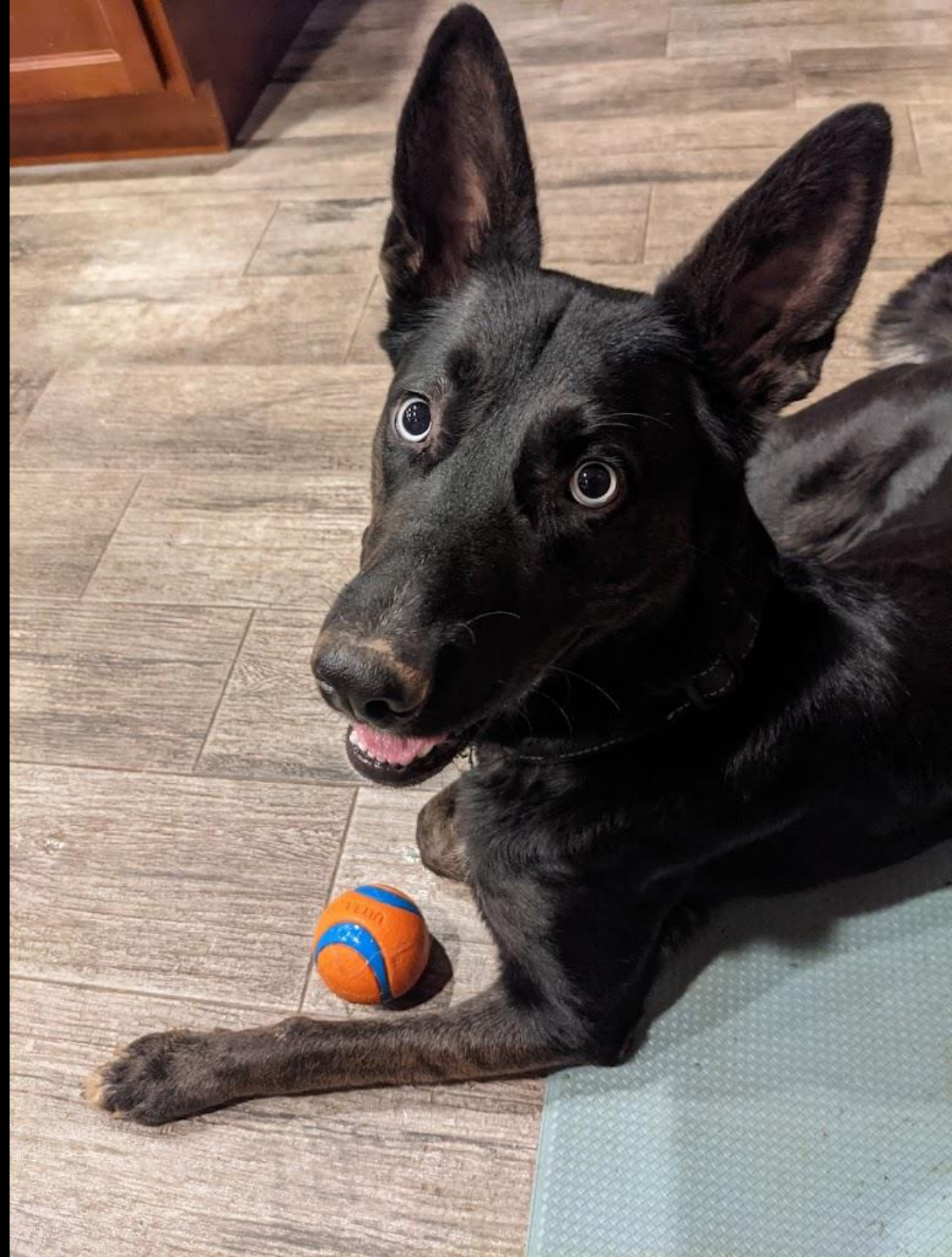
YES, NODE.JS *IS* A PART OF THE WEB PLATFORM

James M Snell / jasnell

Node.js Core Contributor, TSC

@CloudFlare



HERE'S A QUESTION...

