View online

Download PDF



The bees are important: use SDKs wisely

Paolo Insogna

Node.js TSC, Principal Engineer @ **Platformatic**

Mothers are always right!



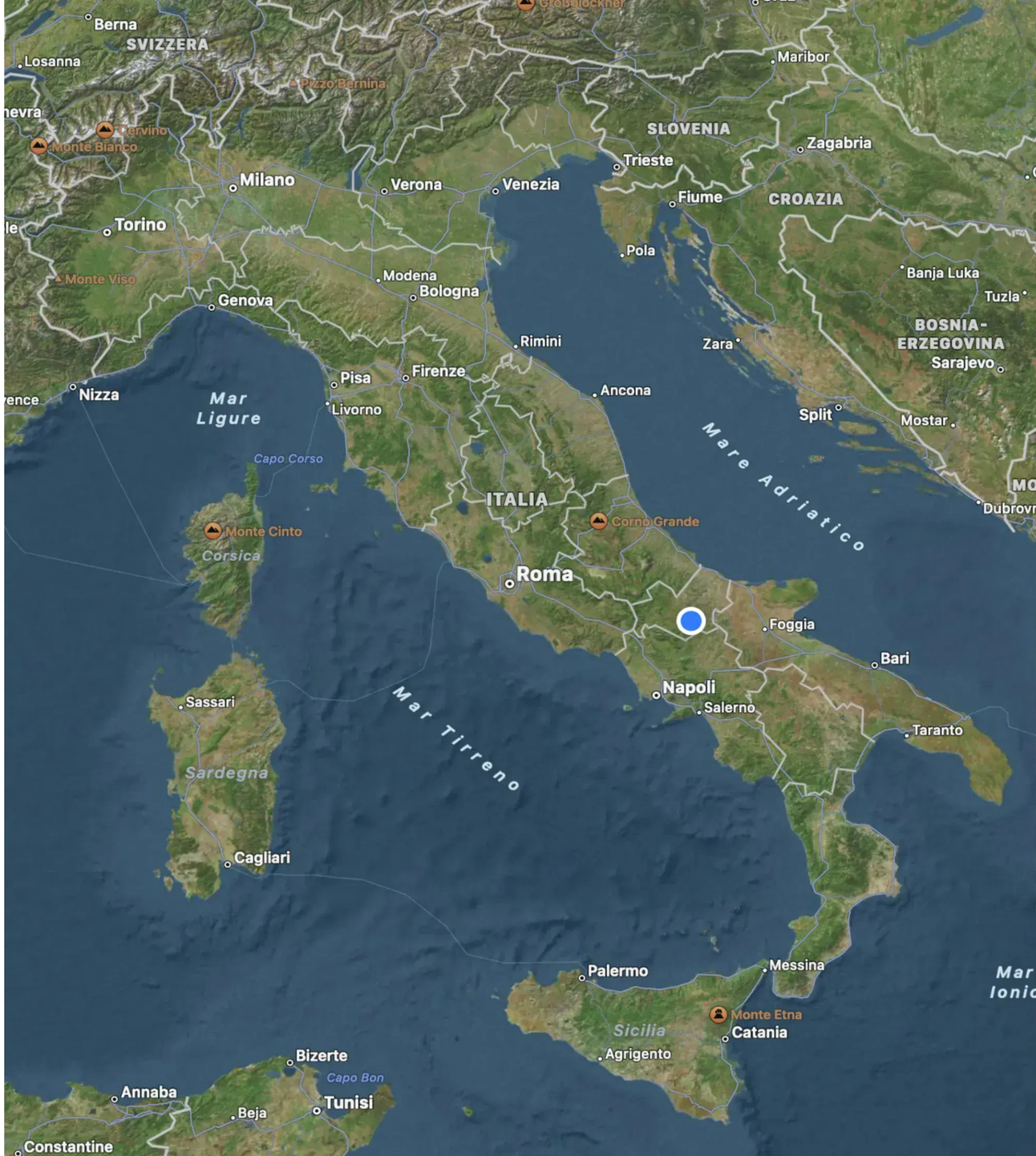
Hello, I'm **Paolo**!



