



**Hono**

**Multi-Runtime Web Framework for  
the Edges**

**Nikolay Pryanishnikov**



# whoami

- Full-Stack Engineer @ Station Labs
- Open-Source Contributor
- Security Researcher

Previously:

- OG Full-Stack Contributor @ Lido Finance
- Travel Startup Founder (Tripfinder.cc)
- Web Dev Studio Founder (Triangle.network)



# Plan

## 1. JS Runtimes

- Engines & Runtimes
- Standalone & Cloud Runtimes
- API Interoperation

## 2. Hono

- What Hono is
- Why Hono is Awesome

# JS Runtimes

# Engine

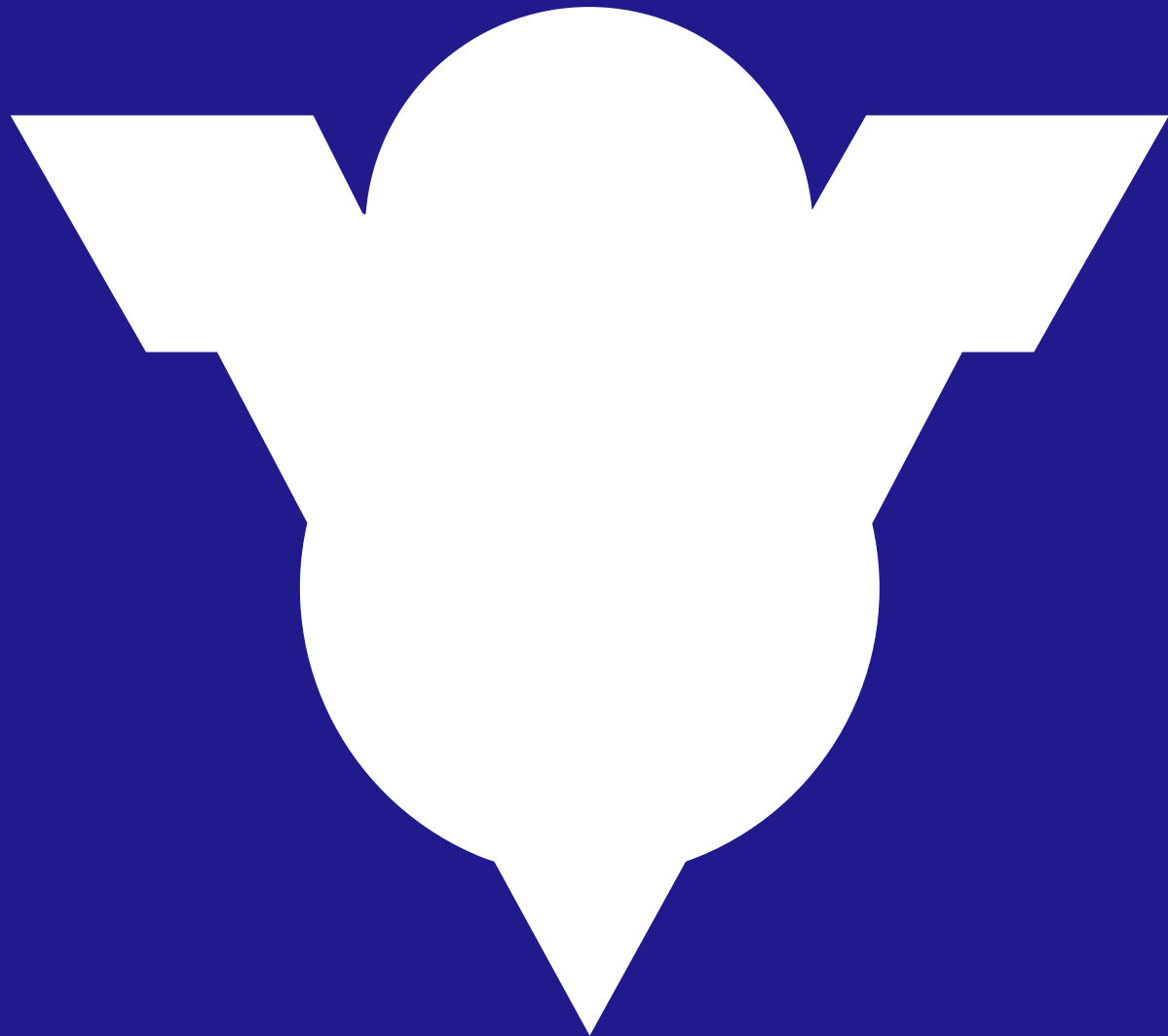
Software that executes code

# Runtime

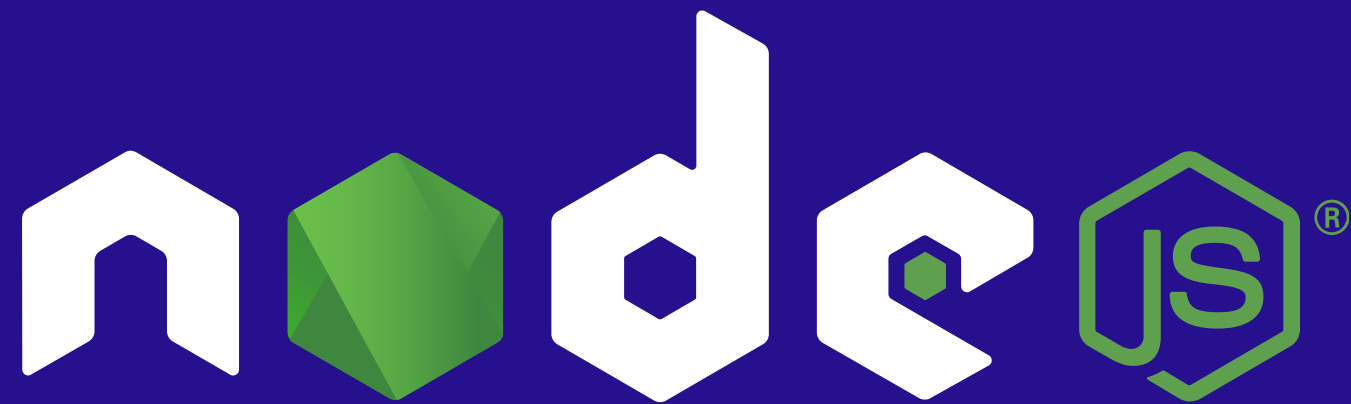
The environment in which code is executed

- event loop
- runtime libraries
- APIs

# Engines



# Standalone Runtimes



# Service Runtimes

aws



CLOUDFLARE®

▲ Vercel



# Standalone Runtimes API Differences

## Node.js

```
import { readFile } from 'fs/promises'  
  
await readFile('file.txt', 'utf-8')
```

## Deno

```
await Deno.readFile('file.txt')
```

# Service Runtimes API Differences

## AWS Lambda

```
const handler = async (event, context) => {  
  return "Hello";  
};  
  
export handler
```

## Vercel Node.js

```
const handler = async (req, res) => {  
  res.send("Hello!");  
}  
  
export handler
```

# 1 Service != 1 Runtime

The screenshot shows the Vercel documentation website. The main heading is "Choosing a Runtime". The page explains that runtimes transform source code into Functions served by the Edge Network. It lists official runtimes: Node.js, Edge, Go, and Python, each with a brief description of how they work. A table lists these runtimes and their descriptions. The page also includes a sidebar with navigation links, a search bar, and a "On this page" section listing various topics like "Community runtimes", "Caching Data", and "Runtime configuration".

