A Study on Data Circulation Mechanism in Smart City Architecture

1st Masami Shinoda
Graduate School of System Design and Technology
Tokyo Denki University
Tokyo, Japan
24amj18@ms.dendai.ac.jp

2nd Kanae MATSUI
Graduate School of System Design and Technology
Tokyo Denki University
Tokyo, Japan
matsui@mail.dendai.ac.jp

***Abstract*—This paper proposes a data circulation mechanism as a core function of the data collaboration platform essential for realizing smart cities. According to the definition by the Cabinet Office of Japan, a smart city is an urban concept aimed at providing personalized services and enhancing management across multiple domains. Digital Transformation (DX) is a key enabler of this advanced urban management. The objective is to create a framework enabling all citizens to utilize data, addressing urban challenges and creating new value.**

***Index Terms*—Smart City, Data Circulation Mechanism, Static and Dynamic Data, Serverless Architecture, Web API Integration**

I. Introduction

The concept of a smart city has various interpretations, but the Cabinet Office of Japan defines its essence as "providing personalized services" and "enhancing management across multiple domains" [1]. This definition is based on three core principles: "citizen-centricity," "emphasis on cross-domain and urban systems," and "solving challenges and realizing visions."

"Enhancing management" essentially refers to Digital Transformation (DX), which aims to enable not only data owners but all citizens to utilize data, contribute to solving societal challenges, and create new value. In this study, DX specifically refers to digitization that enables "data-driven" decision-making, facilitating more effective processes based on data.

However, sufficient data collaboration necessary for a data-driven society is not yet fully realized. This challenge aligns with the smart city's emphasis on cross-domain collaboration and the broader goals of Society 5.0. In Society 4.0 (the Information Society), data was often confined to individuals or specific organizations with limited sharing [2].

To address this, there is a need to open data previously managed in closed systems and enable cross-domain collaboration for all citizens. This study focuses on designing and implementing a data circulation mechanism that promotes secure and efficient data sharing across different sectors.

This paper reviews existing research on smart city architectures and data circulation mechanisms, identifying current challenges. It then presents the design and implementation of the proposed mechanism and reports on experiments evaluating its effectiveness. Finally, the study discusses the results and future prospects.

II. Related Research

A. Related Research

Research on data circulation mechanisms and urban operating systems (Urban OS) has gained significant attention in the context of smart city development. The ability to integrate and share data securely across multiple domains is critical for realizing the smart city concept. Various studies have explored different approaches to achieving this goal.

Aoki et al. [3] proposed an Urban OS architecture designed to manage and securely share heterogeneous data. Their study emphasizes key features such as data standardization, API-based access, and privacy protection, which are essential for data integration across sectors. This research is closely related to our work, particularly in terms of secure data management and sharing.

Several studies have proposed mechanisms for enhancing data interoperability in smart cities, including approaches to real-time data integration and privacy-aware data sharing [4], [5] . These studies underscore the challenges in managing dynamic data streams and ensuring security, which align with the objectives of our work.

Japan's "Super City" initiative leverages the FIWARE platform to enhance data interoperability, security, and collaboration [6], forming the basis for our proposed mechanism.

These studies demonstrate the diverse approaches to designing Urban OS and data circulation mechanisms. Building on these insights, our work focuses on real-time data processing and the integration of heterogeneous data to support efficient and secure data utilization in smart cities.

B. Efforts in Takamatsu City, Kagawa

A notable example of a data-driven initiative is the Smart Map implemented in Takamatsu City [7]. In this case, various data tied to the location information of Takamatsu City are visualized on a map, providing data-driven insights for human decision-making. The map effectively displays both relatively static data, such as the locations of disaster prevention facilities, and more frequently updated IoT sensor data, such as tide levels and precipitation, along with their observation points. This enables individuals to make informed decisions, such as choosing evacuation sites during a disaster, by using the map as a reference.

As the use of real-time data in such cases increases, the demand for data collaboration platforms in smart cities is expected to grow.

III. System Configuration

This study focuses on the data circulation mechanism, one of the core functions essential for realizing smart cities, within the data collaboration platform. The proposed data circulation mechanism primarily handles two types of data:

1) Static data owned by local governments
2) Highly real-time dynamic data obtained from IoT sensors and other sources

Since the characteristics of these data types are fundamentally different, each requires a specific processing method. For the realization of smart cities, it is crucial for the data circulation mechanism to handle both types of data flexibly, enabling collaboration with data analysis platforms and integration with other systems and applications. This study proposes an efficient data processing mechanism that meets these requirements.

