Electronics and Computer Science

Faculty of Engineering and Physical Sciences

University of Southampton Malaysia

COMP3207
CLOUD APPLICATION DEVELOPMENT

Task 5: Monitoring, Logging and Scaling - CloudTalk

Name: Hin Jian Heng

Student ID: 33399948

Email: jhh1e22@soton.ac.uk

Lecturers	Dr. Syed Hamid Hussain Madni
	Dr. Muhamad Najib Zamri

Submission Date: 5/5/2025

1. Table of Contents

1. Table of Contents 2
2. Introduction 3
3. Monitoring 4
   3.1 Firebase Monitoring 4
   3.2 Cloud Function/ Cloud Run Monitoring 6
   3.3 Cloud Monitoring 7
      3.3.1 Application Health Overview Dashboard 9
      3.3.2 Authentication and User Login 10
      3.3.3 Core Feature Performance 10
      3.3.4 User and Group Management 11
4. Logging 13
   4.1 Server Logging 13
   4.2 Cloud Logging 14
5. Scaling 18
   5.1 Firebase & Cloud Run Auto-Scaling 18
   5.2 Kubernetes Scaling 21

2. Introduction

To ensure the reliability, performance, and user experience of the CloudTalk application, it is crucial to implement a comprehensive monitoring, logging, and scaling strategy. Monitoring enables us to understand the application's health status and resource usage in real time and proactively identify potential problems. Detailed logging provides important data support for troubleshooting, performance analysis, and security audits. An effective scaling mechanism ensures that the application can remain responsive and stable under high concurrency or burst traffic to meet user needs. This section will detail the monitoring, logging, and scaling solutions implemented by the CloudTalk application in the Firebase and Google Cloud Platform (GCP) environments.

3. Monitoring

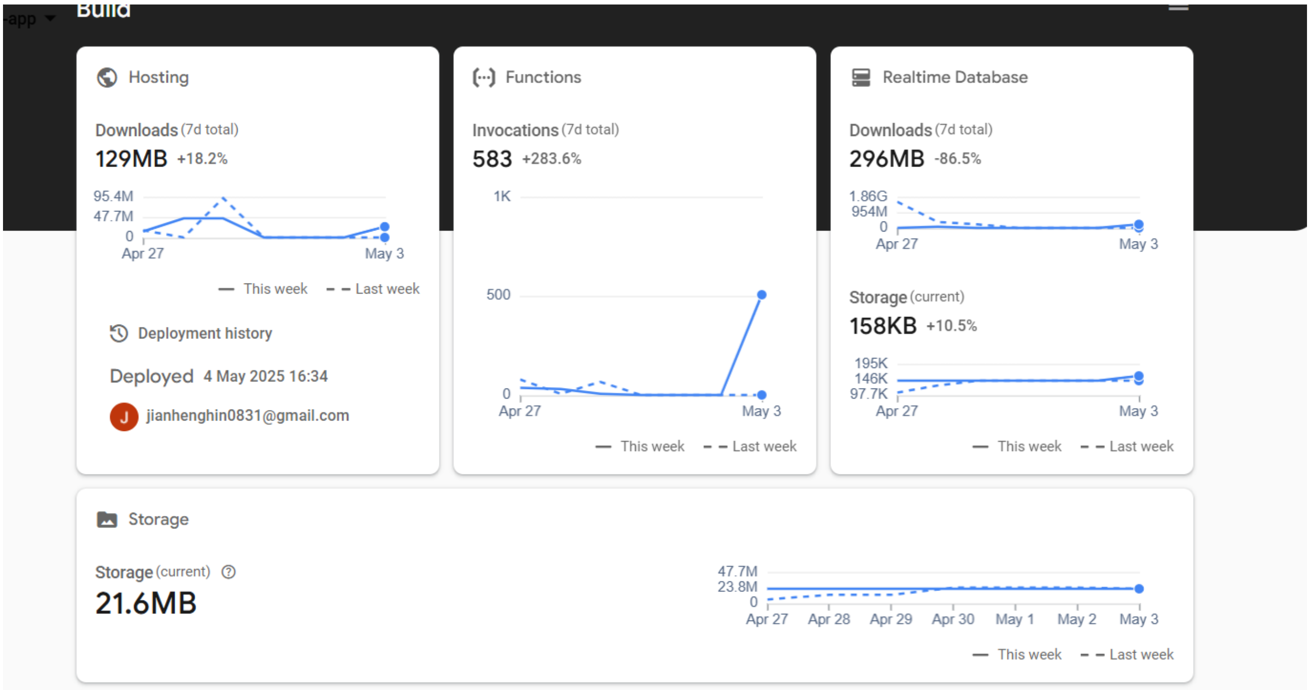
We adopted a multi-layered monitoring strategy, combining Firebase built-in tools and Google Cloud Monitoring to fully understand the CloudTalk application's operating status.

3.1 Firebase Monitoring

Using the automatically set up monitoring dashboard provided by the Firebase console, we can intuitively track the usage of various core Firebase services. This includes:

- Hosting: Monitor the download volume and traffic of website content to evaluate user access patterns and bandwidth consumption.
- Functions: Track the number of Cloud Functions calls to understand the load pressure of backend logic.
- Realtime Database: Monitor the number of database connections, read and write operations, and storage space usage to ensure the performance and capacity of data services.
- Storage: Monitor the usage and download of file storage.

These basic monitoring data help us quickly understand the overall usage trend and resource consumption of the service, providing a basis for capacity planning and cost management.



Hosting

Downloads (7d total)
129MB +18.2%

Deployment history

Deployed 4 May 2025 16:34

jianhenghin0831@gmail.com

Functions

Invocations (7d total)
583 +283.6%

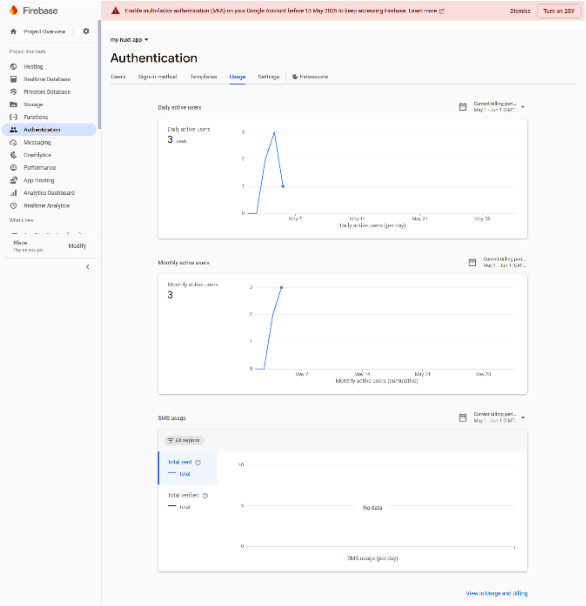
Realtime Database

Downloads (7d total)
296MB -86.5%

Storage (current)
158KB +10.5%

Storage

Storage (current)
21.6MB



Enable multi-factor authentication (MFA) on your Google Account before 13 May 2025 to keep accessing Firebase. Learn more

my-nuxt-app

Authentication

Daily active users — Current billing peri... May 1 - Jun 1 (GMT...

Daily active users: 3 peak

Monthly active users — Current billing peri... May 1 - Jun 1 (GMT...

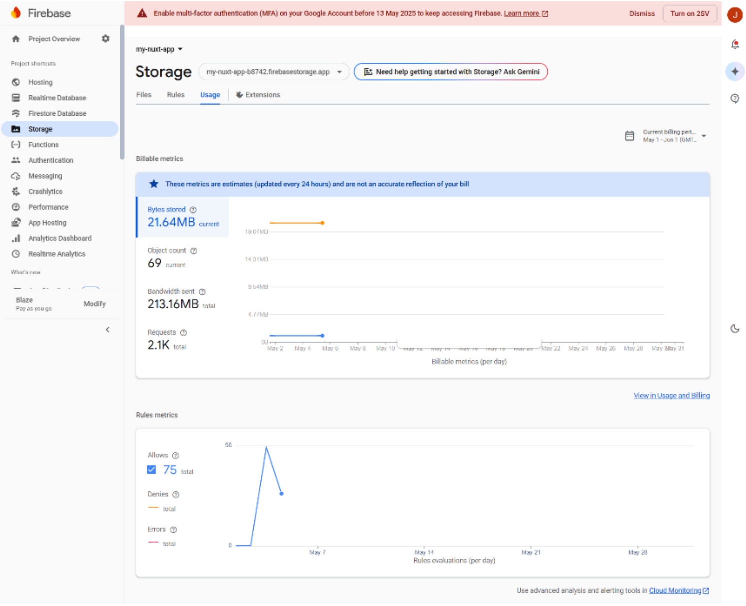
Monthly active users: 3

SMS usage — Current billing peri... May 1 - Jun 1 (GMT...

Total sent — total

Total verified — total

No data



Enable multi-factor authentication (MFA) on your Google Account before 13 May 2025 to keep accessing Firebase. Learn more

my-nuxt-app

Storage — my-nuxt-app-b8742.firebasestorage.app — Need help getting started with Storage? Ask Gemini

Current billing peri... May 1 - Jun 1 (GMT...