YOUR ANSWER WILL LIKELY DEPEND ON WHEN YOU WERE FIRST INTRODUCED TO JAVASCRIPT, NODE.JS, AND WEB DEVELOPMENT

Using JavaScript, what is the *correct* way to open a file, read its content, compress it, encrypt it, then write it back out to another file?

LET'S START WITH THE FIRST PART…

```js

const fs = require('fs');

fs.createReadStream('somefile.txt')
  .setEncoding('utf8')
  .on('data', (chunk) => {
    console.log(chunk);
  });

```

If you've used Node.js for any length of time, this should look familiar.

It's short. It's simple. It works…

In Node.js

LET'S START WITH THE FIRST PART…

```js

const fs = require('fs');

fs.createReadStream('somefile.txt')
  .setEncoding('utf8')
  .on('data', (chunk) => {
    console.log(chunk);
  });

```

There are no fewer than six Node.js specific APIs and conventions at work here:

1. **Node.js Common JS Modules**
2. **Node.js fs module**
3. **Node.js Streams**
4. **Node.js Event Emitter**
5. **Node.js Buffer Object**
6. **Node.js Encodings**

WE CAN SIMPLIFY THIS EXAMPLE…

```js

const fs = require('fs');
const { pipeline } = require('stream');

pipeline(
  fs.createReadStream('somefile.txt'),
  process.stdout);
```

This brings along with it additional Node.js specific APIs such as the Node.js process object and the Node.js pipeline function

Here’s a question though…

Is Node.js the only JavaScript platform that needs to open and read files?

Is Node.js the only JavaScript platform where we need Compression? Encryption? Text encodings? URL parsing? Generating random numbers? Making and responding to HTTP requests?

We use JavaScript to develop at every layer of the Web Platform.

If we're using the same language, and implementing the same capabilities, ***why should we use different APIs***?

Desktop Applications

Servers / Backends

Edge Networks / CDNs / Caches

Web Browsers

IOT Devices

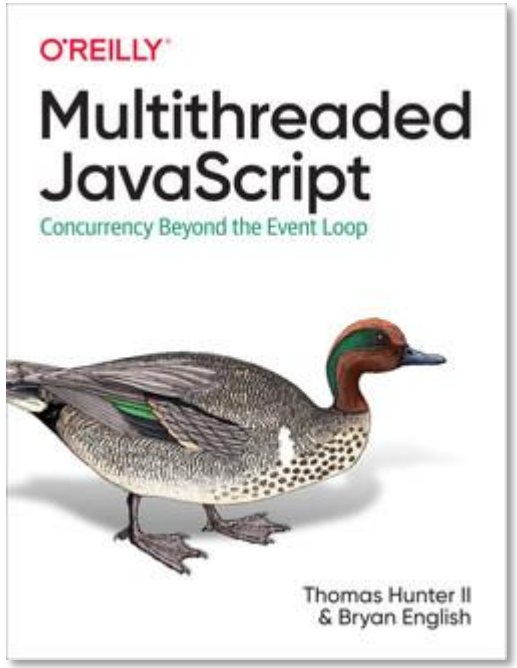
Node.js is not a Web Browser.
Nor should it ever pretend to be.

But Node.js *does* do a lot of the same stuff that Web Browsers do.

And Web Browsers are starting to do a lot of same stuff Node.js does.

And Edge networks are starting to do a lot of the same stuff Node.js *and* Browsers do.

We even have watches that also aren't Web Browsers, Servers, or Edge networks that still do a lot of the same stuff they all do.