The main proposal of this study is as follows: Traditional data collaboration platforms tend to focus on processing real-time data. However, to achieve data-driven systems and evidence-based policy making (EBPM) [8], which are key goals of smart cities, the integration of diverse static and dynamic data is essential.

In the proposed mechanism, these heterogeneous data types are converted into Web APIs to facilitate data circulation. This method provides a unified access approach regardless of the type and characteristics of the data, thereby enhancing interoperability between systems. The following sections describe the details of the proposed mechanism.

A. System Configuration

The system comprises data collection, storage, and distribution in 1.

Fig 1 illustrates the proposed data distribution mechanism. The central API gateway manages requests, while the Lambda functions handle data processing and response formatting. Static data is retrieved directly from SQLite snapshots, whereas dynamic data is fetched from S3 in real-time to ensure up-to-date information.

1) **Data Collection**: Crawls external sources for dynamic data.
2) **Data Storage**: Stores static and dynamic data in databases suited to their characteristics.
3) **Data Distribution**: Provides stored data via Web APIs.

The system is implemented on Amazon Web Services (AWS) as detailed in Table I. All components operate within the AWS environment.

The key feature is the use of a serverless architecture, which minimizes infrastructure management and allows for scalable system construction. TypeScript ensures type safety for API implementation, while Python is used for efficient data collection via crawling. SQLite provides lightweight, fast database operations.

TABLE I
Implementation Environment

Cloud Service	Amazon Web Services (AWS)
Execution Environment	Node.js (16.18.x)
Framework	Serverless Framework (3.26.x)
API Language	TypeScript (5.0.x)
Data Collection Language	Python (3.10.x)
Database System	SQLite (3.x.x)

The following sections will detail the system's functionality and evaluation results.

1) Data Collection: As described in Section 3.1, this study focuses on the collection and storage of two types of data: static data and dynamic data. Static data is stored in the database using SQL based on predefined templates. In contrast, dynamic data, which requires real-time handling, can originate from various sources, including APIs, applications, and system integrations. To address reliance on manual data downloads, we implemented web crawling.

An example of static data is population statistics owned by local governments. These datasets, typically published by national or local government agencies, are provided in human-readable formats such as Excel files or PDFs. To enhance machine readability for the distribution function, we extracted necessary data elements and created templates for each data item. The data items refer to statistical categories such as ”population by gender” or ”population by age,” and the templates define the database (DB) table schema. After creating the templates, data from each year is stored in the database using SQL. This process was carried out manually in this study.

For dynamic data, we also manually created templates that extract necessary data elements. However, unlike static data templates, these templates define CSV file headers instead of DB table schemas. During data collection, the data is stored according to these templates. Since static data typically has an annual update frequency, all data was collected manually. However, dynamic data can be updated multiple times a day, making manual updates impractical. Furthermore, the risk of human error increases with manual processes. Therefore, we implemented an automated data collection function. The system configuration for real-time data updates is shown in 2.

Two primary functions are required for implementing the data collection feature. First, a program and execution environment for crawling websites and storing the data. Second, a mechanism to execute this program at specified intervals.

For the crawling program, AWS Lambda was used as the execution environment, and Python was chosen as the programming language to implement web crawling. The collected data is converted according to the predefined templates for each data item and stored in Amazon S3.

To execute the program at specific intervals, we assumed that the data source websites do not provide update notifications, meaning the system must actively collect data. To address this, AWS Step Functions were used to execute the crawling program according to the update schedules for