Billable metrics

These metrics are estimates (updated every 24 hours) and are not an accurate reflection of your bill

Bytes stored: 21.64MB current

Object count: 69 current

Bandwidth sent: 213.16MB total

Requests: 2.1K total

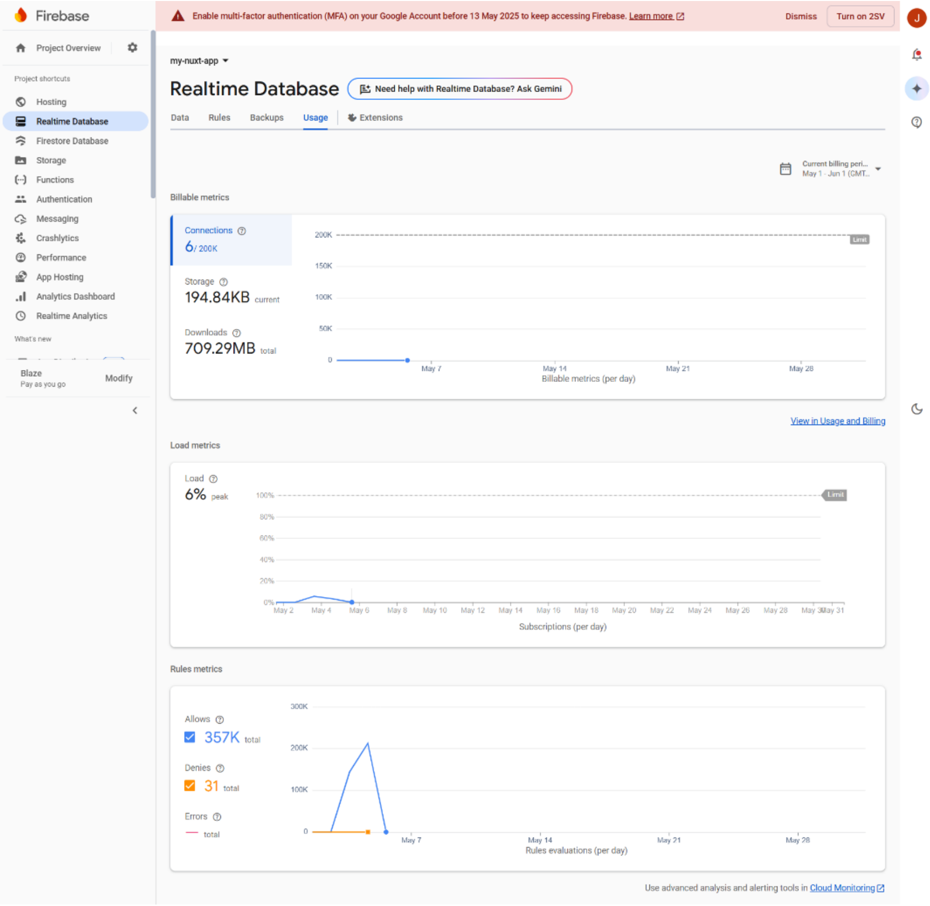
Rules metrics

Allows: ☑ 75 total

Denies: — total

Errors: — total

Use advanced analysis and alerting tools in Cloud Monitoring



Enable multi-factor authentication (MFA) on your Google Account before 13 May 2025 to keep accessing Firebase. Learn more

my-nuxt-app

Realtime Database — Need help with Realtime Database? Ask Gemini

Current billing peri... May 1 - Jun 1 (GMT...

Billable metrics

Connections: 6 / 200K

Storage: 194.84KB current

Downloads: 709.29MB total

Load metrics

Load: 6% peak

Rules metrics

Allows: ☑ 357K total

Denies: ☑ 31 total

Errors: — total

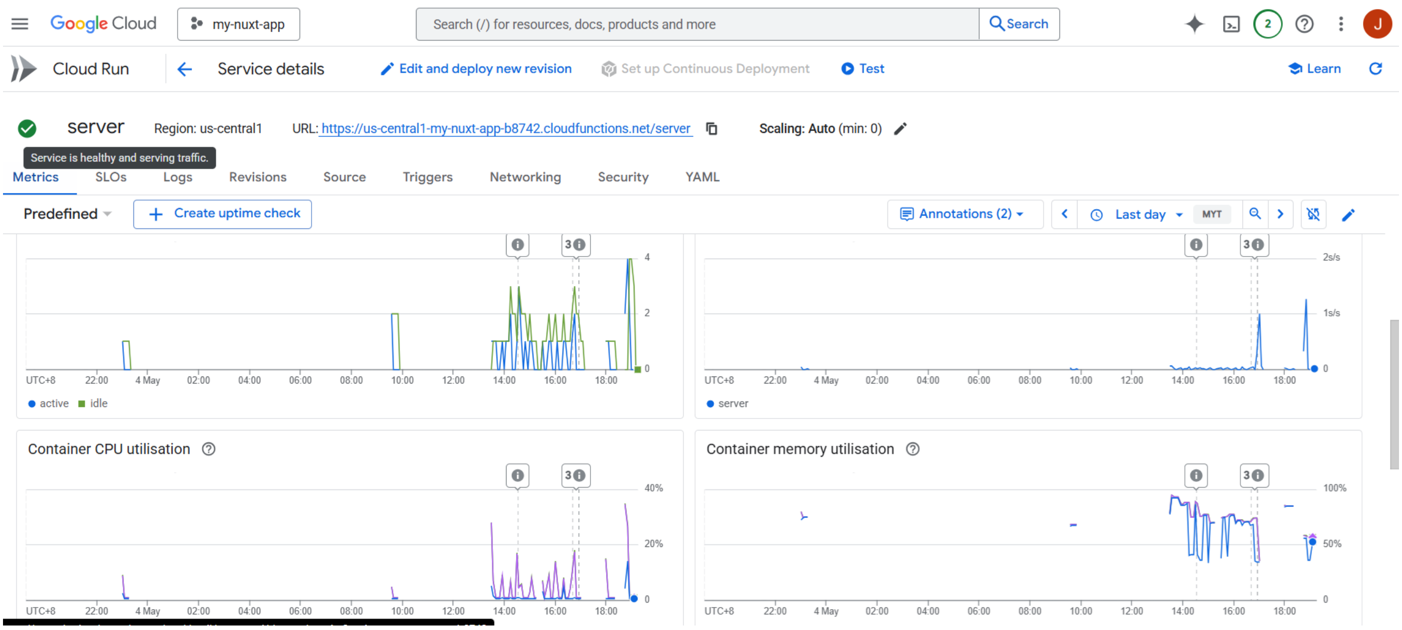
Use advanced analysis and alerting tools in Cloud Monitoring

3.2 Cloud Function/ Cloud Run Monitoring

CloudTalk's backend logic is deployed as Cloud Functions and runs in the Google Cloud Run environment. We use the metrics monitoring feature of the Cloud Run service details page to keep a close eye on key runtime performance indicators, especially:

- Container CPU utilisation: reflects the computing resource consumption of the function instance when processing requests. High utilisation may indicate the need to optimise the code or increase the instance.
- Container memory utilisation: shows the memory usage of the function instance. Sustained high memory usage may lead to performance degradation or errors.

By monitoring these indicators, we can promptly discover the performance bottlenecks of the backend service, evaluate the rationality of resource configuration, and provide data support for the automatic expansion strategy.



server Region: us-central1 URL: https://us-central1-my-nuxt-app-b8742.cloudfunctions.net/server Scaling: Auto (min: 0)

Service is healthy and serving traffic.

3.3 Cloud Monitoring

To gain deeper and more detailed insights into the performance, availability, usage and user experience of specific features within the CloudTalk app, we leveraged the power of Google Cloud Monitoring to define and track a range of custom app metrics, such as network_connectivity and login_success_count.

We created dedicated monitoring dashboards to visualise key metrics so that we can quickly assess the health of our app. These dashboards cover:

- **Application Health Overview Dashboard**: The connectivity status and critical failure overview of network connections and Firebase database connections.
- **Authentication and User Login**: The total number of user logins, success/failure times, login time, MFA (Multi-Factor Authentication) verification (duration, success/ failure times). This helps us monitor the health and security of user activity.
- **Core Feature Performance**: The execution time (e.g., P95, P50 latency) and success/failure counts of core feature performance operations such as sending messages, uploading files. This allows us to accurately understand the performance of the core feature.
- **User and Group Management**: The execution time (e.g., P95, P50 latency) and success/failure counts of user and group management operations such as adding/removing group members, approving/rejecting users, confirming members, and disbanding groups and user group count. This allows us to accurately understand the performance of user and group management features.

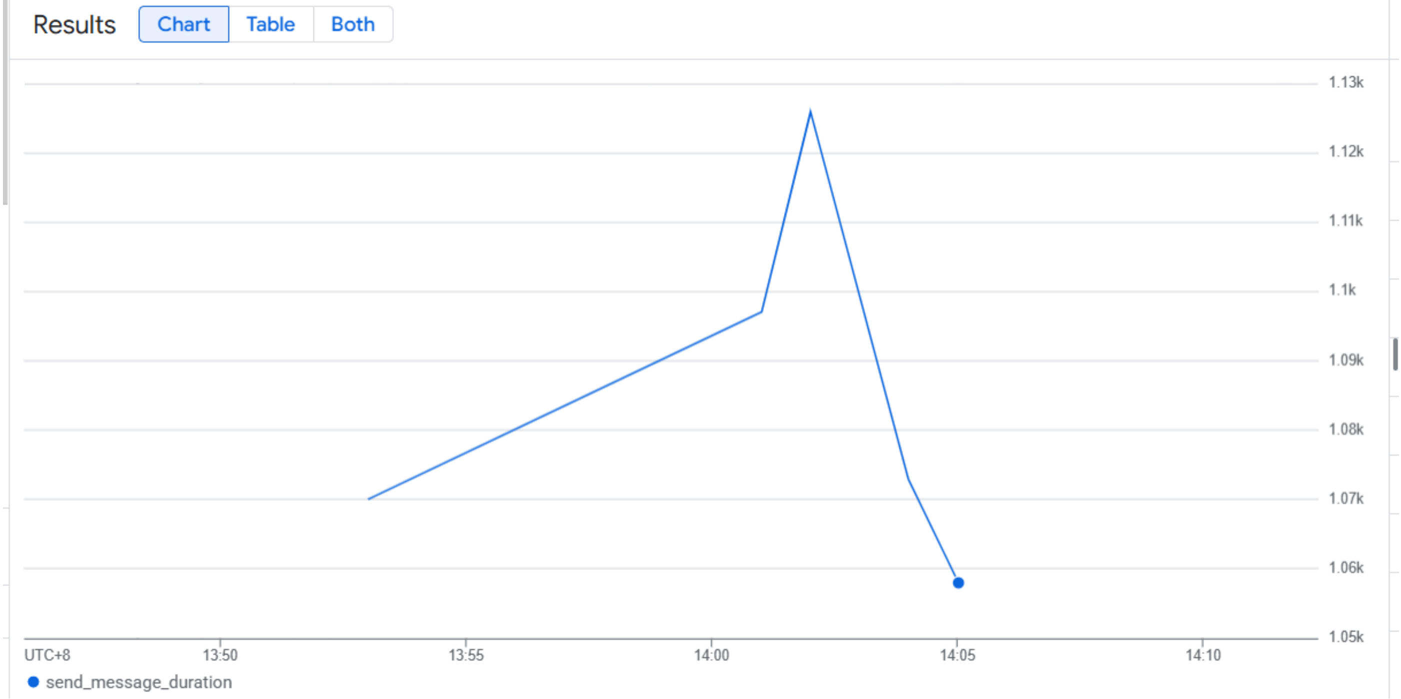
These customised monitoring metrics and dashboards allow us to go beyond the infrastructure level and gain insight into specific behaviours at the application level, allowing us to locate problems more quickly, optimise performance, and improve user experience.

