1. **AWS performance comparison (CloudFront function) and Lambda@edge:**

Comparing AWS CloudFront Functions and Lambda@Edge involves understanding their respective roles and capabilities within the AWS ecosystem, particularly in terms of performance and use cases:

**1. AWS CloudFront Functions:**

* **Role:** CloudFront Functions allow you to deploy lightweight JavaScript functions across AWS's global edge locations, integrated directly with the CloudFront content delivery network (CDN).
* **Use Cases:** They are primarily used for manipulating HTTP requests and responses at the edge, such as header manipulation, URL rewriting, A/B testing, and lightweight processing of content before it reaches the origin server or the client.
* **Performance:** CloudFront Functions are designed for low-latency execution at the edge. They operate within the CloudFront infrastructure, leveraging the global network of edge locations for fast execution close to end users.

**2. Lambda@Edge:**

* **Role:** Lambda@Edge enables you to run Lambda functions at AWS's edge locations (CloudFront edge locations), allowing you to customize the content that CloudFront delivers to end users.
* **Use Cases:** Lambda@Edge can be used for a wide range of tasks including dynamic content generation, authentication and authorization, origin request and response handling, and real-time image transformation.
* **Performance:** Lambda@Edge functions also benefit from the global network of CloudFront edge locations, providing low-latency execution. They are suitable for tasks that require more complex processing or integration with other AWS services.

**Performance Comparison:**

* **Latency:** Both CloudFront Functions and Lambda@Edge execute at the edge, minimizing latency by processing requests closer to end users. Both are optimized for low-latency operations.
* **Execution Environment:** CloudFront Functions are simpler in scope compared to Lambda@Edge, focusing on HTTP request/response manipulation. This simplicity can lead to faster execution times for lightweight tasks compared to Lambda@Edge, which can handle more complex computations and integrations.
* **Complexity:** Lambda@Edge allows you to leverage the full capabilities of AWS Lambda, including integration with other AWS services like DynamoDB, S3, and others. This flexibility comes with the overhead of managing more complex deployments and potentially higher costs depending on usage patterns.

**Choosing Between CloudFront Functions and Lambda@Edge:**

* **CloudFront Functions** are ideal for lightweight edge processing tasks that do not require extensive computation or integration with other AWS services. They offer simplicity and lower operational overhead.
* **Lambda@Edge** is suitable for more complex use cases that require deeper integration with AWS services or more intensive computations at the edge. It provides flexibility and scalability but may involve higher management complexity.

**Conclusion:**

* For simple HTTP request/response manipulation tasks and lightweight edge processing, CloudFront Functions can provide fast performance with minimal overhead.
* For more complex tasks that require integration with other AWS services or involve heavier computations at the edge, Lambda@Edge offers greater flexibility and capability, albeit potentially with slightly higher latency due to its broader scope and flexibility.

1. **S3 and DynamoDB for Lambda@edge:**

The speed of reading data from AWS S3 versus DynamoDB depends on several factors such as the size of data, the way data is structured, and the type of operations you are performing.

**AWS S3 (Simple Storage Service):**

* **Use Cases:** S3 is commonly used for storing large amounts of unstructured data such as images, videos, backups, and log files.
* **Performance:** S3 is optimized for high durability and scalability but typically has higher latency compared to DynamoDB for individual read operations, especially if the data is large or spread across many objects.
* **Access Pattern:** Reading data from S3 involves fetching entire objects or ranges of objects. While S3 can handle high throughput for large-scale data retrieval, retrieving smaller amounts of data might incur higher latency due to network overhead and object retrieval times.

**AWS DynamoDB:**

* **Use Cases:** DynamoDB is a NoSQL database service designed for applications that need low-latency, consistent performance at any scale.
* **Performance:** DynamoDB excels in fast and predictable read and write operations, especially for small to medium-sized data sets. It provides single-digit millisecond latency for reads and writes, making it suitable for real-time applications where responsiveness is crucial.
* **Access Pattern:** DynamoDB allows fine-grained control over read operations, supporting both strongly consistent and eventually consistent reads. It is optimized for both primary key queries and secondary index queries, providing flexibility in how data is accessed.

**Conclusion:**

* For small to medium-sized datasets or when low-latency reads are critical, DynamoDB generally offers faster read performance compared to S3.
* S3 is optimized for storing and retrieving large objects or massive amounts of data at scale, but individual read operations might be slower compared to DynamoDB, especially for smaller datasets or real-time use cases.