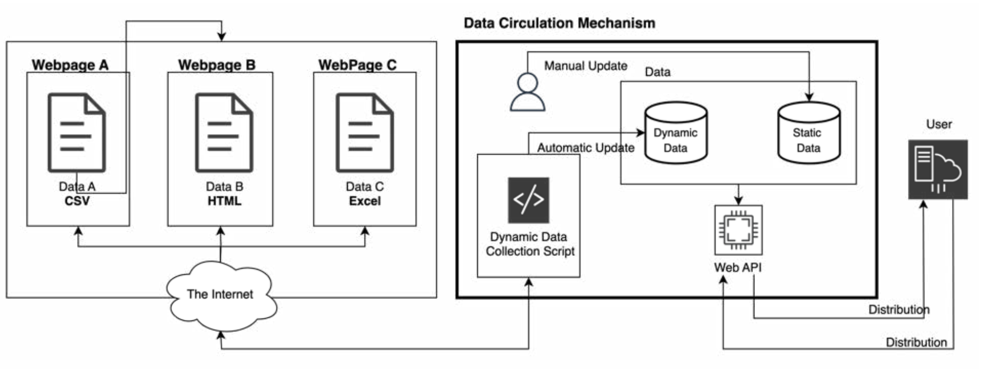

Fig. 1. Proposed Data Distribution Mechanism.

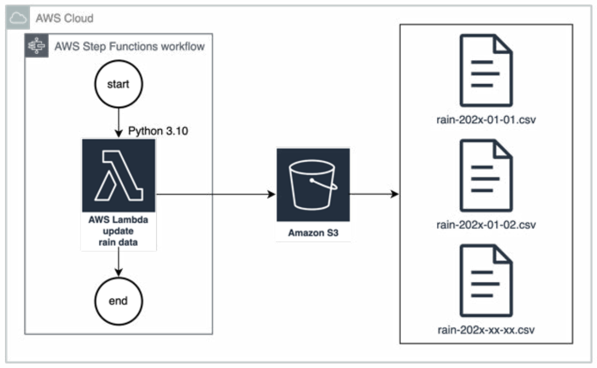

Fig. 2. Real Time Data Update Configuration Diagram.

each data item. In the environment used for this study, the data updates occurred at fixed intervals, such as 12:00 and 12:10. Using AWS Step Functions, we configured a cron-based schedule to automate data collection at these times.

2) Data Storage: This section describes the contents of two types of databases used to handle both static and dynamic data.

Static data, such as population statistics that are updated annually, is stored in an SQL-based database for cost efficiency. SQLite is chosen as the database management system due to its lightweight nature compared to other systems. Since static data is updated at intervals of approximately one year, the overall data volume is relatively small. Therefore, it is managed as a single file and stored in an SQL-based database, allowing the use of SQL queries and offering the convenience of flexible data operations.

For dynamic data, instead of using an SQL-based database, the data is stored in daily segmented CSV files. "Daily segmented" refers to managing each data record in file units, where each file covers a specific period from 12:00 AM to 12:00 AM (Japan Standard Time) the following day. Since dynamic data tends to accumulate large volumes over time, managing it in a single file could cause issues with data size and handling. By dividing the data into daily segments, the size of each file is kept manageable, which also improves efficiency when distributing the data through the Web API, as discussed in the next section. Thus, the accumulation method differs from that used for static data.

3) Data Distribution: This section describes the Web API used to distribute the accumulated data. A Web API is a mechanism that allows part of a program's functionality to be accessed by other programs via the web. In this study, a Web API was implemented to handle the collected and stored data. The configuration diagram is shown in 3.

As a common implementation, when the system receives a request to the Web API, it passes through CloudFront and API Gateway, triggering the corresponding Lambda function for each data item. Each Lambda function is designed to format the data according to a predefined structure and respond in JSON format based on the request. This allows the system to function as a Web API and redistribute the data.

However, there is a significant difference in how static and dynamic data are handled in the stored database files.

For static data, at the time the system is deployed, the SQLite files corresponding to each data item are stored alongside the executable files within the Lambda environment. These files are snapshots stored in S3 and placed inside the Lambda execution environment, allowing the system to operate without retrieving data from S3 every time a Lambda function is executed. This approach enhances response speed and reduces unnecessary resource usage. However, this method is only effective when the total data volume is small and the snapshot data is not frequently updated. When the snapshot data is updated, the Web API must be redeployed, and the data inside the Lambda function must be manually updated, making it impractical for non-static data.

In contrast, for dynamic data, the large data volume and the need to provide the latest data promptly means that the data is not stored inside the Lambda environment. Instead, data is fetched from S3 on-demand based on the request, processed, and returned in the response. This ensures that users receive the latest data with minimal delay from the time of the update to the time of distribution.

