Node.JS Web Application Hosted on AWS with IPFS

Braden Vester

Purdue University in Indianapolis, CIT 50300

Indianapolis, Indiana

[bjvester@purdue.edu](mailto:bjvester@purdue.edu)

***Abstract –* The Interplanetary File System protocol offers a decentralized and redundant method for data discovery. Through peer-to-peer networking and content addressing it is possible to recall files stored on other devices. *IPFS* clusters provide automatic replication of stored data across a number of participant peers. Implementation of an *IPFS* cluster through *AWS* offers a convenient solution for a decentralized data transfer point. The creation of a web-based interface with Node.JS allows users to both upload and download data from the cloud-based cluster.**

NOMENCLATURE

|  |  |
| --- | --- |
| *IPFS* | Interplanetary File System |
| *CID* | Content Identifier |
| *Kubo* | Go-IPFS |
| *ICS* | IPFS-Cluster-Service |
| *ICC* | IPFS-Cluster-CTL |
| *AWS* | Amazon Web Services |
| *EC2* | Elastic Compute Cloud |
| *NPM* | Node Package Manager |

I. INTRODUCTION

The Interplanetary File System Protocol offers an intriguing method of data decentralization and protection. Through its redundant systems, *IPFS* maintains data availability and safety. Content identification hashes are used to query nearby peers for desired information. Clusters are created for data replication and ease of access.

As *IPFS* itself is not a data storage system, individual machines are required to store any replicated data. Cloud servers are optimal choices for such machines due to their near constant states of wakefulness [1]. Three *AWS EC2* nodes have been created to host an *IPFS* cluster. These nodes have been configured with *Ubuntu 24.04* and utilize a t2.large instance. For further details on these nodes, reference can be made to Appendix A. *Figure 1* shows a simple diagram of the created cluster.

A diagram of a network

AI-generated content may be incorrect.

Fig 1. Diagram of AWS hosted IPFS cluster

It should be noted that, within the diagram, alphabetically labeled nodes refer to the server itself; numerically labeled nodes refer to the *IPFS* service; single solid lines refer to a stable network connection between nodes; double solid lines refer to the running service on the server; and the single dotted line refers to the existing cluster service.

To interact with any server running an *IPFS* node, it is important to present a clear user interface. This would allow easy communication and querying of target nodes. To this end, a primitive web app developed with Node.JS. This app has been created to facilitate the transfer of data between the cloud-based cluster and a local machine.

II. DISCUSSION OF RELEVANT TECHNOLOGIES

A. *Discussion of Interplanetary File System and Related Technologies*

*IPFS* is a file sharing protocol that seeks to solve the issue of centralized data servers [2]. When working with *IPFS* users can pin files to an *IPFS* node. This action enters the file into a virtual web of interconnected devices, each of which contains a variety of saved documents and files. As a file becomes more popular to users on this network, it is pinned on more and more nodes. This tendency makes the file easier to access and protects it from becoming unavailable.

When a user wishes to download a file from the *IPFS* network, a *CID* is required to query the data. This query searches nearby nodes for the file and travels to the closest node with the file. This means that it will be easier to obtain the file the more often a file is pinned and the more nodes posses the file. Additionally, since the file is saved to numerous nodes, there is a measure of redundancy far exceeding other file protocols such as HTTP.

1) *go-IPFS (Kubo)*

*IPFS* has been implemented in a number of different languages and methodologies. This project uses a specific implementation of *IPFS* created using the language Go. This implementation is also referred to as *Kubo*.

Though *Kubo* offers the same capabilities of standard *IPFS*, this implementation contains several differences. One difference is the RPC API which assists in controlling the *IPFS* daemon. Another is the system requirements. *Kubo* requires at least 6 GiB of memory on 2 CPU cores [3]. This differs from standard *IPFS*, which only requires 512 MiB, and is a result of *Kubo’s* parallel nature.

2) *IPFS-Cluster-Service*

*IPFS* networks often contain clusters, or swarms, of individual nodes connected together. These clusters offer additional redundancy through automating pin replication and tracking [4]. To establish a cluster, a generic *IPFS* node must be created, and the *ICS* software must be installed.

3) *IPFS-Cluster-CTL*

*IPFS-Cluster-CTL* is the primary command suite through which interaction with the *ICS* is achieved. *ICC* works through a REST API and offers a number of essential capabilities [5]. Some key commands provided by *ICC* are the ‘pin’ command, the ‘add’ command and ‘sync’.

