View online

Download PDF

Horizontal Scaling of a Web3 system to the sky and beyond in AWS

Paolo Insogna

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

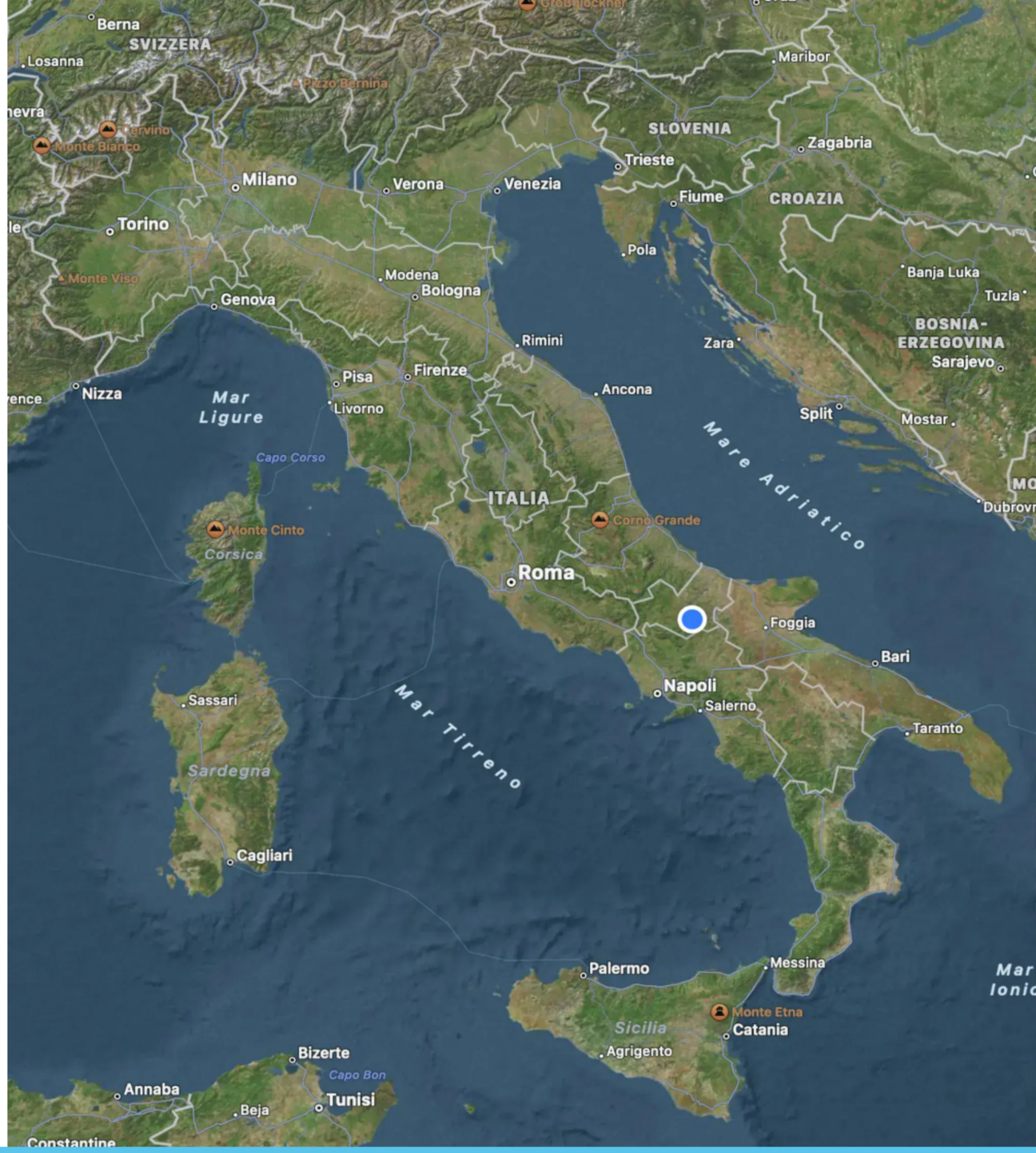
All you need to succeed is love!

Hello, I'm **Paolo**!