**Choosing a Runtime**

Runtimes transform your source code into Functions, which are served by our Edge Network. Learn about the official runtimes supported by Vercel.

Vercel supports multiple runtimes for your functions. Each runtime has its own set of libraries, APIs, and functionality that provides different trade-offs and benefits.

Runtimes transform your source code into [Functions](#), which are served by our [Edge Network](#).

Runtime configuration is usually only necessary when you want to use the Edge runtime.

Vercel supports these official runtimes:

Runtime	Description
<a href="#">Node.js</a>	The Node.js runtime takes an endpoint of a Node.js function, builds its dependencies (if any) and bundles them into a Serverless Function.
<a href="#">Edge</a>	The Edge runtime is a lightweight JavaScript runtime that exposes a set of Web Standard APIs that make sense on the server.
<a href="#">Go</a>	The Go runtime takes in a Go program that defines a singular HTTP handler and outputs it as a Serverless Function.
<a href="#">Python</a>	The Python runtime takes in a Python program that defines a singular HTTP handler and outputs it as a Serverless Function.

**On this page**

- Community runtimes
- Caching Data
- Runtimes comparison
  - Node.js
  - Edge
- Infrastructure
  - Runtime
  - Cold starts
  - Location
  - Failover mode
  - Automatic concurrency scaling
- Burst concurrency limits
- Isolation boundary
- File system support
- Archiving
- Functions created per deployment
- Size limits
  - Bundle size limits
  - Max duration
  - Memory size limits
  - Environment variables
  - Request body size

# Possible Service Runtime Limitations

- Limited NPM package access
- Limited API set (Web Standard APIs only)
- No direct database access



# Vercel Edge Limitations

Also, `Buffer` is globally exposed to maximize compatibility with existing Node.js nodules.

