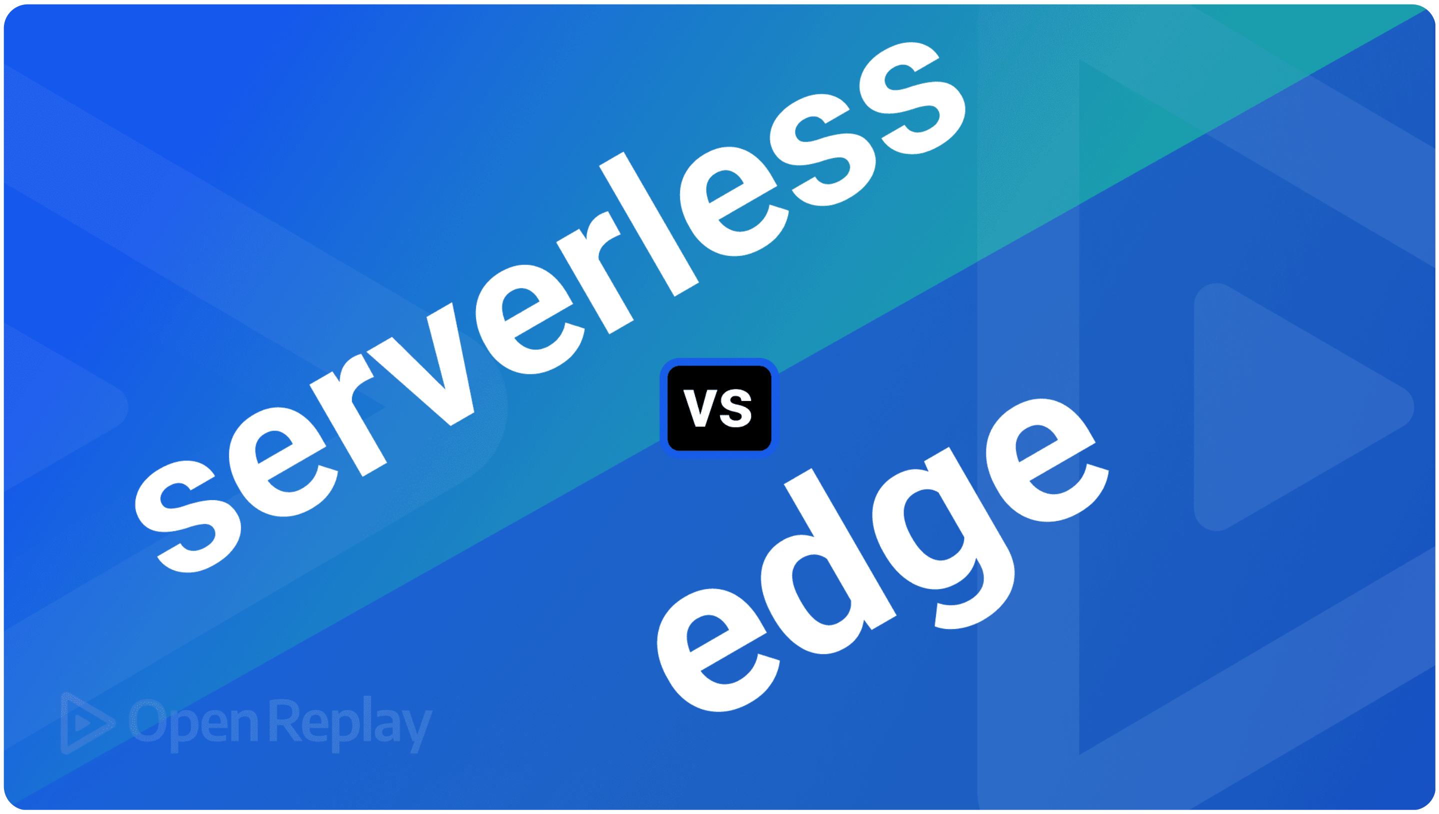
< BACK

Serverless Vs. Edge Functions

Goodluck Woha Feb 15, 2023 · 3 min read



In this comparison article, we'll take a deep look into the strengths and weaknesses of serverless and edge functions for web development. For a long time, serverless functions have been the norm for developers to write programmatic functions on a server without managing anything related to the server it is running on. Then came Edge functions in 2019, a newer version of serverless functions run at the edge. When we say "run at the edge," we mean that these functions are run at a server closest to the function caller.

A serverless function is a cloud computing service that allows a developer to run code without managing or maintaining a server's underlying infrastructure. It is an alternative to a traditional server-side application and is typically invoked in response to an event, such as a user action.