Node.js Technical Steering Committee Member

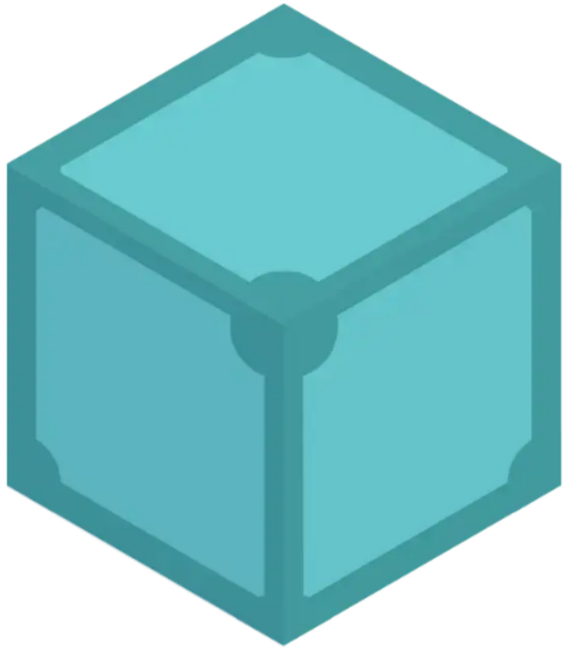
Platformatic Principal Engineer

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



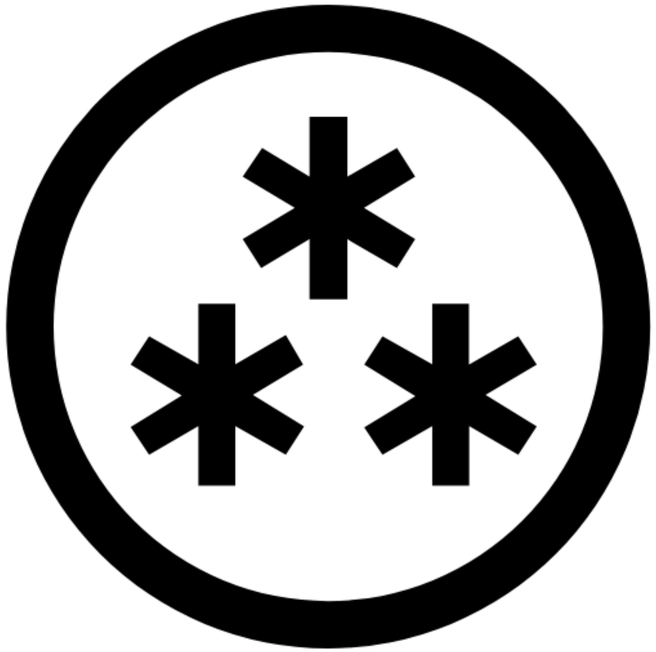
IPFS (InterPlanetary FileSystem)

A protocol designed with the intention to preserve and grow humanity's knowledge by making the web upgradeable, resilient, and more open.



web3.storage

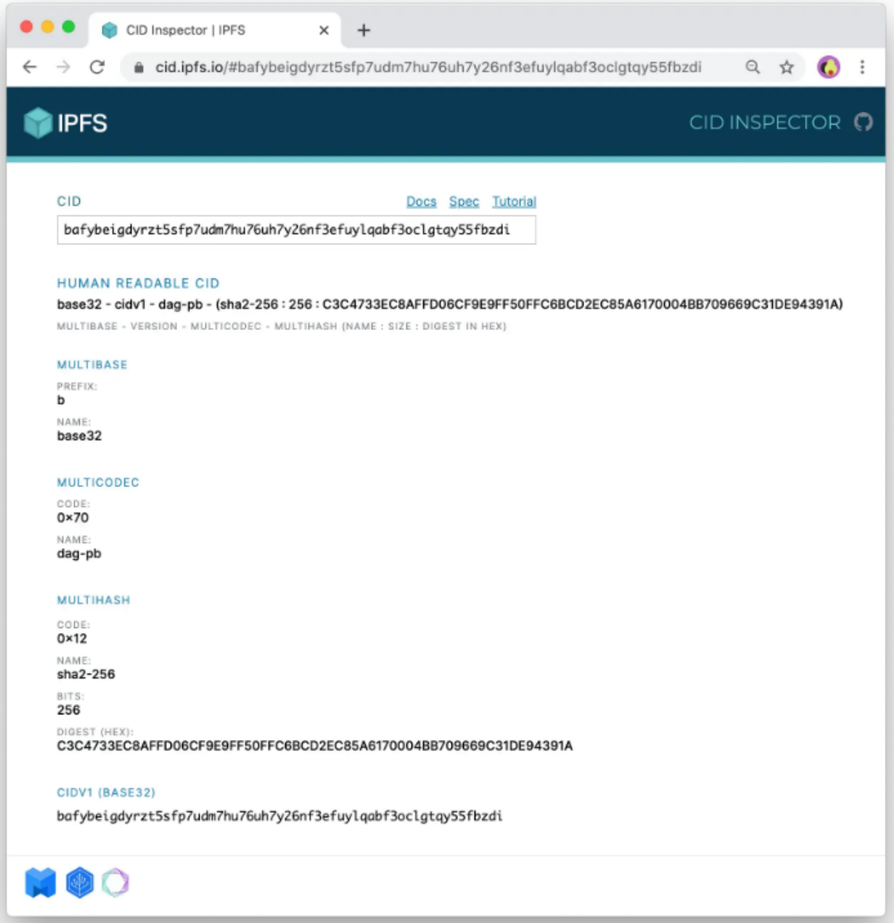
A service that makes storing files on IPFS easier and more accessible for non-technical people.



