# Case study 1

## Big Data Systems and Intelligence Analytics (DAMG 7245)

| Asawari Kadam | kadam.asa@northeastern.edu | 2711195 |
| --- | --- | --- |
| Rutuja Kute | kute.r@northeastern.edu | 2711730 |
| Hariharan Sundaram | sundaram.ha@northeastern.edu | 2915360 |

# WanderPlan: Pioneering Affordable and Tailored Travel Adventures

### **Press Release (PR)**

“Today, we are delighted to unveil WanderPlan, a revolutionary travel itinerary app designed to make every journey not only cost-effective but also uniquely tailored to individual preferences. WanderPlan leverages cutting-edge technology, data-driven insights, and user-centric features to redefine the travel planning experience”

**Key Features:**

1. Year-round Budget-Friendly Travel: WanderPlan helps users discover the most economical travel itineraries for a given destination, ensuring affordable adventures throughout the year.

2. Global Exploration Made Affordable: Explore the world on a budget with WanderPlan's capability to find the most cost-effective travel destinations globally for a specified duration, unlocking endless possibilities for your next vacation.

3. Stay in the Know with Real-time Price Alerts: WanderPlan keeps travellers informed with real-time price alerts for their chosen itinerary, ensuring they never miss out on the best travel deals.

4. Effortless Destination Discovery:Find your perfect destination effortlessly with WanderPlan's powerful search feature, considering user preferences, budget constraints, and more.

5. Tailor Your Terrain with Tags: Utilize tags for destinations based on terrain, helping you filter and find places that match your preferred landscapes and activities.

# 

# FAQ

## External FAQ:

Q: How can WanderPlan help me find the cheapest travel itineraries?

A: WanderPlan utilizes advanced algorithms to analyze pricing data and identify the most budget-friendly travel itineraries for a given destination throughout the year. Simply input your desired location, and let WanderPlan do the rest.

Q: Can WanderPlan recommend budget-friendly destinations globally for a specific travel duration?

A: Yes, WanderPlan's global destination explorer feature allows you to specify your travel duration, and the app will suggest the most economical destinations anywhere in the world.

Q: How do I receive real-time price alerts for my selected itinerary?

A: Enable real-time price alerts in the app, and WanderPlan will notify you instantly when there are price drops or special deals for your chosen itinerary.

### 

## Internal FAQ:

1. General Architecture:

Q: What are the main components of our architecture?

A: Our architecture consists of various components for data ingestion, processing, storage, and analysis. Key components include UI Event Logs API, Destination Service API, Itinerary Service API, Property and Resource Management Service API, Search Service API, CloudWatch Logs, Kafka, MongoDB, Airbyte, Snowflake, Airflow, Spark, RabbitMQ, Elasticsearch, and PostgresSQL.

Q: What are the benefits of this architecture?

A: This architecture offers scalability, performance, flexibility, and real-time insights. Components are decoupled, enabling parallel processing and independent development. Kafka buffers slow tasks, improving API performance. Spark and Snowflake handle large datasets efficiently. MongoDB and Airbyte provide flexible data handling.

2. Data Flow and Processing:

Q: How does user data flow through the system?

A: User interactions captured by the UI Event Logs API are streamed to Kafka. Airflow orchestrates ETL processes to transform and load data into Snowflake. Spark analyzes data for various use cases like user engagement, A/B testing, and dynamic pricing. Results are available in Snowflake and dashboards for visualization.

Q: How are hotel and location data managed?

A: Destination Service API and Property and Resource Management Service API manage hotel and location data. Search Service API utilizes Elasticsearch for indexed data, enabling fast search responses. Updates are processed through Kafka and ETL pipelines to keep Snowflake and databases synchronized.

3. Specific Components and Technologies:

Q: Why do we use Kafka?

A: Kafka decouples APIs from slow tasks like database inserts and data lake transfers, improving API performance and scalability. It buffers messages asynchronously, handling large data volumes efficiently.

Q: What are the advantages of using Snowflake?

A: Snowflake combines both data lake and data warehouse functionalities. This provides flexibility for storing raw and structured data while enabling efficient analytics on historical and real-time data.

Q: What are some use cases for Apache Spark?