B. *Discussion of Amazon Web Services and Relevant Services*

Amazon Web Services offer a number of cloud-based solutions for individual needs. The platform offers options for data processing file storage, web hosting, etc. *AWS* offers a number of free items with paid services as additional options [6].

1) *Elastic Compute Cloud*

*AWS* Elastic Compute Cloud is a service by which it is possible to create and manage virtual servers. *EC2* offers a large pool of machine images and node instances allowing users to modify their server’s capabilities. *EC2* is noted as offering 400 Gbps network speeds [7].

2) *Elastic IP*

By default, *EC2* nodes are relegated to dynamic public IP addresses. This means that anytime the server shuts down, the address users use to access the device may be changed. To prevent such inconveniences, *AWS* offers an elastic IP system. This system allows users to reserve a single, or multiple, addresses and individually assign them to their *EC2* servers [8]. Addresses can be assigned and unassigned at will.

3) *Network Security Group*

Network Security Groups control traffic to and from an *EC2* instance. Essentially, a security group acts as a firewall for any nodes assigned to it. These groups allow users to modify both inbound and outbound traffic rules with configurations for destinations, ports, and traffic protocols. [9]

C. *Discussion of* *Node.JS and Related Services*

Node.JS is a runtime environment for the JavaScript language. It allows server creation and dynamic app hosting. The service is open-source and available on most platforms. This makes it a useful tool for experimentation and production. The service uses an event loop to update any code running within the environment and is designed to be asynchronous and scalable. [10]

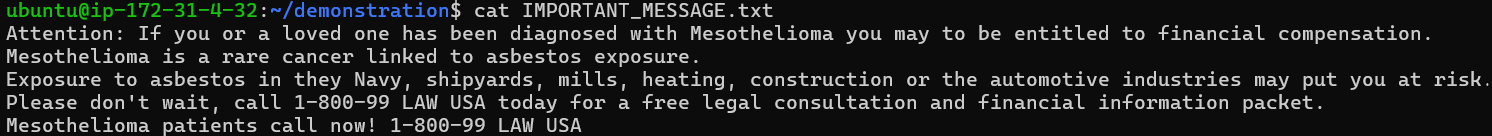


Fig 2. Contents of test text file after having been read to the command line.



Fig 3. Use of 'ipfs-cluster-ctl add' to upload test file to cluster storage.

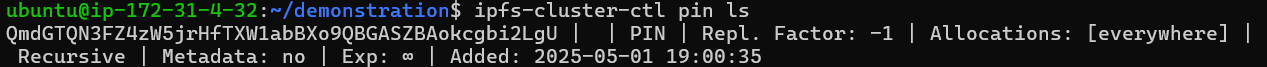


Fig 4. Output of the 'ipfs-cluster-ctl pin ls' command as viewed from Node A.

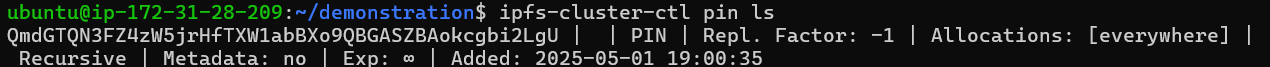


Fig 5. Output of the 'ipfs-cluster-ctl pin ls' command as viewed from Node B.

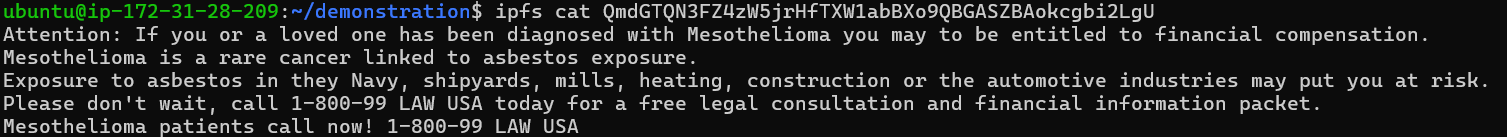


Fig 6. Pinned file viewed through 'ipfs cat' command.

1) *Node Package Manager*

*NPM* is rather self-explanatory. It is a service which offers management capabilities for JavaScript packages. [11] This service works in conjunction with Node.JS. *NPM* is similar in role to Python’s pip service.III. PRACTICAL DEMONSTRATIONS

To understand the abilities of *IPFS* as a file storage protocol, it is important to complete a practical test of the typical operations of an *IPFS* node. To this end, a demonstration was completed on a previously created *IPFS* cluster swarm. A secondary demonstration, this one showcasing the functionality of the Node.JS web application, was also completed.