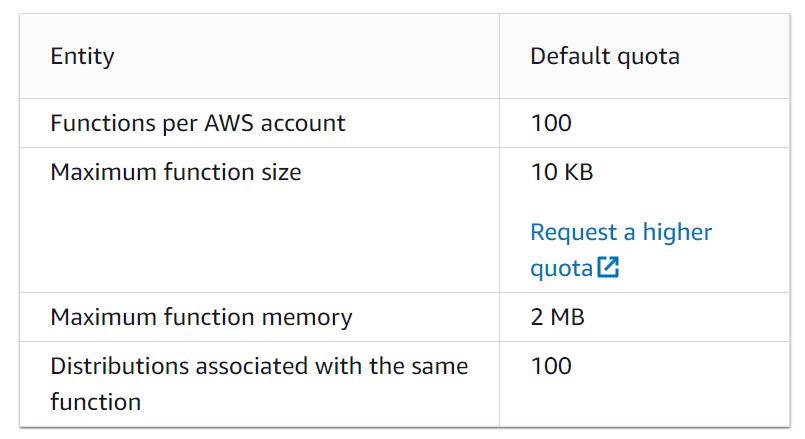
1. **Comparative analysis between CloudFront (CFF+KVS) and Lambda@Edge:**

In summary, choose **Lambda@Edge for complex scenarios** where you need deep integration with AWS services and more control over your edge logic. Opt for **CloudFront Functions for simpler operations** that can be handled within a lightweight JavaScript execution environment at the edge.

Below is the table of comparison:

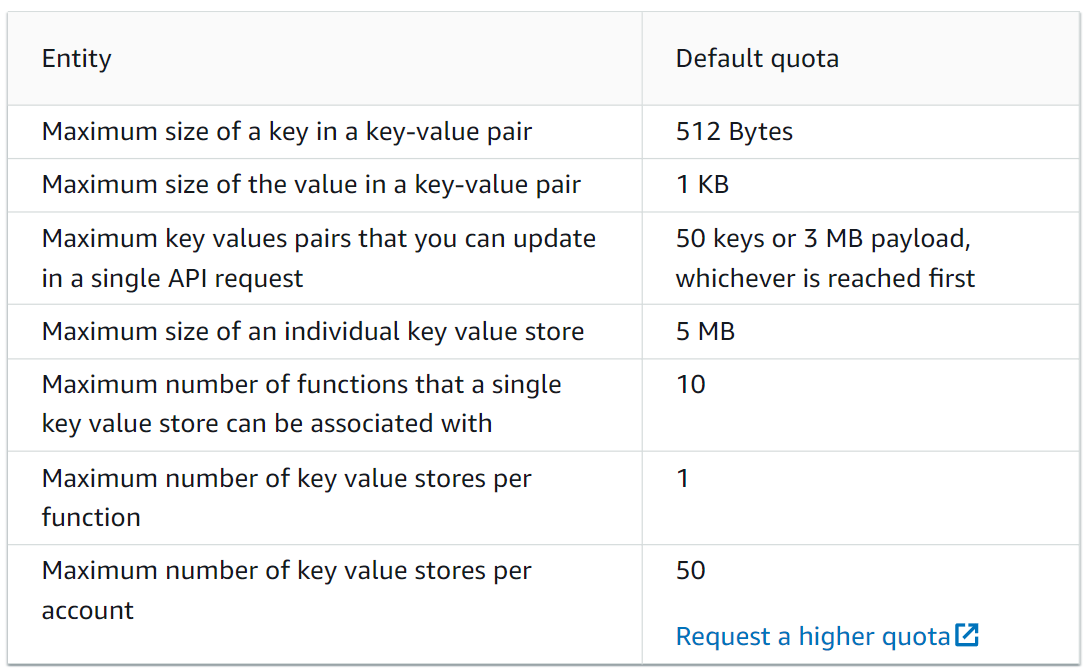
|  |  |  |  |
| --- | --- | --- | --- |
| **Serial** | **Description** | **CFF+KVS** | **Lambda@Edge** |
| 1 | Event sources | * Viewer request * Viewer response | * Viewer request * Viewer response * Origin request * Origin response |
| 2 | Supports [Amazon CloudFront KeyValueStore](https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/kvs-with-functions.html) | Yes | No |
| 3 | Maximum size of the function code and included libraries | 10KB | 1 MB (viewer request and viewer response)  50 MB (origin request and origin response) |
| 4 | Pricing | Free tier available; charged per request | No free tier; charged per request and function duration |
| 5 | **Flexibility** | They are designed to provide lightweight, simpler, and faster ways to modify the content delivered through CloudFront.  They are primarily used for lightweight operations like modifying headers, modifying URLs, or redirecting requests. | More control over the code and can implement complex logic. For example, you can modify headers, rewrite URLs, or customize content based on viewer characteristics or geographic location. |
| 6 | Environment | CloudFront Functions run in an isolated, sandboxed JavaScript environment optimized for low latency. |  |
| 7 | Complexity vs. Simplicity | CloudFront Functions are simpler and quicker to set up but offer limited capabilities compared to Lambda@Edge. | Lambda@Edge is more powerful and flexible but requires more setup and management overhead. |
| 8 | Use Cases | Functions are ideal for lightweight operations and quick modifications. | Lambda@Edge is suitable for complex operations and custom logic based on specific CloudFront events. |
| 9 | Integration | CloudFront Functions are more self-contained and optimized for specific edge operations within CloudFront. | Lambda@Edge can integrate more deeply with other AWS services.  It integrates closely with other AWS services and resources, allowing you to access and manipulate S3 buckets, DynamoDB tables, and other AWS resources directly. |