Content Identifier

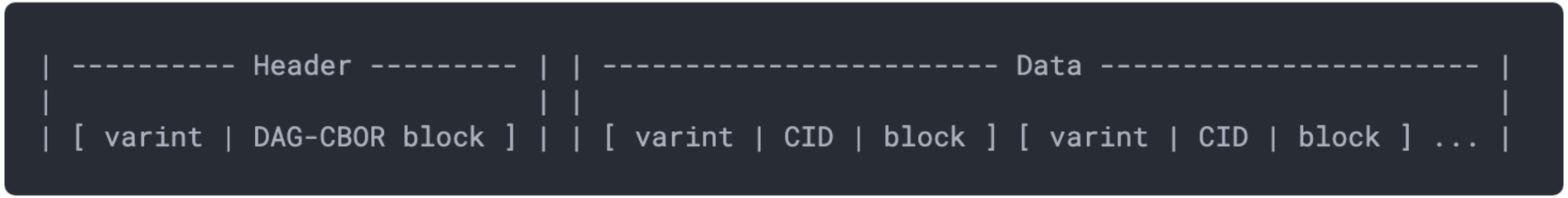
A CID uniquely identifies a chunk of data in IPFS and provides informations on how to verify the integrity of the data.

It does not contain any information about the location.



Content Archives

The CAR file format is optimized for sequential reading and it's essentially a concatenation of blocks delimited using byte length prefixes.



What was wrong?

Previous Challenges

Increasing traffic

The original implementation struggled to handle the massive daily upload traffic.

Hard to scale

New nodes could not be quickly added on-demand due to the long bootstrap phase.

What was the problem?

One simple question

“How can we use AWS services to make the service horizontally scalable with no limits?”

Goals

Handle growth

The system should be able to handle the massive amount of data uploaded on web3.storage.

Stateless services

Having stateless services helps horizontal scaling and greatly improves fault tolerance.

Cloud based

The entire architecture should leverage any modern cloud services and try to be affordable.

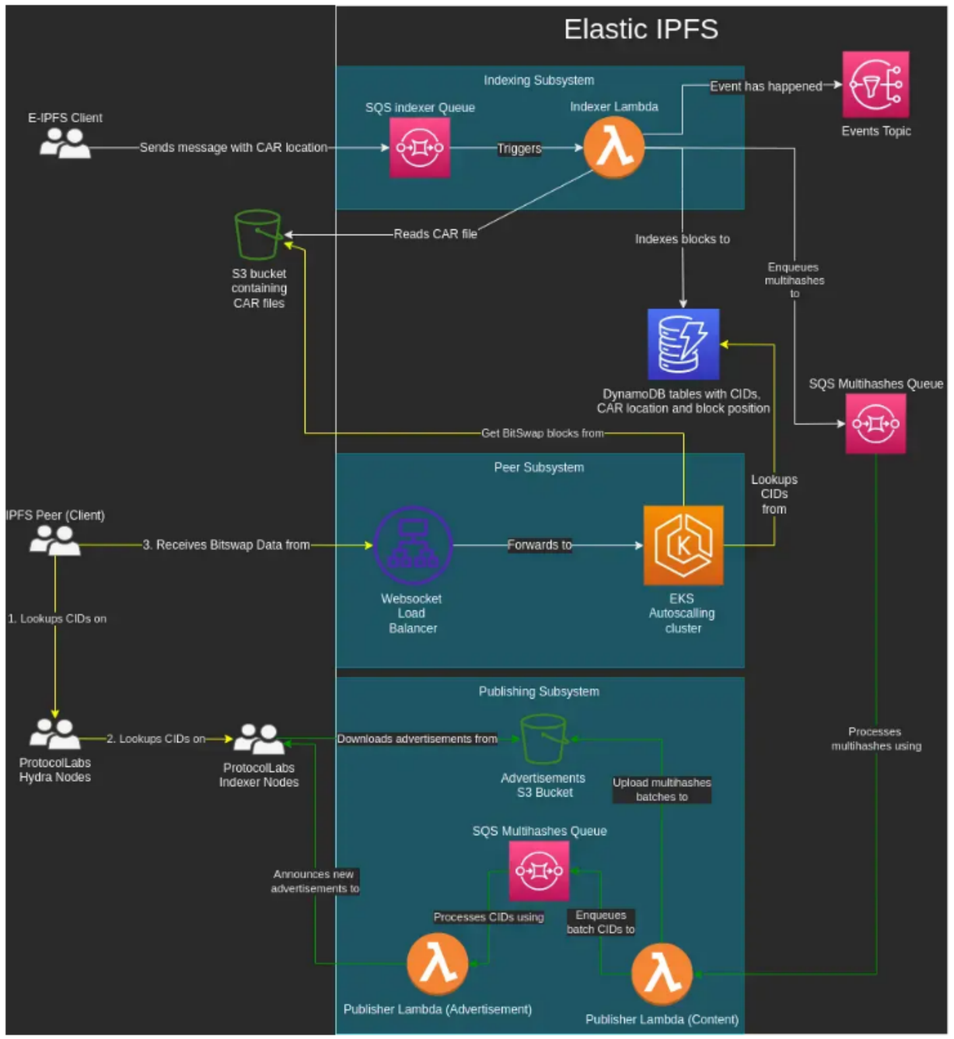
The solution

Hi, I'm Elastic IPFS!