An edge function is another type of serverless function, but its code is run closer to the end user. It operates on the edge of the cloud and utilizes the computing power of connected devices or services within the cloud. Edge functions are designed to provide faster and more efficient computing operations by allowing data processing and analysis to happen where the data is generated rather than sending the data to the cloud for processing.

Serverless and edge functions are both supported by cloud platforms such as Amazon Web Services(AWS), Microsoft Azure, IBM Cloud, Google Cloud, Oracle Cloud, and more.

Edge functions claim lower latency and improved performance compared to traditional serverless functions. Today, we'll find out if there is truth to these claims.

Use Cases

Serverless and edge functions offer similar use cases, such as the IoT, image, and video processing, real-time data analytics, backend for mobile and web apps, and chatbots. However, specific scenarios exist in which each technology excels against the other.

Serverless functions are best suited for event-driven applications that require rapid scaling and minimal infrastructure management. This makes it popular for applications such as image and video processing, IoT, and real-time data analytics.

On the other hand, edge functions are ideal for applications that require low latency and high bandwidth, such as gaming, augmented reality, and live streaming. They are also useful for applications that involve working with sensitive data, as they allow for data

Scalability

Serverless functions take the lead in terms of scalability because most, like AWS Lambda, Azure Functions, and Google Cloud Functions, run on a cloud-based infrastructure with a large number of servers that allow them to automatically scale up and down based on their given workload.

Edge functions run on edge devices like gateways, routers, or IoT devices, which are limited in number and resources. Unlike serverless functions, which automatically scale up and down based on the workload, scaling edge functions requires manual intervention.

Latency Speeds

In this section, we'll put both functions' latency speeds to the test using Postman and APImetrics. Postman is an API platform that can easily be used to build and test APIs. When testing with Postman, it will tell us the exact time it took for our call to get to the function and back to our local computer. Since your application will likely be used by people worldwide, we'll also make calls with APImetrics to better benchmark the latency speeds in other countries.

I have a NextJS project created with two folders named "serverless" and "edge" in NEXT's default api folder. Each contains its respective function type.

```
latencytest
    ├── pages
    │   └── api
    │       └── serverless
    │           └── hello.js
    │       └── edge
    │           └── hello.js
    │
```

NEXT is a popular React framework for deploying server-side rendered applications. It is well known that files in its api folders are deployed as serverless functions. However, NEXT has added a new runtime feature that allows functions in the api folder to be deployed on the edge.

Inside the hello.js of the serverless folder is the following code:

```
export default function handler(req, res) {
      res.status(200).json({ message: "Hello Serverless!" });
    }
```

And inside the hello.js of the edge folder is:

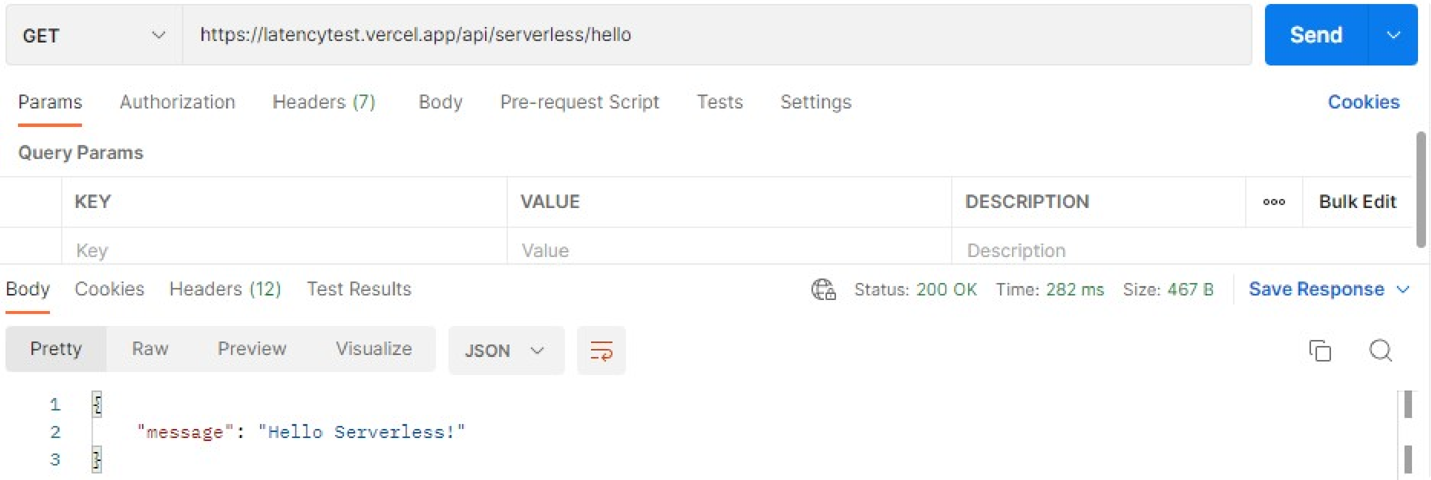
```
import { NextResponse } from "next/server";

export default async function handler() {
  return NextResponse.json({ message: "Hello Edge!" });
}

export const config = {
  runtime: "experimental-edge",
};
```

