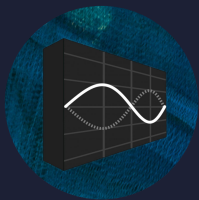


DEVELOPING APPS WITH IPFS APIs

CORE COURSE C



IPFS Camp



Marcin Rataj

@lidel



Jim Pick

@jimpick



Hugo Dias

@hugomrdias

Using **IPFS**
as a **building block**
in **your application**

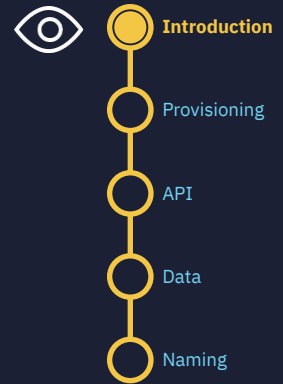






Introduction

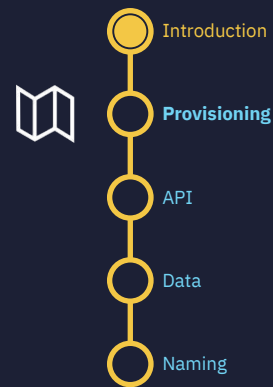
Goal: build basic understanding about **IPFS** in apps..





Introduction

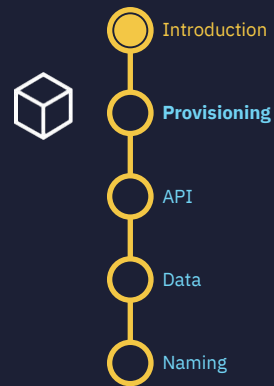
Where to run it





Introduction

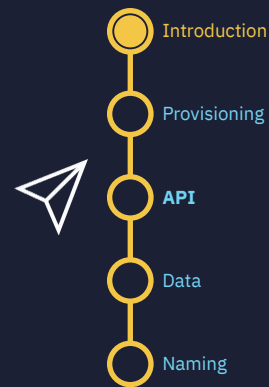
What to run





Introduction

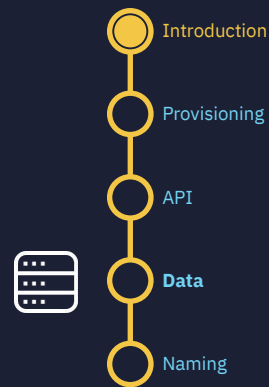
How to **talk** to it





Introduction

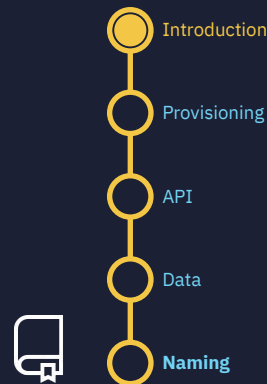
How to store data





Introduction

How to track updates (spoiler: by naming things)



Provisioning



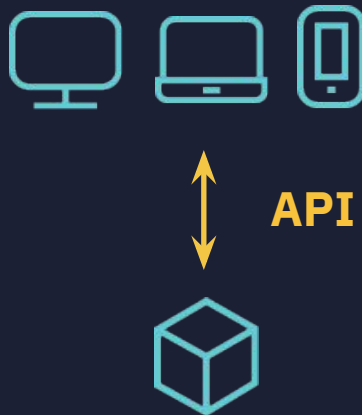
- Introduction
- Provisioning
- API
- Data
- Naming

Where to **run** IPFS



IPFS in Apps:
**High
Level
Patterns**

App with API Client



IPFS
Service

App



Embedded
IPFS Node



IPFS

as a

Backend Service



IPFS node running as a service
within your cloud infrastructure

Backend Service: Self-hosted

Roll your own:

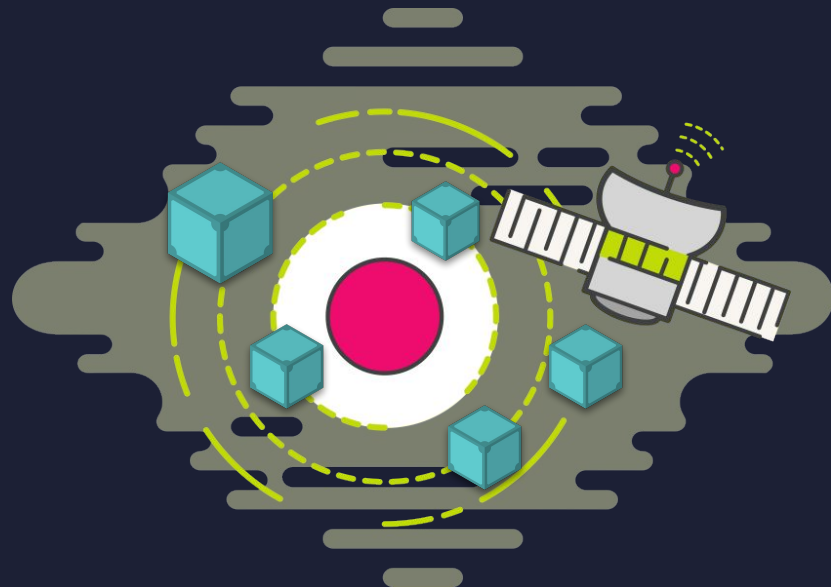
go-ipfs github.com/ipfs/go-ipfs

ipfs-cluster cluster.ipfs.io

Learn more:

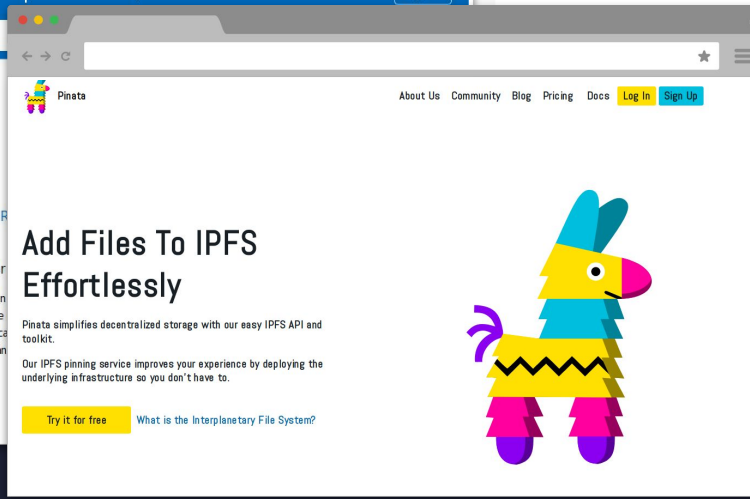
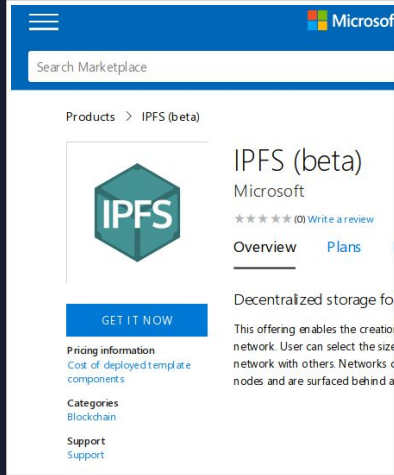
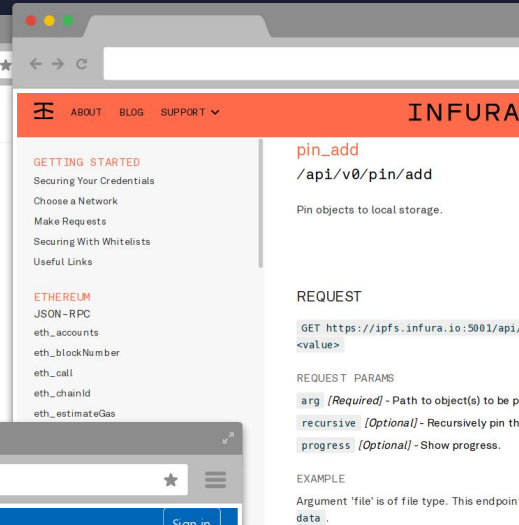
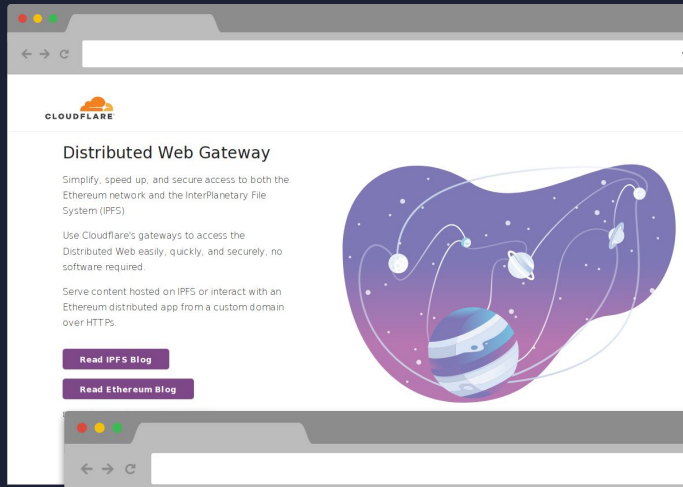
Elective E: Deploying IPFS Infrastructure

Elective B: IPFS Cluster



Backend Service: Cloud Providers

IPFSaaS:
Infura
Pinata
Microsoft Azure
Cloudflare



OR..