The architecture is divided into three independent and stateless subsystems.

New nodes can be added or remove at any time, global replication can be easily achieved.

https://github.com/elastic-ipfs/elastic-ipfs



E-IPFS is democratic and cloud agnostic

No vendor lock-in

The AWS implementation is just the first implementation and it can also serve as reference implementation for testing.

Generic components

As long as you have object storage, queueing and serverless systems, you can adapt and deploy E-IPFS.

The DevOps are your limit

Mixed or on-premise approach are also possible, but be gentle with the people responsible to maintain the infrastructure.

CARs Table Item

It contains information about indexed CAR files, including the S3 full path.

```
interface Car {
  path: string
  bucket: string
  bucketRegion: string
  createdAt: Date
  fileSize: number
  key: string
  roots: Array<string>
  version: number
}
```

Blocks Tables Item

It stores the multihash and the type of the block, used for debugging.

```
interface Block {
  multihash: string
  createdAt: Date
  type: 'raw' | 'dag-cbor' | 'dag-json'
}
```

Blocks Position Item

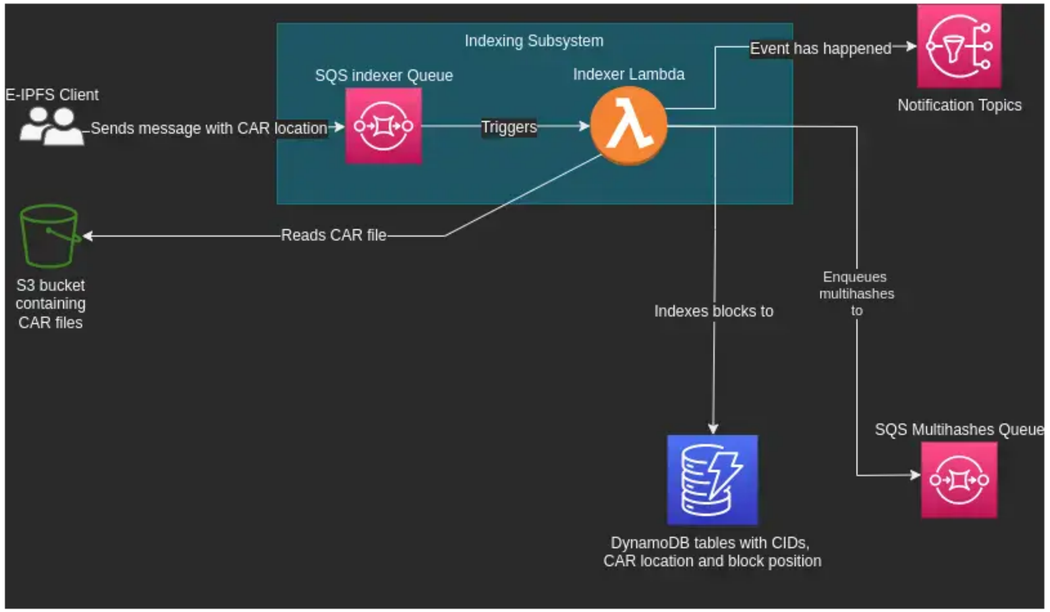
The most important table in the architecture

It links blocks to CARs, including offset and length.

```
interface BlockToCars {
  blockmultihash: string
  carpath: string
  length: number
  offset: number
}
```

Indexing subsystem

Overview


Indexing Subsystem
SQS indexer Queue
Indexer Lambda
Event has happened
Notification Topics
E-IPFS Client
Sends message with CAR location
Triggers
Reads CAR file
S3 bucket containing CAR files
Indexes blocks to
Enqueues multihashes to
SQS Multihashes Queue
DynamoDB tables with CIDs, CAR location and block position

Indexing flow

1 **CAR file is copied to S3**
The event is recorded on a SQS topic which will later trigger Indexing Lambda executions.

2 **Indexing Lambda executes**
The Lambda processes the CAR file and copies all the relevant informations in the DynamoDB tables showed earlier.

3 **Blocks are enqueued for publishing**
For each block encountered in the CAR file, the Indexer Lambda enqueues its multihash in a SQS topic which will later trigger Publishing Subsystem execution.

Bye, Idempotence!

The Lambda was idempotent

It required lots of additional reads on DynamoDB and made batching unavailable.

