Unified Framework for End-User Authentication Protocol in Feature-as-a-Service Models

Jaehyung Ahn, Junhong Min, Hyung Tae Lee, and Jeongyeup Paek

Department of Computer Science & Engineering,
Chung-Ang University,
Seoul, Republic of Korea
{lbr0452000, dmc93, hyungtaelee, jpaek}@cau.ac.kr

***Abstract*—Feature-as-a-Service (FaaS) is a new cloud computing model that simplifies the implementation of new features in existing apps with minimal code modifications. One of the most important characteristics of FaaS is that the FaaS server manages its data independently of the service server. However, such a FaaS scheme has a potential issue. Specifically, in the FaaS server, authenticating end-users can be a problem because the service server usually manages user and authentication data. Although several FaaS products requiring end-user authentication have their own authentication methods to address this problem, there is no industry standard for a FaaS server to authenticate an end-user; i.e., commercial FaaS products have individualized end-user authentication methods. In this paper, we analyze existing FaaS products and propose an introductory guideline about end-user authentication methods for FaaS. This guideline aims to help FaaS customers understand the FaaS's authentication and suggest implementation direction.**

***Index Terms*—Feature as a Service (FaaS), System as a Service (SaaS), Cloud Computing, Authentication, Protocol, Guideline**

I. Introduction

Cloud computing has transformed the industry by abstracting away the complexity associated with server configuration [1]. Such abstraction can reduce the concern of constructing and maintaining server configuration. However, there are still some difficulties with developing new features in the service. Developing a new feature entails a considerable amount of varied work, such as modifying the database scheme, developing the front-end components, and handling all the interactions with existing systems. Furthermore, certain features may require extensive data collection or involve complex collaborations between vendors and clients. For example, developing geographic information systems or GIF sticker features demands significant time for data collection while implementing a chat feature necessitates intricate synchronizations between the server and client. Therefore, instead of developing them from scratch, integrating a pre-built feature module can derive substantial benefits because it can reduce the time and effort required for implementing complex features.

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2022R1A4A5034130 & No. 2021R1A2C1008840), and also by the MSIT (Ministry of Science and ICT), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2023-RS-2022-00156353) supervised by the IITP (Institute for Information & Communications Technology Planning & Evaluation)

Feature-as-a-Service (FaaS) is a cloud computing model where individual software functionalities or features are provided as a standalone service. Since FaaS offers web API (e.g. [2]) or even client UI library (e.g. [3]), it is considered a simple and efficient approach for adding new features to existing services. To simplify these development and implementation process, FaaS has unique characteristics. Notably, it is noteworthy that FaaS has a distinct server (called *'FaaS server'*), and it manages its data independently of the service servers. Due to this characteristic, there is a potential problem. Specifically, the FaaS server cannot access the end-user information in the service server. It creates a significant challenge for certain FaaS products which require end-user authentication. For instance, chatting FaaS needs to authenticate every end-user and prevent other users from accessing specific user's data. For this reason, it is essential to have some end-user authentication method for FaaS servers.

Despite the demand for end-user authentications in FaaS, there is no industry standard. Therefore, companies have developed and used unique authentication methods [4] [5], and in some cases, they may choose not to implement authentication at all [6]. However, distinct authentication methods should be adjusted to employ multiple FaaS in a single service. In this case, as the number of FaaS in a service increases, the associated logic and data maintenance costs can increase significantly. As a result, the main advantage of FaaS can be offset.

This paper focuses on the potential problem within FaaS concerning the absence of a unified guideline for end-user authentication. By creating a unified description and standard terminologies for the authentication method, FaaS customers can better comprehend the authentication logic and streamline the related codes in their applications. To do this, we first establish a guideline and terminologies for the end-user authentication method of FaaS. Then, we briefly compare the implementation methods of commercial products to provide references for developers creating new FaaS products.

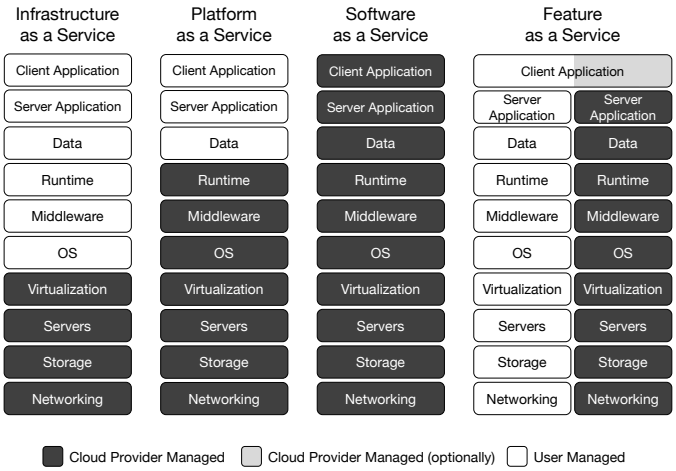

Fig. 1: Comparison between the cloud computing models [7]