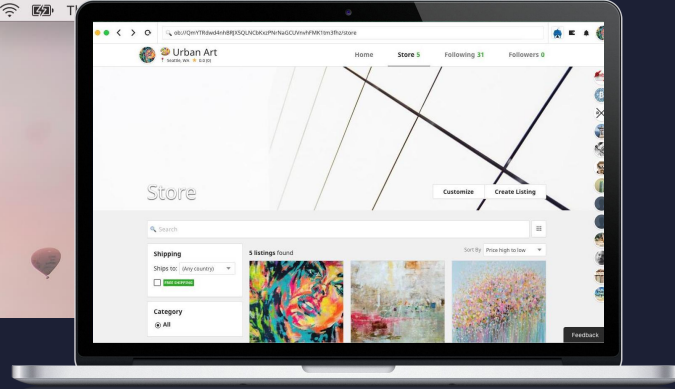
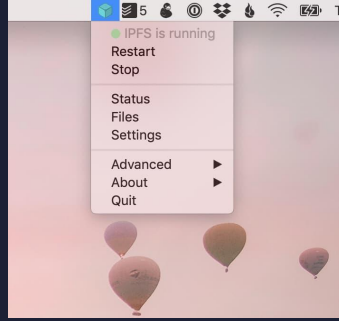


Embedded IPFS Node

shipping with your app



IPFS Desktop
OpenBazaar



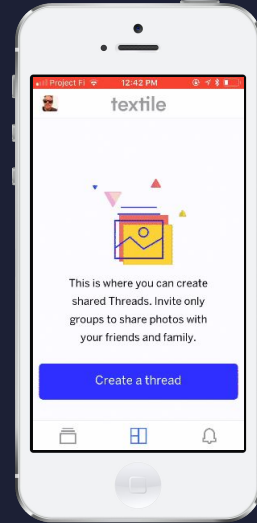
Embedded go-ipfs



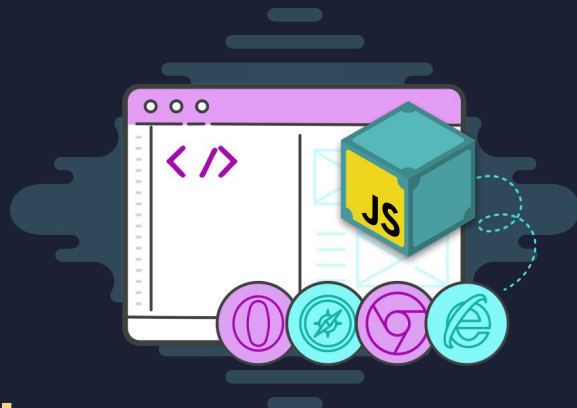
Textile Photos

Elective D

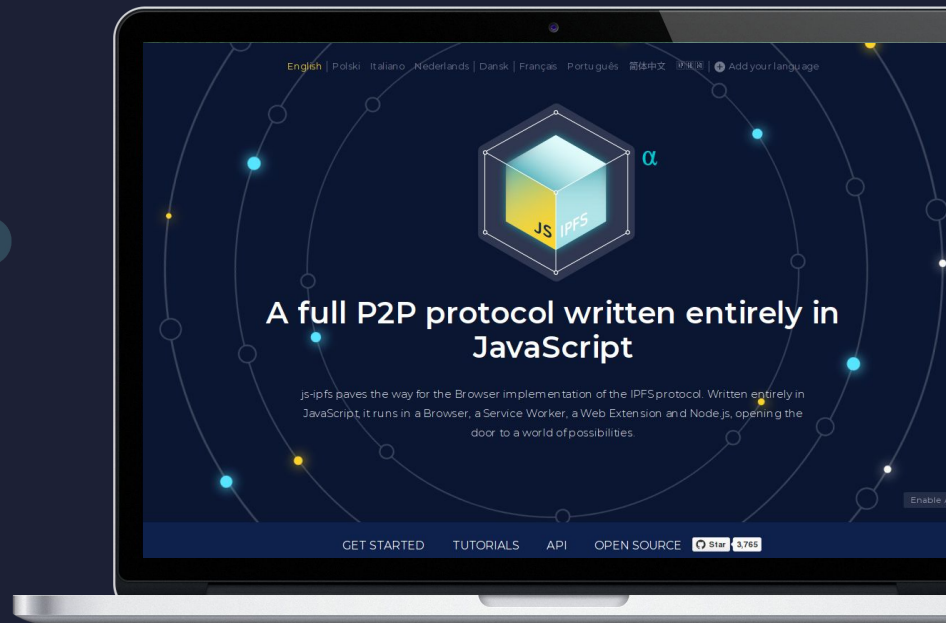
Building DApps with Textile



No need to include the entire thing!
github.com/hsanjuan/ipfs-lite












Embedded js-ipfs



Full IPFS node running in a web context (Web Browser, Electron, Node)