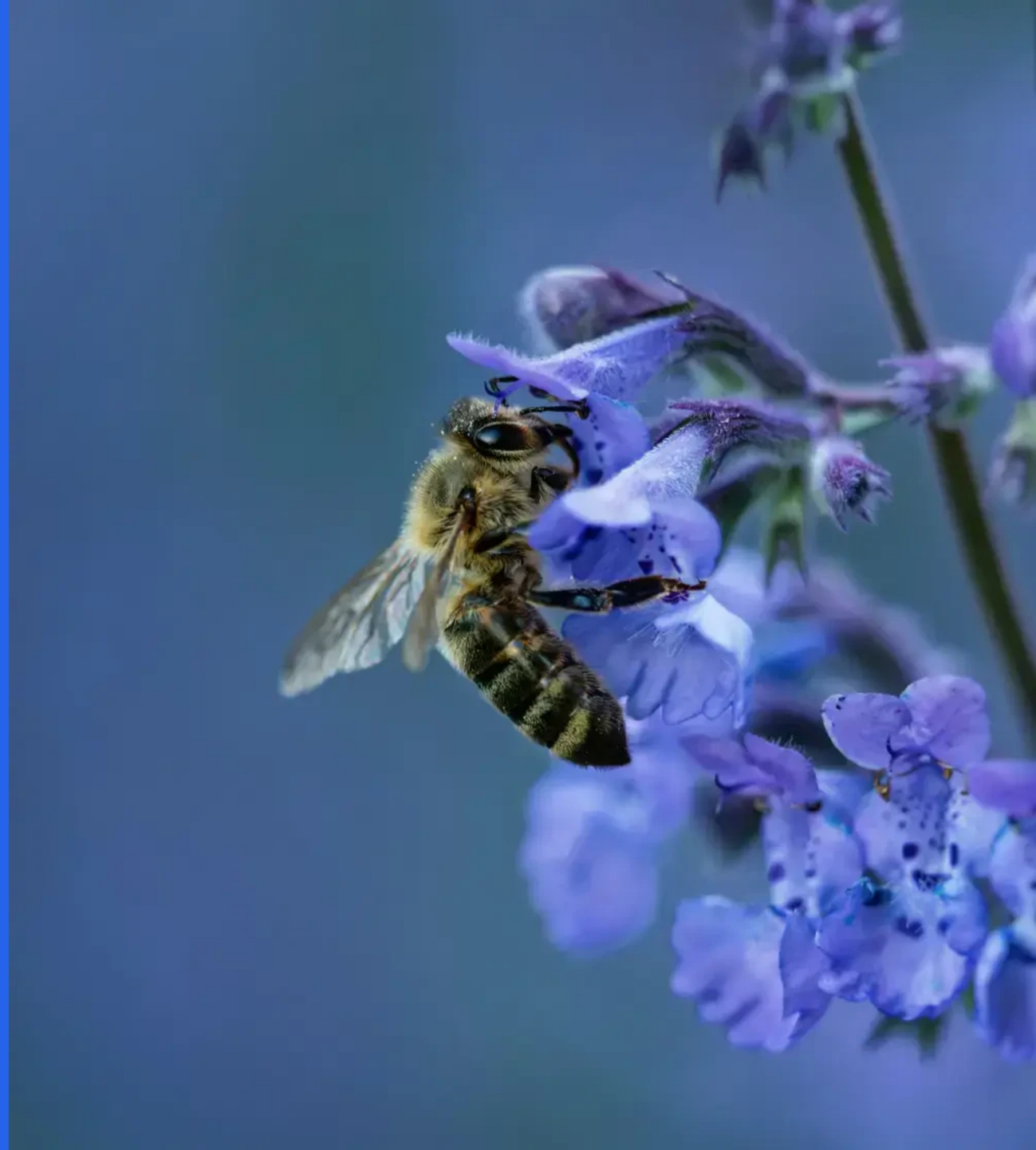
Node.js Technical Steering Committee Member

Platformatic Principal Engineer

`paoloinsogna.dev` `ShogunPanda` `p_insogna` `pinsogna`



Our entire existence is on their shoulders!



How is that?



They are related to our food

Over 70% of the food we eat is directly or indirectly related to their pollination.



We are endangering them

Pesticides and climate change made them an endangered species.



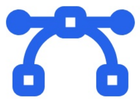
No coming back

If they disappear, we have no natural or artificial replacement.

Aren't we here for dev stuff?



Yes, we do!



APIs are related to our application

Over 100% of the application we use is directly or indirectly related to their quality.



We are endangering them

Adding complexity between applications and APIs made them an endangered species.