Custom Metrics

Name	Purpose
network_connectivity	Tracks current network connection (1/0).
firebase_connectivity	Tracks Firebase DB connection state (1/0).
user_login_count	Tracks total login count.
user_group_count	Tracks number of groups user belongs to.
login_duration	Tracks login duration (email or Google).
login_success_count	Tracks successful login count.
login_failure_count	Tracks failed login count.
mfa_verification_failure_count	Tracks MFA verification failures.
mfa_verification_success_count	Tracks MFA verification success.

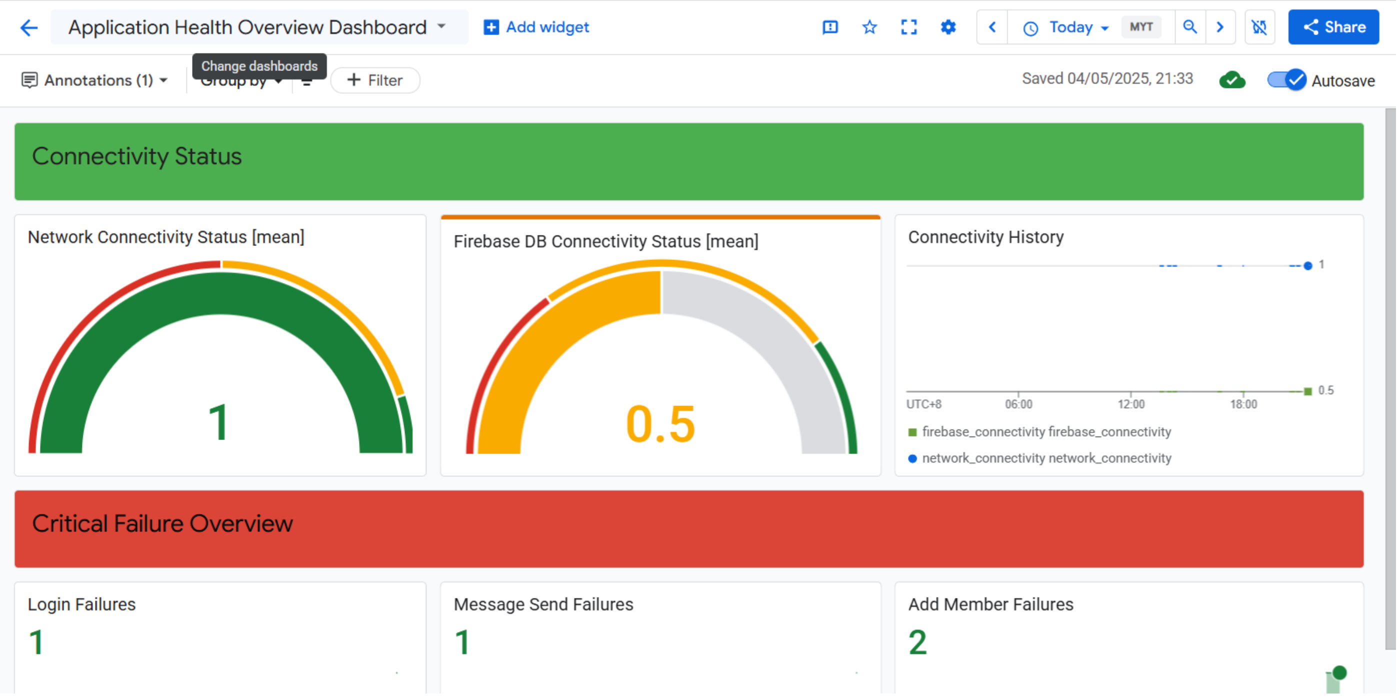
send_message_duration	Tracks duration to send a text message.
send_message_success_count	Tracks successful message sends.
send_message_failure_count	Tracks failed message sends.
file_upload_duration	Tracks duration of single file upload.
add_member_duration	Tracks duration to add a member.
add_member_success_count	Tracks successful member adds.
add_member_failure_count	Tracks failed member adds.
approve_pending_user_duration	Tracks time to approve a pending user.
approve_pending_user_success_count	Tracks successful approvals.
approve_pending_user_failure_count	Tracks failed approvals.
reject_pending_user_duration	Tracks time to reject a pending user.
reject_pending_user_success_count	Tracks successful rejections.
reject_pending_user_failure_count/ **reject_pending_user_success_count**	Tracks failed/ success rejections.
confirm_add_member_duration	Tracks confirmation duration.
confirm_add_member_success_count	Tracks successful confirmations.
confirm_add_member_failure_count	Tracks failed confirmations.
remove_member_duration	Tracks duration to remove member.
remove_member_success_count	Tracks successful removals.
remove_member_failure_count	Tracks failed removals.
disband_group_duration	Tracks time taken to disband group.
disband_group_success_count	Tracks successful disbands.
disband_group_failure_count	Tracks failed disbands.
global_mute_duration	Tracks time to toggle global mute.