A: Spark utilizes distributed clusters for fast analytics on large datasets. We use it for various tasks like usability analysis, system performance monitoring, dynamic pricing, destination popularity recommendations, product performance, demand forecasting, and price forecasting.

4. Data Governance and Security:

Q: How do we ensure data quality and accuracy?

A: Airflow pipelines include data cleansing and validation steps to ensure data quality. We monitor data quality metrics and utilize data lake cleansing tools to maintain historical data integrity.

Q: How is data security protected?

A: We implement strong security measures like secure data storage, authentication, and authorization mechanisms to protect user data and prevent unauthorized access. CloudWatch Logs and system monitoring help detect potential security breaches.

# Architecture Diagram

Components used:

**UI Event Logs API:** Web API to handle UI Event Logs and screens viewed by user on web and mobile interfaces. This data can be used to analyse app engagement, time on each screen, usability research and A/B testing

**Destination Service admin API:** This service can be used to add new locations and hotels by admin to the backend

**Itinerary Service API:** This is a group of services which together help user complete their itinerary creation journey

**Property and Resource Management Service admin API(Booking.com):** This API can be used to update the prices of hotels and other transportation services

**Search Service API:** For performing hotel and destination search. It is indexed using elastic search for fast search performance

**CloudWatch Logs:** For obtaining system logs and performance metrics especially useful for performance evaluation analytics, finding uptimes, potential bugs and security breaches through data analysis

**Kafka:** Used for holding messages as a broker and eventually streaming them to the consumers. It is used to decouple the slower parts like database insertion and analytics to improve API performance

**MongoDB:** Used as a log store. This data is periodically transferred to the data lake using connectors like AirByte

**Airbyte:** Connector to transfer data from MongoDB to data lake and also acts like a message broker sink to transfer data from RabbitMQ to the data lake

**Snowflake:** It is used as both a data lake and data warehouse. A data lake stores all data in raw, unstructured format so that data analysts and data scientists can refer back to the original data form for best decisions and analysis. Data Warehouse is more of structured and readable data which has been formatted according to the use cases of the data analysis team

**Airflow:** This is used to perform the ETL tasks required to build the data warehouse using raw data in the data lake.

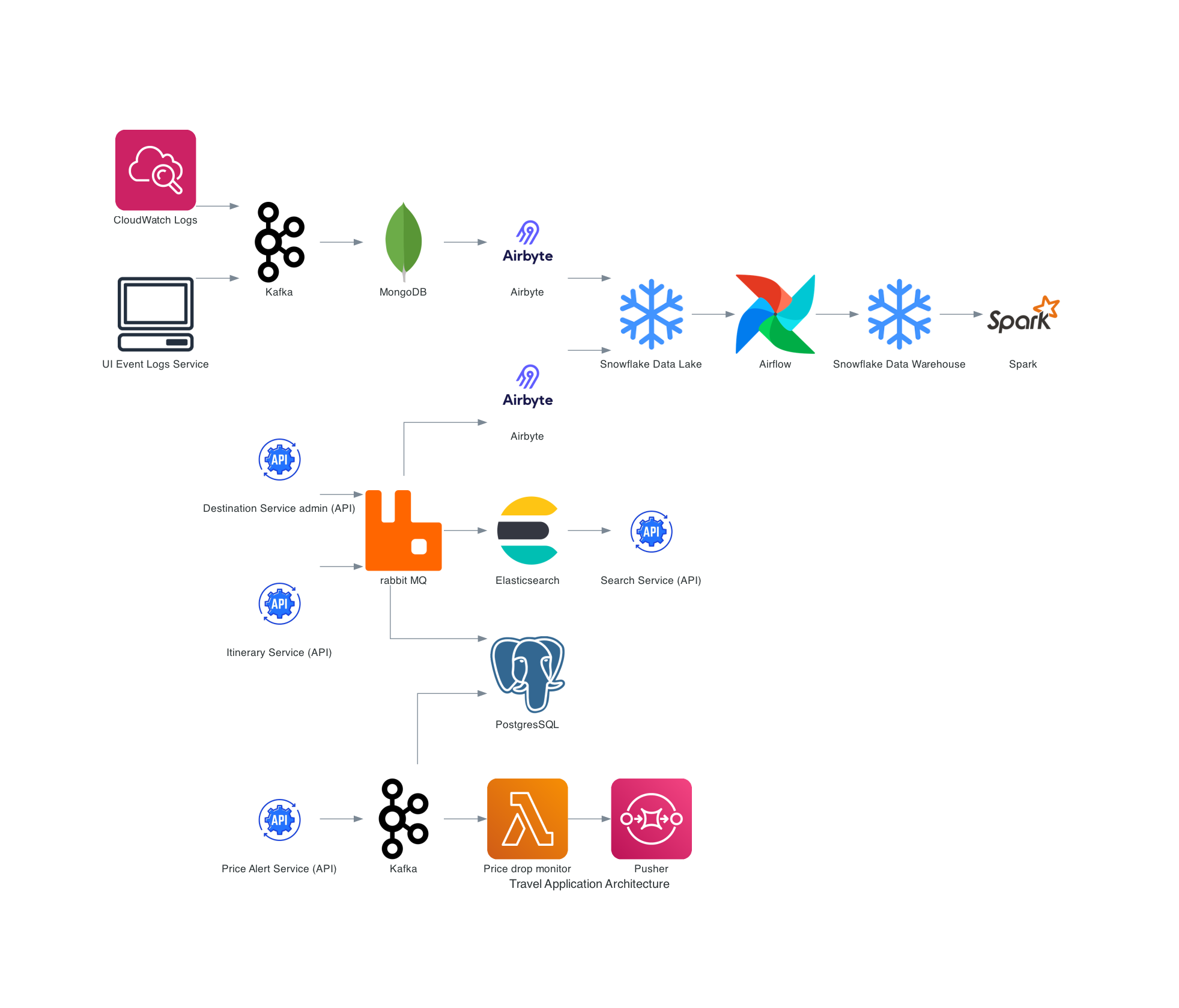
**Apache Spark:** Used to run the analytics jobs over large distributed clusters for accelerated performance. Usability analytics, system performance, dynamic pricing, destination popularity recommendations, product performance, demand forecasting and price forecasting are some of the use cases.