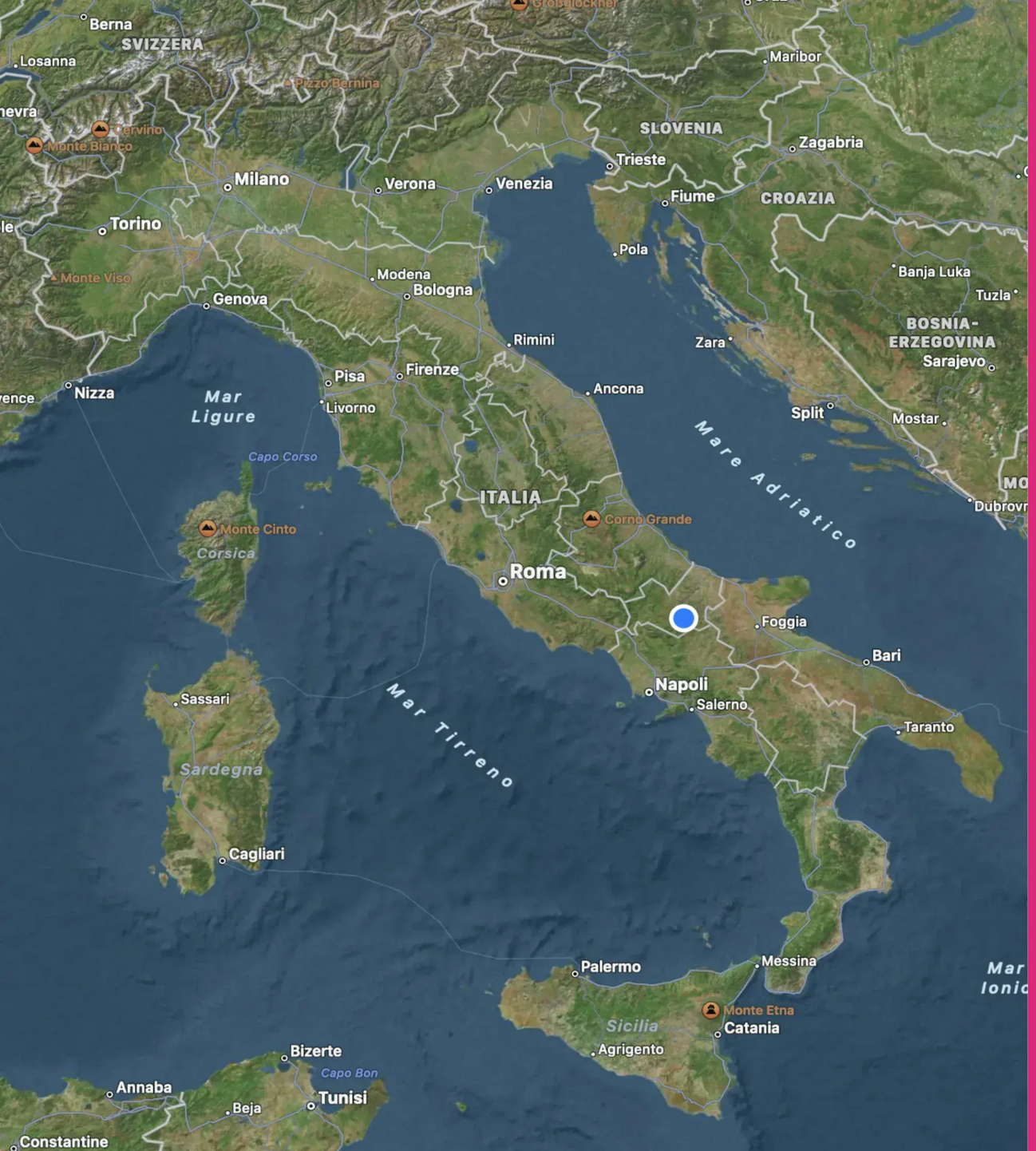


No coming back

If they disappear, we have no other replacement.

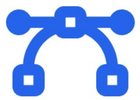
Are you trolling us?





Well, try to translate **bees** in Italian!

Say hello to SDKs



We don't do everything by ourselves

Our application and servers often talk to third party services.



Third parties usually give us an SDK

The kit makes our life easier by simplifying integration.



Lot of goodies

Authentication, retry logic and other boring tasks are already implemented.

What do we get, in short?



Focus

All the API services are made simpler to use, hiding all implementation details.



Feel at home

Each SDK is designed to be easily adopted by your favorite programming language.

Are there just pros?



You already know the answer...



What do we ALSO get, in short? (1/2)



Roadmap dependency

Since you don't control the SDK development, you can't embrace features at your pace.



Supply chain

If you have a bug or vulnerability in the SDK, you have to wait for the vendor to solve it.