Custom Metrics Result

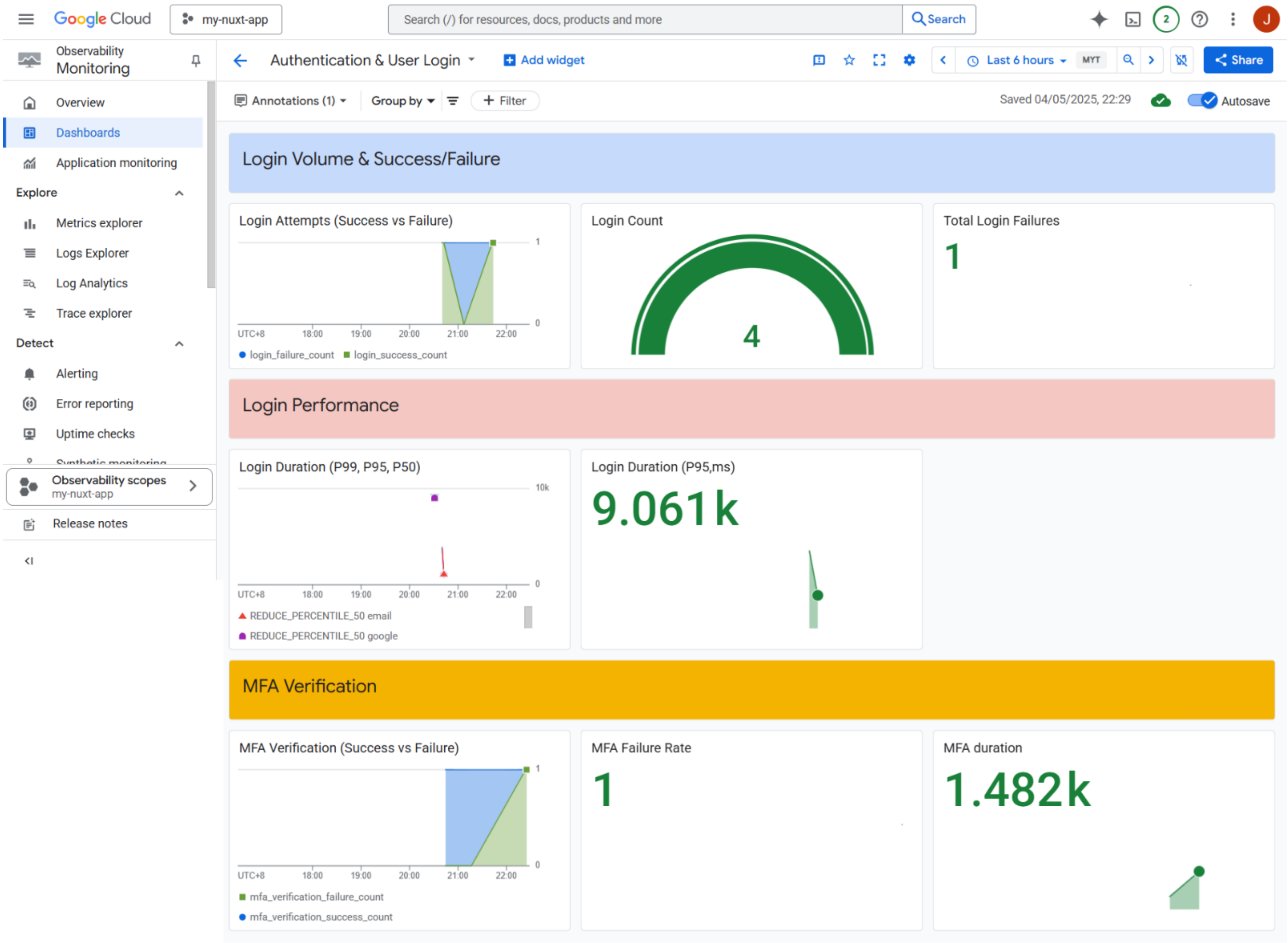


Custom Dashboard Result

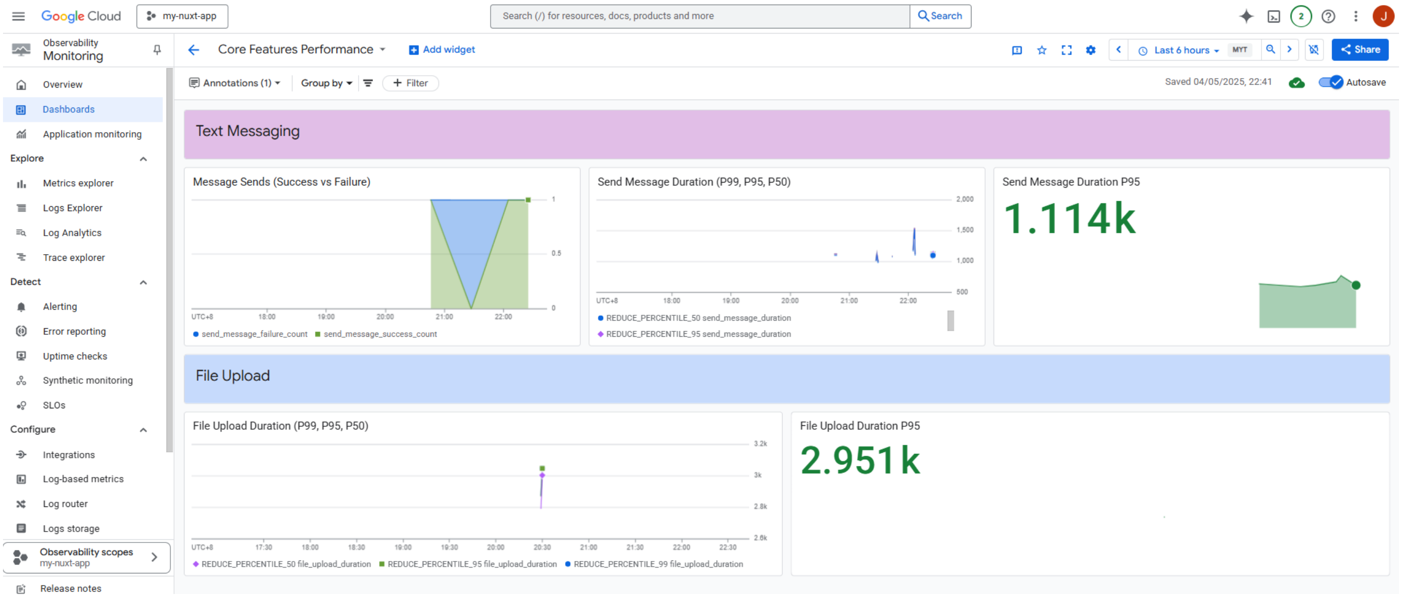
3.3.1 Application Health Overview Dashboard