Recall that this swarm is comprised of three nodes hosted on *AWS* EC2 servers. These servers are easily accessible through a command-line interface using SSH. This text-based interaction contrasts with the graphical interface presented by the Node.JS web application.

A. *General Demonstration of IPFS and IPFS-Cluster Capabilities*

To begin the demonstration, a file, IMPORTANT\_MESSAGE.txt’ was created on Node A with some filler text. This file will act as a test file to be pinned and copied by the *IPFS* cluster. *Figure 2* shows this test file read through the command line.

With the test file created, it is possible to “upload” the file to the *IPFS* system. Recall that this can be done through the use of the IPFS-Cluster-CTL command suite. The *ICC* command ‘add’ is specifically used to accomplish this upload. *Figure 3* shows the use of this command to load the test file. Note the output of this command which contains the file’s *CID* hash

In order to verify the file upload, another *ICC* command, ‘pin’, with the additional action ‘ls’, can be used to list all files pinned to the cluster. *Figure 4* displays the results of this command. Specific attention should be paid to the various fields this output shows. First among these fields is the *CID* hash, which remains unchanged from *Figure 3.* There are, however, additional fields for upload date, expiration time, and allocation locations. Additionally, to verify that the services of the cluster are working, the same command was run from Node B. *Figure 5* shows this output on the new node.

*IPFS* offers a command, ‘cat’, which can be used to preview the contents of a pinned file. This can be helpful when deciding whether a file should be saved locally. *Figure 6* shows the output of this command when run with the *CID* of the test file.

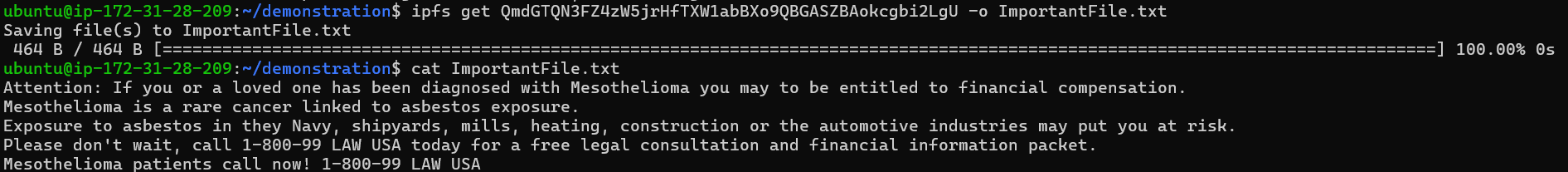


Fig 7. Result of 'ipfs get' command used to save a local copy of the test file.

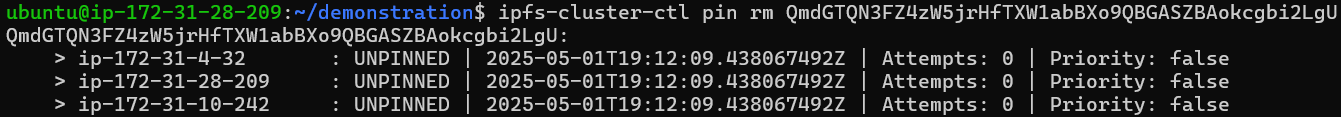


Fig 8. Use of the 'ipfs-cluster-ctl pin rm' command to unpinthe test file from the cluster swarm.

A screenshot of a computer

AI-generated content may be incorrect.

Fig 9. Dashboard of the Node.JS web application.

Considering that it might be necessary to download a file from *IPFS* storage, another command in the *IPFS* suite, ‘get’, can be used to make a local copy. This command uses the tag -o to specify the path of the output file. *Figure* 7 shows the output from this command along with a verification to show that the file remains the same.

On occasion, when a file no longer needs to remain pinned within the *IPFS* system, it might be necessary to remove the pinned file. This can be accomplished through another use of the *ICC* command ‘pin’, albeit with the supplementary command ‘rm’. *Figure 8* shows the use of this command. Specific note should be made of the three responses, each of which display the unpinning from an individual node in the cluster swarm

B. *Demonstration of Node.JS Web Application and Functionalities*

Moving on now to the demonstration of the web-based Node.JS application, it should be recalled that the application itself is meant to act as a medium through which users can interact with the *IPFS* cluster. Additionally, it should be understood that this application is a more primitive tool for understanding how it is possible to interact with such clusters. In fact, this specific implementation executes the very same commands shown in the earlier demonstration.