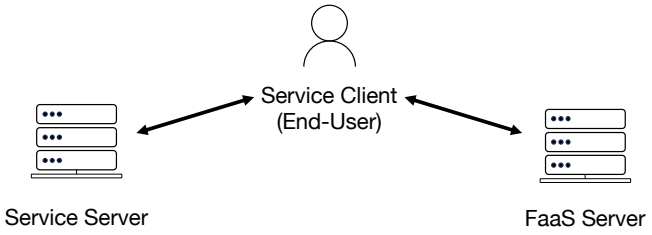

Fig. 2: Roles in feature-as-a-service (FaaS) model

II. Feature-as-a-Service

Cloud computing has three representative models, such as Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and SaaS (Software-as-a-Service) [8]. As a new model, the FaaS has a unique architectural concept at the intersection of PaaS and SaaS. In this section, we explain the FaaS architecture to comprehend why FaaS requires end-user authentication, unlike other cloud computing models.

Fig. 1 illustrates the architectural configuration of FaaS and other models. The principal concept of FaaS is that it offers an API (Application Programming Interface) or an SDK (Software Development Kit), with the backend server fully managed by the cloud provider. FaaS is similar to SaaS in providing an operational server, but they have some differences. While SaaS fully manages client-side front-end operations, FaaS can only support libraries or APIs for client-side applications because it is designed for service developers. For example, Slack [9], a SaaS-based product, provides a chatting feature that customers can directly use as end-users. On the other hand, in the case of Sendbird Chat [10], which offers a chatting feature as a service, it provides a development kit for incorporating the chatting feature. Specifically, Sendbird's customers have their own service, and within that service, end-users can seamlessly utilize Sendbird's chatting feature as if it is an integrated part of the service software.

FaaS has three entities, as shown in Fig. 2. Their roles in FaaS are as follows:

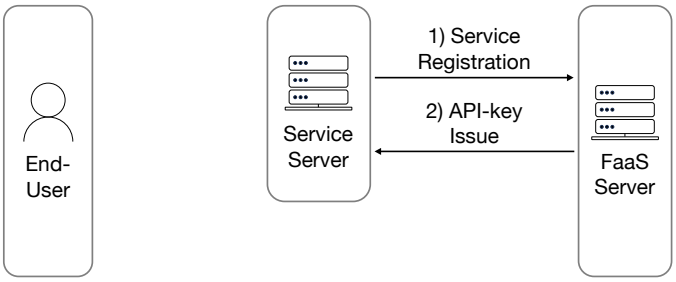

Fig. 3: Authentication between FaaS and service server

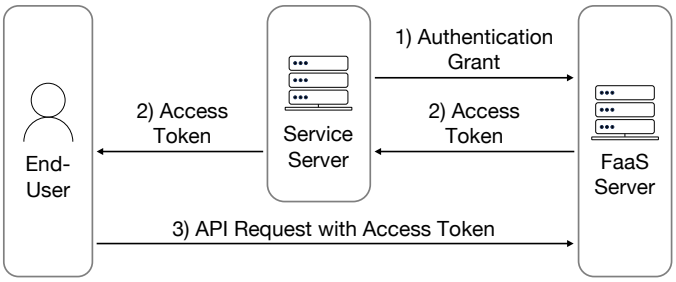

Fig. 4: Authentication between FaaS server and end-user

- **Service server** is a backend server of a service. The service server is needed to provide a login feature for end-users and other features that FaaS does not provide.
- **FaaS server** is a backend server for features that a FaaS product provides.
- **End-user** is a service application operating on an end-user's device. It can communicate with the service server and the FaaS server. End-user can interact with the FaaS server through FaaS's library or API.

In this structure, the FaaS server cannot distinguish whether a request from an end-user is a malicious impersonation or legitimate access without the assistance of the service server. Therefore, by furnishing the FaaS server with end-user information obtained from the service server, it becomes possible for the FaaS server to implement an end-user authentication method, such as sessions or JSON Web Token (JWT) [11], which can effectively identify and secure end-user information against forgery, tampering, and identity spoofing.

III. End-user Authentication in FaaS

Several companies providing FaaS have implemented their own end-user authentication methods. However, despite their implementations being similar, there is no unified description. This lack of descriptions or guidelines about designing and implementing end-user authentication methods can confuse FaaS customers and impose unnecessary burdens on developers of new FaaS products. Therefore, in this section, we propose a standardized implementation guideline and terminologies of authentication in FaaS based on their similarity.