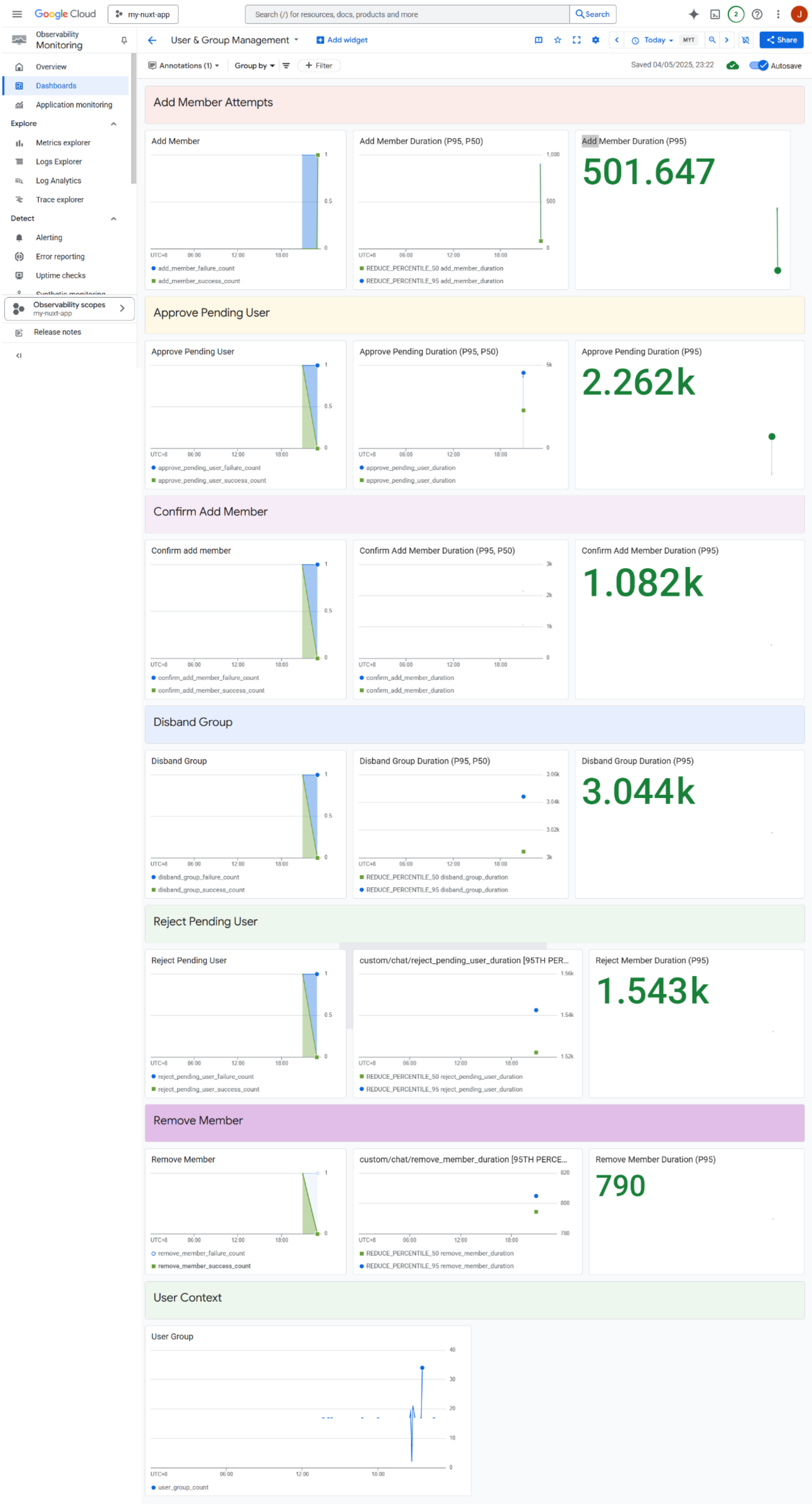
3.3.2 Authentication and User Login



3.3.3 Core Feature Performance

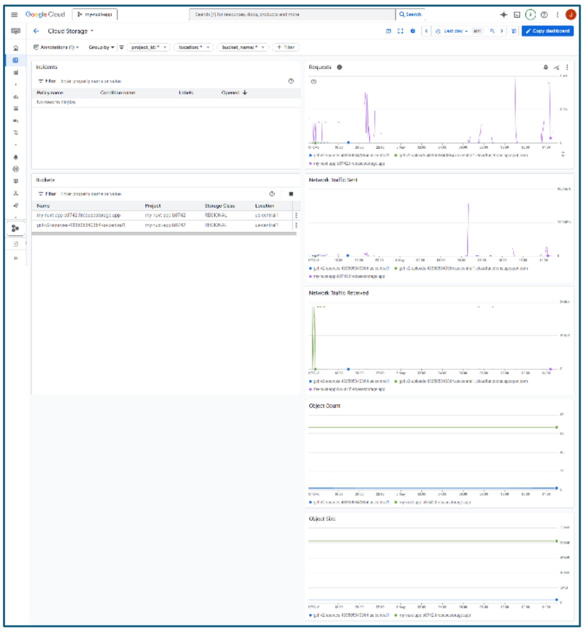


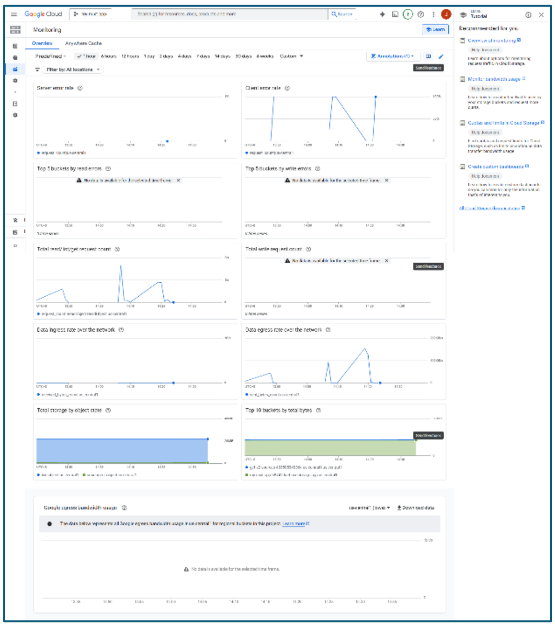
3.3.4 User and Group Management

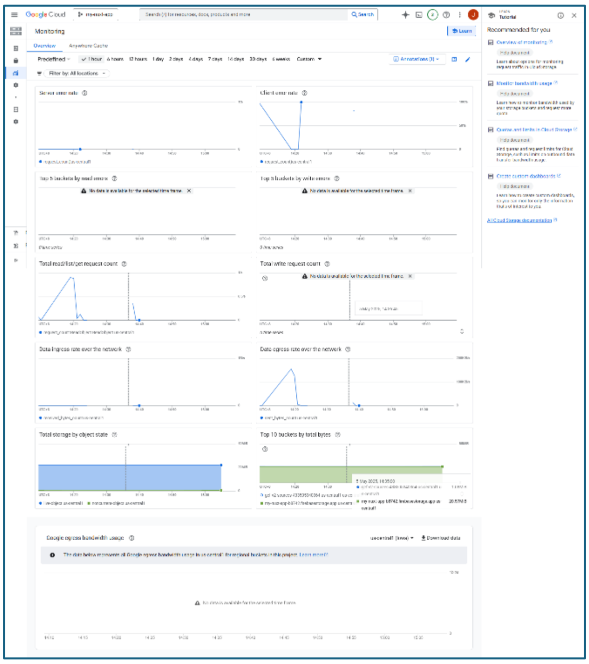

Google Cloud
my-nuxt-app
Search (/) for resources, docs, products and more
Search
Observability Monitoring
Overview
Dashboards
Application monitoring
Explore
Metrics explorer
Logs Explorer
Log Analytics
Trace explorer
Detect
Alerting
Error reporting
Uptime checks
Synthetic monitoring
Observability scopes
my-nuxt-app
Release notes
User & Group Management
Add widget
Today
MYT
Share
Annotations (1)
Group by
Filter
Saved 04/05/2025, 23:22
Autosave
Add Member Attempts
Add Member
add_member_failure_count
add_member_success_count
Add Member Duration (P95, P50)
REDUCE_PERCENTILE_50 add_member_duration
REDUCE_PERCENTILE_95 add_member_duration
Add Member Duration (P95)
501.647
Approve Pending User
Approve Pending User
approve_pending_user_failure_count
approve_pending_user_success_count
Approve Pending Duration (P95, P50)
approve_pending_user_duration
approve_pending_user_duration
Approve Pending Duration (P95)
2.262k
Confirm Add Member
Confirm add member
confirm_add_member_failure_count
confirm_add_member_success_count
Confirm Add Member Duration (P95, P50)
confirm_add_member_duration
confirm_add_member_duration
Confirm Add Member Duration (P95)
1.082k
Disband Group
Disband Group
disband_group_failure_count
disband_group_success_count
Disband Group Duration (P95, P50)
REDUCE_PERCENTILE_50 disband_group_duration
REDUCE_PERCENTILE_95 disband_group_duration
Disband Group Duration (P95)
3.044k
Reject Pending User
Reject Pending User
reject_pending_user_failure_count
reject_pending_user_success_count
custom/chat/reject_pending_user_duration [95TH PER...
REDUCE_PERCENTILE_50 reject_pending_user_duration
REDUCE_PERCENTILE_95 reject_pending_user_duration
Reject Member Duration (P95)
1.543k
Remove Member
Remove Member
remove_member_failure_count
remove_member_success_count
custom/chat/remove_member_duration [95TH PERCE...
REDUCE_PERCENTILE_50 remove_member_duration
REDUCE_PERCENTILE_95 remove_member_duration
Remove Member Duration (P95)
790
User Context
User Group
user_group_count
UTC+8
06:00
12:00
18:00

3.3.5 System Management

Besides that, cloud monitoring also provides a few of dashboard that allows for monitoring, which include: Cloud Storage, Google Cloud Storage, and Cloud Run Monitoring.




Google Cloud
Monitoring
Learn
Overview
Anywhere Cache
Predefined
1 hour
6 hours
12 hours
1 day
2 days
4 days
7 days
14 days
30 days
6 weeks
Custom
Annotations (1)
Filter by: All locations
Send feedback
Server error rate
Client error rate
Top 5 buckets by read errors
Top 5 buckets by write errors
No data is available for the selected time frame.
Total read/ list/get request count
Total write request count
Data ingress rate over the network
Data egress rate over the network
Total storage by object state
Top 10 buckets by total bytes
Google egress bandwidth usage
Download data
No data is available for the selected time frame.
Tutorial
Recommended for you
Overview of monitoring
Monitor bandwidth usage
Quotas and limits in Cloud Storage
Create custom dashboards
All Cloud Storage documentation

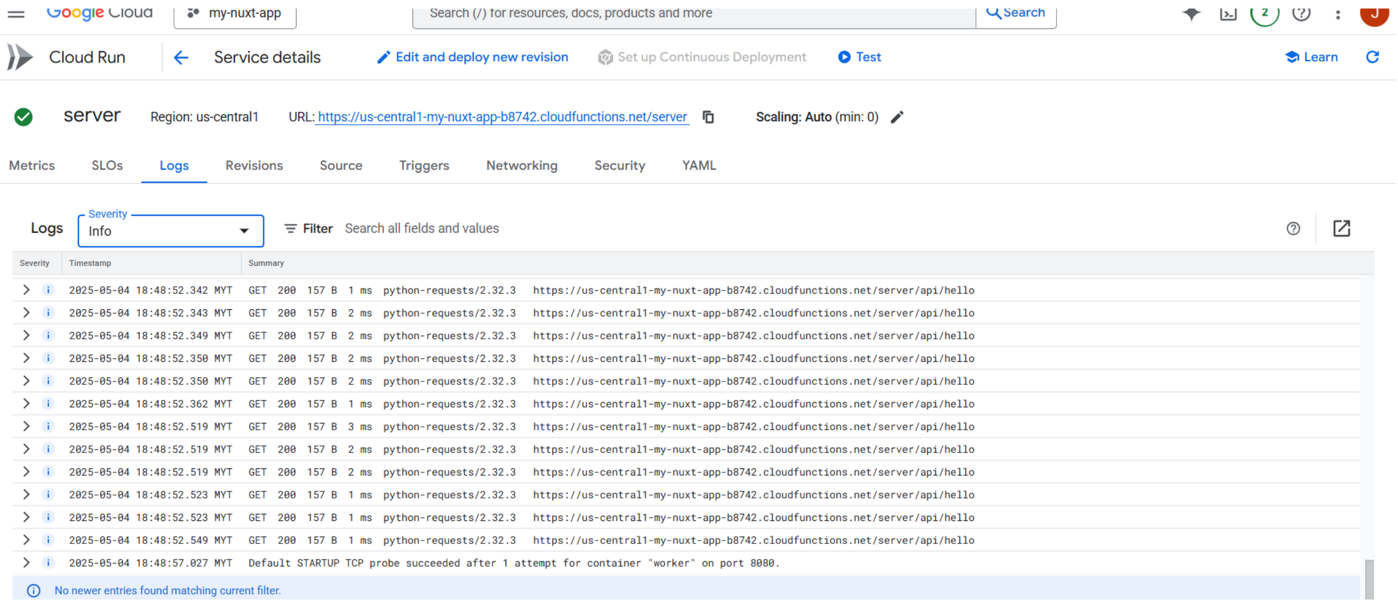

Google Cloud
my-nuxt-app
Search (/) for resources, docs, products and more
Search
Monitoring
Learn
Overview
Anywhere Cache
Predefined
1 hour
6 hours
12 hours
1 day
2 days
4 days
7 days
14 days
30 days
6 weeks
Custom
Annotations (1)
Filter by: All locations
Server error rate
Client error rate
Top 5 buckets by read errors
Top 5 buckets by write errors
No data is available for the selected time frame.
Total read/list/get request count
Total write request count
Data ingress rate over the network
Data egress rate over the network
Total storage by object state
Top 10 buckets by total bytes
Google egress bandwidth usage
Download data
No data is available for the selected time frame.
Tutorial
Recommended for you
Overview of monitoring
Monitor bandwidth usage
Quotas and limits in Cloud Storage
Create custom dashboards
All Cloud Storage documentation

4. Logging

Detailed logging is a key component of application observability, providing a foundation for troubleshooting, behavioural analysis, and security auditing.

4.1 Server Logging

In CloudTalk's backend services (running on Cloud Run/Cloud Functions), we have implemented server-side logging. We use a standard console. log/console.error to log the start and end of key operations, important information during processing, and any errors and exceptions that occur. These logs are automatically captured by Cloud Logging.


Google Cloud
my-nuxt-app
Search (/) for resources, docs, products and more
Search
Cloud Run
Service details
Edit and deploy new revision
Set up Continuous Deployment
Test
Learn
Metrics
SLOs
Logs
Revisions
Source
Triggers
Networking
Security
YAML
Logs
Filter
Search all fields and values

server Region: us-central1 URL: https://us-central1-my-nuxt-app-b8742.cloudfunctions.net/server Scaling: Auto (min: 0)

Severity: Info

```
Severity  Timestamp                  Summary
  i  2025-05-04 18:48:52.342 MYT  GET  200  157 B  1 ms  python-requests/2.32.3   https://us-central1-my-nuxt-app-b8742.cloudfunctions.net/server/api/hello
  i  2025-05-04 18:48:52.343 MYT  GET  200  157 B  2 ms  python-requests/2.32.3   https://us-central1-my-nuxt-app-b8742.cloudfunctions.net/server/api/hello
  i  2025-05-04 18:48:52.349 MYT  GET  200  157 B  2 ms  python-requests/2.32.3   https://us-central1-my-nuxt-app-b8742.cloudfunctions.net/server/api/hello
  i  2025-05-04 18:48:52.350 MYT  GET  200  157 B  2 ms  python-requests/2.32.3   https://us-central1-my-nuxt-app-b8742.cloudfunctions.net/server/api/hello
  i  2025-05-04 18:48:52.350 MYT  GET  200  157 B  2 ms  python-requests/2.32.3   https://us-central1-my-nuxt-app-b8742.cloudfunctions.net/server/api/hello
  i  2025-05-04 18:48:52.362 MYT  GET  200  157 B  1 ms  python-requests/2.32.3   https://us-central1-my-nuxt-app-b8742.cloudfunctions.net/server/api/hello
  i  2025-05-04 18:48:52.519 MYT  GET  200  157 B  3 ms  python-requests/2.32.3   https://us-central1-my-nuxt-app-b8742.cloudfunctions.net/server/api/hello
  i  2025-05-04 18:48:52.519 MYT  GET  200  157 B  2 ms  python-requests/2.32.3   https://us-central1-my-nuxt-app-b8742.cloudfunctions.net/server/api/hello
  i  2025-05-04 18:48:52.519 MYT  GET  200  157 B  2 ms  python-requests/2.32.3   https://us-central1-my-nuxt-app-b8742.cloudfunctions.net/server/api/hello
  i  2025-05-04 18:48:52.523 MYT  GET  200  157 B  1 ms  python-requests/2.32.3   https://us-central1-my-nuxt-app-b8742.cloudfunctions.net/server/api/hello
  i  2025-05-04 18:48:52.523 MYT  GET  200  157 B  1 ms  python-requests/2.32.3   https://us-central1-my-nuxt-app-b8742.cloudfunctions.net/server/api/hello
  i  2025-05-04 18:48:52.549 MYT  GET  200  157 B  1 ms  python-requests/2.32.3   https://us-central1-my-nuxt-app-b8742.cloudfunctions.net/server/api/hello
  i  2025-05-04 18:48:57.027 MYT  Default STARTUP TCP probe succeeded after 1 attempt for container "worker" on port 8080.
```

No newer entries found matching current filter.