What do we ALSO get, in short? (2/2)



Quality

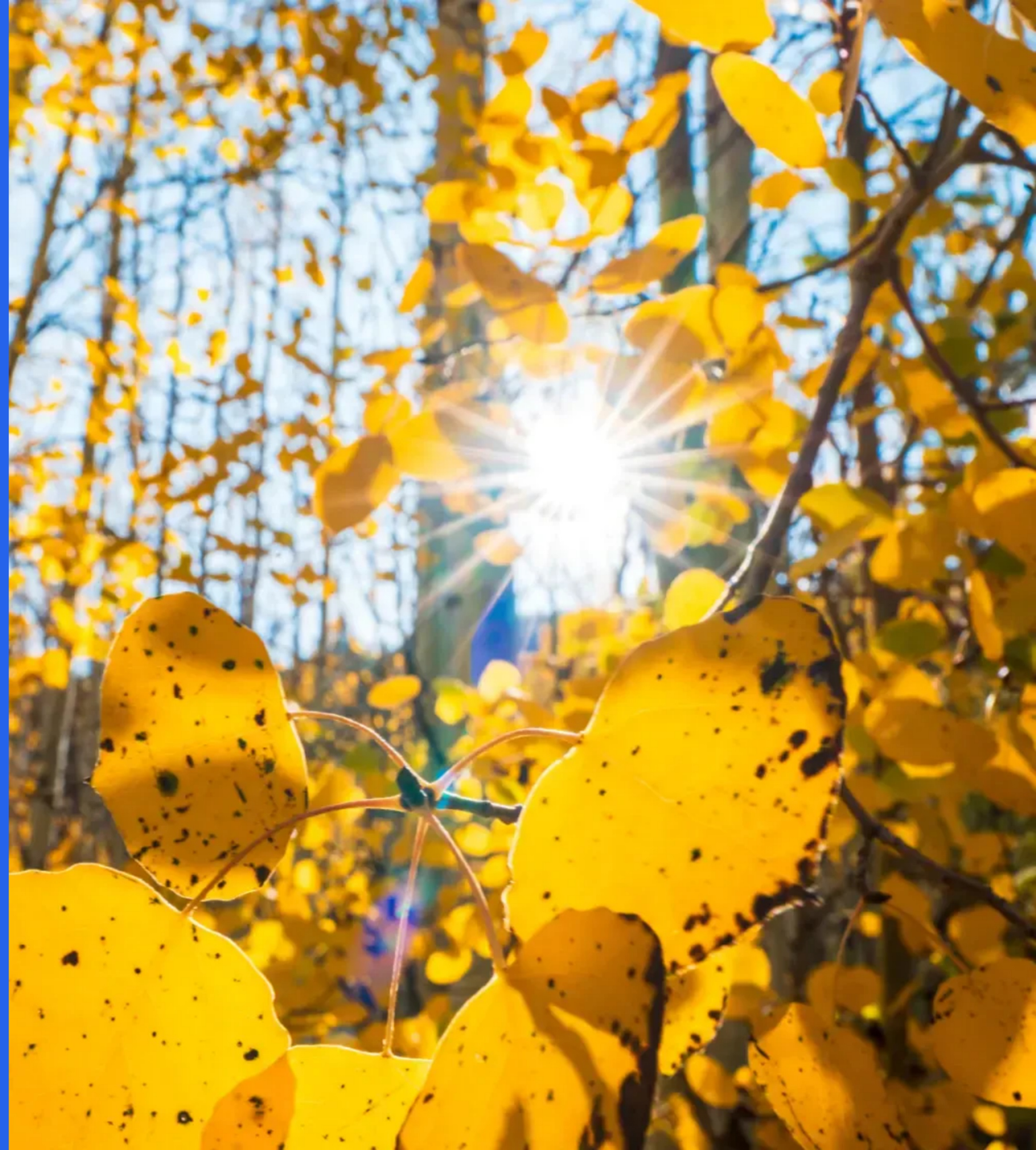
Often vendors automate the generation of SDKs out of API definitions.
Native language experience is lost.



Performance

SDKs are often underperforming due to the attempt to keep a unique architecture across languages.

Only an example can enlighten us!



Case study: uploaded a file to AWS S3



Why?
It's a pretty simple use case that we all faced sometimes.



Which one?
I'm gonna analyze two different versions of the AWS JavaScript SDK.



How?
I'm gonna show you how many dependencies the SDK brings and how it performs.

Let's go!



AWS SDK v2: Dependencies in theory

We only add dependency in the `package.json` file ...

```
{
  "name": "aws-sdk-v2",
  "version": "1.0.0",
  "main": "index.mjs",
  "scripts": {},
  "dependencies": {
    "aws-sdk": "^2.1508.0"
  }
}
```

AWS SDK v2: Dependencies in reality

... but we actually get a lot more!

```
shogun@panda:~/example$ pnpm list --depth=2 | wc -l
28
```