After deploying both functions with `vercel deploy`, the latency speed of the serverless function was around 287 ms, and the latency speed of the edge function was around 167 ms from my local computer.

Serverless:

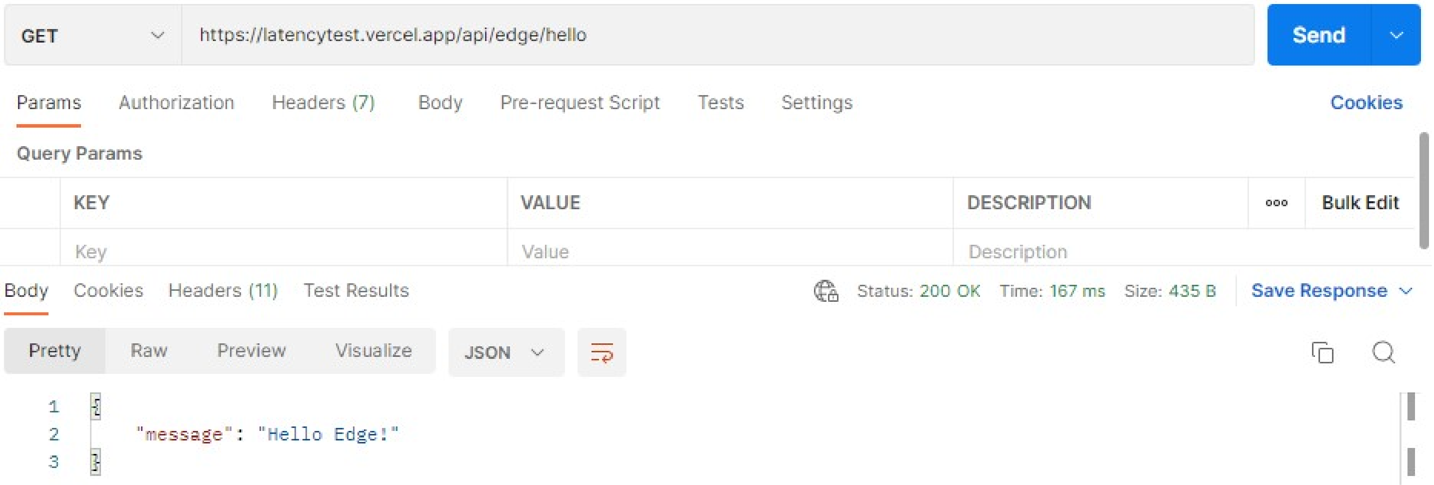


https://latencytest.vercel.app/api/serverless/hello

Status: 200 OK Time: 282 ms Size: 467 B

```
{
    "message": "Hello Serverless!"
}
```

Edge:



https://latencytest.vercel.app/api/edge/hello

Status: 200 OK Time: 167 ms Size: 435 B

```
{
    "message": "Hello Edge!"
}
```

Let’s see how both functions perform when called from different countries. The results shown below were derived with APImetrics:

Country	Serverless	Edge
United States	111 ms	93 ms
Japan	96 ms	86 ms
Singapore	95 ms	90 ms
Australia	101 ms	89 ms

Edge functions definitely have the advantage in terms of latency speed. Edge functions are also less likely to experience the “cold start” issue common with serverless functions that lead to delayed api responses.

Integration with other Tech Stacks

Regarding integration with other tech stacks, both serverless and edge functions have their own advantages and limitations.

For example, edge functions are designed to run on specific edge devices like gateways. As a result, edge functions are tightly integrated with the underlying hardware and are challenging to integrate with other technologies. However, they can be integrated with CDNs, DNS, and Firewalls as long as they run on the same edge devices.

Serverless functions are designed to be platform-agnostic and can easily integrate with a broader range of other technologies, such as databases, messaging queues, and other cloud services. This makes them a good fit for organizations with various tech stacks that need to integrate with multiple systems. Overall, serverless functions are more flexible when integrating with other technologies.