## Unsupported APIs

The Edge runtime has some restrictions including:

- Some Node.js APIs other than the ones listed above **are not supported**. For example, you can't read or write to the filesystem
- `node_modules` *can* be used, as long as they implement ES Modules and do not use native Node.js APIs
- Calling `require` directly is **not allowed**. Use `import` instead

The following JavaScript language features are disabled, and **will not work**:

API	Description
<code>eval</code>	Evaluates JavaScript code represented as a string
<code>new Function(evalString)</code>	Creates a new function with the code provided as an argument
<code>WebAssembly.compile</code>	Compiles a WebAssembly module from a buffer source
<code>WebAssembly.instantiate</code>	Compiles and instantiates a WebAssembly module from a buffer source

Supported APIs

Network APIs

Encoding APIs

Stream APIs

Crypto APIs

Other Web APIs

Check if you're on the Edge runtime

Compatible Node.js versions

Unsupported APIs

Environment Variables

[⬆️ Back to top](#)



# WinterCG

<https://wintercg.org>

# Members



Bloomberg



# Web Standard APIs

The screenshot shows the MDN Web Docs page for the Fetch API. The page has a dark theme. At the top, there is a navigation bar with links for 'References', 'Guides', 'Plus', 'Curriculum', 'Blog', 'Play', and 'AI Help'. There are also buttons for 'Theme', 'Log in', and 'Sign up for free'. Below the navigation bar, the breadcrumb trail reads 'References > Web APIs > Fetch API'. A search bar and a language selector (English (US)) are also present. On the left side, there is a sidebar with a 'Filter' button and a list of categories: 'Fetch API', 'Guides', 'Interfaces', and 'Methods'. Under 'Guides', there are links for 'Using the Fetch API', 'Fetch basic concepts', and 'Cross-global fetch usage'. Under 'Interfaces', there are links for 'Headers', 'Request', and 'Response'. Under 'Methods', there is a link for 'fetch()'. The main content area features the title 'Fetch API' and a blue note box stating: 'Note: This feature is available in Web Workers'. Below the note, there is a paragraph explaining that the Fetch API provides an interface for fetching resources and is a replacement for XMLHttpRequest. The section 'Concepts and usage' follows, explaining that the Fetch API uses Request and Response objects and related concepts like CORS. It then describes the fetch() method, noting it is a global method available in Window and Worker contexts. Finally, it details the fetch() method's arguments, stating it takes a mandatory path to the resource and optionally an init options object.

mdn web docs\_ References Guides Plus Curriculum <sup>NEW</sup> Blog Play AI Help <sup>BETA</sup> Theme Log in Sign up for free

References > Web APIs > Fetch API English (US)

Filter

## Fetch API

**Note:** This feature is available in [Web Workers](#).

The Fetch API provides an interface for fetching resources (including across the network). It is a more powerful and flexible replacement for [XMLHttpRequest](#).

### Concepts and usage

The Fetch API uses [Request](#) and [Response](#) objects (and other things involved with network requests), as well as related concepts such as CORS and the HTTP Origin header semantics.

For making a request and fetching a resource, use the [fetch\(\)](#) method. It is a global method in both [Window](#) and [Worker](#) contexts. This makes it available in pretty much any context you might want to fetch resources in.

The [fetch\(\)](#) method takes one mandatory argument, the path to the resource you want to fetch. It returns a [Promise](#) that resolves to the [Response](#) to that request — as soon as the server responds with headers — **even if the server response is an HTTP error status**. You can also optionally pass in an `init` options object as the second argument (see [Request](#)).

#### In this article

- Concepts and usage
- Fetch Interfaces
- Specifications
- Browser compatibility
- See also



# Before

```
const handler = async (req, res) => {  
  res.status(200)  
  res.send('Hello!')  
  return  
}  
  
export default handler
```

# After

```
const handler = async (req: Request) =>  
  return new Response('Hello!', 200)  
}  
  
export default handler
```

# Why API Design Matters

```
const handler = (res, req) => {  
  res.status(200)  
  res.json({ ok: true })  
  
  // forget a return and do something else  
  
  res.status(200)  
  res.json({ ok: true, hello: "world" })  
}
```



