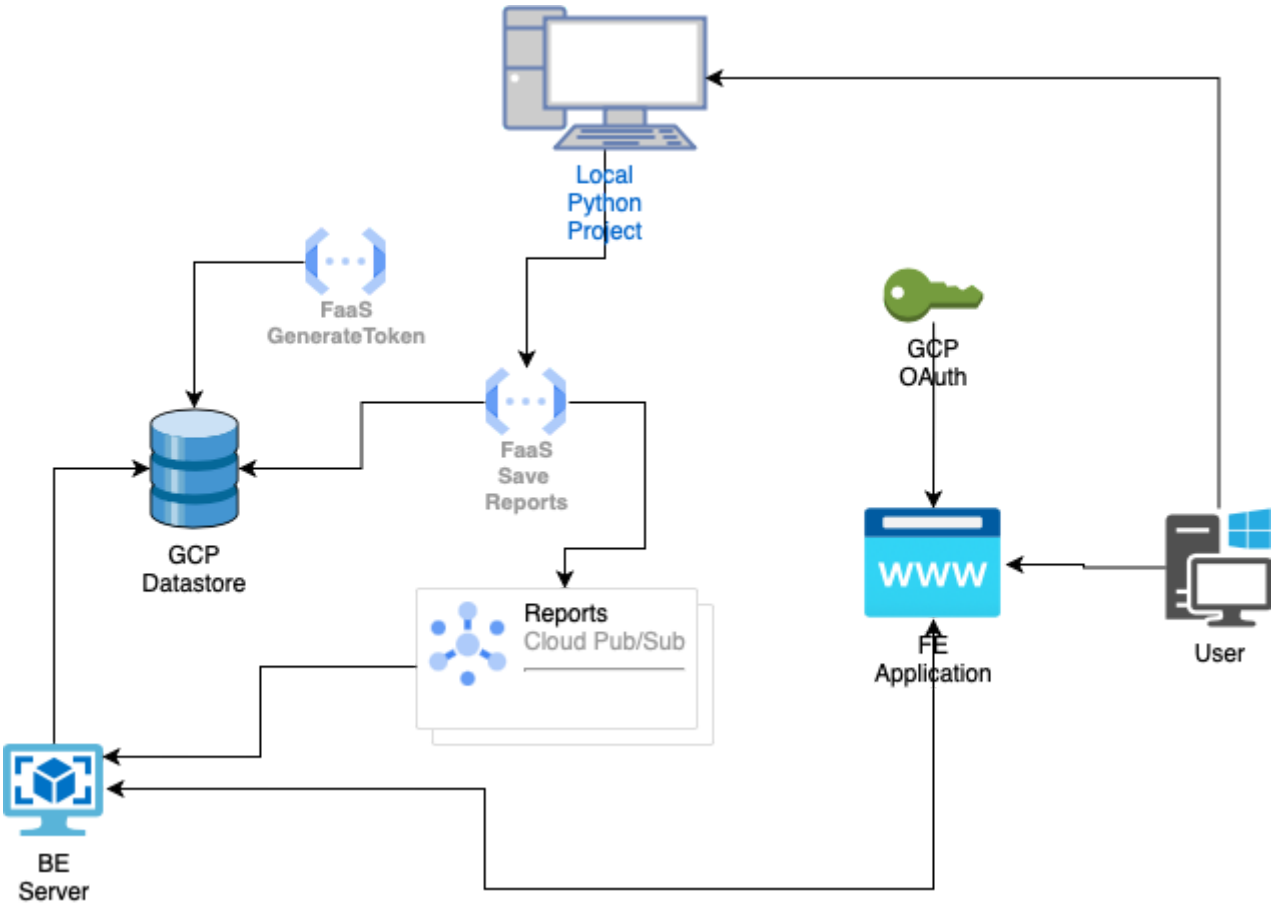
Report

Requirements

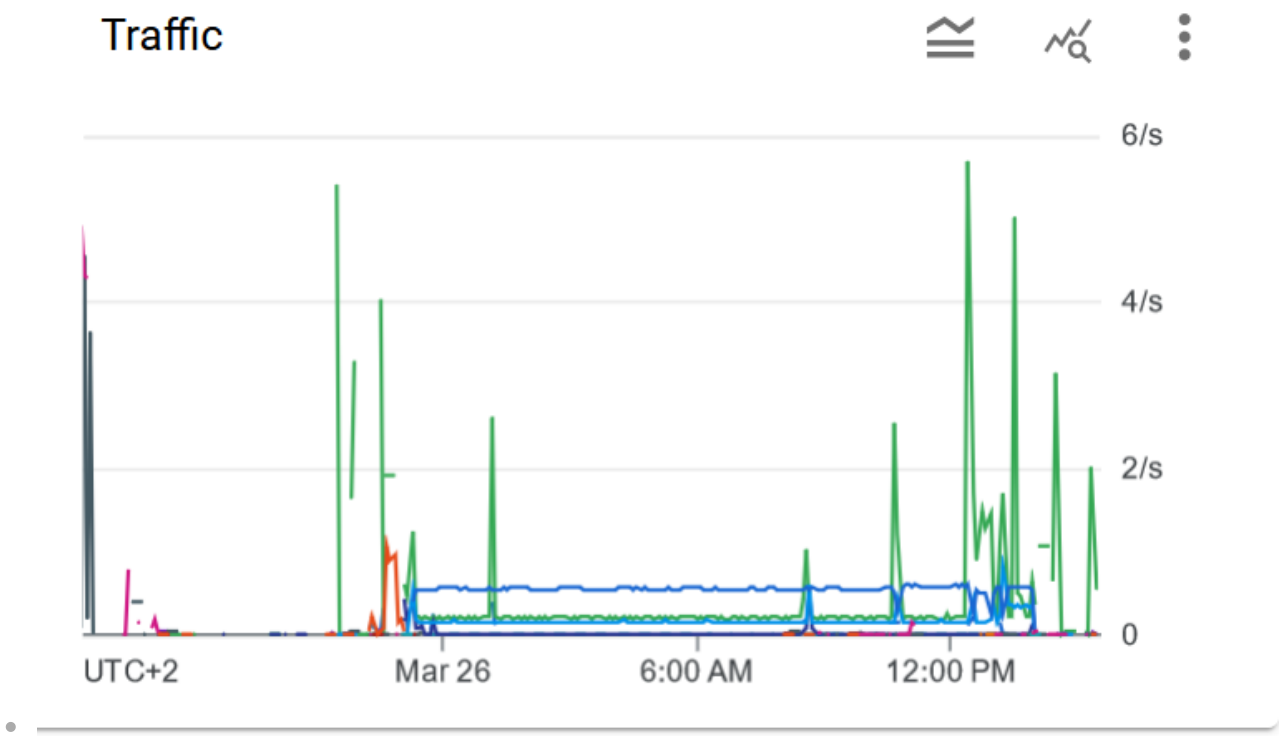
- Create a distributed application that uses a cloud ecosystem (e.g. Google, Amazon, et.al.). Requirements - the application has at least three components:
  - one is using at least three native cloud services (one is Stateful)
  - the other can be an on-premise application but is required to use at least one service from Google Cloud (or similar services for other Clouds Providers): Cloud BigTable, BigQuery, Cloud DataProc, or services such as Cloud Endpoints, Prediction API
  - the other is a FaaS (AWS Lambda, Google Cloud Functions, IBM OpenWhisk or Microsoft Azure Functions)
- At least one Real-Time Web Technology (e.g. WebSocket) must be used in a proper manner in your system. (Don't forget about performance metrics that fit to your distributed application). More details are offered during the laboratory.
- Create a scientific report based on the services chosen in point A, which are used in various distributed architectures of real systems (e.g. Twitter). Analyze the characteristics of your system: performance, latency, reliability, transparency dimensions et.al (see C1-2, C6)

