



# IPFS Constellation: Developer Integration Guide

## Overview

Switching from a standalone IPFS node (or public gateways like Infura/Pinata) to our **Constellation Cluster** requires **zero code changes** to your application logic.

Constellation mirrors the standard IPFS HTTP API. You simply need to update your **API Endpoint Configuration** to point to our highly available fleet instead of a local daemon or single provider.

---

## 1. Connection Details

Replace your current IPFS connection string with the Cluster Endpoint.

Service	Current (Old)	Constellation (New)
API Endpoint	localhost:5001 or ipfs.infura.io	https://api.yourdomain.com
Gateway	localhost:8080 or ipfs.io	https://gateway.yourdomain.com
Protocol	HTTP/HTTPS	<b>HTTPS</b> (Recommended)

(Note: Ensure you have been issued an API Key if our cluster requires authentication).

---

## 2. Code Examples

### A. JavaScript / TypeScript ([ipfs-http-client](#))

Works with the standard Kubo client library.

JavaScript

None

```
import { create } from 'ipfs-http-client'
```

```

// OLD: const ipfs = create({ url: 'http://localhost:5001' })

// NEW: Connect to Constellation
const ipfs = create({
  url: 'https://api.yourdomain.com',
  // If we require Basic Auth (Optional):
  // headers: {
  //   authorization: 'Basic ' + Buffer.from(username + ':' +
password).toString('base64')
  // }
})

async function upload() {
  const { cid } = await ipfs.add('Hello Constellation Fleet!')
  console.log('Data pinned to cluster:', cid.toString())
}

upload()

```

## B. Python (ipfshttpclient)

Python

```

None

import ipfshttpclient

# OLD: client = ipfshttpclient.connect()

# NEW: Connect to Constellation
client =
ipfshttpclient.connect('/dns/api.yourdomain.com/tcp/443/https')

res = client.add('test_file.txt')
print(f"Uploaded to Geo-Redundant Storage: {res['Hash']}")
```

## C. CLI / cURL

Ideal for CI/CD pipelines or quick tests.

Bash

```
None

# Upload a file
curl -X POST -F file=@myfile.png \
"https://api.yourdomain.com/api/v0/add?pin=true"

# Response
# {"Name": "myfile.png", "Hash": "Qm...", "Size": "1234"}
```

---

## 3. Verifying Persistence

Unlike standard IPFS where a file is only on *your* node, Constellation automatically replicates it. You can verify this "Autonomic Healing" behavior:

1. **Upload a file** using the steps above.
2. **Wait 5 seconds** (for the Allocation Engine to route data).
3. **Check Redundancy**:

Bash

```
None

# If you have cluster admin access:
constellation-cli pin ls <CID>

# Output should show multiple allocations:
# CID: Qm...
#   - Node A (US-East): PINNED
#   - Node B (EU-West): PINNED
```

---

## 4. Troubleshooting

- **Error: connection refused:** Ensure you are using <https://> if the endpoint is secured.
- **Slow Uploads:** For files >100MB, ensure your client timeout is set to at least [60s](#). The cluster replicates data synchronously for safety; this takes slightly longer than a single-node pin.
- **CORS Errors:** If calling from a browser, ensure [api.yourdomain.com](http://api.yourdomain.com) allows your specific domain in its CORS headers.

---

## Final Readiness Check

You have now completed the full technical stack:

1. **Architecture:** Defined.
2. **Implementation:** Configured (Systemd/Nginx).
3. **Operations:** Monitoring (Prometheus) & Testing (Chaos Script).
4. **Onboarding:** Developer Guide (Above).