If dynamic data were handled using the same method as static data, the entire system would need to be redeployed, not just individual Lambda functions, leading to longer system downtime proportional to the number of data items returned

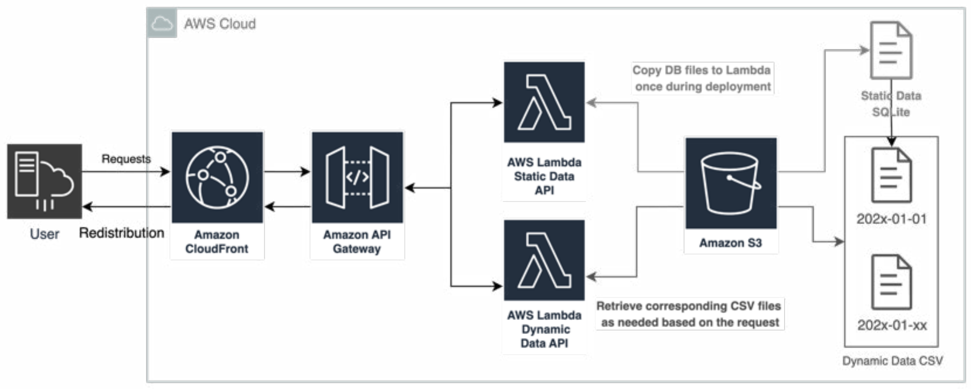
AWS Cloud
Requests
Redistribution
User
Amazon CloudFront
Amazon API Gateway
AWS Lambda Static Data API
AWS Lambda Dynamic Data API
Copy DB files to Lambda once during deployment
Retrieve corresponding CSV files as needed based on the request
Amazon S3
Static Data SQLite
202x-01-01
202x-01-xx
Dynamic Data CSV

Fig. 3. WebAPI Configuration Diagram.

by the Web API, and increased costs. Therefore, different distribution methods are used for static and dynamic data to accommodate their respective characteristics and meet system requirements.

IV. Evaluation Experiment

A. Evaluation Contents

To verify the effectiveness of the proposed data circulation mechanism, we implemented the system assuming the foundation of the open data platform ”Data Platform Kure” in Kure City, Hiroshima Prefecture. ”Data Platform Kure” is a system that publicly provides various administrative data held by Kure City as Web APIs for use by citizens and businesses.

The data to be evaluated includes both static and dynamic data, and their characteristics, periods, and items are summarized in Table II.

The evaluation will be conducted by analyzing the rate and nature of errors occurring during the process from data collection to distribution for the data and periods shown in Table II. Specifically, the following types of errors are anticipated:

1) Errors during data collection: crawling failures, data format mismatches
2) Errors during data storage: database write failures, capacity overflow
3) Errors during data distribution: API response errors, data format inconsistencies

Due to the differing characteristics of static and dynamic data, the evaluation methods will be distinguished accordingly. For static data, since the collection is done manually, collection-related issues are not considered. Therefore, the primary evaluation metric will be the success rate of distribution via the Web API, which will be measured by the percentage of correct data returned in response to API requests.

For dynamic data, the reliability of the automatic collection process is crucial, and the primary evaluation metric will be the data collection rate. This will be calculated by the ratio of successfully collected data to the total scheduled collection attempts.

In both data types, if errors occur, the type and frequency of the errors will be recorded and used to identify areas for system improvement. Through this evaluation, the performance and reliability of the proposed data circulation mechanism will be comprehensively verified, and challenges for practical implementation will be identified.

B. Evaluation Results

1) Results of Static Data: In this section, we present the operational results of the static data distribution system over an 8-month period, from April 1, 2023, to December 31, 2023. The data was aggregated based on the following definitions as evaluation metrics:

- Total Access Count: The total number of HTTP requests to the constructed system.
- Valid Access Count: The number of valid requests, excluding issues caused by the request source.
- Server-side Error Count: The number of cases where the system failed to return an appropriate response due to system-related issues.

The evaluation results are shown in Table III.

The distribution rate was calculated using the following equation 1.

$$\text{Distribution Rate} = \frac{\text{Valid Access Count} - \text{Server-side Error Count}}{\text{Valid Access Count}} \times 100 \ (\%) \tag{1}$$

Table IV shows the corresponding HTTP status codes for each aggregation category.

Here, each status code indicates the following:

- 200: Request successful
- 206: Partial content sent successfully
- 403: Access forbidden
- 503: Service unavailable

”Server-side errors” include temporary server overloads (503) and access permission configuration issues (403).