Results

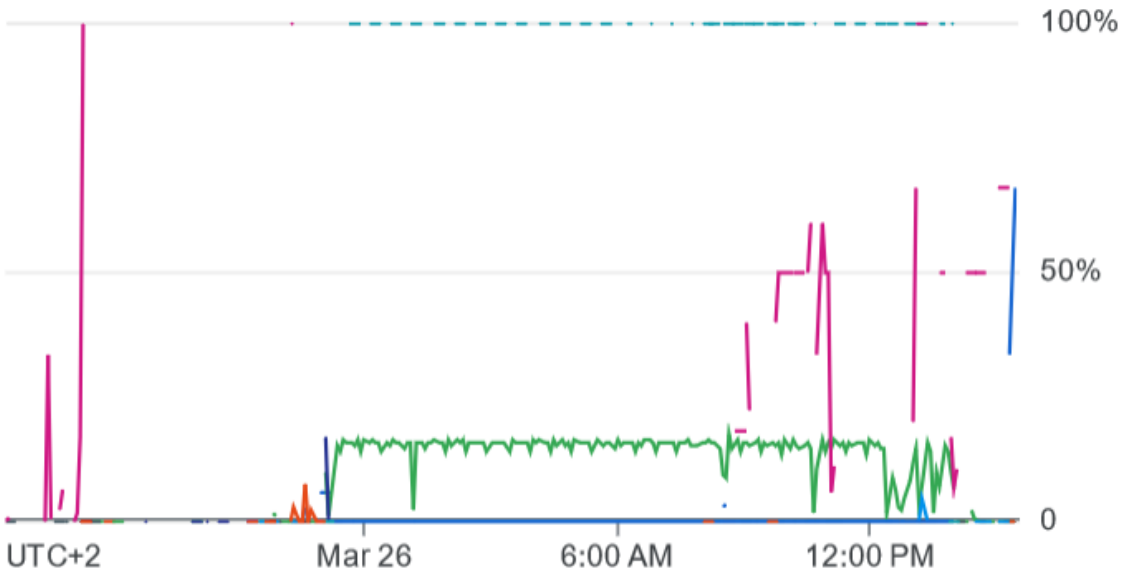
Architecture:


Local
Python
Project
FaaS
GenerateToken
GCP
OAuth
FaaS
Save
Reports
GCP
Datastore
Reports
Cloud Pub/Sub
www
FE
Application
User
BE
Server