*Figure 9* displays the dashboard of the application. Several notable features of this dashboard include fields for uploading both local files and *CID* hashes, interaction buttons for adding, listing, removing, previewing, and downloading data, and finally a response field for command outputs.

When seeking to use the application, it is worth using the ‘List’ button to observe any pinned files. *Figure* 10 shows the dashboard with the response field populated with available *CIDs.* Note that this *CID* is the same as the earlier hash representing the test file.

A screenshot of a computer

AI-generated content may be incorrect.

Fig 10. Application dashboard with response listing pinned files.

A screenshot of a computer

AI-generated content may be incorrect.

Fig 11. Preview of selected CID shown through application dashboard.

Having observed the available files, it might be prudent to preview a file. *Figure* 11 shows such an interaction. Note that the *CID* selection field was filled for this interaction.

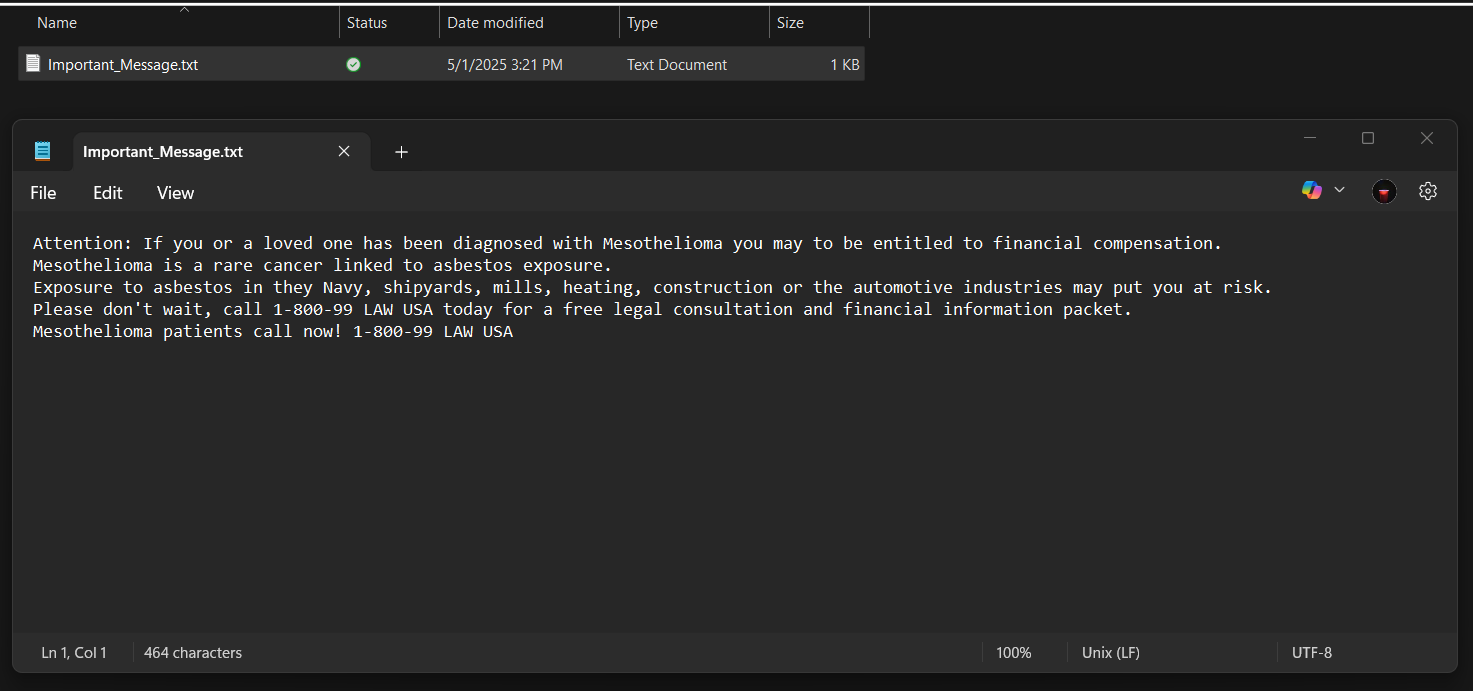


Fig . Test file downloaded from IPFS cluster to local machine.

A screenshot of a computer

AI-generated content may be incorrect.

Fig 13. Dashboard showing file added to the IPFS cluster swarm.

A screenshot of a computer

AI-generated content may be incorrect.

Fig 14. Dialog box used for naming downloaded file.

With a *CID* chosen and previewed, it might be desirable to download the file to a local machine.