The authentication process consists of two steps. First, the FaaS server authenticates the service server to verify the incoming requests. Then, FaaS entities establish end-user authentication.

FaaS and Service Servers: A FaaS server communicates with client applications from multiple services. Therefore, the FaaS server should distinguish every API call from which the service server has requested. Fig. 3 portrays an authentication flow between the FaaS and service servers. The service should conduct a service registration on the FaaS server to use the FaaS product. If the registration is completed, the FaaS server issues an API key. After this, the service server includes the API key on its request, and the FaaS server identifies whether the request from the service server is legitimate by checking the key.

FaaS server and End-user: Upon successful registration with the FaaS server, the service server gains the authority to request an access token for a specific user. The end-user utilizes the token when it calls the FaaS's API. In Fig. 4, steps 1 and 2 illustrate the process of issuing an access token, and step 3 demonstrates the subsequent usage of the token. The details for each step are as follows:

1) *Authentication Grant*: When the service server determines to grant the end-user for accessing FaaS server, it sends an access token request, including the end-user ID, to the FaaS server. Generally, this step is derived from the sign-in request of the end-user to the service server.
2) *Access Token Issue*: The FaaS server issues an access token for the end-user and responds with the access key. Then, the service server transfers the key to the end-user.
3) *API Request with Access Token*: The end-user includes the token when making API calls to the FaaS server. The FaaS server verifies the token and allows access only for resources authorized to the requester.

IV. Comparative analysis

The guideline consolidates the authentication protocols that have already been employed by various FaaS products. Nevertheless, certain implementation details, such as API-key or access token implementation, are left to the discretion of each individual company. While most of the implementation methods may not matter to customers utilizing the product, certain policies do have an impact. For example, in the case of access tokens, whether the product uses JWT or a session token does not significantly affect the customer, as they are merely strings. However, the presence of a token expiration time is of significance to customers, as a token with a finite expiration requires the service server to reissue it periodically. In this section, we illustrate examples of both similarities and dissimilarities in implementations that have a substantial impact on customers, using specific FaaS products as case studies: Sendbird, Twilio Conversation, Stream Chat Messaging and Stipop. Additionally, we summarize the major technical differences in Table I.

Sendbird [12] Sendbird Chat is a FaaS platform that integrates chatting, video calling, live streaming, and other functionalities into mobile applications and websites.

Twilio [13] Twilio is a communication platform known for its SMS and voice call API. Additionally, it provides a conversation and programmable video SDK, which shares similarities with the Sendbird platform.

Stream [14] Stream provides chatting and activity feed features as its core offering. Additionally, it offers video calling and streaming functionalities as beta services.

Stipop [15] Stipop provides image stickers, GIFs and other visual content that can be facilitated to contents in applications. It provides APIs to access sticker images and a dedicated SDK tailored for messaging and social media applications.

A. Service Registration

The service registration process is commonly carried out through the developer dashboard website provided by each FaaS product. FaaS customers sign up to the dashboard and perform the feature management tasks, including service registration and API-key issue.

B. API-key

FaaS products have implemented the concept of the API-key differently from one another. The simplest approach, adopted by Stipop, involves generating an immutable identifier string. However, this method is vulnerable in case the API-key gets leaked. In contrast, Sendbird Chat and Twilio Conversation offer an alternative solution with revokable API-keys, which can be generated multiple times and revoked. Moreover, Stream utilizes JWT tokens that can be generated using a secret API key, serving as tokens to authenticate the service server. The use of JWT specifications allows for the inclusion of an expiration time, enhancing security in the authentication process.

C. API-key attach location in HTTP Header

The HTTP header key for the API key also varies across FaaS products. For new FaaS products, it is advisable to avoid using keys that are already in use, as listed in Table I. Moreover, Stream utilizes the 'AUTHORIZATION' key as the request header field for the OAuth 2.0 Bearer token [16]. Therefore, Stream advises removing the 'Bearer' prefix string, which may be automatically inserted by some HTTP libraries [5].

D. Access Token

There are two types of access token implementations: persistent and expirable. The expirable token has a finite expiration period, unlike the persistent token, which does not expire. Twilio exclusively employs an expirable token, with a default TTL (Time To Live) set to 3600 seconds and a maximum validity period of 24 hours. On the other hand, Sendbird provides the flexibility for customers to choose between these options. They label the access token as 'Auth Token' and further categorize the persistent token as the 'Access Token' and the expirable token as the 'Session Token'. Meanwhile, Stipop has not made their documentation about the authentication

TABLE I: The differences of FaaS products in authentication

	API-key	API-key attach location in HTTP Header	Access Token
Sendbird	Master API Token(immutable) or Secondary API Token(revokable)	Api-Token	Access Token(persistent) or Session Token(expirable)
Twilio	Auth Token(immutable) or API key/secret pair(revokable)	TWILIO_API_KEY	Access Token(expirable)
Stream	Token(revokable and expirable)	AUTHORIZATION	Token(JWT, expirable)
Stipop	API key(immutable)	api_key	Not opened to public

grant publicly accessible; however, the documentation for the *API Request with Access Token* is available. It is presumed that the authentication method is exclusively accessible to private customers.