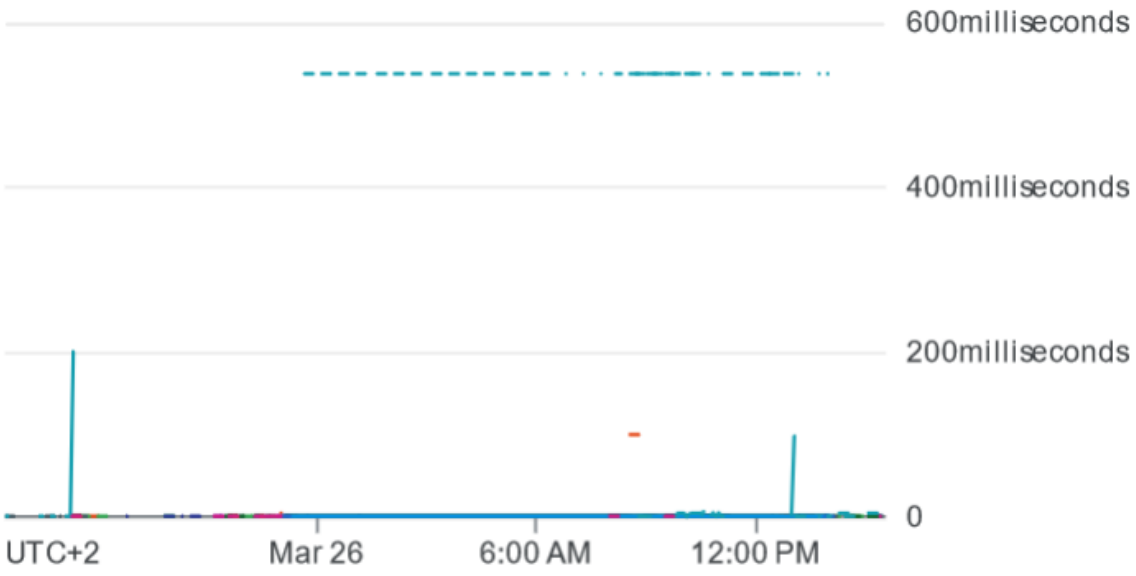
Server


Traffic
6/s
4/s
2/s
0
UTC+2
Mar 26
6:00 AM
12:00 PM

Errors



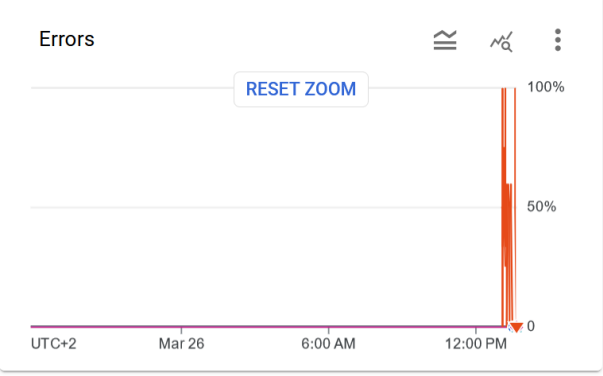
Median latency

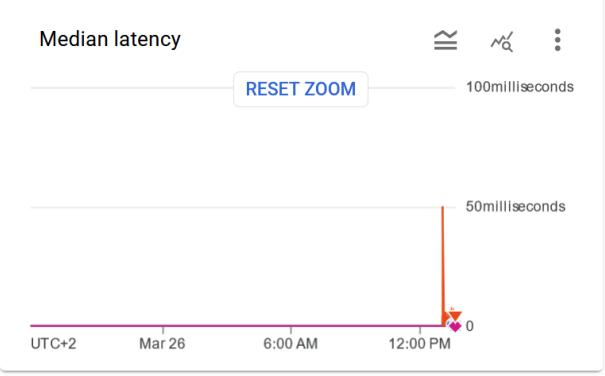


Filter Filter

Name	Requests	Errors (%)	Latency, median (ms)	Latency, 95% (ms)
Cloud Monitoring API	29,593	0	32	62
Compute Engine API	27,778	7	126	478
Cloud Logging API	9,394	0	68	126
Cloud Pub/Sub API	7,137	1	1	31
Cloud Datastore API	3,181	0	13	60
Kubernetes Engine API	1,604	1	31	497
Cloud Run Admin API	1,195	0	27	187
Artifact Registry API	759	1	29	185
Cloud Functions API	101	0	85	491
Cloud Autoscaling API	94	100	536,870	536,870
Cloud Firestore API	36	0	51	429
Network Connectivity API	16	0	93	2,516
Cloud DNS API	10	0	49	480
Identity and Access Management (IAM) API	8	0	54	117
Cloud AI Companion API	5	0	6,291	117,440
Binary Authorization API	2	0	196	255

Frontend:





Name	Requests	Errors (%)	Latency, median (ms)	Latency, 95% (ms)
Cloud Logging API	17,269	0	61	123
Cloud Monitoring API	11,310	0	28	60
Cloud Pub/Sub API	73	49	4,128	72,980
Compute Engine API	25	0	116	917

Services

Pub/Sub for Event-Driven Communication:

- Google Cloud Pub/Sub is a fully-managed real-time messaging service that allows you to send and receive messages between independent applications. It enables asynchronous messaging between system components through topics and subscriptions.
- Homework example: Cloud Pub/Sub is utilized as follows: a topic corresponds to an individual project identified by its respective token, whereby a subscriber is configured to await projects, filtering for the specific project of interest. Subsequently, a publication is orchestrated to dispatch the project token alongside serialized project data.
  - **Performance:** Google Cloud Pub/Sub ensures high system efficiency and throughput by enabling asynchronous messaging between system components. It efficiently handles large volumes of messages, ensuring optimal performance even under heavy loads.
  - **Reliability:** Pub/Sub guarantees reliable message delivery, ensuring high availability and fault tolerance. It automatically retries message delivery in case of failures, ensuring message delivery with at-least-once semantics.
  - **Flexibility:** Pub/Sub offers flexibility in message delivery and processing, supporting both real-time and batch processing scenarios. It allows subscribers to control message acknowledgments, enabling customized message processing based on application requirements.
  - **Usability:** Pub/Sub provides a user-friendly interface for managing topics, subscriptions, and message delivery. It offers comprehensive documentation and SDKs for various programming languages, simplifying integration and development processes.
- Real life example: Sky By landing diagnostic data from set-top boxes directly in Cloud Pub/Sub, Sky eliminates data loss caused by bottlenecks in server capacity.