AWS SDK v2: The code

```
import AWS from 'aws-sdk'
import { readFile } from 'node:fs/promises'

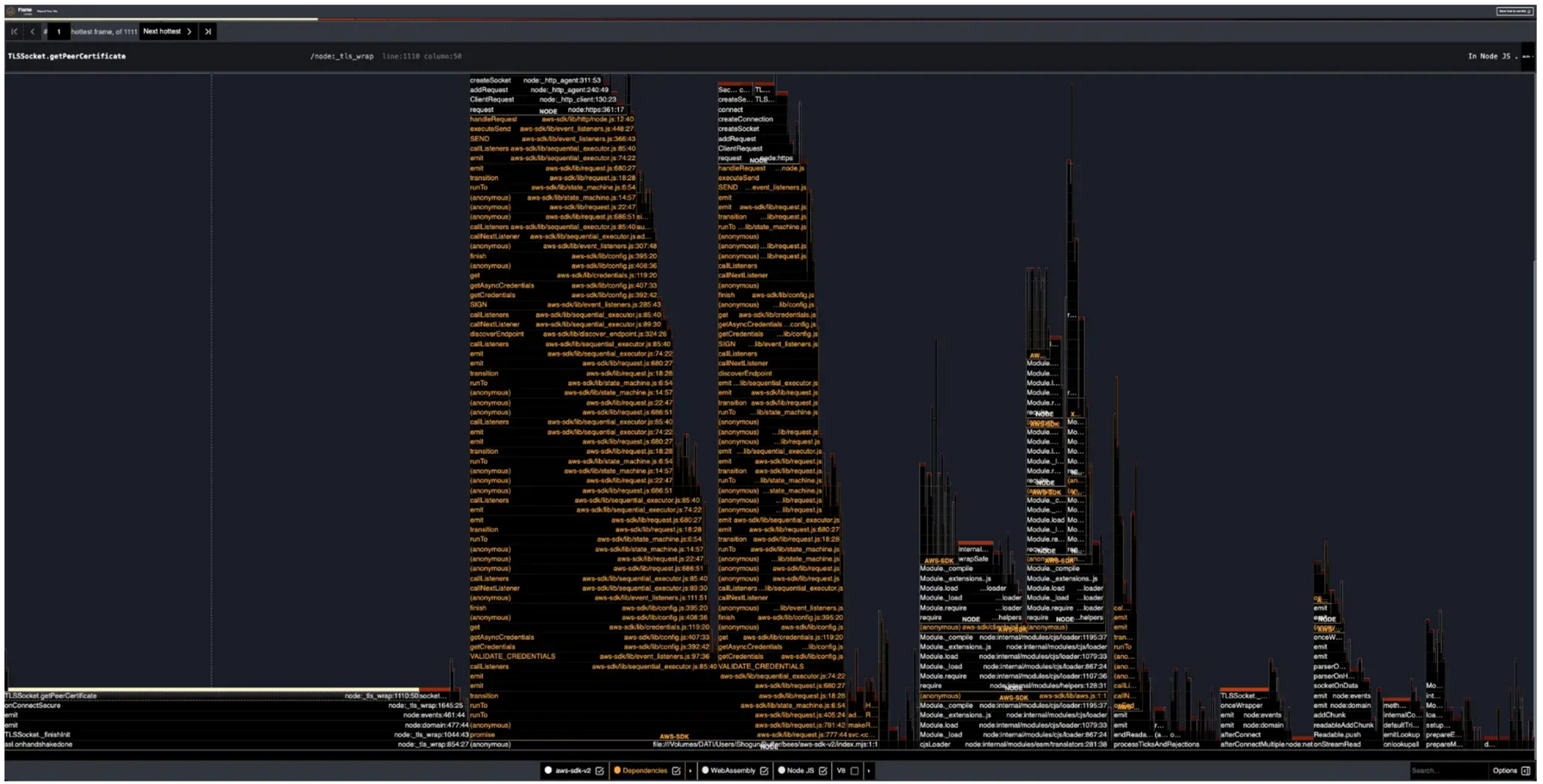
// Create the client, this implicitly reads
// the AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY
// environment variables.
const client = new AWS.S3({
  apiVersion: '2006-03-01',
  region: process.env.AWS_DEFAULT_REGION
})

for (let i = 0; i < 100; i++) {
  // Upload the file
  await client
    .putObject({
      Bucket: process.env.BUCKET,
      Key: process.env.KEY,
      Body: await readFile(process.env.FILE, 'utf-8')
    })
    .promise()
}
```

AWS SDK v2: How does it perform?

```
shogun@panda:~/example$ time node index.mjs
------------------------------------------------------
Executed in   13.76 secs    fish           external
  usr time    1.40 secs   84.00 micros    1.40 secs
  sys time    0.13 secs  732.00 micros    0.13 secs
```

AWS SDK v2: What's happening under the hood?



Let's get modern!