To understand limitations in browsers,
learn about libp2p transports and discovery in **Core Course B**

Transports in js-ipfs & go-ipfs

Transport	Browser	Node	Go
TCP			
Websockets			
WebRTC			
QUIC			

GO- vs JS-

What are the differences?



go-ipfs

interop

.JS-ipfs

Performance:
big data sets, lots of
connections

Interop:
Protocol and CoreAPI

Transports:
TCP and Websockets

Flexible:
Web Browser, Web
Workers, Node,
Electron, Terminal,

Advanced Features:
Filestore, Urlstore,
Private Networks,
Relays, Tunneling, QUIC

Local Discovery:
mDNS

Soon:
DHT

Soon:
DHT, GC, Profiles

APIs

- Introduction
- Provisioning
- API**
- Data
- Naming



IPFS CLI API

docs.ipfs.io/reference/api/cli

\$ ipfs --help

docs.ipfs.io/reference/api/cli

USAGE

ipfs - Global p2p merkle-dag filesystem.

SYNOPSIS

```
ipfs [--config=<config> | -c] [--debug=<debug> | -D] [--help=<help>] [-h=<h>] [--local=<local> | -L]
[--api=<api>] <command> ...
```

OPTIONS

-c,	--config	string	- Path to the configuration file to use.
-D,	--debug	bool	- Operate in debug mode.
--help		bool	- Show the full command help text.
-h		bool	- Show a short version of the command help text.
-L,	--local	bool	- Run the command locally, instead of using the daemon.
--api		string	- Use a specific API instance (defaults to /ip4/127.0.0.1/tcp/5001).
--enc,	--encoding	string	- The encoding type the output should be encoded with (json, xml, or text).

Default: text.

--stream-channels	bool	- Stream channel output.
--timeout	string	- set a global timeout on the command.

SUBCOMMANDS

BASIC COMMANDS

init	Initialize ipfs local configuration
add <path>	Add a file to IPFS
cat <ref>	Show IPFS object data
get <ref>	Download IPFS objects
ls <ref>	List links from an object
refs <ref>	List hashes of links from an object

IPFS HTTP API

docs.ipfs.io/reference/api/http

\$ ipfs daemon

Initializing daemon...

go-ipfs version: 0.4.21-

Repo version: 7

System version: amd64/linux

Golang version: go1.12.5

Swarm listening on /ip4/127.0.0.1/tcp/4001

Swarm listening on /ip4/172.17.0.1/tcp/4001

Swarm listening on /ip4/192.168.0.16/tcp/4001

Swarm listening on /p2p-circuit

Swarm announcing /ip4/127.0.0.1/tcp/4001

Swarm announcing /ip4/172.17.0.1/tcp/4001

Swarm announcing /ip4/192.168.0.16/tcp/4001

Swarm announcing /ip6>:::1/tcp/4001

API server listening on /ip4/127.0.0.1/tcp/5001

WebUI: http://0.0.0.0:5001/webui

Gateway (readonly) server listening on /ip4/127.0.0.1/tcp/8080

Daemon is ready

API port

Gateway port

```
$ ipfs id
```

```
$ ipfs --api /ip4/127.0.0.1/tcp/5001 id
```

```
$ curl http://127.0.0.1:5001/api/v0/id
```

