Serverless Functions: Transitioning to a Dynamic Web Publishing System

A Technical Report submitted to the Department of Computer Science

Presented to the Faculty of the School of Engineering and Applied Science
University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree
Bachelor of Science, School of Engineering

Taher Calcuttawala

Fall, 2022

Rosanne Vrugtman, Department of Computer Science

Serverless Functions: Transitioning to a Dynamic Web Publishing System

CS4991 Capstone Report, 2022

Taher Calcuttawala
Computer Science
The University of Virginia
School of Engineering and Applied Science
Charlottesville, Virginia USA
tyc7eu@virginia.edu

Abstract
A brand management technology company's existing website publishing system was incapable of generating pages dynamically, requiring the use of inefficient workarounds to overcome this challenge. I worked on a new dynamic web publishing system that allowed for serverless function integration to generate web pages. I built multiple Kafka consumers to record plugin metadata; then designed various websites to test the system end-to-end to ensure acceptable functionality. This new system significantly reduced overhead, allowing pages to be generated much more efficiently, using much less storage. Additional advantages include increased security and flexibility. The system is currently in its nascent stages and in-depth testing is still required to ensure the product works as intended at scale.

1. Introduction/Background

The brand management technology company's core platform allows businesses to collect and organize their data. The company has various products that leverage the data stored on the core platform, such as AI-powered search, SEO-ready web pages, listings, and data analytics. The team I worked on focuses on website publishing and serving, ensuring that generated pages are hosted to their corresponding geographic servers correctly. For the new dynamic web publishing system, our team was tasked with generating, storing, and serving serverless function data.

The motivation for creating this new system was the many drawbacks in the existing system. First, the new system allows for third party interaction with the company's products. The old system required an in-house consulting team to build web pages for the client on the company's platform. The new system, however, allows anyone to leverage the company's product and platform to build a webpage. Second, the new system supports serverless functions. These functions are dynamic, stored on the cloud, and rendered server side, which provides a variety of functionalities not present in the existing system.

2. Related Works

The goal of serverless architecture is to host single purpose applications that scale on demand. Traditionally, these serverless functions follow a lifecycle in which they are created, perform their task when called, and then are destroyed. According to Hall and Ramachandran (2019), this allows users to only pay when their functions are executing, and hosts gain the benefit of reduced resource consumption as the functions are not running constantly.

This defines the approach the brand management company took to solve the drawbacks with the existing system. The ease of accessibility serverless architecture provides benefits not only for the company's clients, but also greatly reduces resource consumption for the company. A serverless function implementation like this could have been used a starting point for the new dynamic publishing system.

While the concept of serverless architecture is new, there have already been substantial commercial applications. All the major cloud providers have begun implementing their own platforms for serverless architecture, allowing users to deploy applications to the market without having the infrastructure in place to support them. As a result of this widespread adoption, generally accepted conventions are now in place to optimize their implementation and use, according to Nupponen and Taibi (2020). Following these guidelines would be essential to the development of the new dynamic system, as many of the limitations of serverless functions have already been identified and researched.

3. Project Design

The overall goal of this project was to create a new web publishing system that could generate pages dynamically, using serverless functions. The company's approach to serverless functions differed from existing implementations as it was built into the existing ecosystem, leveraging already built microservices to generate, deploy and host the serverless functions. While this lack of flexibility seems like a drawback, it allows clients to seamlessly integrate serverless functions into their existing web pages for specific use cases and using static pages where the extra features are not necessary.

My team was responsible for publishing and serving a given plugin, a single instance of a serverless function. The main requirement for this system was to take a JavaScript function written by an end-user and publish it. This process would turn the client's code into a plugin our system would recognize. Once the plugin was properly generated, our system would need to keep track of it, recording whether it was up and running and which regions it was published in.

3.1 System Architecture

As shown in Figure 1, the plugin publishing system is comprised of five major components: The user interface, the publishing server, the publishing regions, the feedback processor, and the database.

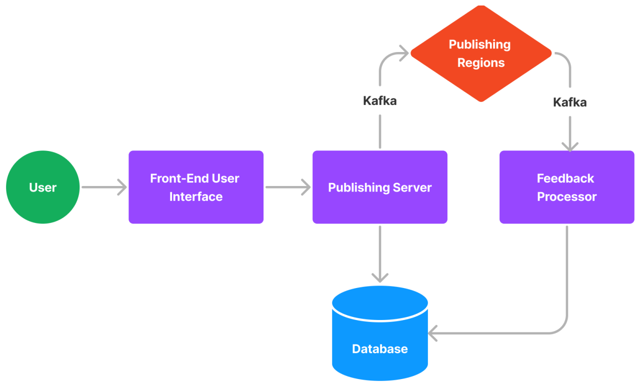


Figure 1: System Architecture

A user interacts with the front-end user interface, where they create their JavaScript function. The user-defined function is then sent to the publishing server, where the code is published into a plugin. Once the plugin data has been created, it is stored in the database. The publishing server also sends the plugin to each of the geographic publishing regions, where they are ready to be invoked. This is done using a Kafka consumer in each publishing region, which consumes the data sent by the publishing server. The geographic publishing regions are copies of each other and should ideally hold the same information. Their redundancy and strategic geographic positioning allow for high-speed access of plugins stored within them.