Lastly, Stream adopts a slightly different approach in creating an access token for the end-user. In contrast to our guideline, where the FaaS server is requested to issue an access token by service servers, Stream allows the service server to generate a JWT access key containing the end-user's identifier and encrypts it using the secret API-key.

V. Conclusion

The FaaS market is experiencing rapid growth, exemplified by Sendbird, one of the largest FaaS companies, securing $100 million in series C funding [17]. This rapid expansion suggests the likelihood of more FaaS products being introduced in the future. As a result, we anticipate that an increasing number of developers will engage with one or more FaaS products in the future. Among other factors, the necessity of end-user authentication in certain FaaS business models inevitably contributes to the significant burden faced by FaaS developers when developing individual authentication methods for each product.

Therefore, we have suggested a guideline for end-user authentication protocol for FaaS products. The unified guideline facilitates the development of secure FaaS products for emerging companies, streamlining the process and enabling faster and easier creation of robust FaaS solutions. Moreover, developers creating new FaaS products can save significant time by adopting standardized authentication methods, eliminating the need to analyze other companies' authentication processes and create accompanying documentation.

References

[1] L. Qian, Z. Luo, Y. Du, and L. Guo, "Cloud computing: An overview," in *Cloud Computing*, M. G. Jaatun, G. Zhao, and C. Rong, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 626–631.

[2] "Sendbird Chat/Platform API Docs," 2023, [last accessed Jul. 2023]. [Online]. Available: https://sendbird.com/docs/chat/v3/platform-api/overview

[3] "Sendbird Docs/UIKit Docs," 2023, [last accessed Jul. 2023]. [Online]. Available: https://sendbird.com/docs/uikit

[4] "Sendbird Docs/Chat/Platform API/Create a user/Access token vs Session token," [last accessed Aug. 2023]. [Online]. Available: https://sendbird.com/docs/chat/platform-api/v3/user/creating-users/create-a-user#2-access-token-vs-session-token

[5] "Stream Docs/Chat/Authentication," 2023, [last accessed Aug. 2023]. [Online]. Available: https://getstream.io/chat/docs/rest/#authentication

[6] "Onymos API Docs/," [last accessed Aug. 2023]. [Online]. Available: https://onymos.com/api/onymos-chat-functions/

[7] "IaaS vs. PaaS vs. SaaS - Red Hat Technology Topics," 2022, [last accessed Jul. 2023]. [Online]. Available: https://www.redhat.com/en/topics/cloud-computing/iaas-vs-paas-vs-saas

[8] C. M. Mohammed and S. R. Zeebaree, "Sufficient Comparison Among Cloud Computing Services: IaaS, PaaS, and SaaS: A Review," *International Journal of Science and Business*, vol. 5, no. 2, pp. 17–30, 2021. [Online]. Available: https://ideas.repec.org/a/aif/journl/v5y2021i2p17-30.html

[9] "Slack - Features/Messaging," [last accessed Aug. 2023]. [Online]. Available: https://slack.com/team-chat

[10] "Sendbird Docs/Chat," [last accessed Aug. 2023]. [Online]. Available: https://sendbird.com/docs/chat

[11] M. Jones, J. Bradley, and N. Sakimura, "Json web token (jwt)," Tech. Rep., 2015.

[12] "Sendbird," https://sendbird.com, [last accessed Aug. 2023].

[13] "Twilio," https://www.twilio.com, [last accessed Aug. 2023].

[14] "Stream," https://getstream.io, [last accessed Aug. 2023].

[15] "Stipop," https://stipop.io, [last accessed Aug. 2023].

[16] M. B. Jones and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage," RFC 6750, Oct. 2012. [Online]. Available: https://www.rfc-editor.org/info/rfc6750

[17] I. Lunden, "Sendbird raises $100M at a $1B+ valuation, says 150M+ users now interact using its chat and video APIs," 2021, [last accessed Aug. 2023]. [Online]. Available: https://techcrunch.com/2021/04/06/sendbird-raises-100m-at-a-1b-valuation-says-150m-users-now-interact-using-its-chat-and-video-apis/