Demo Abstract: A Serverless Approach to Publish/Subscribe Systems

Faisal Hafeez
Technical University of Munich
faisal.hafeez@tum.de

Pezhman Nasirifard
Technical University of Munich
p.nasirifard@tum.de

Hans-Arno Jacobsen
Technical University of Munich

ABSTRACT

Building reliable and scalable publish/subscribe (pub/sub) systems require tremendous development efforts. The serverless paradigm simplifies the development and deployment of highly available applications by delegating most of the operational concerns to the cloud providers. The serverless paradigm describes a programming model, where the developers break the application downs into smaller microservices which run on the cloud in response to events. In this paper, we propose a design of a serverless pub/sub system based on the Amazon Web Services Lambdas and Microsoft Azure Functions. Our pub/sub system performs topic-based, content-based and function-based matchings. The function-based matching is a novel matching approach where the subscribers can define highly customizable subscription function which the broker applies to the publications in the cloud. We also provide an evaluation application for investigating the scalability of the designed brokers on different serverless platforms.

CCS CONCEPTS

• **Software and its engineering → Publish-subscribe / event-based architectures**; **Cloud computing**;

KEYWORDS

Serverless, Function as a service (FaaS), topic-based pub/sub, content-based pub/sub, function-based pub/sub

1 INTRODUCTION

A pub/sub system relies on several middleware applications for delivering the matching publications. However, developing a large-scale distributed pub/sub system raises several concerns with scalability, availability and fault tolerance. The serverless computing, also known as the Function-as-a-Service (FaaS), represents a programming model where the developers decompose the applications into microservices, also known as functions, and deploy the functions to the cloud. Next, cloud providers are responsible for executing the functions in response to events and also for performing the automatic maintenance and scaling the resources. Despite several apparent benefits of serverless resources, the short-lived stateless nature of serverless functions introduces a new set of development challenges. In this paper, we present a pub/sub broker based on AWS Lambdas[1] and Azure Functions[2] using JavaScript and C# languages. We employ two additional cloud resources to overcome the limitation of serverless platforms. A Database-as-a-Service to persist the state of the system and the cloud's messaging queues for delivering the publications. On Amazon, we use DynamoDB[3], and Azure provides Table storage[4] to store data. Besides, Amazon provides Simple Queue Service[5] while Azure provides Service Bus[6] messaging queues. Furthermore, we provide an evaluator application for investigating the scalability and reliability of the brokers on both platforms.

Middleware '18, December 10–14, 2018, Rennes, France
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-6109-5/18/12…$15.00
https://doi.org/10.1145/3284014.3284019

2 RELATED WORK

Currently, all primary cloud providers offer serverless resources. Although the studies on pub/sub systems based on serverless paradigm is limited, some studies investigated the performance, scalability, and applicability of various serverless platforms and different factors affecting the performance of the platforms [1, 2]. In [3], the authors proposed a basic pub/sub broker based on IBM Cloud Functions. We extend their work by introducing function-based matching and improving the performance by employing AWS and Azure cloud providers to overcome the vendor-specific constraints of IBM Cloud.

3 TECHNICAL APPROACH

A pub/sub system consists of three primary components: broker, publishers, and subscribers. The broker is responsible for the delivery of publications based on a matching scheme. We decompose the broker's tasks into microservices which can be developed as serverless functions. These functions can be divided into three categories: subscribers registration or deregistration, creating subscriptions, and submitting publications.

Once a subscriber registers to the system, the broker creates a message queue instance for the subscriber and returns the connection details. Each subscriber uses a dedicated message queue for receiving the publications. The subscriber can subscribe to various subscriptions. The broker stores the subscriber message queue information and the subscription type in the database. When a publisher submits a publication along with the subscription type, broker fetches the list of subscribers for the given subscription type from the database and performs the matching scheme accordingly, and finally forwards the matching publications.