4.2 Cloud Logging

The CloudTalk application is fully integrated with Google Cloud Logging, which records many operations and event logs in structured or unstructured formats, such as user actions, error logs, and system performance. These logs cover all aspects of application operation:

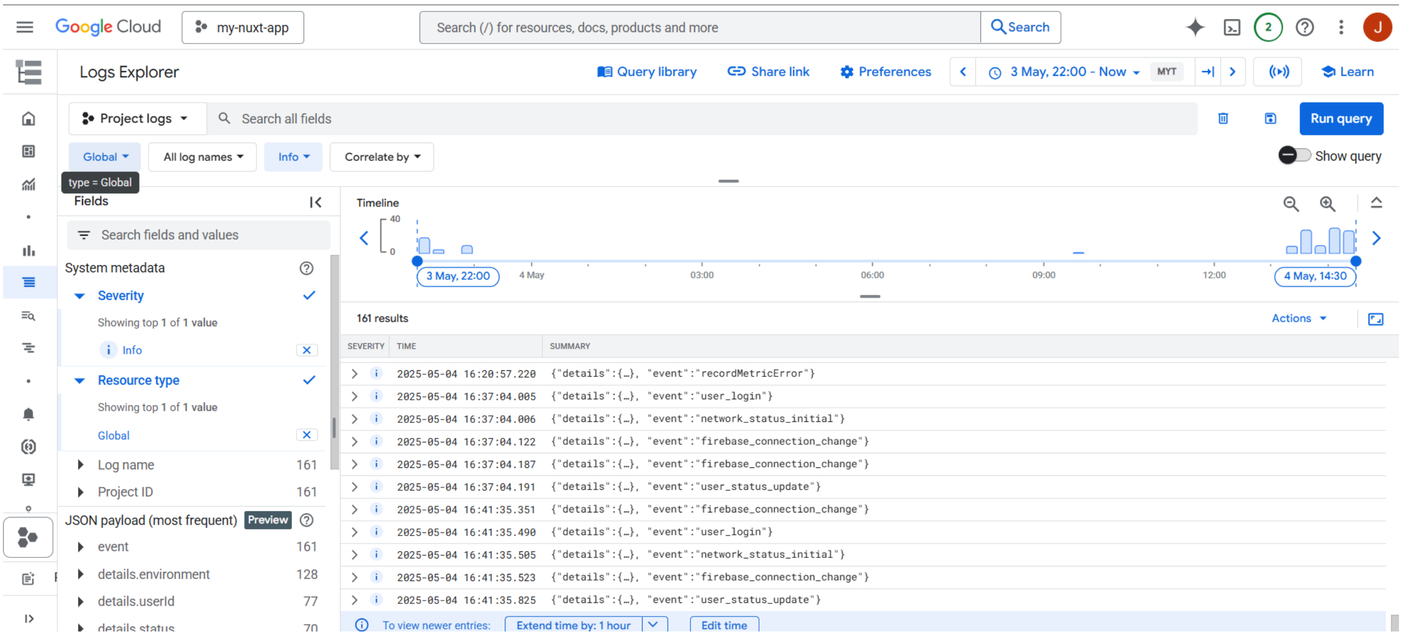
- Infrastructure and connection: network status changes, initial value and change of Firebase database connection status.
- User authentication and status: user successful login/logout, token expiration, online/offline status update, email/Google login attempt, MFA request and verification error, new user creation, etc.
- Core function interaction: message sending attempt/success/failure, file upload success, batch upload completion, group member addition/approval/rejection/confirmation/removal success and failure, group disband attempt/success/failure, scheduled message creation/update/deletion, etc.
- User interface operation: opening a specific modal box (such as disbanding a group, scheduled message, global mute confirmation), success/failure of copying a group ID, etc.
- Error and monitoring: general application-level errors and monitoring indicators, tracking failure or precondition missing logs.

All these logs are stored in Cloud Logging and can be searched, analysed, and filtered using a powerful query language. We can also create alerts based on specific log patterns to be notified when critical errors or abnormal events occur. This greatly facilitates rapid response and problem resolution

Name	Purpose
network_status_change	Logged when network status changes.
network_status_initial	Logged at app load to capture initial network state.
firebase_connection_change	Logged when Firebase DB connection changes.
user_login	Logged when user successfully logs in.
user_status_update (online)	Logged when user status updates to online.
token_expired	Logged when Firebase ID token expires.
user_logout	Logged when user logs out or is unauthenticated.
user_status_update (offline)	Logged when user leaves app and status updates to offline.
mfa_verification_requested (email)	Logged when MFA is required after email login.
login_success (email)	Logged on successful email login.
login_failure (email)	Logged on failed email login.
login_duration (email)	Logged on email login duration.
mfa_verification_error	Logged on MFA verification error.
login_attempt (google)	Logged when user tries Google login.

new_user_created (google)	Logged when new user is created via Google login.
mfa_verification_requested (google)	Logged when MFA is required after Google login.
login_success (google)	Logged on successful Google login.
login_failure (google)	Logged on failed Google login.
login_duration (google)	Logged on Google login time
send_message_attempt	Logged when user tries to send a text message.
send_message_success	Logged after successfully sending a text message.
send_message_failure	Logged on text message send failure.
send_message_duration	Logged on duration sending a text message.
file_upload_success	Logged on successful file upload.
batch_file_upload_complete	Logged after a batch of files is fully uploaded.
file_upload_duration	Logged on file upload duration.
add_member_success	Logged when admin successfully adds a member.
add_member_failure	Logged on failure to add member.
approve_pending_user_success	Logged on successful approval of join request.
approve_pending_user_failure	Logged on failure to approve join request.
reject_pending_user_success	Logged on successful rejection of join request.
reject_pending_user_failure	Logged on failure to reject join request.
confirm_add_member_success	Logged on successful member confirmation.
confirm_add_member_failure	Logged on member confirmation failure.
remove_member_success	Logged on successful member removal.
remove_member_failure	Logged on failure to remove member.
disband_group_attempt	Logged when admin attempts to disband group.
disband_group_success	Logged on successful group disband.
disband_group_failure	Logged on group disband failure.
open_disband_group_modal	Logged when disband confirmation modal is opened.
open_scheduled_messages_modal	Logged when scheduled messages modal is opened.
open_global_mute_confirmation	Logged when global mute confirmation is opened.
global_mute_success	Logged on successful global mute toggle.
global_mute_failure	Logged on global mute toggle failure.
leave_group_attempt	Logged when user tries to leave group.
leave_group_success	Logged on successful group leave.

leave_group_failure	Logged on failure to leave group.
update_scheduled_message_success	Logged on successful update of scheduled message.
create_scheduled_message_success	Logged on successful creation of scheduled message.
update_scheduled_message_failure	Logged on failure to update scheduled message.
create_scheduled_message_failure	Logged on failure to create scheduled message.
delete_scheduled_message_success	Logged on successful deletion of scheduled message.
delete_scheduled_message_failure	Logged on failure to delete scheduled message.
update_group_success	Logged when group details are successfully updated.
update_group_failure	Logged when group update fails.
copy_group_id_success	Logged when group ID is successfully copied.
copy_group_id_failure	Logged when group ID copy fails.
application_error	Logged for general application-level errors.
monitoring_error	Logged when metric tracking fails or has missing preconditions.