**Rabbit MQ:** Used to decouple APIs from elastic search indexing, database insertion and data lake transfer

**Elasticsearch:** Used to index data of locations and hotels for faster search response

**PostgresSQL:** A relational database for storing hotel data, location data, user data, itinerary records and payment information

**Pusher:** Used for sending price change notifications to users who have subscribed to the service



# Research findings and design decisions

1. Real-time Demand Prediction and Dynamic Pricing:

* Research Findings:
  + User engagement data from UI Event Logs API and historical booking data revealed significant fluctuations in demand across different locations and times.
  + Traditional pricing models based on static seasonal adjustments were deemed insufficient for capturing these dynamic trends.
* Decisions:
  + Implemented micro-batch ETL with Apache Spark to analyse near real-time user activity and predict short-term demand variations.
  + Integrated the demand predictions with the Property and Resource Management Service API to enable dynamic pricing adjustments for hotels and services.

2. User Engagement and User Interface/User Experience (UI/UX) Optimization:

* Research Findings:
  + UI Event Logs API analysis identified underutilised features and screen navigation bottlenecks.
  + A/B testing further validated the impact of specific UI/UX elements on user engagement and conversion rates.
* Decisions:
  + Prioritized development efforts based on insights from UI event log analysis and A/B testing results.
  + Streamlined user journeys and optimized key UI elements to improve engagement and conversion rates.

3. System Performance and Scalability:

* Research Findings:
  + CloudWatch Logs analysis revealed performance bottlenecks during peak loads, impacting API response times and user experience.
  + Traditional monolithic architecture struggled to handle increasing data volumes and processing demands.
* Decisions:
  + Could possibly use a microservices-based approach with Kafka for decoupling and asynchronous processing.
  + Employed Airflow and Spark for scalable ETL and analytics on large datasets.

4. Data Management and Insights Generation:

* Research Findings:
  + Scattered data sources across MongoDB and PostgresSQL hampered comprehensive analysis and insights generation.
  + Lack of standardization in data formats and structures impeded efficient data exploration and reporting.
* Decisions:
  + Implemented Airbyte for automated data transfer and consolidation from MongoDB to Snowflake.
  + Established data governance policies and standardized data formats across all storage systems.
  + Utilized Spark for advanced analytics and generated insightful reports on various aspects like destination popularity, competitor comparison, and payment failures.

