HiPFiSH Documentation

Release 0.0

Chris Gough

January 23, 2017

Contents

CHAPTER 1

Overview

HiPFiSH is essentially two things:

- A high reliability archive
- A multi-protocol Content Delivery Network (CDN)

The archive is versatile, reliable and cost effective. It's composed of multiple 3rd party storage and backup services, achieving a higher level of redundancy than is possible with any single provider. The hipfish **Archive API** hides the complexity of dealing with multiple platforms behind a unified interface and security model.

Using the hipfish **Publishing API**, users can publish archived files via a conventional, best of breed HTTP/HTTPS CDN. The unique thing about hipfish is that it also provides Inter Planetary File System (IPFS) publishing, via the worlds largest IPFS CDN. This combines all the benefits of a traditional HTTP/HTTPS CDN with the benefits of IPFS (performance, efficiency, resiliance), ensuring the best possible experience for IPFS users.

1.1 Archive

Users can load files into the archive using a variety of methods;

- directly HTTP POST the file into an API
- provide the URL of a file (a worker will fetch it)
- provide an IPFS address (a worker will fetch it)

The archive is replicated across multiple networks and providers, ensuring high availability even if some providers go offline (or out of business).

Users have the option of ensuring archives not published (using a "privacy lock" feature). This is an excellent way to manage off-site backups, or retire

1.2 HTTP Publishing

Users can publish their files via a conventional HTTP/HTTPS CDN. This works in the usual way; files are replicated to regional centers and from there to edge servers, to minimise latency. A conventional CDN provides the best possible performance and availability for HTTP/HTTPS traffic.

This is as simple as toggling a single bit on the individual file (or archive) using the **Publishing API**. If the file (or archive) is "privacy locked", it is not possible to publish it via conventional HTTP/HTTPS CDN.

HiPFiSH is opinionated about file names. All files are published as https://cdn.hipfish.io/ipfs/<hash>/, where <hash> is the unique identifier for a specific version of the file. The <hash> is generated when the file is first saved to the archive. It is actually equal to the IPFS address of the file, if it were to be published via IPFS (the existance of the hash does not indicate that the file is or is not published via IPFS, it is a unique hash of the file contents).

1.3 IPFS Publishing

Users can publish an individual archived file (or entire archive) via the hipfish IPFS CDN, by simply toggling a single bit on the file (or archive) using the **Publishing API**. If the file (or archive) is "privacy locked", it is not possible to publish it via the IPFS CDN.

The HiPFiSH IPFS CDN is similar to a conventional HTTP/HTTPS CDN, in that files are replicated to edge servers in multiple regions. But IPFS is not a client/server protocol like HTTP/HTTPS, it is a peer to peer protocol. This means the edge servers are actually a globally distributed network freeloader-friendly peers.

In theory, IPFS does not need this. In theory, once your files have been downloaded by a number of fair-playing consumers that go on to serve those files to their neighbours, your own IPFS server can go offline and the data will remain available due to a grid of fair-playing peers. There are certainly circumstances where this works out, but in reality there are also circumstances where some users, particularaly early users of your data, do not enjoy the benefits of a healthy grid of peers serving your data.

The HiPFiSH CDN is a kind of booster or accellerator for the IPFS grid. By publishing your data to our distributed network of peers, you can be sure the worst case scenario is that only our peers are hosting your data. That's a very high minimum standard, it's roughly equivalent to the best possible case for HTTP/HTTPS! Any peering on top of the HiPFiSH CDN is an improvement, especially for remote networks or networks with a sudden rush of interest in your data (for example a science conference). So you have the best of both worlds; At the very least, your data will always be served with a CDN level of performance. If a user has fair-playing peers in their network neighourborhood, their performance will be even better.

1.4 Web Interface

The hipfish.io website allows users to sign up, manage their account and API access tokens. These API access tokens can be used via the API or CLI tool.

The web site also allows users to perform many of the management features available via the CLI tool and API from their workstation, tablet or phone.

1.5 Command Line Tool

There is a CLI tool (hipfish console) that uses the API, so it also needs an access token. Apart from manageing API access tokens, everything is possible via the CLI.

The CLI tool is called *hipfish*, and it is organised around sub-commands such as *archive* and *publish*. *hipfish –help* will list and describe the various subcommands.

hipfish subcommands also support the *–help* option, such as *hipfish archive –help*, which describe how to use the subcommands.

1.6 Application Programming Interface

The API uses a HATEOS design. This means the design incorporates natural HTTP verbs (GET, POST, PUT, PATCH, DELETE) with resources that can be identified by their URL.

The CLI tool uses the REST API. Therefor, everything that is possible via the CLI is also possible via the API.

1.7 More Information

This documentation covers:

- getting started: Try this out to get a feel for we web UI
- applications: Example user-stories, who might find hipfish useful
- console tool: Understand the features and concepts behind them
- API: Online resources and verbs for accessing them programatically

The documentation itself:

- is published at https://hipfish.readthedocs.io/
- is managed at https://github.com/hipfish/hipfish-docs/

If there are issues with the documents or there is something you would like help with, please feel free to raise a ticket at http://github.com/hipfish/hipfish-docs/issues

The http://hipfish.io/ website is obviously a key resource, and the email helpdesk at support@hipfish.io is also available.