Development and Deployment

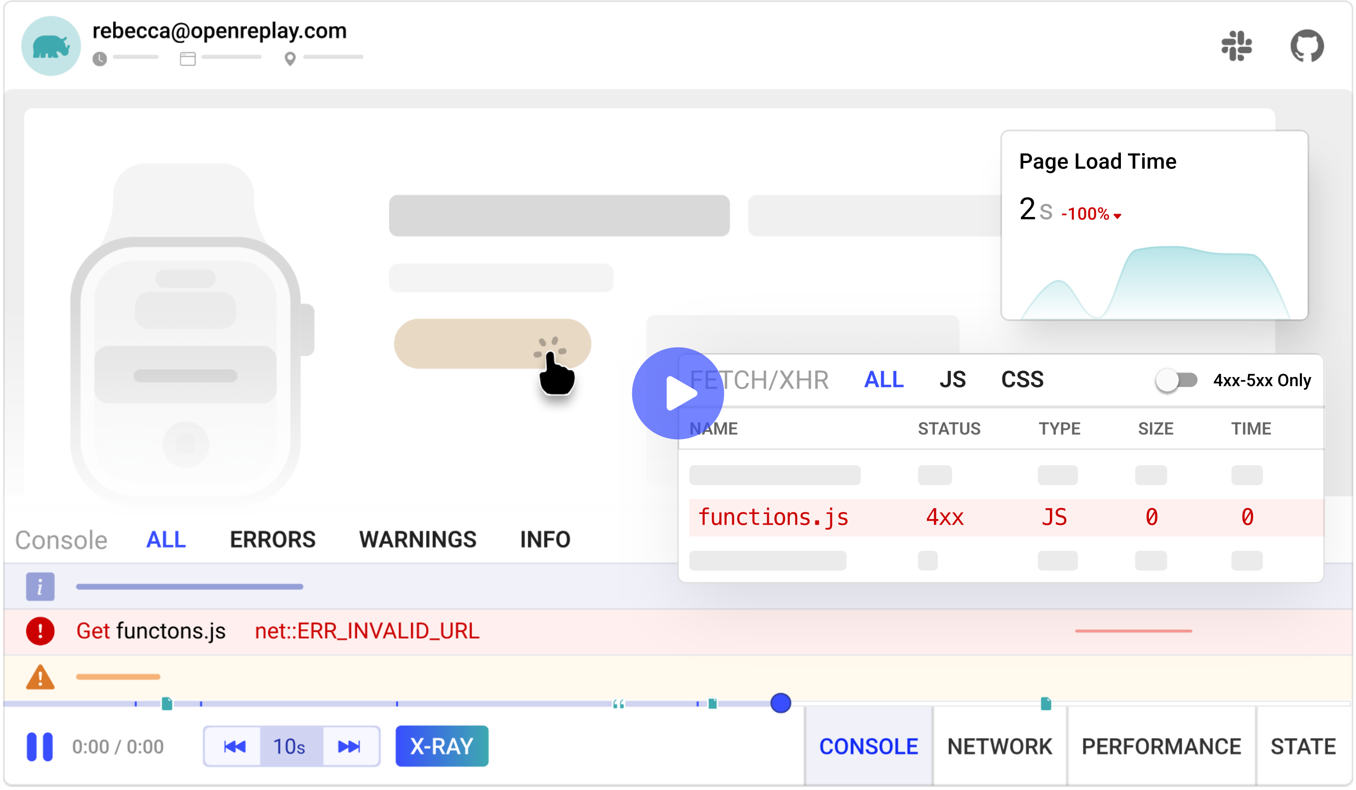
Serverless functions offer a more streamlined development process, and this is because they abstract away more of the underlying server infrastructure.

When working with edge functions, developers need to consider the constraint and limitations of the edge devices running the function because edge functions often require more manual intervention and configuration for deployment.

Final Verdict

While edge functions do, in fact, offer lower latency and improved performance, serverless functions provide greater scalability, easier integration, and development experience than edge functions. Edge functions are best suited for applications that require low latency and high bandwidth.

If you need a more scalable solution with more out-of-the-box integrations, serverless functions might be the way to go. On the other hand, if you need low latency and high bandwidth for your application, edge functions are a better option. You can have the best of both worlds by creating a hybrid serverless application. For example, both functions can be used for an e-commerce site. Serverless functions can handle payment processing, while edge functions can handle real-time inventory management and delivery tracking. Another example is an online game that uses edge functions for its game logic and physics simulations while utilizing serverless functions for user account management and leaderboard functionality.