5. Future Research and Development:

* Investigate streaming analytics platforms for near real-time insights and automated decision-making.
* Explore serverless technologies for cost-effective scaling and dynamic resource allocation.
* Implement data cleansing and quality assurance processes to ensure reliable analysis and avoid biases.
* Conduct ongoing user research and A/B testing to continuously improve user experience and engagement.

# Justifications for the chosen architecture

Here are some key justifications for the chosen architecture in your case study:

1. Balancing Real-time and Historical Needs:

* Micro-batch ETL for real-time demand prediction: Allows for accurate, near real-time predictions of short-term demand fluctuations through frequent data processing intervals.
* Larger batching for historical analysis: Efficiently analyzes larger datasets for A/B testing, system performance evaluations, and historical trends without sacrificing real-time performance.

2. Scalability and Performance:

* Kafka for decoupling: Improves API performance by buffering slow tasks like database updates and data lake transfers asynchronously.
* Apache Spark for distributed analytics: Enables fast processing of large datasets for various use cases, improving overall analytical performance.
* Snowflake as both data lake and data warehouse: Offers flexibility for storing raw and structured data while supporting efficient analytics.

3. Data Management and Flexibility:

* MongoDB for unstructured data: Provides a flexible storage solution for UI event logs, effectively handling unstructured data formats.
* Airbyte for automated data transfer: Simplifies data movement between MongoDB and Snowflake, reducing manual effort and streamlining the pipeline.

4. User Experience and System Monitoring:

* Pusher for real-time price updates: Enhances user experience by delivering immediate notifications about price changes for subscribed users.
* CloudWatch Logs for performance and troubleshooting: Provides valuable insights into system health and allows for proactive identification and resolution of potential issues.

5. Modularity and Maintainability:

* Microservices architecture with Airflow and Spark: Facilitates independent development and deployment of components, simplifying updates and maintenance.
* Decoupling with RabbitMQ and Airbyte: Improves fault tolerance and scalability by isolating components from failures and enabling parallel processing.

# Considerations for future improvements

1. Caching and Data Optimization:

* Explore data partitioning and compression techniques to optimize storage requirements and improve query performance on large datasets in Snowflake.

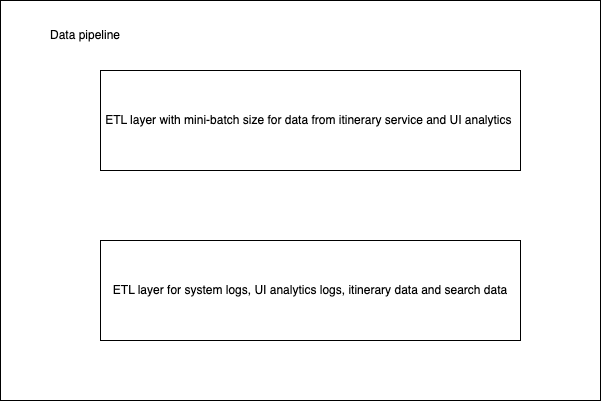
2.Real-time Weather Updates:

* Display current weather conditions and forecasts for chosen destinations directly on the search results page.
* Implement dynamic icons or color coding to visually showcase weather highlights (sunny, rainy, etc.).
* Allow users to filter destinations based on desired weather preferences (e.g., warm and sunny vs. cool and rainy).

3. AI and Machine Learning Integration:

* Implement AI-powered recommendation engines to personalize user experiences by suggesting relevant destinations, activities, and accommodation based on their preferences and historical data.
* Utilize machine learning models for anomaly detection in system logs and performance metrics to proactively identify and troubleshoot potential issues.

# Data pipeline



In our use case we needed two kinds of data processing speeds.

1. Fast ETL and analytics for real time demand prediction for dynamic pricing of properties and services. This can be achieved by using an ETL layer with micro-batch processing size. We chose ETL over other architectures since it would be sufficient for our use case having small batch size to perform the transformation before loading, making the further use of data in Spark easier and quicker.Data refresh would be on hourly basis
2. Periodic ETL and analytics with a larger batch size (considering data for a day and historical) for A/B testing, system performance evaluations, UI/UX usability research, popular destinations, demand and price forecasting, payment failures, user engagement and competitor comparison in terms of price and hits.