering abilities are accessible through GraphQL queries, enabling precise data retrieval.
4. Allowing Users to See Their Log Entries (Frontend Frameworks - React, Next.js, etc.): User-friendly web interfaces are built using frontend frameworks like React or Next.js. They interact with the backend to provide users with intuitive access to log entries, including filtering and searching.
5. Web Server (Node.js with Express.js): Node.js with Express.js serves as the efficient web server. It handles log submissions, exposes the GraphQL API for querying. This combination ensures the system operates reliably and can scale as needed.
6. Redis can also be added in for faster caching of frequently accessed logs from the server.

Scenario 2: Expense Reports

1. Data Storage (Google Cloud SQL): I'll use Google Cloud SQL to store expense data using MySQL or PostgreSQL for structured and reliable data management. When an expense is marked as reimbursed in the database, it serves as the trigger for further actions.
2. Web Server (Google App Engine with Node.js): The web application will be hosted on Google App Engine using the Node.js runtime environment. This provides a scalable and serverless backend. This is where the logic for managing expenses and changing their status resides.
3. Email Handling (Google Cloud SendGrid with RESTful API): To efficiently handle email notifications, I'll integrate Google Cloud SendGrid. Its RESTful API will be used to send email notifications when an expense is marked as reimbursed.
4. PDF Generation (Google Cloud Functions with Puppeteer): I'll create a Google Cloud Function in python to generate PDFs when expenses are marked as reimbursed. The trigger for this function is the change in the expense status in the database.
5. Templating (Google Cloud Firebase Hosting with React): For the frontend, I'll host it on Google Cloud Firebase Hosting and build it using the React framework. React offers dynamic templating for user interfaces. This frontend is where the users interact with the application and may initiate the reimbursement process, which, in turn, triggers the backend processes.

Scenario 3: A Twitter Streaming Safety Service

1. Twitter API: To address the scenario, I'd consider using the Twitter API that aligns with the requirements for real-time tweet monitoring. The Twitter Streaming API, especially the Filter Stream, can be utilized to track specific keywords and phrases for triggering investigations.
2. Web Server Technology: For the web server technology, I'd opt for Node.js and Express.js. Node.js is a solid choice for web development, and Express.js serves as an efficient framework for building web applications.
3. Databases: Storing trigger combinations can be accomplished with a NoSQL database like MongoDB, offering the necessary flexibility for dealing with keyword combinations. As for the historical log of tweets, I'd go for a relational database such as PostgreSQL, which is well-suited for structured data and complex queries.
4. Expandability: To ensure the system's expandability beyond the local precinct, I'd consider implementing microservices and containerization using Docker and Kubernetes. This approach enables a scalable and manageable architecture, allowing for the easy addition of resources as needed.
5. Stability: Stability measures would be essential. This includes proper error handling and logging, monitoring and alerting systems like Prometheus and Grafana, load balancing, regular stress testing, and keeping the system updated with security patches.
6. Real-Time Streaming Incident Report: For real-time incident reporting, WebSockets or Server-Sent Events (SSE) combined with Express.js can be employed. This approach allows for real-time updates and dynamic visualization of threat levels.
7. Media Storage: To store media files, I'd utilize cloud-based object storage services like Amazon S3 or Google Cloud Storage. These services offer scalable and cost-effective storage for images and other media associated with tweets.

Scenario 4: A Mildly Interesting Mobile Application

Geospatial Data:

- For geospatial data, I would utilize Google Cloud Firestore, a serverless, NoSQL database with geospatial indexing and queries.
- I'd store latitude and longitude attributes for 'interesting events' and user locations to handle geospatial data efficiently.

Image Storage:

- To store images for long-term and cost-effective storage, I'd choose Google Cloud Storage.
- Google Cloud Storage offers scalability, durability, and security.
- To ensure fast image retrieval, I'd use Google Cloud CDN, which caches images closer to users for quick access.

API Development:

- I'd write the API using Google Cloud Functions, a serverless compute service.
- Google Cloud Functions would enable me to build a scalable API without managing servers.
- I'd use Google Cloud Endpoints to create a RESTful or GraphQL API that handles CRUD operations for users and 'interesting events.'

Database:

- My choice for the database would be Google Cloud Firestore.
- Firestore supports geospatial data and offers real-time synchronization, making it well-suited for mobile applications. It also scales automatically based on demand.

Administrative Dashboard:

- To develop the administrative dashboard, I'd employ Google Cloud Firebase Hosting, a static web hosting service.
- I'd use web technologies like React or Angular for building an intuitive dashboard.
- To ensure secure access to the dashboard, I'd implement Firebase Authentication for authentication and authorization.

By exclusively utilizing Google Cloud services, I can establish a streamlined, scalable, and secure backend for the mobile application, efficiently managing geospatial data, image storage, API development, and content administration.