[1]https://aws.amazon.com/lambda/
[2]https://azure.microsoft.com/en-us/services/functions/
[3]https://aws.amazon.com/dynamodb/
[4]https://azure.microsoft.com/en-us/services/storage/tables/
[5]https://aws.amazon.com/sqs/
[6]https://azure.microsoft.com/en-us/services/service-bus/

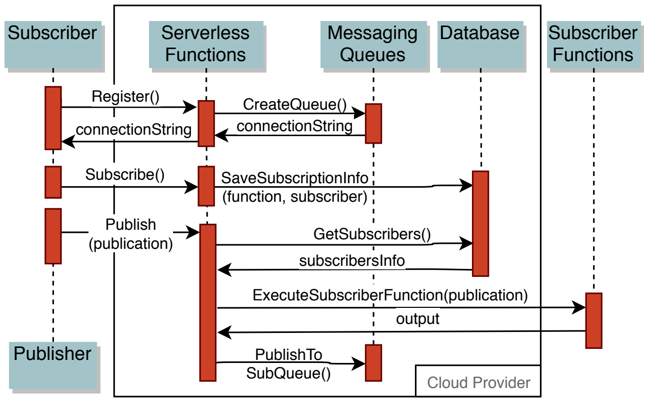

Figure 1: Function-based matching sequence diagram.

The broker provides three types of matching methodologies. While topic-based and content-based matchings are standards in pub/sub systems, we introduce function-based matching in this paper. Function-based matching is a novel and extremely expressive matching approach. The subscribers submit matching functions to a host, e.g., a serverless function, and send the address to the function to the broker as a subscription. The matching function can contain a Turing-complete code, which the serverless broker applies to the content of the publication. If the content of the publication satisfies the logic of the function, the broker forwards the matching publication to the filtered subscriber's message queue. We should emphasize that the matching function can contain any arbitrary logic, from simple calculations to performing complex image recognition methods. Figure 1 shows the system sequence diagram for function-based matching.

We open-sourced the system, and the source code is publicly available[7].

4 SOFTWARE DEMONSTRATION

To investigate the scalability, extensibility, and latency of the brokers on Amazon and Azure cloud providers, we provide a distributed evaluation application that creates heavy workloads for the brokers. The application can create several publishers and subscribers and allows subscribers to create various subscriptions. The application also allows the publishers to send publications for different subscription types. Users can monitor the evaluation result in real-time. The application provides an interface for the user to configure the evaluation parameters such the number of publishers and subscribers and the publications throughput. The application enables the user to quantify the scalability and latency of the broker in response to the increasing number of subscribers and publications. Since we are using two different cloud providers and two programming languages for developing the functions, the application also helps with comparing results for different implementations of the broker and finding the potential bottlenecks of different cloud providers. As an example, Figure 2 demonstrates the latency for delivery of 5 publications per second to 50 subscribers on AWS and Azure.

[7]https://github.com/i13tum/serverless-pubsub

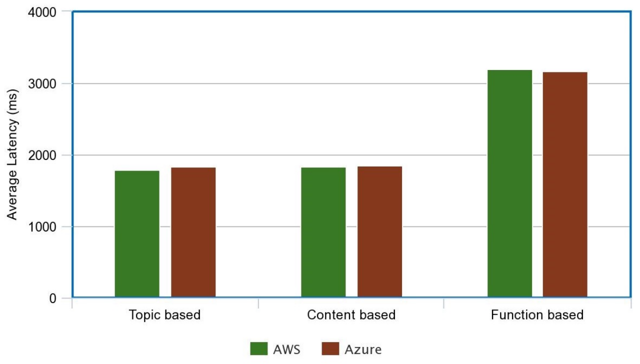

Figure 2: Latency for delivering five publications per second to 50 subscribers on AWS and Azure.

5 CONCLUSIONS

In this demo, we presented a pub/sub broker based on AWS and Azure serverless platforms. The broker is capable of performing topic-based, content-based, and function-based schemes. Furthermore, we provided evaluator application for investigating the performance and latency of the broker on different platforms. As part of our future work, we will compare the performance and scalability of the proposed system with existing systems such as RabbitMQ, ZeroMQ, and Apache Kafka.

ACKNOWLEDGMENTS

Alexander von Humboldt Foundation supported this project.

REFERENCES

[1] W. Lloyd, S. Ramesh, S. Chinthalapati, L. Ly, and S. Pallickara. 2018. Serverless Computing: An Investigation of Factors Influencing Microservice Performance. In *2018 IEEE International Conference on Cloud Engineering (IC2E)*. 159–169.
[2] G. McGrath and P. R. Brenner. 2017. Serverless Computing: Design, Implementation, and Performance. In *2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW)*. 405–410.
[3] P. Nasirifard, A. Slominski, V. Muthusamy, V. Ishakian, and H. A. Jacobsen. 2017. A Serverless Topic-based and Content-based Pub/Sub Broker: Demo. In *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference: Posters and Demos*. ACM, 23–24.