It should be understood that by downloading the desired file it is saved to a machine outside of the cloud-based cluster swarm. For this demonstration a Windows 11 laptop was used as the outside device. *Figure 12* shows a dialog box requesting a name for the downloaded file. *Figure 13* shows the file downloaded to the Windows 11 machine.

The web application also supports uploading files from a local machine to the cloud cluster. A second test file was created on the Windows 11 machine. This file was uploaded to the cloud cluster through the dashboard’s functionality. This is shown in *Figure 14.* To verify the upload, the *CID* hash shown in *Figure 14* can be compared to the most recently uploaded hash in *Figure 15*. Additionally, *Figure 16* shows the new test file downloaded and read to the command line on Node C.

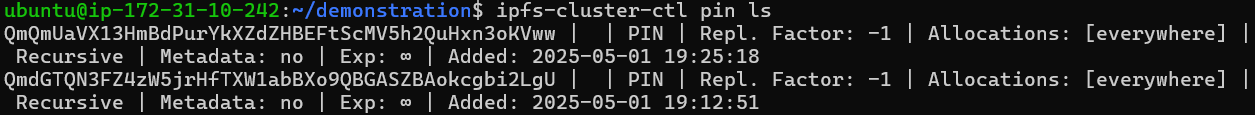


Fig 15. Pinned files listed on Node C after upload of second test file.

A screen shot of a computer

AI-generated content may be incorrect.

Fig 16. Second test file locally saved and read on Node C.

IV. NOTEABLE ISSUES AND CONSIDERATIONS

A. *Differences Between Cluster Setup on Local and Cloud-Based Devices*

Due to the potential cost associated with utilizing *AWS* based services, an initial test setup of the *IPFS* cluster was completed on a series of virtual machines. When connecting *IPFS* nodes in a cluster format, a secret must be shared to each node. This allows the cluster to identify other participating peers. Sharing this secret is the only step needed when setting up a cluster wherein all nodes are on the same subnet. However, in a case where each node resides on a different subnet, such as with the private addresses of the *AWS* cluster (reference can be made to *Table 3, Table 4,* and *Table 5* of AppendixAfor node addresses) it is necessary to create a bootstrap node. This is a node that acts as the central loadstone for the cluster. To specify the bootstrap node, a tag ‘—bootstrap’ must be added to the *ICS* daemon’s execution. This tag is followed by a string with the format /ip4/<bootstrap address>/tcp/9096/p2p/ <bootstrap node id>.

B. *Technology Age and Package Deprecation*

Initial efforts at developing the *IPFS* web application sought to use more developed means of interacting with the cloud cluster. Early efforts sought to use the *NPM* package ipfs-http-client within the JavaScript backend. Unfortunately, the package is noted as being deprecated on the *NPM* website [12]. Attempts to use the technology in opposition to the warning resulted in failure. Another attempt was made to use the package Helia, which was recommended on the ipfs-http-client webpage. This package, while newer than ipfs-http-client, is also aged and resulted in a number of errors [13].

Though it is likely still possible to use these packages, the numerous errors encountered presented severe and unnecessary hurdles. For the sake of a simple demonstration, the decision was made to design the application with direct console command execution.

C. *Prevention of Cluster Formation by Security Group Disagreement*

When configuring the *AWS* hosted *IPFS* cluster, issues arose with a lack of communication between the EC2 servers. Initial configuration of the servers left each node in a separate network security group. These separate groups isolated each node from the other, thus preventing assimilation into the cluster. While modification of the group policies could have resolved the issue, each node was moved to the same network security group for easier management.

V. CONCLUSION

It should be noted that original plans had been made to create a news display app that utilizes *IPFS* to store data. However, due to technical difficulties and the realization that the web scraping nature of that idea meshed poorly with the functionality of *IPFS,* a new route was chosen. The web application demonstrated above became the replacement. Though truly limited in its scope and capabilities, this application provided an easier medium through which to understand the operations of *IPFS* and its clusters.

The abilities of *IPFS* to store data across multiple servers at once makes it useful for maintaining data security. Coupling this technology with a cloud environment creates a stable source for data storage and retrieval.

In future work it might be useful to design a system wherein the application can switch between nodes. This would better present the real strength of *IPFS,* that being continued data access with offline servers. Currently, the web application is only operation on the bootstrap node. Should that server be interrupted, the user interface would be unreachable, although the data would be retained on the other servers.

APPENDIX A

VERSIONS AND CONFIGURATIONS OF TECHNOLOGY

TABLE 1.