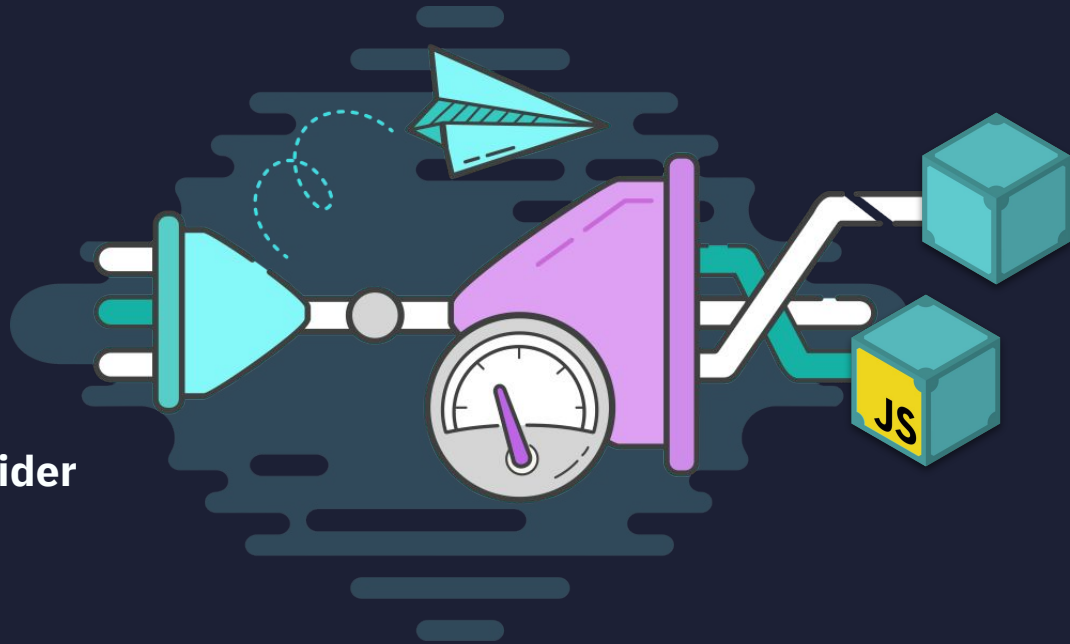
 API port

docs.ipfs.io/reference/api/http

Look for **Poster Session** about **IPFS HTTP API**

API Fallback

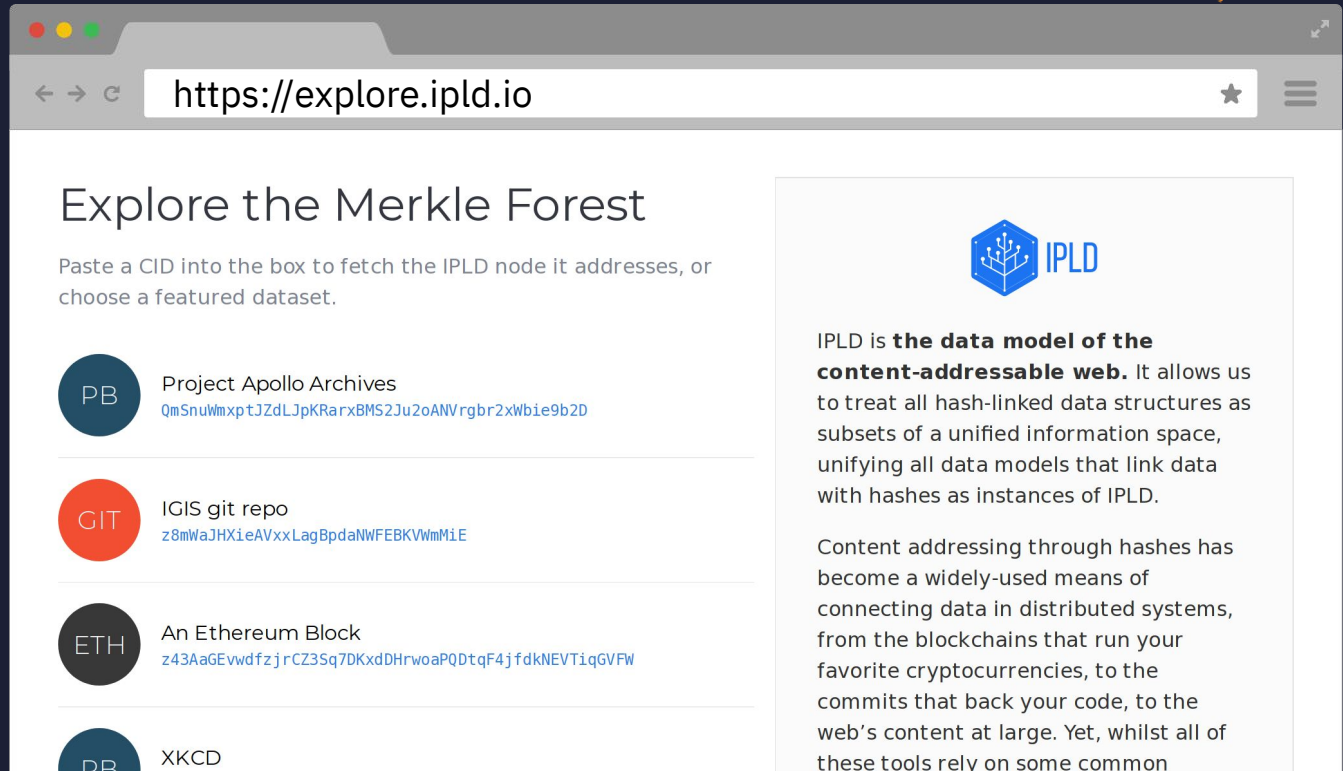
Support **more than one API Provider**



Fallback example: IPLD Explorer

Open the console to see how the app is looking for the best API

- Application does not care about details of the node, just needs to get the data
- Tries to use API of existing local node via `js-ipfs-http-client`
- Falls back to spawning embedded `js-ipfs`



