Edge Functions

Edge Functions are Vercel Functions that run on the Edge Runtime, a minimal JavaScript runtime that exposes a set of Web Standard APIs.

- **Lightweight runtime**: With a smaller API surface area and using V8 isolates, Edge runtime-powered functions have a slim runtime with **only a subset** of Node.js APIs are exposed
- **Globally distributed by default**: Vercel deploys all Edge Functions globally across its Edge Network, which means your site's visitors will get API responses from data centers geographically near them

⚠ We recommend migrating from edge to **Fluid compute** for improved performance and reliability.

Edge Functions and your data source

Edge Functions execute in the region closest to the user, which could result in longer response times when the function relies on a database located far away. For example, if a visitor triggers an Edge Function in Japan, but it depends on a database in San Francisco, the Function will have to send requests to and wait for a response from San Francisco for each call.

To avoid these long roundtrips, you can limit your Edge Functions to **regions near your database**, or you could use a globally-distributed database.

Feature support

Feature	Support Status
Secure Compute	Not Supported
Streaming	Supported
Cron jobs	Supported
Vercel Storage	Supported
Edge Config	Supported
OTEL	Not supported

Streaming

Streaming refers to the ability to send or receive data in a continuous flow.

The **Edge runtime** supports streaming by default.

Edge Functions **do not** have a maximum duration, but you **must** send an *initial* response within 25 seconds. You can continue **streaming a response** beyond that time.

All streaming functions support the `waitUntil` **method**, which allows for an asynchronous task to be performed during the lifecycle of the request.

Cron jobs

Cron jobs are time-based scheduling tools used to automate repetitive tasks. When a cron job is triggered through the **cron expression**, it calls a Vercel Function.

From your function, you can communicate with a choice of **data stores**. To ensure low-latency responses, it's crucial to have compute close to your databases. Always deploy your databases in regions closest to your functions to avoid long network roundtrips. For more information, see our **best practices** documentation.

Edge Config

An **Edge Config** is a global data store that enables experimentation with feature flags, A/B testing, critical redirects, and IP blocking. It enables you to read data at the edge without querying an external database or hitting upstream servers.

Location

Edge Functions are executed close to the end-users across Vercel's global network.

When you deploy Edge Functions, there are considerations you need to make about where it's deployed and executes. Edge Functions are executed globally and in a region close to the user's request. However, if your **data source** is geographically far from this request, any response will be slow. Because of this you can opt to **execute your function closer to your data source**.

Failover mode

Vercel's **failover mode** refers to the system's behavior when a function fails to execute because of data center downtime.

Vercel provides **redundancy** and automatic failover for Edge Functions to ensure high availability.

File system support

Isolation boundary

In Vercel, the isolation boundary refers to the separation of individual instances of a function to ensure they don't interfere with each other. This provides a secure execution environment for each function.

As the Edge runtime is built on the **V8 engine**, it uses V8 isolates to separate just the runtime context, allowing for quick startup times and high performance.

Bundle size limits

Vercel places restrictions on the maximum size of the deployment bundle for functions to ensure that they execute in a timely manner. Edge Functions have plan-dependent size limits. This is the total, compressed size of your function and its dependencies after bundling.

Memory size limits

Edge Functions have a fixed memory limit. When you exceeds this limit, the execution will be aborted and we will return a `502` error.

The maximum size for a Function includes your JavaScript code, imported libraries and files (such as fonts), and all files bundled in the function.

If you reach the limit, make sure the code you are importing in your function is used and is not too heavy. You can use a package size checker tool like **bundle** to check the size of a package and search for a smaller alternative.

Request body size

In Vercel, the request body size is the maximum

Edge Functions have the following limits applied to the request size:

Name
Maximum URL length
Maximum request body length
Maximum number of request headers
Maximum request headers length

Edge Function API support

Edge Functions are neither Node.js nor browser applications, which means they don't have access to all browser and Node.js APIs. Currently, the Edge runtime offers **a subset of browser APIs** and **some Node.js APIs**.