TABLE II
DATA CONTENT TO BE EVALUATED.

Data Type	Period	Data Items
Static Data	2023/04/01∼2023/12/31	31 items (e.g., population statistics, financial status, tourism statistics)
Dynamic Data	2023/11/05∼2023/12/05	Rainfall, river water level, tide level
Dynamic Data	2023/12/02∼2024/01/02	Pedestrian flow

TABLE III
STATIC DATA DISTRIBUTION RATE

Total Access Count	Valid Access Count	Server-side Error Count	Distribution Rate (%)
7544	6755	60	99.1

TABLE IV
STATUS CODE CORRESPONDENCE BY AGGREGATION CATEGORY

Total Access Count	Valid Access Count	Server-side Error Count
All	200, 206, 403, 503, Others	403, 503, Others

The analysis results show a distribution rate of 99.1%, indicating that the constructed system can distribute static data with high reliability. Factors preventing a 100% distribution rate include temporary network delays, momentary server overloads, and minor software bugs. However, the frequency of these issues is extremely low and is not considered to have a significant impact on overall system performance.

A 99.1% distribution rate confirms the mechanism's effectiveness and robustness over 8 months. The system's ability to maintain stable performance over an 8-month period further supports its robustness and reliability.

As future work, a detailed analysis of the remaining 0.9% error cases will be necessary to identify potential improvements. Additionally, collecting more long-term operational data and continuously evaluating the system's stability will be essential.

2) Results of Dynamic Data: During the verification period of dynamic data summarized in Table II, the number of requests to the Web API matched the number of responses. However, several issues were discovered in the data collection function. The results are shown in Table V.

The river water level data achieved a 100% success rate due to its consistent update schedule and minimal dependency on external services. In contrast, rainfall and tide level data faced challenges such as source service delays and format inconsistencies, impacting their collection reliability.

The evaluation was conducted by counting the success or failure of the data collection program on each day and calculating the success rate as the ratio of successful operations to the total evaluation days. Here, success refers to cases where all scheduled data was collected correctly, while failure refers to cases where any part of the data collection encountered issues.

As a result, approximately 20% of the dynamic data distribution exhibited deficiencies across all four data types. The success rate is insufficient for providing the system as a practical service, indicating the need for improvements in the data collection function. After analyzing the failures, two primary issues were identified:

First, the system faced issues with unexpected data update deviations from external sources.

Second, a lack of understanding of AWS-specific characteristics in the constructed system caused problems. Specifically, the crawling program implemented using AWS Lambda transitioned to an "Inactive" state after a certain period of inactivity, preventing immediate execution. This phenomenon is documented in AWS documentation, but it was difficult to account for all requirements in advance, and this issue was not addressed during the implementation stage.

To address the issues identified in this study, particularly the automatic transition of AWS Lambda to the "Inactive" state, the following improvements are under consideration. To enhance flexibility in collaboration with external services, we plan to implement a scheduling feature that accommodates variable data update times and introduce alternative processes for anomaly detection. Rare failures impacted the stability of dynamic data processing.

In the future, these improvements will be implemented to improve system stability and reliability through long-term operation. We will also work on modularizing the data collection function and improving its versatility to adapt to various types of dynamic data.

V. DISCUSSION

The proposed system can be seamlessly integrated into existing urban infrastructures by providing standardized APIs for various municipal departments. For non-technical stakeholders, user-friendly dashboards could simplify data access and enhance decision-making processes.

TABLE V
DYNAMIC DATA DISTRIBUTION RATE

Dynamic Data	Success	Failure	Evaluation Days	Success Rate (%)	Failure Rate (%)
Rainfall	25	6	31	80.6	19.4
Tide Level	19	12	31	61.2	38.8
River Water Level	31	0	31	100.0	0.0
Pedestrian Flow (KLA)	1	1	2	50.0	50.0

In the process of collecting dynamic data, integration with external applications and systems is essential. This study has revealed the practical difficulties in fully understanding complex services like AWS and data source services that are difficult to modify. Additionally, unforeseen issues, such as rare failures under specific conditions, were identified. These challenges significantly impact the stability and reliability of dynamic data processing systems.

