Edge and Node.js Runtimes

In the context of Next.js, "runtime" refers to the set of libraries, APIs, and general functionality available to your code during execution.

Next.js has two server runtimes where you can render parts of your application code:

- Node.js Runtime
- Edge Runtime

Each runtime has its own set of APIs. Please refer to the Node.js Docs and Edge Docs for the full list of available APIs. Both runtimes can also support streaming depending on your deployment infrastructure.

By default, the `app` directory uses the Node.js runtime. However, you can opt into different runtimes (e.g. Edge) on a per-route basis.

Runtime Differences

There are many considerations to make when choosing a runtime. This table shows the major differences at a glance. If you want a more in-depth analysis of the differences, check out the sections below.

	Node	Serverless	Edge
Cold Boot	/	~250ms	Instant
HTTP Streaming	Yes	Yes	Yes
IO	All	All	`fetch`
Scalability	/	High	Highest
Security	Normal	High	High
Latency	Normal	Low	Lowest

	Node	Serverless	Edge
npm Packages	All	All	A smaller subset

Edge Runtime

In Next.js, the lightweight Edge Runtime is a subset of available Node.js APIs.

The Edge Runtime is ideal if you need to deliver dynamic, personalized content at low latency with small, simple functions. The Edge Runtime's speed comes from its minimal use of resources, but that can be limiting in many scenarios.

For example, code executed in the Edge Runtime on Vercel cannot exceed between 1 MB and 4 MB, this limit includes imported packages, fonts and files, and will vary depending on your deployment infrastructure.

Node.js Runtime

Using the Node.js runtime gives you access to all Node.js APIs, and all npm packages that rely on them. However, it's not as fast to start up as routes using the Edge runtime.

Deploying your Next.js application to a Node.js server will require managing, scaling, and configuring your infrastructure. Alternatively, you can consider deploying your Next.js application to a serverless platform like Vercel, which will handle this for you.

Serverless Node.js

Serverless is ideal if you need a scalable solution that can handle more complex computational loads than the Edge Runtime. With Serverless Functions on Vercel, for example, your overall code size is 50MB including imported packages, fonts, and files.

The downside compared to routes using the Edge is that it can take hundreds of milliseconds for Serverless Functions to boot up before they begin processing requests. Depending on the amount of traffic your site receives, this could be a frequent occurrence as the functions are not frequently "warm".