Cloud Functions for Serverless Computing:

- Google Cloud Functions is a lightweight compute solution for developers to create single-purpose, stand-alone functions that respond to Cloud events without the need to manage a server or runtime environment. It allows developers to execute code in response to events triggered by Pub/Sub messages or HTTP requests.

- Homework example: Cloud functions were selected for dual primary functionalities within the system architecture: Firstly, to generate tokens, facilitating secure authentication and access control mechanisms. Secondly, they were utilized to receive reports, enabling streamlined data transmission and processing workflows essential for comprehensive system monitoring and analysis.
    - **Performance:** Cloud Functions execute code rapidly, minimizing response time and enhancing overall system efficiency. They are designed to scale automatically based on demand, ensuring optimal resource utilization and responsiveness.
    - **Reliability:** Cloud Functions operate in a highly available environment, with built-in redundancy and fault tolerance mechanisms.
    - **Security:** Cloud Functions ensure confidentiality, integrity, and accountability through secure coding practices and IAM roles. They support granular access controls, allowing developers to define fine-grained permissions for accessing resources.
    - **Flexibility:** Cloud Functions offer flexibility in resource allocation and scaling, adapting to varying workload demands seamlessly. Developers can configure function triggers and timeouts, customize runtime environments, and define dependencies as needed.
- Real life example: Auchan France*(bonus: Pub-Sub)*. Auchan uses Cloud Functions to further optimize data management and analytics and improve performance across its 500 applications. This, in addition to the collaboration with Elastic, has enabled the company’s data flow analysts to work almost entirely in its data exchange platform to push data without the need for virtual machines or other infrastructure elements.

Datastore for NoSQL Data Storage:

- Datastore is a highly scalable NoSQL database for your applications. Datastore automatically handles sharding and replication, providing you with a highly available and durable database that scales automatically to handle your applications' load. Datastore provides a myriad of capabilities such as ACID transactions, SQL-like queries, indexes, and much more.
- Homework example: Employing a datastore, particularly a NoSQL solution, was deemed indispensable for the storage and organization of heterogeneous project information. The decision to opt for such a solution stemmed from the inherent flexibility it offers, accommodating the varying structures inherent in error data dispatched to the server. This adaptability ensures robust data management capabilities, enabling effective analysis and retrieval processes essential for the seamless operation and maintenance of the system.
    - **Performance:** Google Cloud Datastore offers fast read and write operations, optimizing system throughput and efficiency. It scales horizontally to handle growing datasets and user traffic, ensuring consistent performance under varying workloads.

    - **Reliability:** Datastore ensures data availability and fault tolerance through automatic replication and failover mechanisms. It provides strong consistency guarantees, ensuring that data is always up-to-date and accessible.
    - **Security:** Datastore provides granular access controls and encryption options, safeguarding data privacy and integrity. It supports IAM roles and access policies, allowing administrators to control access at the dataset and entity level.
    - **Transparency:** Datastore offers transparency through monitoring and logging capabilities, providing visibility into data access and modifications. It generates audit logs for all API operations, enabling administrators to track changes and diagnose issues effectively.
- Real life example: Airbus Defence and Space*(bonus: Pub-Sub)*. Over time, as technology grew more sophisticated and Airbus added to its fleet of satellites, it built the capacity to provide better quality data and imagery at increasingly large volumes. the data was held in Cloud Datastore, for easy scalability. With the images processed, Airbus extracted the data and placed it in the existing library. From here, the customer could access the imagery as part of a seamless, connected digital atlas for streaming or downloading.