There are some restrictions when writing Edge Functions:

- Use ES modules
- Most libraries which use Node.js APIs as dependencies can't be used in Edge Functions yet. See **available APIs** for a full list
- Dynamic code execution (such as `eval`) is not allowed for security reasons. You must ensure **libraries used in your Edge Functions don't rely on dynamic code execution** because it leads to a runtime error. For example, the following APIs cannot be used:

API	Description
`eval`	Evaluates JavaScript code
`new Function(evalString)`	Creates a new function wit argument
`WebAssembly.instantiate`	Compiles and instantiates a buffer source

See the **Edge Runtime supported APIs** for more information.

Limits on fetch API

- You cannot set non-standard port numbers in the fetch URL (e.g., `https://example.com:8080`). Only `80` and `443` are allowed. If you set a non-standard port number, the port number is ignored, and the request is sent to port `80` for `http://` URL, or port `443` for `https://` URL.
- The maximum number of requests from `fetch` API is **950** per Edge Function invocation.
- The maximum number of open connections is **6**.
  - Each function invocation can have up to 6 open connections. For example, if you try to send 10 simultaneous fetch requests, only 6 of them can be processed at a time. The remaining requests are put into a waiting queue and will be processed accordingly as those in-flight requests are completed.
- If in-flight requests have been waiting for a response for more than 15 seconds with no active reads/writes, the runtime may cancel them based on its LRU (Least Recently Used) logic.
  - If you attempt to use a canceled connection, the `Network connection lost.` exception will be thrown.

`AbortController` API to set timeouts for `fetch` requests.

Limited Date API

To avoid CPU timing attacks, like Spectre, date and time functionality is not generally available. In particular, the time returned from `Date.now()` only advances after I/O operations, like `fetch`. For example:

app/api/date/route.ts TypeScript

```typescript
export const runtime = 'edge';

export async function GET(request: Request)
  const currentDate = () => new Date().toIS
  for (let i = 0; i < 500; i++) {
    console.log(`Current Date before fetch:
  }

  await fetch('https://worldtimeapi.org/api
  console.log(`Current Date after fetch: ${

  return Response.json({ date: currentDate(
}
```

Limits

The table below outlines the limits and restrictions of using Edge Functions on Vercel:

Feature	Edge Runtime
Maximum memory	128 MB
Maximum duration	25s (to begin returning a response **streaming** data.)
Size (after gzip compression)	Hobby: 1 MB, Pro: 2 MB, Ent: 4 MB

Feature	Edge Runtime
Concurrency	Unlimited concurrency
Cost	Pay for CPU time
Regions	Executes global-first, **can specify**
API Coverage	Limited API support

Edge Middleware CPU Limit

Edge Middleware can use no more than **50 ms** of CPU time on average.

This limitation refers to actual net CPU time, which is the time spent performing calculations, not the total elapsed execution or "wall clock" time. For example, when you are blocked talking to the network, the time spent waiting for a response does *not* count toward CPU time limitations.

Logs

See the Vercel Functions **Logs** documentation for more information on how to debug and monitor your Edge Functions.

Pricing

The Hobby plan offers functions for free, within **limits**. The Pro plan extends these limits, and charges CPU Time for Edge Functions.

Edge runtime-powered functions usage is based on **CPU Time**. CPU time is the time spent actually processing your code. This doesn't measure time spent waiting for data fetches to return. See "Managing usage and pricing for **Edge Functions**" for

Functions using the Edge Runtime are measured in the number of **execution units**, which are the amount of CPU time — or time spent performing calculations — used when a function is invoked. CPU time does not include idle time spent waiting for data fetching.

A function can use up to 50 ms of CPU time per execution unit. If a function uses more than 50 ms, it will be divided into multiple 50 ms units for billing purposes.

See **viewing function usage** for more information on how to track your usage.

Resource pricing

The following table outlines the price for each resource according to the plan you are on.

Edge Functions are available for free with the included usage limits. If you exceed the included usage and are on the Pro plan, you will be charged for the additional usage according to the on-demand costs:

Resource	Hobby Included	Pro Included	Pr
Edge Function Execution Units	First 500,000	First 1,000,000	$2 Ex
Function Invocations	First 100,000	First 1,000,000	$0 Inv

Hobby

Vercel will send you emails as you are nearing your usage limits. On the Hobby plan you **will not pay for any additional usage**. However, your account may be

When your **Hobby team** is set to **paused**, it remains in this state indefinitely unless you take action. This means **all** new and existing **deployments** will be paused.

