Integrating IPFS into your app.

1. Use **‘ipfs-core’** node package
   1. ipfs-core contains the core api and is intended to be used to run an IPFS node as part of your application without the need to start external processes or manage API ports and servers.
   2. READ MORE: <https://github.com/ipfs-examples/js-ipfs-examples#ipfs-or-ipfs-core>
2. IPFS Core API documentation for the **‘ipfs-core’** package: <https://github.com/ipfs/js-ipfs/tree/master/docs/core-api>

Maybe just implement JS-IPFS in a node.js app?

* Send file to a sever with an IPFS node running?



‘**ipfs-http-client’** - client library that controls an active IPFS node (Kubo or JS-IPFS) running through its RPC API.

Use ‘**ipfs-http-client’** to connect the client web app to an IPFS node.

A Node.js application will interact with the Kubo IPFS Node via its exposed RPC HTTP API.

The Kubo IPFS Node itself HAS TO BE LOCAL (NOT REMOTE) during development. You need to have a running local daemon to interact with the IPFS Node’s API for RPC, by getting the local multiaddr or localhost url and passing it as an argument to the ‘create’ method of ‘ipfs-http-client’ when creating an instance of the HTTP API client.

* Problem: How to deploy the Node.js app that needs to interact with **the supposed local Kubo IPFS Node**, UNLESS if we could somehow run and deploy a Kubo IPFS daemon on Heroku with its own link so that it can be used to specify the URL argument of the ‘ipfs-http-client’s ‘create’ method in the Node.js app.
* POSSIBLE ALTERNATIVE:
  + Deploy DM-FS system web app on fleek.
  + Use 3rd party pinning services to upload files to the IPFS network.
  + Users can choose to run their own local IPFS node
    - Users install Kubo IPFS in their own machine,
    - Work on including the ‘ipfs-http-client’ module in the react web app, and specify the “multiaddr” of the local HTTP RPC API in the ‘create’ method of the module.