Once a plugin is received by a publishing region, it interacts with the feedback processor, confirming that the plugin has been successfully published. This is done using another Kafka consumer, located in the feedback processor, which consumes the confirmation sent by a publishing region. The feedback processor then updates the database with this confirmation.

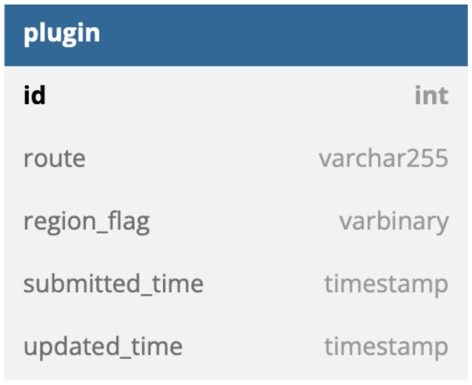
3.2 Database Interaction

The publishing server and the feedback processor read and write information about a plugin to the database. This is crucial in informing internal services about the health and status of a published plugin, as querying the internal database is much quicker than pinging the publishing regions. For security and ease-of-use, I wrote custom query functions into both the publishing server and feedback processor to query the database. These custom functions allowed a specific plugin to be accessed or modified given its primary key.

Additionally, since the database was being accessed indirectly through function calls, there was protection from malicious attacks such as SQL-Injections. In the publishing server, these query functions added a plugin to the database. In the feedback processor, these functions updated the correct plugin's data, confirming that it had been successfully published to the publishing region.

3.3 Database Design

To allow plugins to be reliably queried, a robust database design is required. A SQL database was used to store plugin data, which includes the id, route, region flag, submitted time, and updated time, as shown in Figure 2.



plugin	
id	**int**
route	varchar255
region_flag	varbinary
submitted_time	timestamp
updated_time	timestamp

Figure 1: Plugin Database Table

The id field represents the primary key and can be used to solely identify a specific plugin. The route refers to the URL path the plugin resides. The submitted time is the field that records the timestamp the plugin was published. The region flag is a binary field that represents whether a given plugin has been published to a specific geographic region. Consider a system where there are five regions, each with an index 0 through 4. If the region flag field is “00000,” we know that the plugin has not been published to any region. If the region flag is “10111,” we know that the plugin has been published to every region except the one with index 1. This field notifies the internal system which regions the plugin has been published to. The updated time field is also a timestamp field, which is updated to the latest time whenever the region flag is updated. It is important to note that the actual plugin data is not stored in the database; only information about the status of a plugin is stored. The actual plugin code is stored within the publishing regions, where it would be invoked.

4. Results

The system is currently in the testing stages. At the end of my time with the company, I wrote various functions to test the system end-to-end to ensure proper functionality for each component. I was successful in generating multiple serverless functions, which showcased the new system working, and the various advantages the dynamic system had over the static one.

5. Conclusion

This new publishing system provides many new features. One of these functionalities includes API key protection through server-side rendering, which greatly reduces API key usage and costs associated with usage thresholds. Another feature of serverless functions is dynamic routing, which allows different serverless functions to be mapped to different URL paths and allows URL parameters to be passed to those functions if necessary, greatly reducing the number of static pages that need to be stored. These are just a couple examples of the various advantages of serverless functions allowing the company to cater to a much larger host of potential clients.

6. Future Work

The new publishing system still requires various features to be ready for client use. The current system only supports GET requests, meaning plugins can only display information. While URL arguments can be used as a workaround to send data, it is not secure. An important feature to add in the future would be POST request support, allowing plugins to send data securely to a server or another plugin. This feature would greatly increase their versatility and security.

A dedicated web interface would be necessary for users to create, update and monitor their plugins. While this feature is purely visual, it would increase user engagement and ease of use. Adding this feature would not be difficult, as the company has existing status pages for its other products. Building this feature into a user’s existing portal would be seamless.

Finally, although the functionality of the publishing system was tested thoroughly, a load test has not been done. This kind of testing would be essential in determining whether the new system could handle many concurrent users. One of the strengths of serverless functions is their ability to scale on demand, so it would be imperative to test this.

References

[1] Hall, A. and Ramachandran, U. 2019. “An execution model for serverless functions at the edge,” Proceedings of the International Conference on Internet of Things Design and Implementation.

[2] Nupponen, J. and Taibi, D. 2020. “Serverless: What it is, what to do and what not to do,” 2020 IEEE International Conference on Software Architecture Companion (ICSA-C).