(Go buy Bryan and Thomas' new book. It's very good and has an amazing foreword)

LET’S GO BACK TO THE EXAMPLE

```js

const fs = require('fs');

fs.createReadStream('somefile.txt')
  .setEncoding('utf8')
  .on('data', (chunk) => {
    console.log(chunk);
  });

```

LET'S GO BACK TO THE EXAMPLE

```mjs

import { createReadStream } from 'fs';

fs.createReadStream('somefile.txt')
  .setEncoding('utf8')
  .on('data', (chunk) => {
    console.log(chunk);
  });

```

EcmaScript Modules replace Common JS...

Yes, ESM is not perfect.

Yes, ESM is not as flexible.

But ESM will work for more than just Node.js, and that's important.

The more it's used, the better it will get.

LET'S GO BACK TO THE EXAMPLE

```mjs

import { open } from 'fs/promises';

const file = await open('somefile.txt');
file.createReadStream('somefile.txt')
  .setEncoding('utf8')
  .on('data', (chunk) => {
    console.log(chunk);
  });

```

Promises replace Node.js style callbacks

LET'S GO BACK TO THE EXAMPLE

```mjs

import { open } from 'fs/promises';

const file = await open('somefile.txt');
for await (const chunk of file) {
  console.log(chunk.toString('utf8'))
};

```

Use Async Iterators

LET’S GO BACK TO THE EXAMPLE

```mjs

import { open } from 'fs/promises’;
const dec = new TextDecoder();

const file = await open('somefile.txt');
for await (const chunk of file) {
  console.log(dec.decode(chunk));
};

```

Use TextDecoder and Typed Arrays instead of the Buffer API and Node.js encodings

LET’S GO BACK TO THE EXAMPLE

```mjs

import { open } from 'fs/promises’;
const file = await open('somefile.txt');

const dec = new TextDecoder();
for await (const chunk of file) {
  console.log(dec.decode(chunk));
};

```

The fs/promises API is still Node.js specific…

But ESM, TextDecoder, Async Iterators, Console… these are all JavaScript and Web Platform Standards.

MDN Web Docs
moz://a

▸ Technologies ▸ References & Guides ▸ Feedback

Site search... (Press "/" to

Web technology for developers > Web APIs

Table of contents

Specifications
Interfaces
See also

Web APIs

When writing code for the Web, there are a large number of Web APIs available. Below is a list of all the APIs and interfaces (object types) that you may be able to use while developing your Web app or site.

Web APIs are typically used with JavaScript, although this doesn't always have to be the case.

MDN lists 91 separate API specifications as part of the collection of “Web APIs”,

And 1017 separate interfaces and object types implemented by browsers.

MDN Web Docs
moz://a

▸ Technologies ▸ References & Guides ▸ Feedback

Site search... (Press "/" to

Web technolo

Table of

Specifications
Interfaces
See also

Node.js will never come close to implementing anywhere close to all of these.

MDN lists 91 separate API specifications as part of the collection of "Web APIs",

And 1017 separate interfaces and object types implemented by browsers.