https://explore.ipld.io

Explore the Merkle Forest

Paste a CID into the box to fetch the IPLD node it addresses, or choose a featured dataset.

- PB** Project Apollo Archives
`QmSnuWmxptJZdLJpKRarxBMS2Ju2oANVrgbr2xWbie9b2D`
- GIT** IGIS git repo
`z8mWaJHXieAVxxLagBpdaNWFEBKVWmMiE`
- ETH** An Ethereum Block
`z43AaGEvwdfzjrCZ3Sq7DKxdDHRwoaPQDtqF4jfdkNEVTiqGVFW`
- PB** XKCD

IPLD

IPLD is **the data model of the content-addressable web**. It allows us to treat all hash-linked data structures as subsets of a unified information space, unifying all data models that link data with hashes as instances of IPLD.

Content addressing through hashes has become a widely-used means of connecting data in distributed systems, from the blockchains that run your favorite cryptocurrencies, to the commits that back your code, to the web's content at large. Yet, whilst all of these tools rely on some common

New library: ipfs-provider

\$ npm install ipfs-provider



- Framework-agnostic
- Customizable fallback
- Extracted from our internal web apps
- Available on a growing number of providers

The screenshot shows the npm package page for `ipfs-provider`. The browser address bar displays `https://www.npmjs.com/package/ipfs-provider`. The package name `ipfs-provider` is prominently displayed, along with its version `0.2.1`, which is marked as `Public` and published `9 days ago`. Navigation tabs include `Readme` (highlighted), `4 Dependencies`, `0 Dependents`, and `3 Versions`. Below the package name, it is noted as being `made by Protocol Labs` for a `project IPFS`, with `travis passing` and `dependencies up to date`. The main description states: "Connect to IPFS via an available provider." and "This module tries to connect to IPFS via multiple providers, in order:". Two bullet points describe the providers: `webext` (for browser extensions) and `window.ipfs` (provided by the `IPFS Companion` browser extension). On the right side, there is an `install` section with a terminal snippet `> npm i ipfs-provider`, a `weekly downloads` chart showing `8` downloads, and a table with columns for `version` and `license`, showing `0.2.1` with a `MIT` license. At the bottom, there are links for `open issues` and `pull requests`.



IPFS Camp

Demo App



Try on your own!

**GET THE
APP TEMPLATE
CODE**

bit.ly/2FCAcWp

**Work with those at your table
to complete the code challenges**





IPFS Camp

Coding Challenge #1

connect to the API with `js-ipfs-http-client`



Coding Challenge #1:

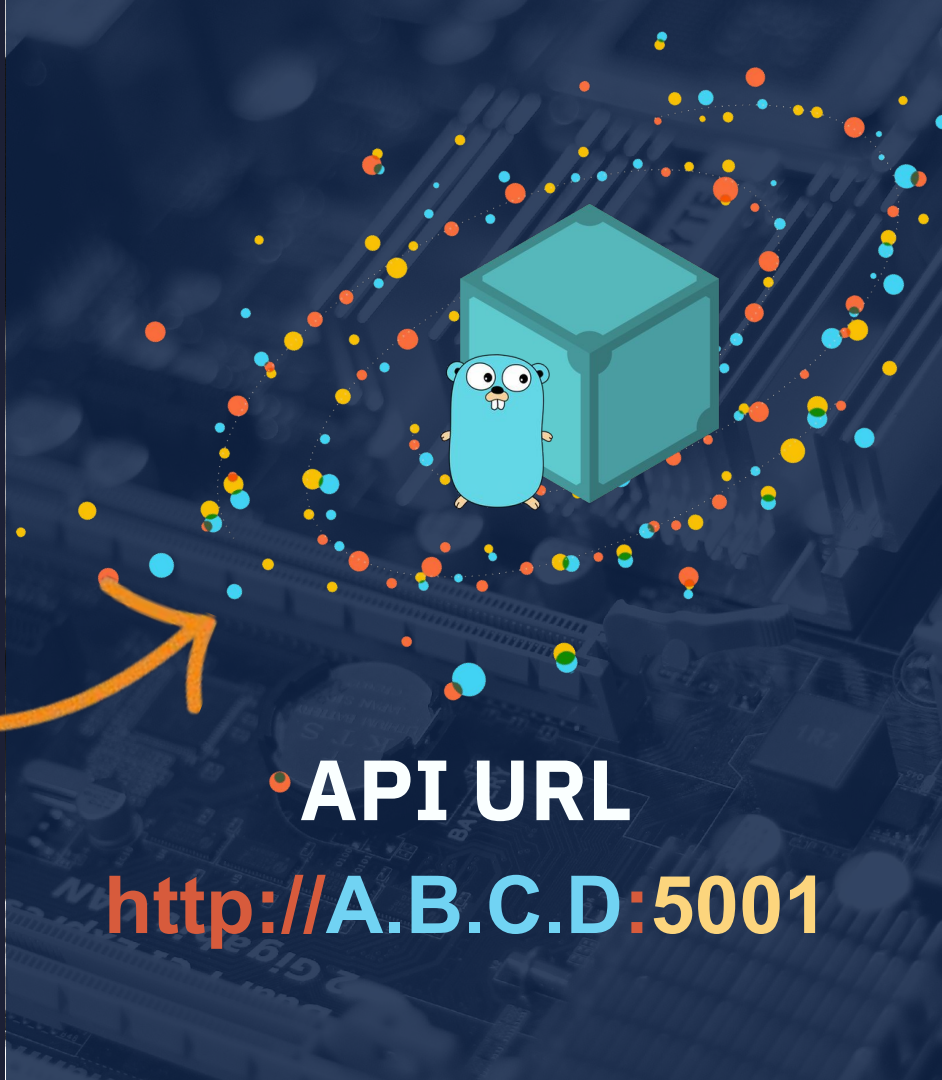
Connect to a remote HTTP API



Resources:

docs.ipfs.io/reference/api/http

github.com/ipfs/js-ipfs-http-client



• **API URL**

http://A.B.C.D:5001



IPFS Camp

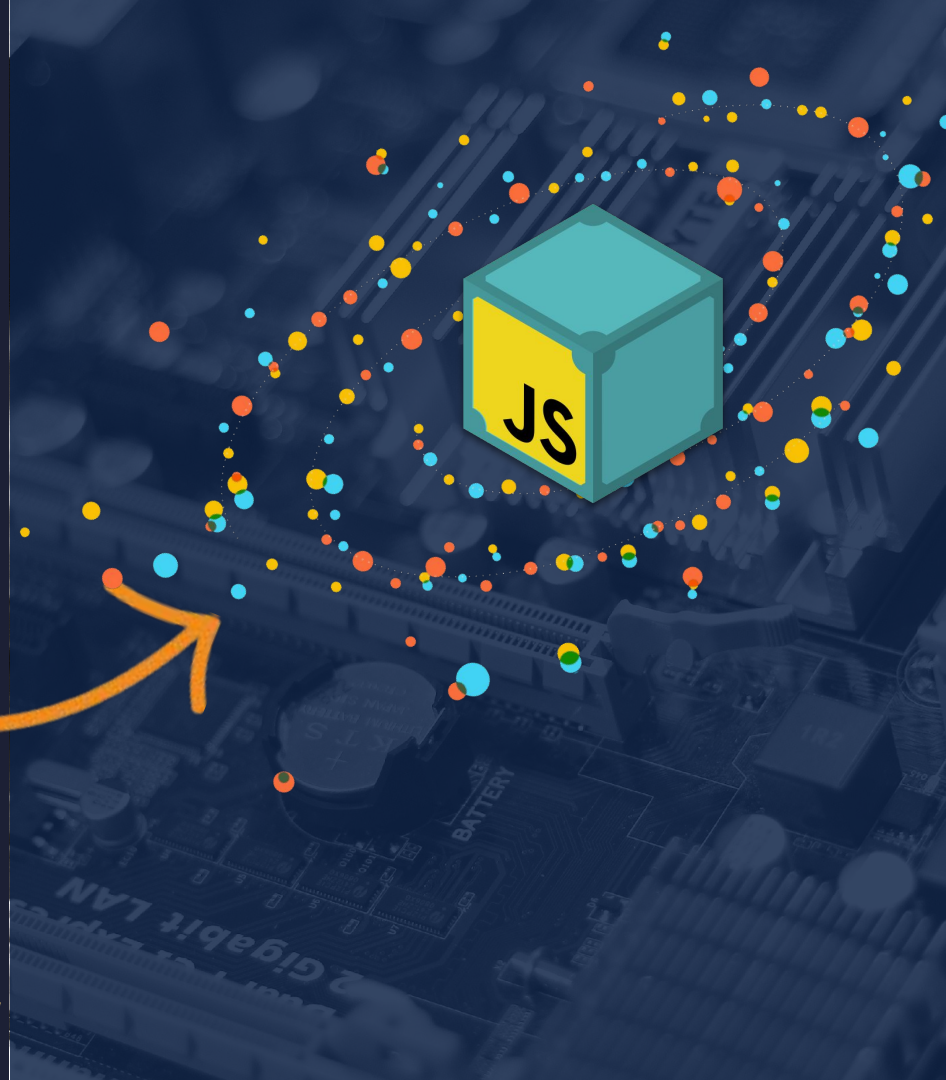
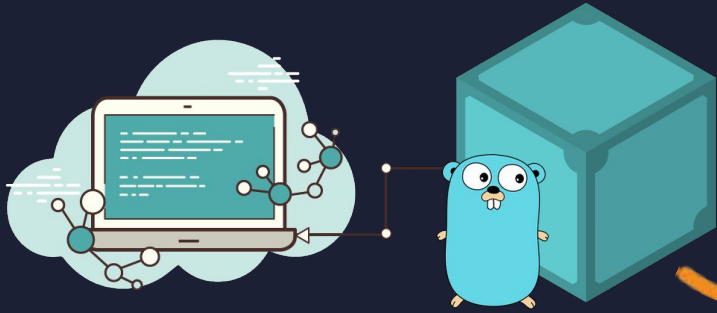
Coding Challenge #2