> If you have reached this state, your application is likely a good candidate for a Pro account.

To unpause your account, you have two main options:

- **Contact Support**: You can reach out to our **support team** to discuss the reason for the pause and potential resolutions
- **Transfer to a Pro team**: If your Hobby team is paused, you won't have the option to initiate a **Pro trial**. Instead, you can set up a Pro team:
  1. **Create a Pro team account**
  2. Add a valid credit card to this account. Select the **Settings** tab, then select **Billing** and **Payment Method**

Once set up, a transfer modal will appear, prompting you to **transfer your previous Hobby projects** to this new team. After transferring, you can continue with your projects as usual.

Pro

For teams on a Pro trial, the **trial will end** when your team reaches the **trial limits**.

Once your team exceeds the included usage, you will continue to be charged the on-demand costs going forward.

Pro teams can **set up Spend Management** to get notified or to automatically take action, such as **using a webhook** or **pausing your projects** when your usage hits a set spend amount.

Enterprise

  - Custom **execution units**
  - Multi-region deployments

See **Vercel Enterprise plans** for more information.

Viewing function usage

Usage metrics can be found in the **Usage tab** on your **dashboard**. Functions are invoked for every request that is served.

You can see the usage for **functions using the Edge Runtime** on the **Edge Functions** section of the **Usage tab**. The dashboard tracks the usage values:

Metric	Description
Invocations	The number of times your Functions have been invoked
Execution Units	The number of execution units that your Edge Functions have used. An execution unit is 50 m of CPU time.
CPU Time	The time your Edge Functions have spent computing responses to requests

Managing Functions invocations

You are charged based on the number of times your **functions** are invoked, including both successful and errored response status codes, and excluding cache hits.

When viewing your Invocations graph, you can group by **Count** to see the total of all invocations across your team's projects.

- Use the **Projects** option to see the total number of invocations for each project within your team. This can help you identify which projects are using the most invocations and where to optimize.
- Cache your responses using **edge caching** and **Cache-Control headers**. This reduces invocations and speeds up responses for users.

Managing execution units

You are charged based on number of **execution units** that your Edge Functions have used. Each invocation of an Edge Function has a **Total CPU time**, which is the time spent running your code (it doesn't include execution time such as spent waiting for data fetches to return).

Each execution unit is 50ms. Vercel will work out the number of execution units (**total CPU time of the invocation / 50ms**) used for each invocation. You will then be charged based on anything over the limit.

For example:

- If your function gets invoked *250,000* times and uses *350* ms of CPU time at each invocation, then the function will incur **(350 ms / 50 ms) = 7** execution units each time the function gets invoked.
- Your usage is: 250,000 * 7 = **1,750,000** execution units Pro users have 1,000,000 execution units included in their plan, so you will be charged for the additional 750,000 execution units. The cost is $2.00 for each additional 1,000,000 execution units.

Optimizing execution units

- Execution units are comprised of a calculation of

invocations through caching and the **CPU time** used per invocation.

Managing CPU time

There is no time limit on the amount of CPU time your Edge Function can use during a single invocation. However, you are charged for each **execution unit**, which is based on the compute time. The compute time refers to the actual net CPU time used, not the execution time. Operations such as network access do not count towards the CPU time.

You can view CPU time by **Average** to show the average time for computation across all projects using Edge Functions within your team. This data point provides an idea of how long your Edge Functions are taking to compute responses to requests and can be used in combination with the invocation count to calculate execution units.

Optimizing CPU time

- View the CPU time by **Project** to understand which Projects are using the most CPU time
- CPU time is calculated based on the actual time your function is running, not the time it takes to respond to a request. Therefore you should optimize your code to ensure it's as performant as possible and avoid heavy CPU-bound operations

Last updated on April 28, 2025

Next
Account Management ›

Was this helpful?

Products

AI

Enterprise

Fluid Compute

Next.js

Observability

Previews

Rendering

Security

Turbo

v0

Resources

Community

Docs

Guides

Help

Integrations

Pricing

Resources

Solution Partners

Startups

Templates

Company

About

Blog

Careers

Changelog

Contact Us

Customers

Partners

Privacy Policy

Legal

Social

GitHub

LinkedIn

Twitter

YouTube

Loading status…