CHAPTER 2

Getting Started

TODO: document simple common use-cases

2.1 Create an Account

TODO: walk through with screen shots

2.2 Create a private Archive

TODO:

- walkthrough with screen shots
- observe the opinionated resource name
- mention the CLI and API

2.3 Make the archive public

TODO:

- walkthrough with screen shots
- explain public vs. private archives
- mention the other archive options

2.4 Publish file via IPFS

TODO:

- walkthrough with screen shots
- explain HTTP vs. IPFS publishing
- mention can also publish entire archive

CHAPTER 3

Applications

TODO:

- describe the various user stories
- accelerated ipfs publishing
- highly durible storage, public or private

3.1 Software/Media/Games companies

IPFS:

- many users, great experience regardless of the state of the peer grid.
- performance benefits when peering kicks in

HTTPS:

- best of breed tradistionsl CDN with th the same simpinterface

3.2 Scientific/Industrial big data publishing

- minimise local network traffic
- durable storage, reliable hosting
- benefits of IPFS

3.3 Records Management

- public or private
- redundant across multiple networks
- programmatic hooks for next generation index management

3.4 Corporate data at rest

It's cheaper and easier to export the data to us than it is to sort it out. Free that expensive SAN.

3.5 Open Data

If you have an open data policy, we can make it easy (and great!).

CHAPTER 4

Console Tool

TODO: describe the CLI using feature semantics.

CHAPTER 5

Application Programming Interface

TODO:

- document the API using HATEOS pattern (organise by resource subheading, then resource/verb sub-subheading)
- cleanup this gumph

Path	Description
/archive/	Unlike a traditional HTTP(S) CDN, an IPFS CDN can not assume the file is available as a URL. Resources must somehow be placed in the hipfish archive before they can be reliably hosted. This archive is actually quite useful on it's own (it's durable, and can be kept private for record keeping use-cases). This is a low-level API.
/host-ing/	Once a file is in the archive, it can be reliably published on IPFS using the hosting API. This low-level API provides fine-grained controll over hosting.
/collec-tions/	A high-level API that allows users to archive and host collections of files as a group. This API is quite similar to traditional CDN APIs, and is designed for maximum convenience. It's probably the one you want :)

Low level *archive* api:

- POST file to archive, returns archive job GUID
- POST URL to spider that does *HTTP GET* into archive, returns archive job GUID
- POST IPFS address to spider that does *IPFS GET* into archive, returns archive job GUID
- GET archive job GUID, returns status of archive job
- *GET /archive/ipfs-hash/config* returns status/config of archive (if possible)
- *PUT /archive/upfs-hash/config* updates config of archive

Low level "hosting" api:

- POST ipfs address to hosting collection, if status of address is "archived" then idempotently adds file to hosting collection

5.1 Authentication

Access to the HiPFiSH API is controlled via API tokens. These can be managed via the Web UI.

TODO:

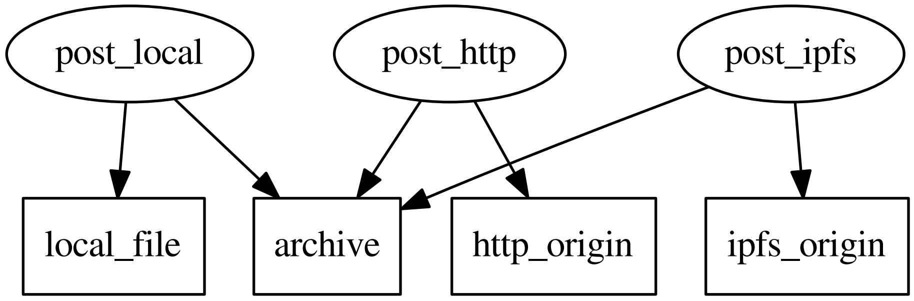
- insert screen shots and HOWTO for API token management and account management

- feature specification for API token management and account management
- would we ever use the API to manage API tokens (how to bootstrap it - would require account creation, credit card etc, the whole thing?)

5.2 Archive

One difference between HiPFiSH and a traditional (HTTP/HTTPS) Content Delivery Network is the concept Archive. With a traditional CDN, resources are proxied from an *origin* (or *upstream*). With IPFS, the origin resource may or may not exist, or may or may not be available.

With HiPFiSH, resources are first loaded into the archive, and then they can be served via IPFS.



We should have multiple methods to get a file into the archive:

- POST the file directly
- POST a URL, which will be subject to a HTTP(S) GET request to pull the file into the archive.
- POST an IPFS address, which will be subject to a an IPFS GET request to pull the file into the archive.

The first method (POST file) would work with a single file. The other two methods (POST URL/IPFS address) could reasonably work with a (potentially very large) collection of addresses. Conceivably, these collection of addresses could be of mixed type (URLs and IPFS addresses).

All these methods could take some time (non-blocking I/O), so they should immediately return a URL that can be used to check

- a single URL (returning a collection of statuses); or
- a collection of URLs, (each returning a single status)

The HTTP(S) and IPFS addresses may or may not succeed, due to file availability. So the

5.3 Hosting

TODO:

- document hosting requirements
- design hosting API

5.4 Search

TODO:

- document search requirements

5.5 Statistics

TODO:

- document statistics requirements
- design statistics API
- note similarity and difference to search API (comment elements/paramaters please)

5.6 Billing

TODO:

- document billing requirements
- design billing API