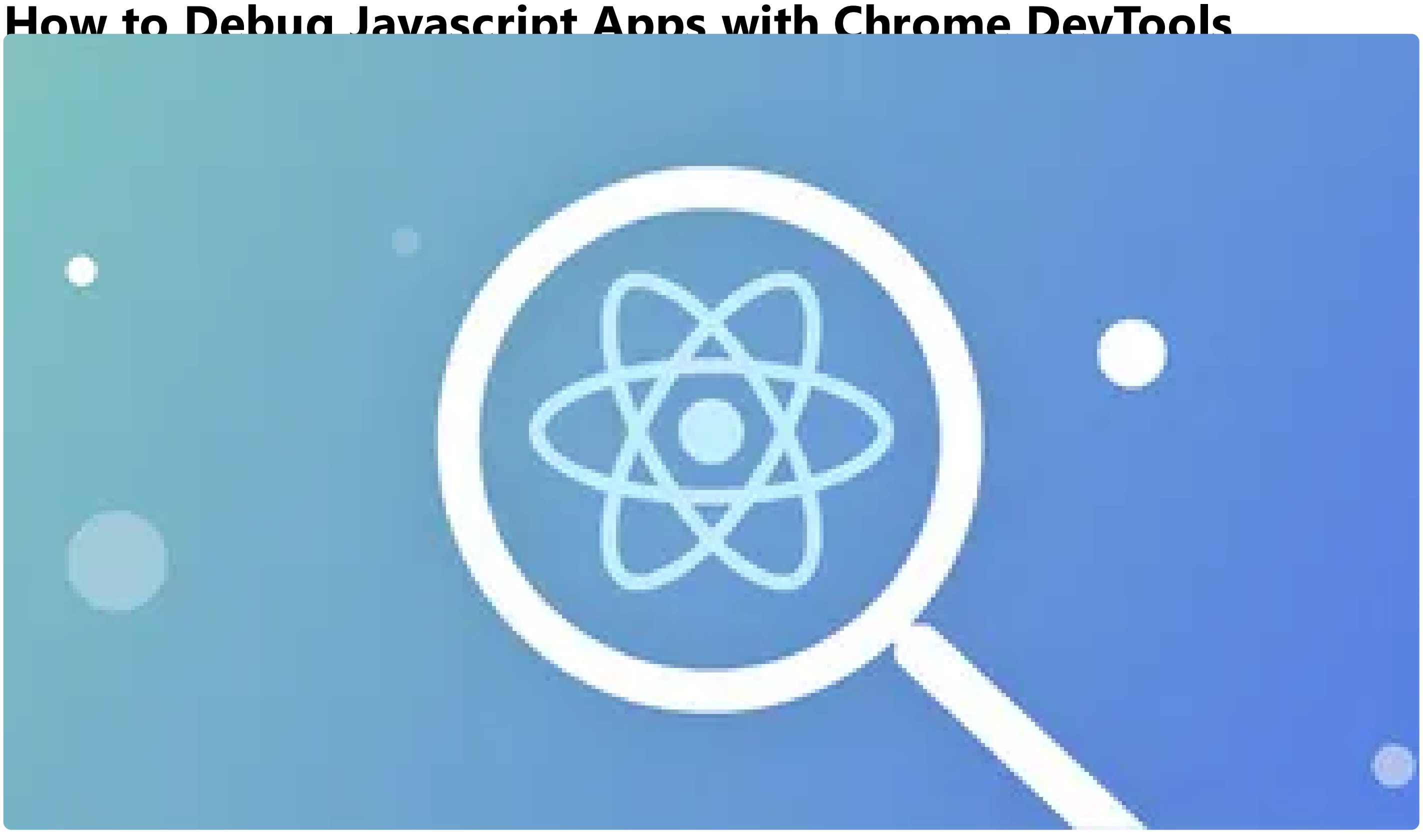

rebecca@openreplay.com
Page Load Time
2 s -100%
FETCH/XHR ALL JS CSS 4xx-5xx Only
NAME STATUS TYPE SIZE TIME
functions.js 4xx JS 0 0
Console ALL ERRORS WARNINGS INFO
Get functons.js net::ERR_INVALID_URL
0:00 / 0:00
10s
X-RAY
CONSOLE NETWORK PERFORMANCE STATE

Check our GitHub Repo

More articles from OpenReplay Blog

Sep 18, 2020, 6 min read

How to Debug Javascript Apps with Chrome DevTools



Mar 4, 2020, 6 min read

How to Debug ReactJS with Chrome DevTools