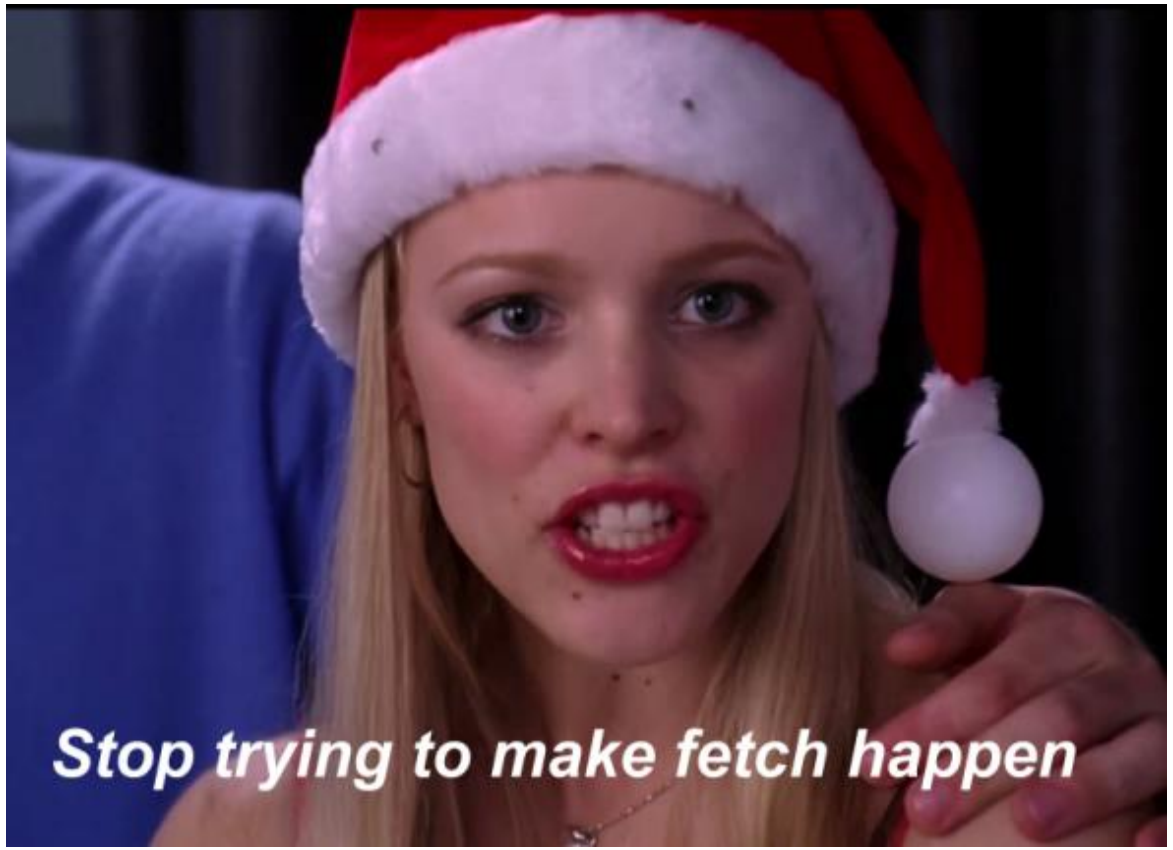
Bryan is 100% Absolutely, Unquestionably, Completely Correct.

Different types of platforms, different types of runtimes, Running in different environments for different reasons will always have need for their own APIs, conventions, patterns, and their own legacy concerns.

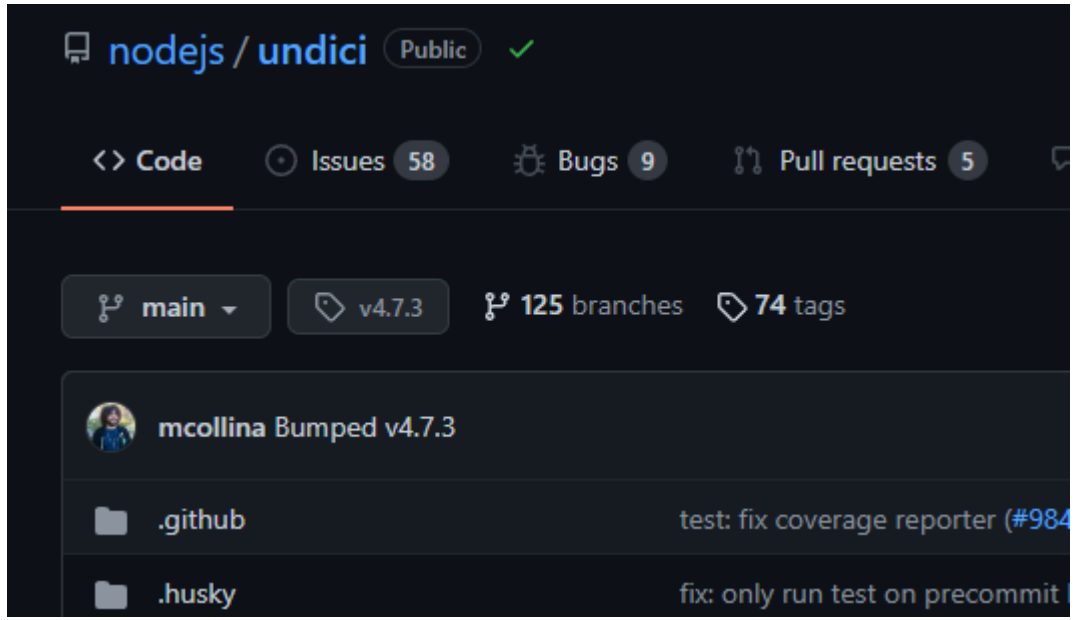
So, What Web Platform APIs work in Node.js today?