**Hono**

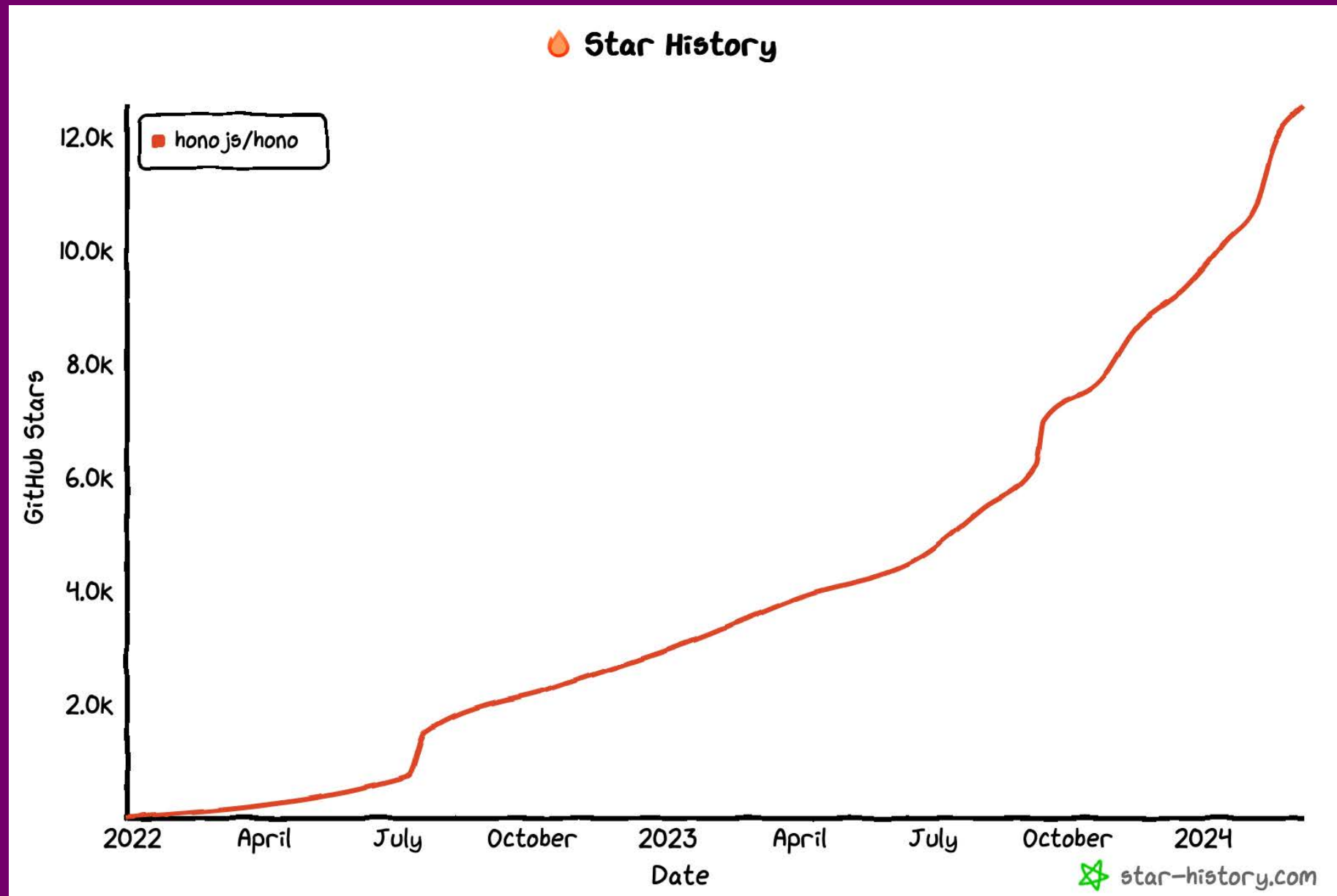
Hono is flame 🔥 in Japanese

Created by Yusuke Wada, Developer  
Advocate @ Cloudflare

- small
- simple
- fast
- flexible
- great developer experience



# Gaining Popularity



# What it Looks Like

```
import { Hono } from 'hono'

const app = new Hono()

app.get('/hello', (c) => {
  return c.json({
    message: `Hello!`,
  })
})
```

**Why It's Awesome**

# Flexible

- Can migrate Server -> Serverless and vice versa
- Can switch runtimes
- Can switch hosting services
- Can be standalone or as part of a Next.js project, for example

# Flexible

- Node.js
- Deno
- Bun

- Cloudflare Workers
- Fastly Compute
- Vercel
- Netlify
- AWS Lambda
- Lambda@Edge