Indexing has very low failure rate

Idempotence was removed as indexing rarely failed and this brought big performance gain.

Let's talk about DHT

DHT Challenges

IPFS uses the Kademlia DHT for keeping peers and blocks information.

The Elastic IPFS claims to be a single node

Even though internally a lot of resources are used, only one `PeerID` is used.

No long connection

E-IPFS cannot directly participate to the DHT as it is not capable to connect to GossipSub.

Not self sufficient

In order to advertise content, the E-IPFS systems needs external auxiliary systems.

Hydra Nodes

The Hydra nodes speed up looking up content in the DHT.

When

Hydra nodes are created to have an high chance to incur into one of them when looking up data.

What

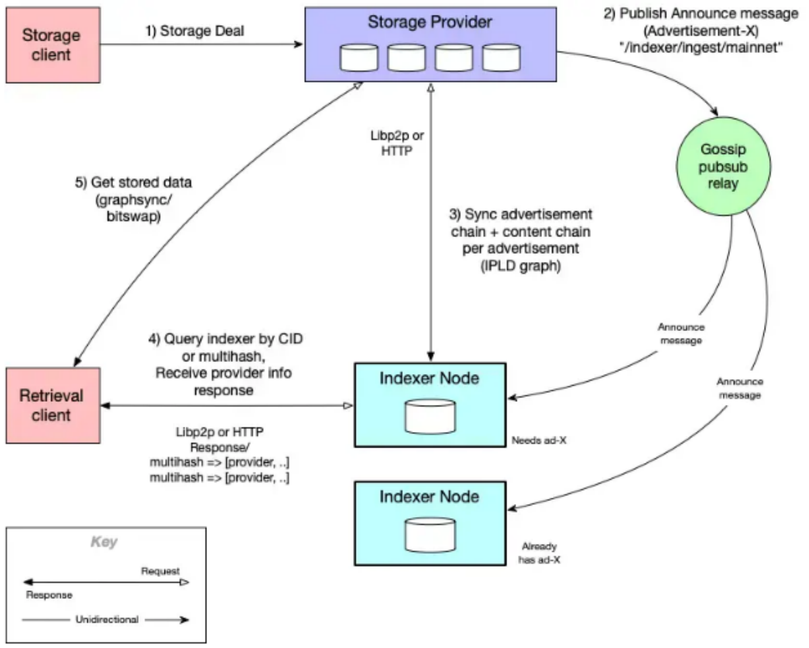
Content Routing Delegation via shared storage and third party systems as the Indexer Nodes.

Where

The Hydra nodes are distributed in the DHT sense, not the geographical one.

Indexer Nodes

A system which maps CID to content providers via libp2p or HTTP API.



What API shall we use?

E-IPFS uses the HTTP API as GossipSub is not viable for the reasons below.

E-IPFS is on the cloud

Maintaining long living connection, when possible, is expensive.

GossipSub is verbose

A lot of data is exchanged and it will impact costs as well.

libp2p optimization conflict

Destination host joins multiple streams with the same `PeerID`.

How does HTTP ingestion work?

The HTTP Ingest API make the Indexer Nodes choose when to sync.

Upload new advertisements and entries data
The data must available over HTTP.

Advertisements are strictly ordered
Each is linked to the previous one. The **`/head`** one is regularly updated.

Notify Indexers on `/ingest/announce` when ready to be queried
`HTTP PUT` requests will signal that new data has been published.

What will the Indexer Node will do?

When ready, the Indexer Node will sync via the HTTP server.

1. **Fetch the latest `/head` URL**
   It contains the latest advertisement data CID.

2. **Fetch the advertisement data and its related entries data.**
   All the data we are interested into is in these files.

3. **Repeat the flow**
   Stop when reaching an already indexed advertisement or the tail of the chain.

Head file

The **/head** just maintains a link to the latest advertisement.

```
{
  "head": {
    "/" : "bagugeeragtbtgeb2iulx2zihnymysaebais6ysziefwih66tyc7pupns2bma"
  },
  "pubkey": {
    "/": {
      "bytes" : "CAASpgIwgg ... 9bAgMBAAE="
    }
  },
  "sig": {
    "/": {
      "bytes": "hq9N33rQSZ ... BSHcoQLcbw=="
    }
  }
}
```

Advertisement file

It contains information about storage provider and entries.


```
{
  "Addresses" : ["/dns4/peer.ipfs-elastic-provider-aws.com/tcp/3000/ws"],
  "ContextID": {
    "/": { "bytes": "YmFndXF1ZXJ . .. dpdnhyYQ==" }
  },
  "Entries": {
    "/" : "bagugeeraijghexgvdvLuk3pwd6axbvzcajhd3pylz7sm3sfaitc7lnwivxra"
  },
  "IsRm": false,
  "Metadata": { "/": { "bytes": "gBI" } },
  "PreviousID": {
    "/" : "bagugeeraii6bahmbnnijhakhyscrlkcblwpwdvahzt53vzhd2jfryj43ebag"
  },
  "Provider": "bafzbeibhqavlasjc7dvbiopygwncnrtvjd2xmryk51aib7zyjor6kf3avm",
  "Signature": { "/": { "bytes": "CqsCCAASpg ... y/yqKWQ" } }
}
```