System metadata

Severity
Showing top 1 of 1 value
Info

Resource type
Showing top 1 of 1 value
Global

Log name	161
Project ID	161

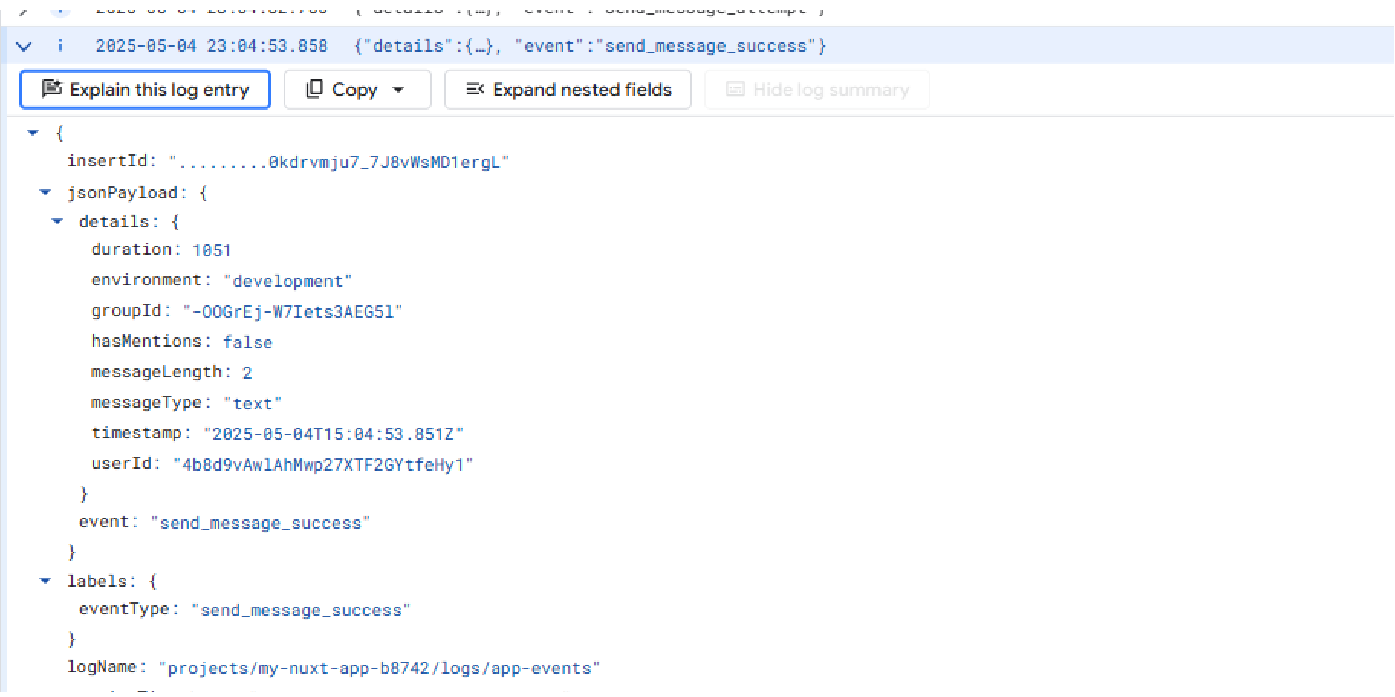
JSON payload (most frequent) Preview

event	161
details.environment	128
details.userId	77
details.status	70

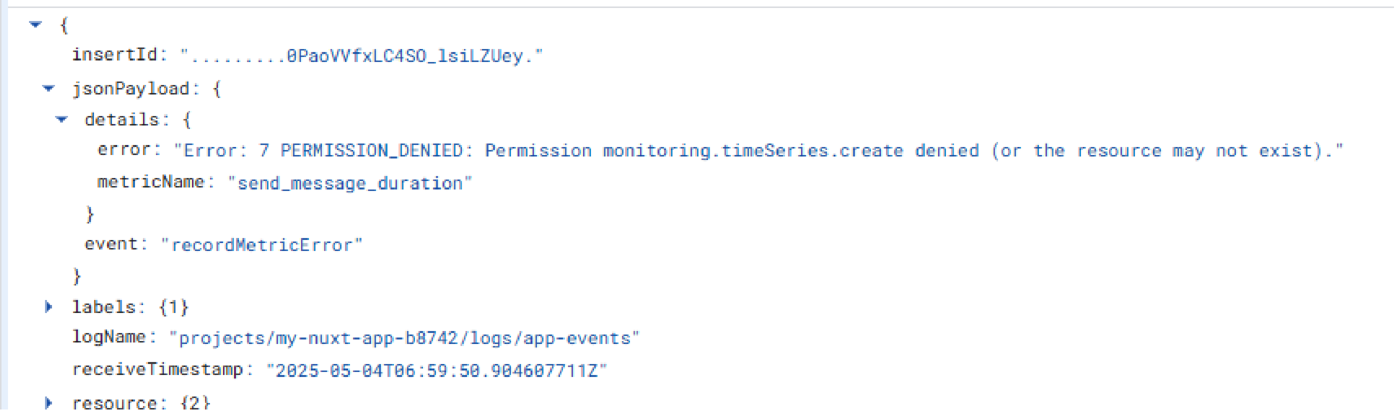
161 results

```
SEVERITY  TIME                     SUMMARY
i         2025-05-04 16:20:57.220  {"details":{…}, "event":"recordMetricError"}
i         2025-05-04 16:37:04.005  {"details":{…}, "event":"user_login"}
i         2025-05-04 16:37:04.006  {"details":{…}, "event":"network_status_initial"}
i         2025-05-04 16:37:04.122  {"details":{…}, "event":"firebase_connection_change"}
i         2025-05-04 16:37:04.187  {"details":{…}, "event":"firebase_connection_change"}
i         2025-05-04 16:37:04.191  {"details":{…}, "event":"user_status_update"}
i         2025-05-04 16:41:35.351  {"details":{…}, "event":"firebase_connection_change"}
i         2025-05-04 16:41:35.490  {"details":{…}, "event":"user_login"}
i         2025-05-04 16:41:35.505  {"details":{…}, "event":"network_status_initial"}
i         2025-05-04 16:41:35.523  {"details":{…}, "event":"firebase_connection_change"}
i         2025-05-04 16:41:35.825  {"details":{…}, "event":"user_status_update"}
```

To view newer entries: Extend time by: 1 hour Edit time



```
2025-05-04 23:04:52.780 {"details":{…}, "event":"send_message_attempt"}
2025-05-04 23:04:53.858 {"details":{…}, "event":"send_message_success"}
{
  insertId: ".........0kdrvmju7_7J8vWsMD1ergL"
  jsonPayload: {
    details: {
      duration: 1051
      environment: "development"
      groupId: "-OOGrEj-W7Iets3AEG5l"
      hasMentions: false
      messageLength: 2
      messageType: "text"
      timestamp: "2025-05-04T15:04:53.851Z"
      userId: "4b8d9vAwlAhMwp27XTF2GYtfeHy1"
    }
    event: "send_message_success"
  }
  labels: {
    eventType: "send_message_success"
  }
  logName: "projects/my-nuxt-app-b8742/logs/app-events"
```



```
{
  insertId: ".........0PaoVVfxLC4SO_lsiLZUey."
  jsonPayload: {
    details: {
      error: "Error: 7 PERMISSION_DENIED: Permission monitoring.timeSeries.create denied (or the resource may not exist)."
      metricName: "send_message_duration"
    }
    event: "recordMetricError"
  }
  labels: {1}
  logName: "projects/my-nuxt-app-b8742/logs/app-events"
  receiveTimestamp: "2025-05-04T06:59:50.904607711Z"
  resource: {2}
```

5. Scaling

To ensure that the CloudTalk application could gracefully cope with changing user loads, we took advantage of the auto-scaling capabilities provided by Google Cloud Platform/ Firebase. Secondly, the Kubernetes scaling method is also provided as a basis for future migration back to Kubernetes.

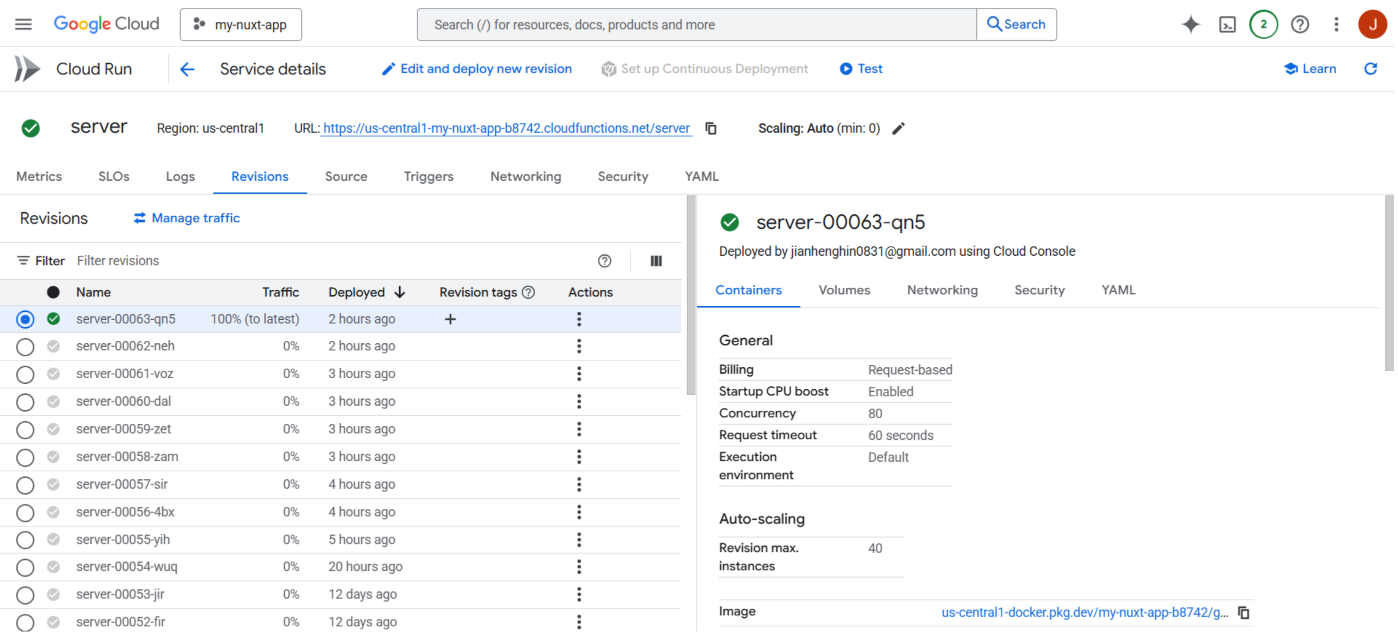
5.1 Firebase & Cloud Run Auto-Scaling

The core backend logic of the CloudTalk application is hosted on Cloud Functions and runs through the Google Cloud Run (fully managed) service. A key advantage of Cloud Run is its built-in auto-scaling capability. This means that the platform automatically adjusts the number of container instances running the application code based on real-time traffic demand.

How it works: Cloud Run triggers scaling based on the number of concurrent requests. Each container instance can handle multiple requests at the same time (configurable concurrency). When the concurrent requests of all existing instances approach or reach their configured upper limit, Cloud Run automatically starts new instances to handle the additional requests. Conversely, when the number of requests decreases and the existing instances are idle for a long time, Cloud Run automatically reduces the number of instances, even to zero (if the configuration allows, such as min-instances: 0), to save costs. In addition, scaling based on CPU utilization can also be configured.