# Very Fast

- Fastest router for Cloudflare Workers
- Fastest router for Deno
- 3x faster than Express even with Node.js adapter

```
• short static - GET /user
-----
Hono RegExpRouter      64 ns/iter (59.19 ns ... 141.52 ns)
Hono TrieRouter        217.17 ns/iter (201.96 ns ... 291.4 ns)
@medley/router         104.57 ns/iter (98.58 ns ... 178.28 ns)
find-my-way            89.87 ns/iter (83.84 ns ... 108.24 ns)
koa-tree-router        87.57 ns/iter (81.6 ns ... 109.58 ns)
trek-router            116.13 ns/iter (110.79 ns ... 135.48 ns)
express (WARNING: includes handling) 633.89 ns/iter (624.01 ns ... 657.1 ns)
koa-router             2 µs/iter (1.98 µs ... 2.05 µs)

summary for short static - GET /user
Hono RegExpRouter
  1.37x faster than koa-tree-router
  1.4x faster than find-my-way
  1.63x faster than @medley/router
  1.81x faster than trek-router
  3.39x faster than Hono TrieRouter
  9.9x faster than express (WARNING: includes handling)
  31.29x faster than koa-router
```

# Different Routers

1. `RegExpRouter` – fastest in the JavaScript world
2. `TrieRouter` – slower than `RegExp`, but supports all patterns
3. `SmartRouter` – auto selection of best router from the above
4. `LinearRouter` – better for "one shot" environments eg `Fastly Compute`
5. `PatternRouter` – smallest, best for environments with limited resources



# Type Safety

```
6
7  const app = new Hono()
8
9  app.get('/entry/:date/:id', (c) => {
10
11     const date = c.req.param("param(key: "date" | "id"): string
12     const id = c.req.param("id")
13
14
15     return c.text('This is permalink')
```

# Autogenerated Client

```
import { Hono } from 'hono'

const app = new Hono()

app.get('/hello', (c) => {
  return c.json({
    message: `Hello!`,
  })
})

type AppType = typeof app

import { hc } from 'hono/client'

const client = hc<AppType>('/api')
const res = await client.hello.$get()
```

# Middleware

```
// match any method, all routes
app.use(logger())

// specify path
app.use('/posts/*', cors())

// specify method and path
app.post('/posts/*', basicAuth())
```

## Included

- Basic Authentication
- Bearer Authentication
- Cache
- Compress
- CORS
- CSRF Protection
- ETag
- JSX Renderer
- JWT
- Timing
- Logger
- Pretty JSON
- Secure Headers

## Third-party

- GraphQL Server
- Sentry
- Firebase Auth
- Zod Validator
- Qwik City
- tRPC Server
- TypeBox Validator
- Verify RSA JWT
- Typia Validator
- Valibot Validator
- Zod OpenAPI
- Clerk Auth
- Swagger UI
- Scalar API Reference
- esbuild Transpiler
- Prometheus Metrics
- Auth.js(Next Auth)

# Custom Middleware

```
// Custom logger
app.use(async (c, next) => {
  console.log(`[${c.req.method}] ${c.req.url}`)
  await next()
})

// Add a custom header
app.use('/message/*', async (c, next) => {
  await next()
  c.header('x-message', 'This is middleware!')
})

app.get('/message/hello', (c) => c.text('Hello
Middleware!'))
```

# Validation

```
import { zValidator } from '@hono/zod-validator'
import { z } from 'zod'

const schema = z.object({
  name: z.string(),
})

app.get(
  '/hello',
  zValidator('query', schema),
  (c) => {
    const { name } = c.req.valid('query')
    return c.json({
      message: `Hello! ${name}`,
    })
  }
)
```

# OpenAPI

```
import { OpenAPIHono } from '@hono/zod-openapi'

const app = new OpenAPIHono()

app.openapi(route, (c) => {
  const { id } = c.req.valid('param')
  return c.json({
    id,
    age: 20,
    name: 'Ultra-man',
  })
})

// The OpenAPI documentation will be available at /doc
app.doc('/doc', {
  openapi: '3.0.0',
  info: {
    version: '1.0.0',
    title: 'My API',
  },
})
```



# General Takeaways

- Standardised Web APIs
- Reusable objects with fetch  
(Request + Response)
- Improved interoperability of  
runtimes

Thanks WinterCG!

# Hono Takeaways

- ✓ Runs everywhere
- ✓ Easy migration
- ✓ Excellent DevEx
- ✓ Easy to get going
- ✓ Easily extendable

Try it!

# Hono Links

<https://github.com/honojs/hono>

<https://twitter.com/honojs>

<https://discord.gg/KMh2eNSdxV>

<https://github.com/yusukebe>

**Thank You!**

# My Links

<https://twitter.com/kolyasapphire>

<https://github.com/kolyasapphire>

<https://ks.gg>