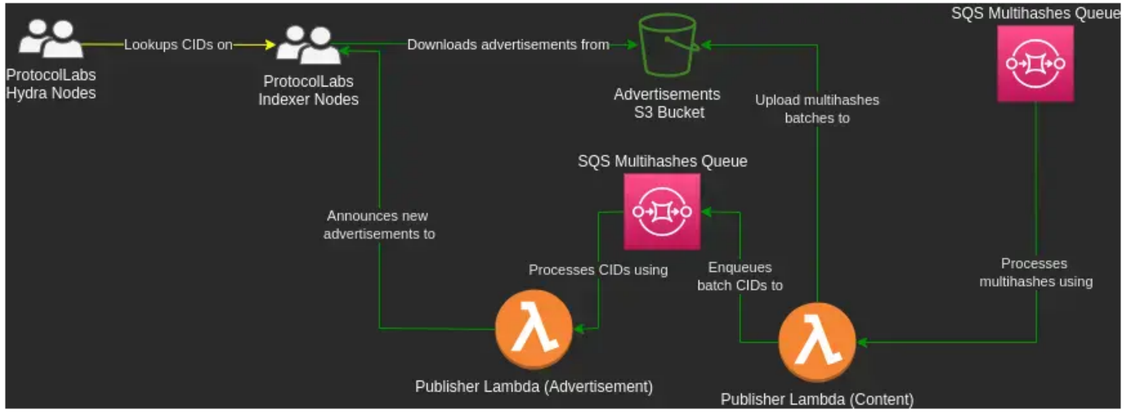
Entries file

Is is just a list of CIDs made available by the provider.

```
{
  "Entries": [
    {
      "/" : { "bytes": "EiCvOxoqEOhmarwWow7nY+bw2JL6SiLEYigT400EnBMeZA==" }
    },
    {
      "/" : { "bytes": "EiBOIwugRMRrvV3LFB7i3ixzaY5XTBASznCwwTWIu9Mx3g==" }
    },
    {
      "/" : { "bytes": "EiDc5NKXBT8ZSNfLDhTs2+72ZZ94qfruS]QEfJVM70XB6w==" }
    }
  ]
}
```

Publishing subsystem

Overview


ProtocolLabs
Hydra Nodes
Lookups CIDs on
ProtocolLabs
Indexer Nodes
Downloads advertisements from
Advertisements
S3 Bucket
Upload multihashes
batches to
SQS Multihashes Queue
SQS Multihashes Queue
Announces new
advertisements to
Processes CIDs using
Enqueues
batch CIDs to
Processes
multihashes using
Publisher Lambda (Advertisement)
Publisher Lambda (Content)

Concurrency Problem

Handle the huge load without losing data is challenging.

Several millions of blocks are uploaded every day
The ingestion queue can easily explode.

No advertisement can be lost
If data is not correctly received by the Indexer, it will be unreachable.

The head of the queue must be updated atomically
Concurrency control must be applied to the processors.

What now?

Publishing flow

The solution was to split the publishing process into two steps in order to reduce the size of the problem.

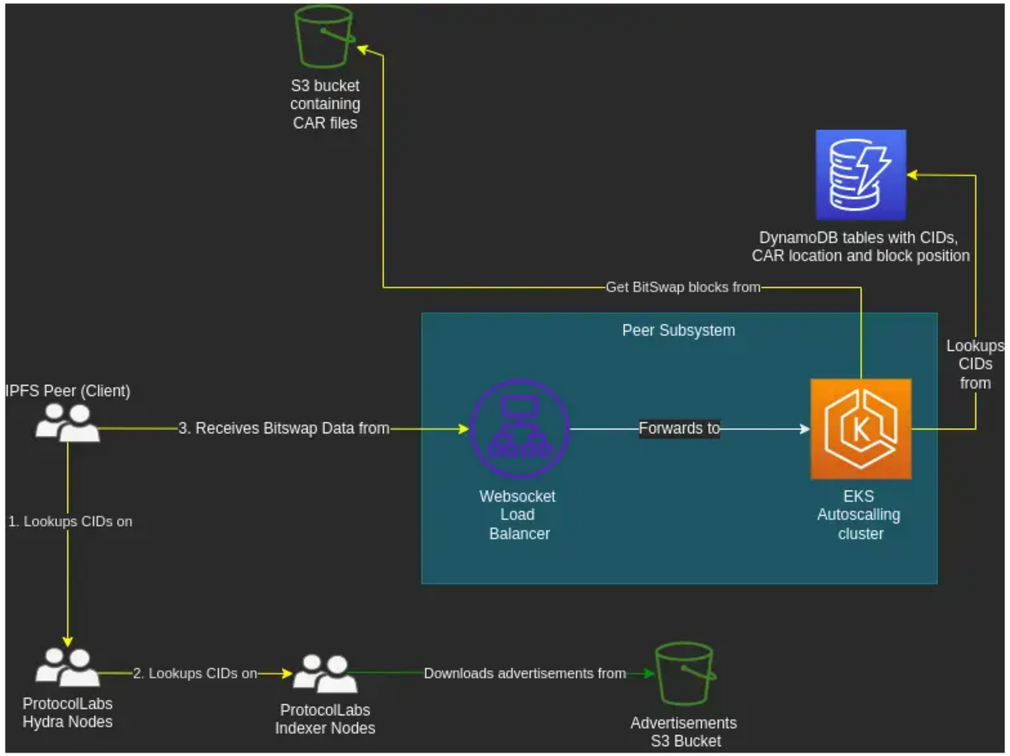
1. **Group indexed blocks into entries file**
   Concurrency is not limited as order does not matter.