AWS IMAGE INFORMATION

|  |  |
| --- | --- |
| Image Field | Version |
| Machine Image Type | Ubuntu Server 24.04 LTS (HVM) |
| System Architecture | 64-bit (x86) |

Presenting information pertaining to the specific AMI chosen for this project.

TABLE 2.

INSTANCE IMAGE INFORMATION

|  |  |
| --- | --- |
| Instance Field | Value |
| Instance Type | t2.Large |
| Processing Cores | 2 |
| Memory | 8 GiB RAM |
| Storage | 10 GiB SSD |
| Run Cost | 0.0928 US/hr |

Presenting information pertaining to the specific instance chosen for this project.

TABLE 3.

NODE A INFORMATION

|  |  |
| --- | --- |
| Node Field | Value |
| Instance Name | CIT503\_Final\_NodeA |
| Hostname | Ubuntu |
| Public IP | 3.131.204.37 |
| Private IP | 172.31.34.32 |
| IPFS Node | Node0 |
| Bootstrap | True |

Presenting information about the first server node.

TABLE 4.

NODE B INFORMATION

|  |  |
| --- | --- |
| Node Field | Value |
| Instance Name | CIT503\_Final\_Nodeb |
| Hostname | Ubuntu |
| Public IP | 3.147.142.205 |
| Private IP | 172.31.10.242 |
| IPFS Node | Node1 |
| Bootstrap | False |

Presenting information about the second server node.

TABLE 5.

NODE C INFORMATION

|  |  |
| --- | --- |
| Node Field | Value |
| Instance Name | CIT503\_Final\_NodeC |
| Hostname | Ubuntu |
| Public IP | 3.136.151.224 |
| Private IP | 172.31.28.209 |
| IPFS Node | Node2 |
| Bootstrap | False |

Presenting information about the third server node.

TABLE 6.

SOFTWARE VERSIONS

|  |  |
| --- | --- |
| Technology | Version |
| Ubuntu Server | 24.04 |
| go-ipfs (kubo) | 0.34.1 |
| ipfs-cluster-service | 1.1.2 |
| Ipfs-cluster-ctl | 1.1.2 |
| Node.JS | 18.19.1 |
| Npm | 9.2.0 |

Software used throughout the project and their versions

REFERENCES

[1] Amazon Web Services, “Amazon EC2 Service Level Agreement (SLA) – Historical Information,” [Online]. Available: <https://aws.amazon.com/ec2/sla/historical/>. [Accessed: May 3, 2025].

[2] IPFS Documentation, “Welcome to IPFS documentation,” [Online]. Available: <https://docs.ipfs.tech/>. [Accessed: May 3, 2025].

[3] IPFS Documentation, “Install IPFS from the command line,” [Online]. Available: <https://docs.ipfs.tech/install/command-line/#system-requirements>. [Accessed: May 3, 2025].

[4] IPFS Cluster, “IPFS Cluster,” [Online]. Available: <https://ipfscluster.io/>. [Accessed: May 3, 2025].

[5] IPFS Cluster, “IPFS Cluster ctl Reference,” [Online]. Available: <https://ipfscluster.io/documentation/reference/ctl/>. [Accessed: May 3, 2025].

[6] Amazon Web Services, “AWS Free Tier,” [Online]. Available: <https://aws.amazon.com/free/>. [Accessed: May 3, 2025].

[7] Amazon Web Services, “Amazon EC2,” [Online]. Available: <https://aws.amazon.com/ec2/>. [Accessed: May 3, 2025].

[8] Amazon Web Services, “Elastic IP addresses - Amazon EC2,” [Online]. Available: <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/elastic-ip-addresses-eip.html>. [Accessed: May 3, 2025].

[9] Amazon Web Services, “Security groups for your VPC,” [Online]. Available: <https://docs.aws.amazon.com/vpc/latest/userguide/vpc-security-groups.html>. [Accessed: May 3, 2025].

[10] Node.js, “About Node.js,” [Online]. Available: <https://nodejs.org/en/about>. [Accessed: May 3, 2025].

[11] Node.js, “An introduction to the npm package manager,” [Online]. Available: <https://nodejs.org/en/learn/getting-started/an-introduction-to-the-npm-package-manager>. [Accessed: May 3, 2025].

[12] npm, “ipfs-http-client,” [Online]. Available: <https://www.npmjs.com/package/ipfs-http-client>. [Accessed: May 3, 2025].

[13] GitHub, “ipfs/helia,” [Online]. Available: <https://github.com/ipfs/helia>. [Accessed: May 3, 2025].