AWS SDK v3: Dependencies in theory

A little more detailed dependencies in `package.json` ...

```
{
  "name": "aws-sdk-v3",
  "version": "1.0.0",
  "main": "index.mjs",
  "scripts": {},
  "dependencies": {
    "@aws-sdk/client-s3": "*",
    "@aws-sdk/credential-providers": "^3.462.0"
  }
}
```

AWS SDK v3: Dependencies in reality

... but we actually get even more packages!

```
shogun@panda:~/example$ pnpm list --depth=2 | wc -l
524
```

AWS SDK v3: The code

```
import { readFile } from 'node:fs/promises'
import { S3Client, PutObjectCommand } from '@aws-sdk/client-s3'
import { fromEnv } from '@aws-sdk/credential-providers'

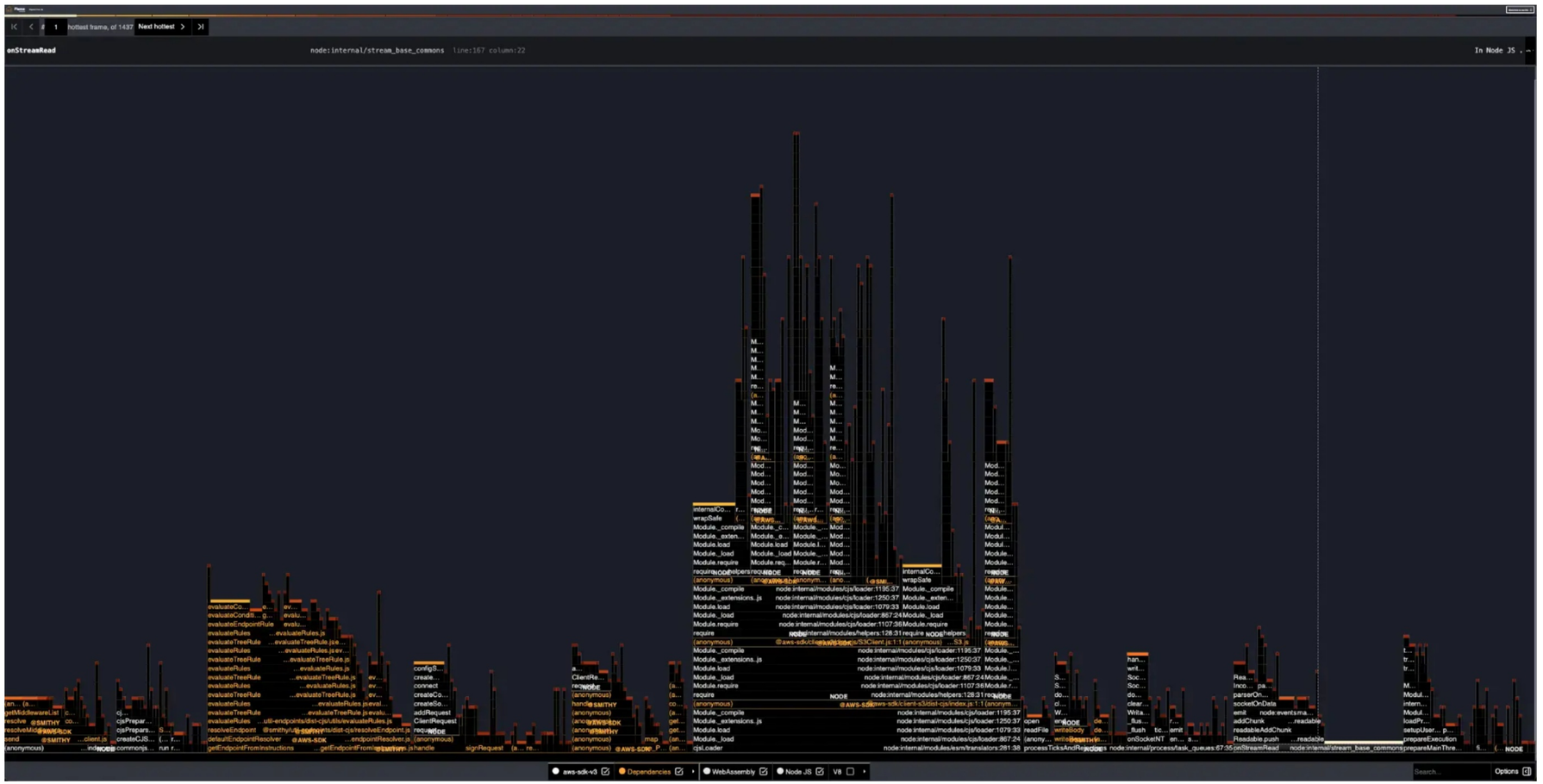
// Create the client
const client = new S3Client({
  region: process.env.AWS_DEFAULT_REGION,
  // This implicitly reads
  // the AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY
  // environment variables.
  credentials: fromEnv()
})

for (let i = 0; i < 100; i++) {
  await client.send(
    new PutObjectCommand({
      Bucket: process.env.BUCKET,
      Key: process.env.KEY,
      Body: await readFile(process.env.FILE, 'utf-8')
    })
  )
}
```

AWS SDK v3: How does it perform?

```
shogun@panda:~/example$ time node index.mjs
------------------------------------------------------
Executed in    8.22 secs      fish           external
  usr time  723.14 millis    0.08 millis  723.06 millis
  sys time   80.60 millis    1.17 millis   79.44 millis
```

AWS SDK v3: What's happening under the hood?