- Web Assembly / WASI
- URL Parser
- Blob
- MessagePort / MessageChannel
- BroadcastChannel
- AbortController / AbortSignal
- Event / EventTarget
- TextEncoder / TextDecoder
- ReadableStream / WritableStream / TransformStream
- CompressionStream / DecompressionStream
- TextEncoderStream / TextDecoderStream
- Web Crypto
- Performance Timeline / User Timing
- Console API
- Timer APIs
- Fetch API
- structuredClone() / atob() / btoa()

- URL Pattern?
- Web Locks?
- File System?
- Raw Sockets?
- Diagnostic APIs?



yarn add undici
npm i undici

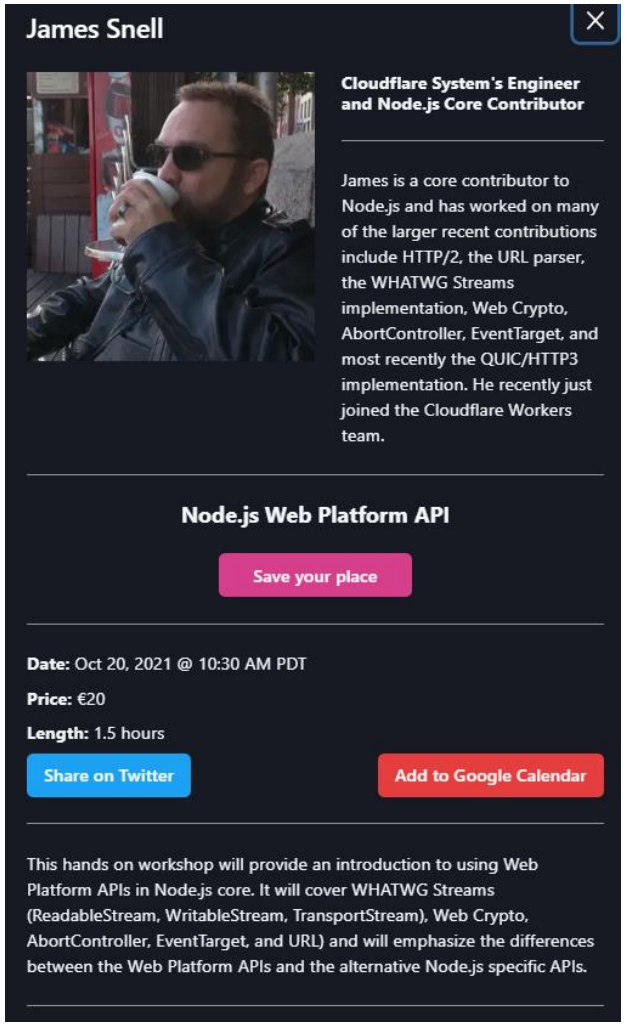


What to expect next?

Conversations have kicked off between Node.js, Deno, and Workers on a common Raw Sockets API.

We've started working to ensure other platforms (like Workers) support the same common subset of Web APIs.

We'll continue to evolve Node.js' support for standard API patterns and specs – without sacrificing backwards compatibility and ecosystem stability.



Join my workshop
Wednesday @ 10:30am pacific time

@jasnell