**Quotas on CloudFront Functions:**

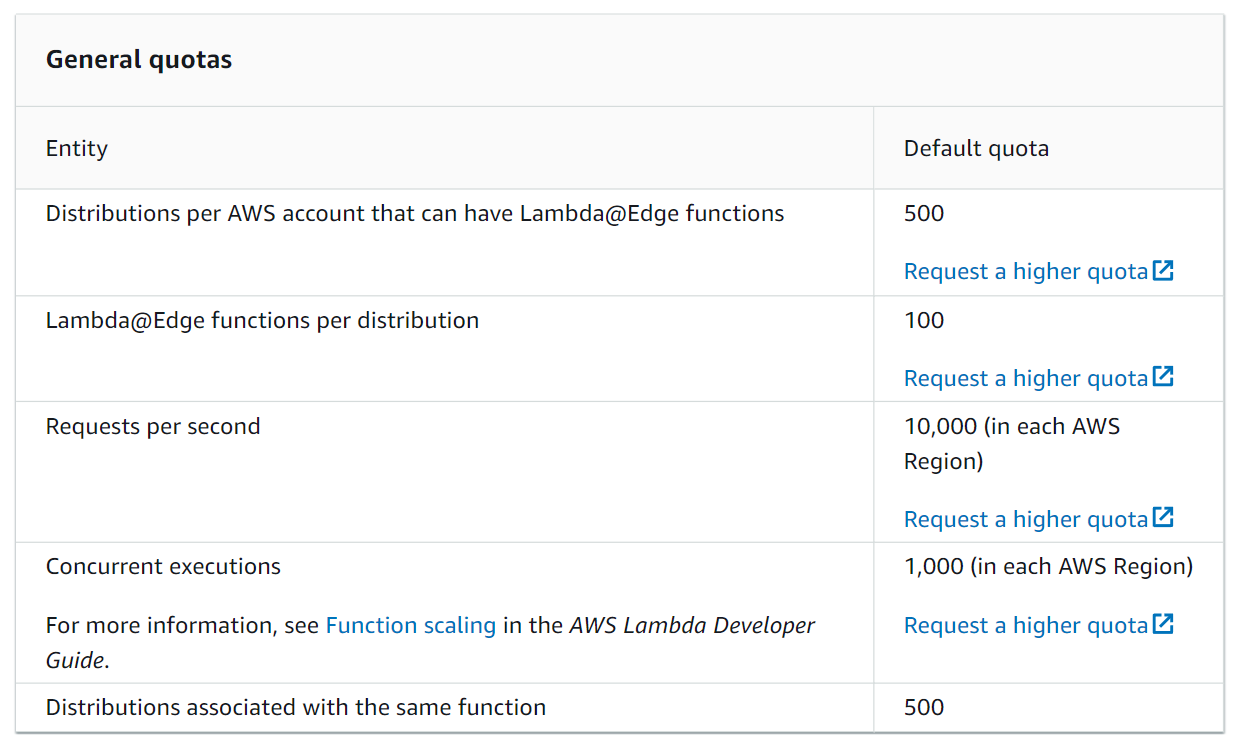
****

[**https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/cloudfront-limits.html#limits-functions**](https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/cloudfront-limits.html#limits-functions)

**Quotas on key value stores:**

****

**Quotas on Lambda@Edge:**

****

[**https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/cloudfront-limits.html#limits-lambda-at-edge**](https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/cloudfront-limits.html#limits-lambda-at-edge)

**Appendix**:

<https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/cloudfront-limits.html#limits-keyvaluestores>

<https://aws.amazon.com/lambda/pricing/>

<https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/edge-functions-choosing.html>

<https://aws.amazon.com/blogs/aws/introducing-cloudfront-functions-run-your-code-at-the-edge-with-low-latency-at-any-scale/>

<https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/cloudfront-limits.html#limits-keyvaluestores>

<https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/cloudfront-limits.html#limits-lambda-at-edge>