WebSocket for Real-Time Bidirectional Communication:

- WebSocket is a real-time technology that enables bidirectional, full-duplex communication between client and server over a persistent, single-socket connection. The WebSocket connection is kept alive for as long as needed (in theory, it can last forever), allowing the server and the client to send data at will, with minimal overhead.
- Homework example: WebSocket was employed as the communication protocol to interface with the server, facilitating the establishment of a reliable and real-time connection. Through this connection, we were able to obtain and exchange essential project information, encompassing user details, error logs, file names, and other pertinent data crucial for seamless collaboration and system functionality.
    - **Latency**: WebSocket technology minimizes latency by enabling persistent, bidirectional communication between clients and servers. It establishes a long-lived connection that allows real-time data exchange with minimal overhead.
    - **Reliability:** WebSocket ensures high availability and fault tolerance, enhancing user experience levels and system reliability.
    - **Security:** WebSocket supports confidentiality, integrity, and audit trails through secure communication protocols and encryption.
    - **Usability:** WebSocket offers a seamless and interactive user experience, facilitating real-time updates and notifications.
- Real life example: Uber. WebSockets play a crucial role in Uber's system architecture by facilitating real-time bidirectional communication between the server and clients (riders and drivers). Unlike traditional HTTP requests that work on a request-response basis, WebSockets establish a persistent connection, enabling continuous data flow without the need for repeated requests. This is essential for Uber, where the real-time

location of drivers, trip status updates, and dynamic pricing information must be communicated instantly to the users.

App Engine for Managed Application Deployment:

- App Engine is a fully managed platform-as-a-service (PaaS) that enables developers to build and deploy scalable web applications and APIs without managing the underlying infrastructure. App Engine automatically handles provisioning, scaling, and load balancing, allowing developers to focus on writing code.
- Homework example: App Engine was chosen as the deployment platform for its managed environment, enabling seamless deployment and scaling of the application without the need for manual intervention. This allowed developers to focus on building and iterating on features rather than managing infrastructure.
    - **Scalability**: App Engine automatically scales resources based on incoming traffic, ensuring optimal performance and availability for applications. It can handle sudden spikes in traffic without manual intervention, providing a seamless experience for users.
    - **Managed Infrastructure**: App Engine abstracts away the underlying infrastructure, allowing developers to focus on application development. It handles tasks such as provisioning, monitoring, and maintenance, reducing operational overhead.
    - **Built-in Services**: App Engine offers built-in services such as Datastore, Memcache, and Task Queues, simplifying common development tasks. Developers can leverage these services to build robust and scalable applications without managing additional components.
    - **Deployment Flexibility**: App Engine supports multiple programming languages, including Java, Python, Node.js, and Go, providing flexibility for developers to choose the language and framework that best suits their needs.
- Real life example: Barilla(bonus: Pub-Sub). Meanwhile Google App Engine enabled the Injenia team to deliver updates and new versions at speed, as part of a feedback process with workers who offered suggestions through a link to Forms embedded in the app. Because CollaborAction is a mobile app built on Google App Engine, scaling to meet new demand has been simple.

OAuth for Secure Authorization and Authentication:

- OAuth is an open standard for secure authorization and authentication, enabling users to grant access to their resources without sharing their credentials. OAuth is widely used in modern web and mobile applications to provide seamless access to third-party services and APIs.
- Homework example: OAuth was implemented for secure authentication and authorization mechanisms within the system architecture. By leveraging OAuth, users

could securely authenticate and authorize access to their data, ensuring confidentiality and integrity throughout the application workflow.

  - **Secure Authentication**: OAuth enables secure authentication without exposing user credentials to third-party applications. Users authenticate with the OAuth provider (e.g., Google, Facebook) and grant permissions to access their resources.
  - **Authorization Flexibility**: OAuth supports fine-grained access control through scopes, allowing users to grant specific permissions to applications. Developers can define scopes that govern the types of actions an application can perform on behalf of the user.
  - **Token-Based Authentication**: OAuth uses tokens (access tokens, refresh tokens) for authentication and authorization, reducing the risk of credential theft and unauthorized access. Tokens have limited lifetimes and can be revoked by the OAuth provider if compromised.
  - **Third-Party Integration**: OAuth facilitates seamless integration with third-party services and APIs by providing a standardized authentication and authorization mechanism. Developers can leverage OAuth to enable single sign-on (SSO) and access to external resources.
- Real life example: Twitter *(bonus: Pub-Sub)*. Twitter utilizes OAuth for secure authentication and authorization on its platform. Users can log in to Twitter using OAuth providers such as Google or Apple, ensuring secure access to their accounts and personalized content. This enhances user privacy and simplifies the login process, leading to a seamless social media experience for Twitter users.