The evaluation results indicate that continuous monitoring of both dynamic data and the collection program is necessary. Implementing a data monitoring program, error detection within the collection program, and alert notifications to administrators when issues arise would enable faster responses to unanticipated problems. Since modifying data source services is often difficult, improving the system's ability to respond quickly to unknown issues is crucial.

For efficient troubleshooting, generalizing error types is important. This involves abstracting similar issues and categorizing them within the system. Such generalization simplifies system status management, enables automation of responses to alerts, and improves the clarity of alert messages. For instance, differentiating between the failure to retrieve data and the failure to process it appropriately allows for more effective problem-solving.

However, the design and implementation of such error abstraction must balance the cost of cloud services. Overly complex systems can increase operational costs, so careful consideration of the scope and cost of implementation is necessary. The design should be tailored to the scale and importance of the system.

Future work includes improving monitoring, automating error handling, and optimizing for cost and performance. Efforts will also be made to modularize the data collection function and improve its versatility to adapt to various types of dynamic data, aiming to enhance the reliability and efficiency of the dynamic data processing system.

Additionally, accumulating and analyzing long-term operational data will contribute to building more accurate failure prediction models, enabling proactive issue avoidance and more efficient resource allocation.

VI. CONCLUSION

In this study, we designed and implemented a data circulation mechanism essential for smart cities, focusing on efficiently handling both static and dynamic data.

The evaluation showed a high distribution rate of 99.1% for static data, demonstrating stable performance. However, dynamic data posed challenges, such as issues with external service integration and AWS Lambda's transition to an "Inactive" state. These findings underscore the need for effective monitoring systems in handling dynamic data.

Key takeaways include the importance of continuous data and program monitoring, error type generalization, and flexible scheduling to improve system stability and reliability. Additionally, understanding cloud service characteristics and optimizing system design are critical.

Moving forward, we will enhance monitoring for dynamic data, automate error handling, and optimize the system, balancing cost and performance. We also plan to modularize the data collection function and improve its adaptability to different types of dynamic data.

ACKNOWLEDGMENT

This research was supported by JSPS KAKENHI Grant Number 22H03572. The data used in this study are available at Dataplatform Kure [9].

REFERENCES

[1] KIRIMTAT, Ayca, et al. Future trends and current state of smart city concepts: A survey. IEEE access, 2020, 8: 86448-86467.

[2] Cabinet Office, Government of Japan, https://www8.cao.go.jp/cstp/english/society5_0/index.html, Available Sep 12rd, 2024.

[3] AOKI, Shunsuke, et al. Time-sensitive cooperative perception for real-time data sharing over vehicular communications: Overview, challenges, and future directions. IEEE Internet of Things Magazine, 2022, 5.2: 108–113.

[4] Xiao, Zhe and Fu, Xiuju and Goh, Rick Siow Mong. Data privacy-preserving automation architecture for industrial data exchange in smart cities. In: IEEE Transactions on Industrial Informatic, 2027. vol. 14, no. 6, p. 2780–2791.

[5] Cirillo, Flavio and Solmaz, Gürkan and Berz, Everton Luís and Bauer, Martin and Cheng, Bin and Kovacs, Ernoe. A standard-based open source IoT platform: FIWARE. In: IEEE Internet of Things Magazine: IEEE, 2019. vol. 2, no. 3, p. 12–18.

[6] HASENBURG, Jonathan; BERMBACH, David. DisGB: Using geo-context information for efficient routing in geo-distributed pub/sub systems. In: 2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC). IEEE, 2020. p. 67–78.

[7] MATSUMOTO, Shimpei, et al. Development of a smartphone application for promoting shopping district using paper maps and Augmented Reality. In: 2021 10th International Congress on Advanced Applied Informatics (IIAI-AAI). IEEE, 2021. p. 619–624.

[8] AKIYAMA, Yuki; OGAWA, Yoshiki; YACHIDA, Osamu. Evidence-Based Policymaking of Smart City: The Case of Challenge in Maebashi City, Japan. In: Society 5.0, Digital Transformation and Disasters: Past, Present and Future. Singapore: Springer Nature Singapore, 2022. p. 55–75.

[9] Kure City, Hirosima Prefecture, Japan; Data Platform Kure URL: https://api.expolis.cloud/docs/opendata/t/kure#auth, Available at Nov, 2024 (Only in Japanese).