In CloudTalk, we implement auto-scaling by setting the configuration:

- min instance: 0
- max instance: 40
- concurrent for each instance: 80



server Region: us-central1 URL: https://us-central1-my-nuxt-app-b8742.cloudfunctions.net/server Scaling: Auto (min: 0)

Name	Traffic	Deployed	Revision tags	Actions
server-00063-qn5	100% (to latest)	2 hours ago	+	⋮
server-00062-neh	0%	2 hours ago		⋮
server-00061-voz	0%	3 hours ago		⋮
server-00060-dal	0%	3 hours ago		⋮
server-00059-zet	0%	3 hours ago		⋮
server-00058-zam	0%	3 hours ago		⋮
server-00057-sir	0%	4 hours ago		⋮
server-00056-4bx	0%	4 hours ago		⋮
server-00055-yih	0%	5 hours ago		⋮
server-00054-wuq	0%	20 hours ago		⋮
server-00053-jir	0%	12 days ago		⋮
server-00052-fir	0%	12 days ago		⋮

server-00063-qn5

Deployed by jianhenghin0831@gmail.com using Cloud Console

General

Billing	Request-based
Startup CPU boost	Enabled
Concurrency	80
Request timeout	60 seconds
Execution environment	Default

Auto-scaling

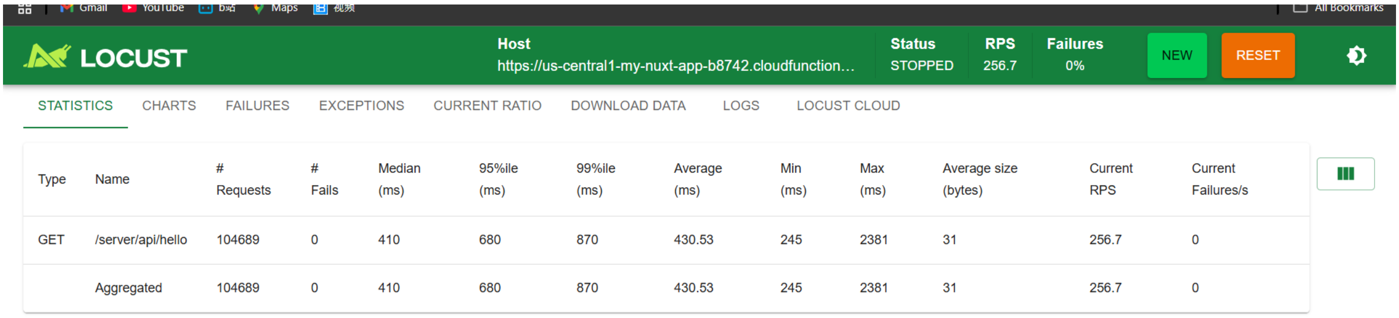
Revision max. instances	40

Image us-central1-docker.pkg.dev/my-nuxt-app-b8742/g...

Load Balancing Testing

We use Locust to simulate many users sending messages at the same time. When the number of users increases by 50 per second, up to 350 users, the system still processes at a fast speed and maintains a 0% failure rate, which shows the stability of the system.

Through cloud function monitoring, we can see that around 19:00, when the load test is carried out, the number of container instances will automatically increase (to 4) when the max concurrent request reaches 80 to handle the load, while maintaining low request latency and CPU/memory usage. When the load decreases, the number of active instances will automatically decrease accordingly. This intuitively demonstrates that Cloud Run's automatic scaling mechanism can effectively cope with changing traffic.



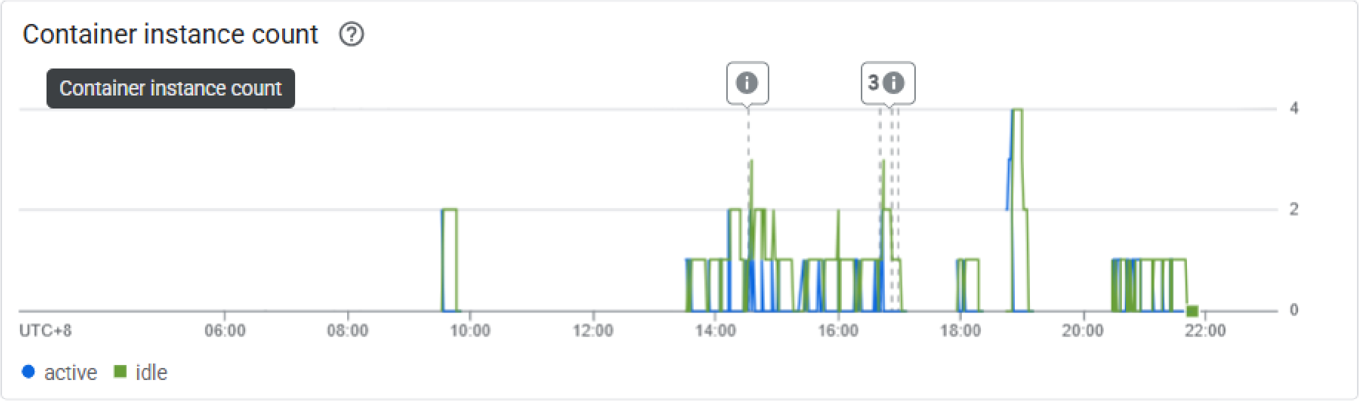
Host https://us-central1-my-nuxt-app-b8742.cloudfunction...

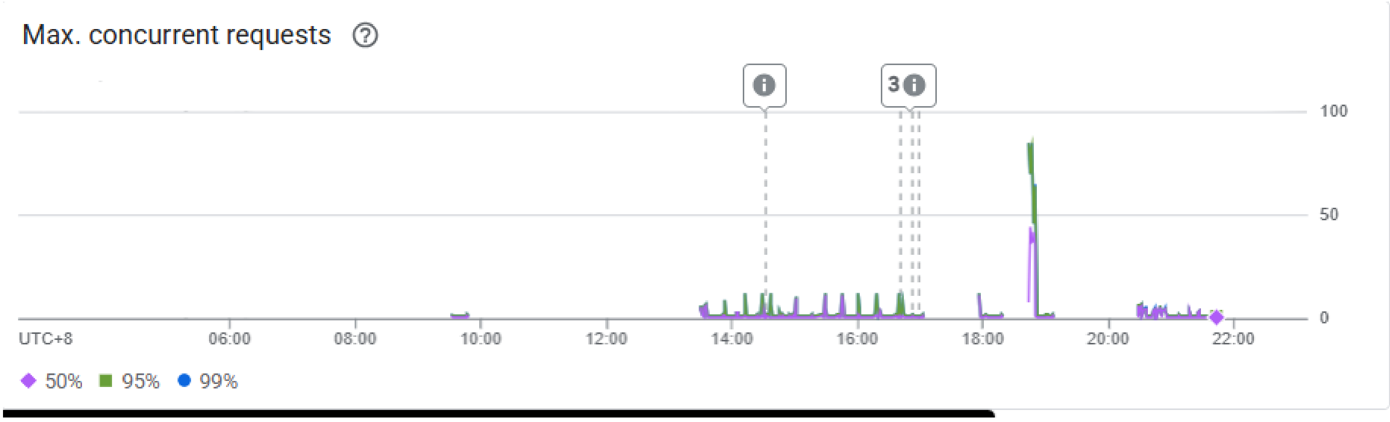
Status STOPPED

RPS 256.7

Failures 0%

Type	Name	# Requests	# Fails	Median (ms)	95%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/server/api/hello	104689	0	410	680	870	430.53	245	2381	31	256.7	0
	Aggregated	104689	0	410	680	870	430.53	245	2381	31	256.7	0





The underlying architecture of other core Firebase services, such as Realtime Database and Firebase Hosting, is also designed to support large-scale concurrency and automatic scaling. Users do not need to manage servers or manually scale; Firebase will handle the scaling of resources in the background. For example, Realtime Database can handle millions of concurrent connections, and Hosting can provide low-latency content delivery through a global CDN. Although the specific scaling mechanisms of these services are internal details, their design goal is to provide a seamless scaling experience.

5.2 Kubernetes Scaling

Due to cost considerations, I currently close Kubernetes Engine and deploy the main application on Firebase Hosting and Cloud Functions. Although it is currently running on the Firebase platform, I still retain the Kubernetes-related deployment and scaling configuration as the basis for future migration back to the Kubernetes environment

Scaling Action	Trigger Conditions	Cooldown Time	Pod Limits
Scale Up	Triggered when **any** of the following conditions are met:	**1 minute**	Max Pods: **20**
	- Average **CPU usage** > **70%**		
	- Average **Memory usage** > **80%**		
	- Average **firebase_operations per Pod** > **500**		
Scale Down	Triggered when **all** of the following are true:	**5 minutes**	Min Pods: **3**
	- **CPU usage** is significantly below 70%		
	- **Memory usage** is significantly below 80%		
	- **firebase_operations per Pod** is significantly below 500		
	- These conditions have stayed true continuously for a period of time		
	- At least **300 seconds** have passed since the last scale-down action (to avoid frequent downscaling during temporary traffic drops)		

Firebase/Cloud Run → Kubernetes

	Step	What to Do
1	**Dockerize**	Dockerize both Backend (Nuxt Server) and Frontend (Nginx serving static assets)
2	**Deploy**	Create Kubernetes Deployments (one for backend, one for frontend)
3	**Env Vars**	Use Secrets/ConfigMaps to manage environment variables
4	**Services**	Define Services for internal access to frontend and backend
5	**Ingress**	Set up routing rules (e.g., /api to backend, others to frontend) and handle HTTPS
6	**Redis**	Deploy Redis inside the cluster and configure connection

7	**Identity**	Use Workload Identity for secure access to Firebase/GCP services
8	**DNS**	Point your DNS to the external IP provided by the Ingress controller