2. **Assign entries to ads, then update the head**
   Concurrency is limited to **1**, with **10000** advertisement sent per execution.

The resulting subsystem is capable to easily advertise **a billion blocks per day**.

Peer subsystem

Overview


S3 bucket containing CAR files
DynamoDB tables with CIDs, CAR location and block position
Get BitSwap blocks from
Peer Subsystem
Lookups CIDs from
IPFS Peer (Client)
3. Receives Bitswap Data from
Forwards to
Websocket Load Balancer
EKS Autoscalling cluster
1. Lookups CIDs on
ProtocolLabs Hydra Nodes
2. Lookups CIDs on
ProtocolLabs Indexer Nodes
Downloads advertisements from
Advertisements S3 Bucket

Peer subsystem characteristic

Fully automatic

The subsystem is an autoscaling EKS Node.js cluster managed by an Elastic Load Balancer.

Stateless and lean

Searches for blocks on DynamoDB and leverages plain HTTP Byte-Range on S3 for serving.

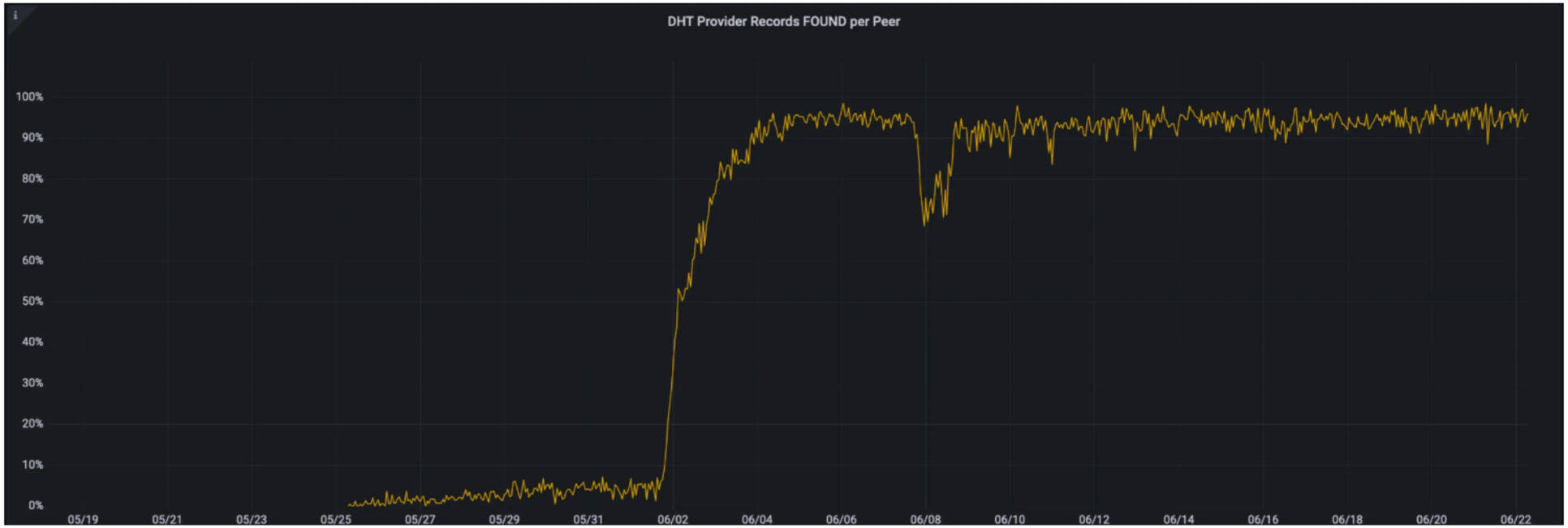
Simplified BitSwap

No external data is fetched, allowing to ignore wantlists and ledgers management.