Quod Erat Demonstrandum

You only see a bit of it
Each SDK brings a lot more complexity which is hidden under the rug.

Not really a monolith
How many point of failures can you afford?



Do we have the solution?



Yes, use the APIs directly!



How many will you use?

1

Unified signing

Each AWS API uses the same **algorithm**.

2

One REST HTTP call

Uploading a file to S3 only uses one **API call**.

Let's get to the action!



No SDK: Setup

No dependencies in **`package.json`**, Node.js already got our backs ...

```
{
  "name": "direct",
  "version": "1.0.0",
  "main": "index.mjs",
  "scripts": {},
  "dependencies": {}
}
```

No SDK: Signing the call

```
1   import { createHash, createHmac } from 'node:crypto'
2   import { readFile } from 'node:fs/promises'
3
4   const awsS3UnsignedPayload = 'UNSIGNED-PAYLOAD'
5
6   function sha256(contents) {
7     return createHash('sha256').update(contents).digest('hex')
8   }
9
10  function hmacSha256(key, contents) {
11    return createHmac('sha256', key).update(contents).digest()
12  }
13
14  // Implements the Authenticating Requests (AWS Signature Version 4) algorithm.
15  function signS3Upload(keyId, accessKey, region, method, rawUrl, headers, payloadHash) {
16    // CanonicalRequest calculation
17    // StringToSign calculation
18    // Signature calculation
19  }
```

No SDK: CanonicalRequest

```
// CanonicalRequest calculation
const url = new URL(rawUrl)
const path = encodeURIComponent(url.pathname).replaceAll('%2F', '/')
// The last one is for the queryString, which is empty
const canonicalRequestComponents = [method, path, '']
const signedHeadersComponents = []

// Next step: StringToSign calculation
```

No SDK: StringToSign

```
// Previous step: CanonicalRequest calculation

// Create the StringToSign
const timestamp = headers['x-amz-date']
const date = timestamp.slice(0, 8)
const scope = `${date}/${region}/s3/aws4_request`
const stringToSign = [
  'AWS4-HMAC-SHA256',
  timestamp,
  scope,
  sha256(canonicalRequest)
].join('\n')

// Next step: Signature calculation
```

No SDK: Signature

```
// Previous step: StringToSign calculation

// Signature calculation
const dateKey = hmacSha256(`AWS4${accessKey}`, date)
const dateRegionKey = hmacSha256(dateKey, region)
const dateRegionServiceKey = hmacSha256(dateRegionKey, 's3')

const signingKey = hmacSha256(dateRegionServiceKey, 'aws4_request')
const signature = hmacSha256(signingKey, stringToSign).toString('hex')

return [
  `AWS4-HMAC-SHA256 Credential=${keyId}/${date}/${region}/s3/aws4_request`
  `SignedHeaders=${signedHeaders}`,
  `Signature=${signature}`
].join(',')
```

No SDK: REST API call

```
function uploadFile(keyId, accessKey, region, bucket, key, contents) {
  const host = `${bucket}.s3.${region}.amazonaws.com`
  const payloadHash = 'UNSIGNED-PAYLOAD'
  const headers = {
    'x-amz-content-sha256': payloadHash,
    'x-amz-date': new Date()
      .toISOString()
      .replace(/\.\d{0,3}/, '')
      .replaceAll(/[:-]/g, ''),
    host
  }

  const url = `https://${host}/${key}`
  headers.authorization = signS3Upload(
    keyId, accessKey, region,
    'PUT', url, headers, payloadHash
  )

  return fetch(url, { method: 'PUT', headers, body: contents })
}
```