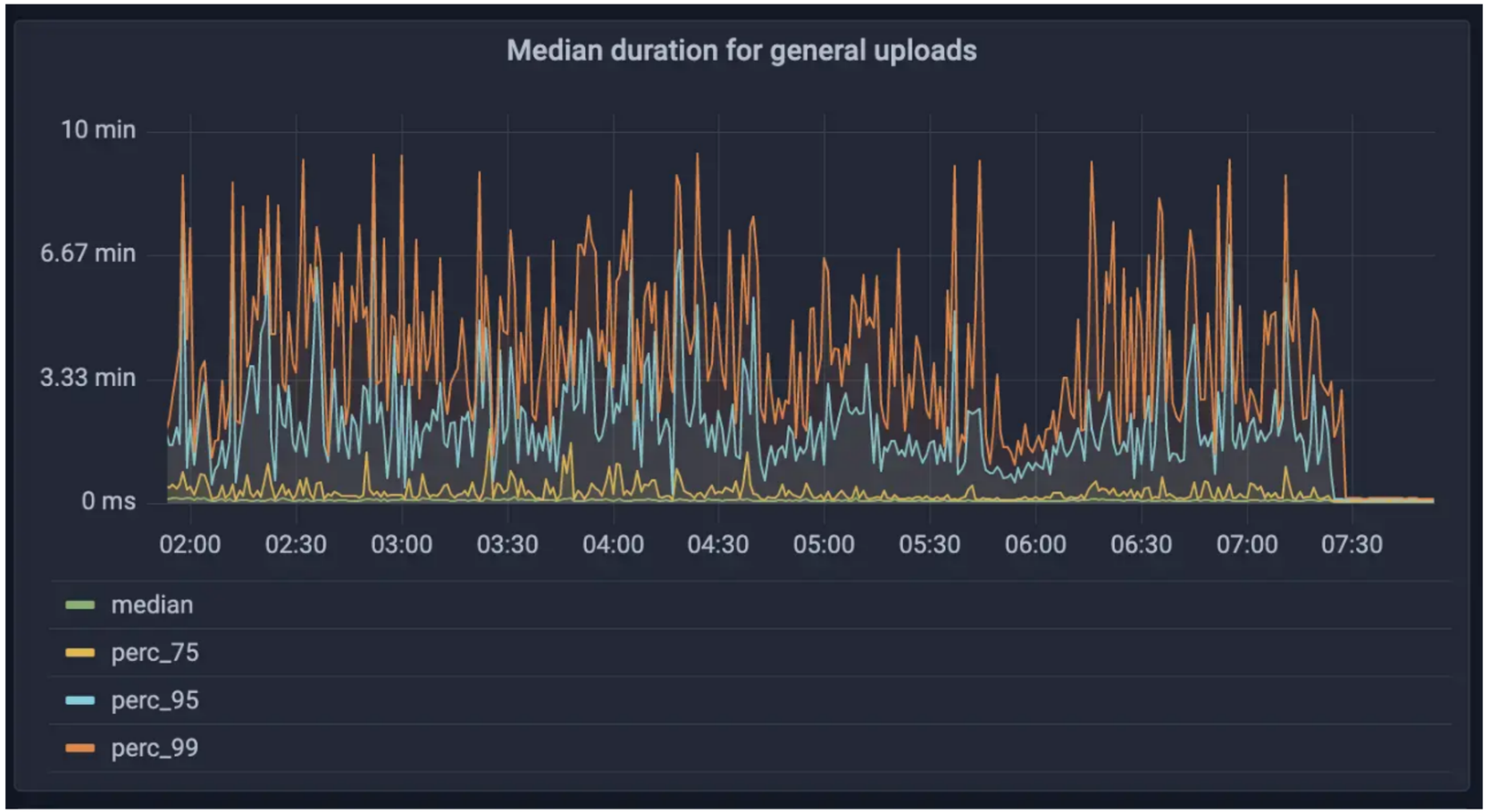
How does E-IPFS perform?

Hit rate went almost to 100%

The graph shows the hit-rate across **200TB** of data.



The average indexing time has dropped



Give me the numbers!

Here are the numbers after **six months** in production.

180M	16B	24B
CARs	**Blocks**	**Blocks to Cars**

Take home lessons

What can we learn from this long journey?

K.I.S.S.

IPFS protocols are very complex. Simplifying is the only way to have viable products.

HTTP is not dead yet

New protocols are very fashionable, but the good old HTTP can still be very performant.

Democracy is good

Being unable to lock in a single cloud provider led to flexible and lean architecture.

One last thing™

“Keep your eyes on the stars, and your feet on the ground.”

Theodore Roosevelt

Thank you!

Paolo Insogna

Node.js TSC, Principal Engineer

@p_insogna

paolo.insogna@platformatic.dev

Platformatic