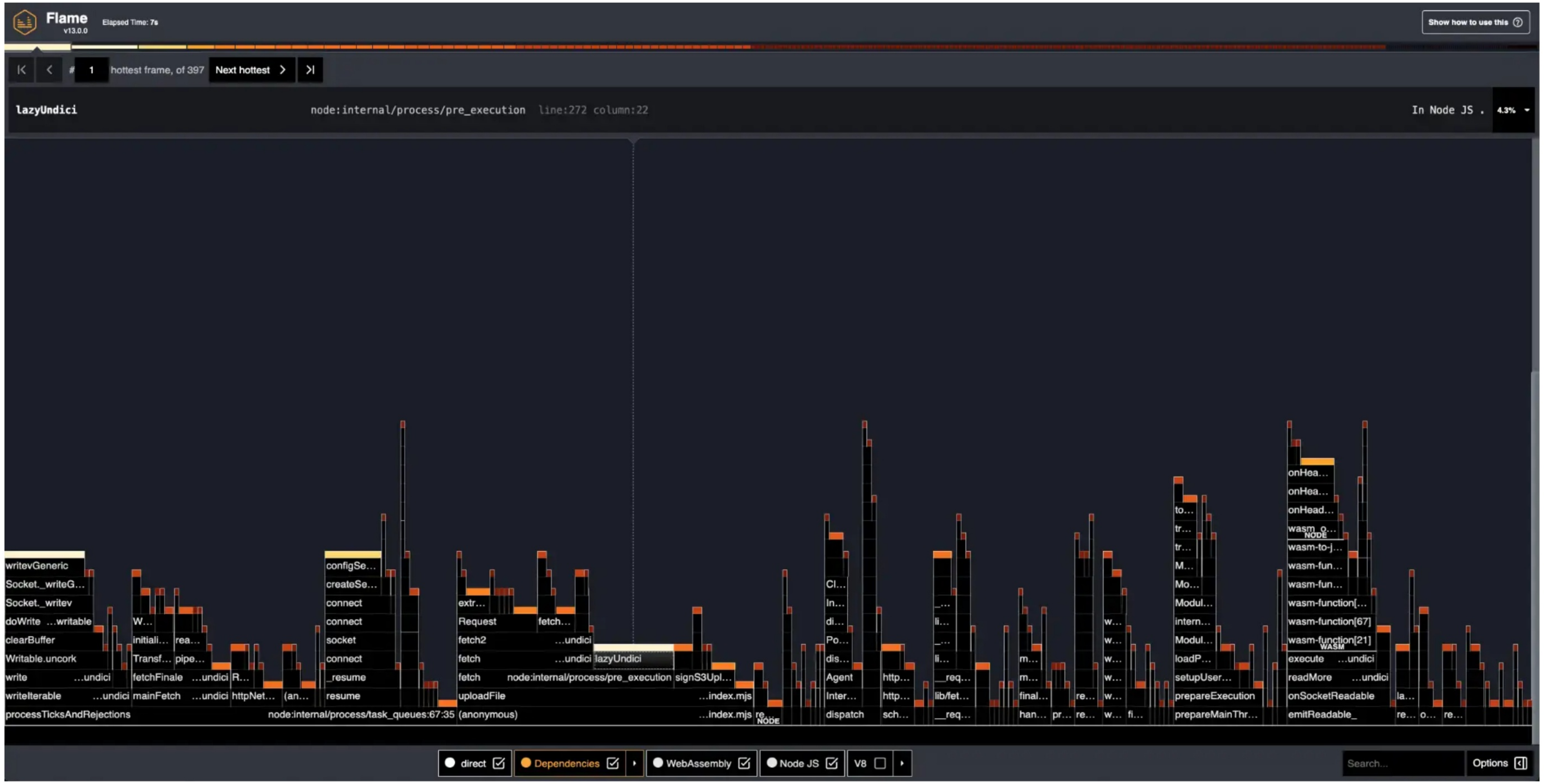
No SDK: Main loop

```
for (let i = 0; i < 100; i++) {
  await uploadFile(
    process.env.AWS_ACCESS_KEY_ID,
    process.env.AWS_SECRET_ACCESS_KEY,
    process.env.AWS_DEFAULT_REGION,
    process.env.BUCKET,
    process.env.KEY,
    await readFile(process.env.FILE, 'utf-8')
  )
}
```

No SDK: How does it perform?

```
shogun@panda:~/example$ time node index.mjs
------------------------------------------------------
Executed in    6.64 secs      fish           external
  usr time  353.39 millis   54.00 micros  353.34 millis
  sys time   42.81 millis  513.00 micros   42.30 millis
```

No SDK: What's happening under the hood?



Mission completed!



Take home lessons

What can we learn from this long journey?



SDKs are not always bad

Sometimes making things manually will **cost too much time**.



Be flexible

You don't have to remove all SDKs. But you can **optimize some calls**.



Read the API docs

If it's simple to use, maybe you **don't need an SDK**.

One last thing™

“Don't take candies from strangers”

Every mother in the world



Thank you!

Paolo Insogna

Node.js TSC, Principal Engineer

@p_insogna

paolo.insogna@platformatic.dev

Platformatic