run js-ipfs in the browser



Coding Challenge #2:

Replace HTTP Client with Embedded JS IPFS



Resources:

github.com/ipfs/js-ipfs#api

github.com/ipfs/js-ipfs#use-in-the-browser

Working with data

- Introduction
- Provisioning
- API
- **Data**
- Naming



Know your APIs:

Use the best API for your data type

Regular files

- `ipfs add`
- `ipfs cat`

Course A: How IPFS deals with Files

Course D: The lifecycle of data in IPFS

Complex data structures

- `ipfs dag put`
 - `ipfs dag get`
- } **IPLD.io**

Posters: IPLD Resolver, IPLD Selectors

Deep Dive: Cross-protocol DAG traversal

Adding files

`ipfs add`

- Introduction
- Provisioning
- API
- Data**
- Naming



Adding files:

IPFS ADD in the command line

```
$ ipfs add file.jpg
```

```
added QmbWqxBEKC3P8tqsKc98xmWNzrzDtRLMiMPL8wBuTGsMnR file.jpg
```

```
$ curl -F file=@file.jpg "http://127.0.0.1:5001/api/v0/add"
```

```
{  
  "Name": "file.jpg",  
  "Hash": "QmbWqxBEKC3P8tqsKc98xmWNzrzDtRLMiMPL8wBuTGsMnR",  
  "Size": "119776"  
}
```

API port



docs.ipfs.io/reference/api



IPFS Camp

Coding Challenge #3

adding a file with js-ipfs



Coding Challenge #3:

Add an avatar **image**, get a **CID**

```
> const result = await ipfs.add(file)
```

```
> result
```

```
[{  
  "path": "file.jpg",  
  "hash": "QmbWqxBEKC3P8tqsKc98xmWNzrzDtRLMiMPL8wBuTGsMnR",  
  "size": "119776"  
}]
```

Relevant JS API docs:

github.com/ipfs/js-ipfs#files

Reading files

`ipfs cat`

- Introduction
- Provisioning
- API
- Data**
- Naming



Reading files:

IPFS CAT in the command line

```
$ ipfs cat QmCID > file.jpg
```

```
$ curl "http://{IP}:5001/api/v0/cat?arg=QmCID" > file.jpg
```


API port

\$ ipfs daemon

Initializing daemon...

go-ipfs version: 0.4.21-

Repo version: 7

System version: amd64/linux

Golang version: go1.12.5

Swarm listening on /ip4/127.0.0.1/tcp/4001

Swarm listening on /ip4/172.17.0.1/tcp/4001

Swarm listening on /ip4/192.168.0.16/tcp/4001

Swarm listening on /p2p-circuit

Swarm announcing /ip4/127.0.0.1/tcp/4001

Swarm announcing /ip4/172.17.0.1/tcp/4001

Swarm announcing /ip4/192.168.0.16/tcp/4001

Swarm announcing /ip6>:::1/tcp/4001

API server listening on /ip4/127.0.0.1/tcp/5001

WebUI: http://0.0.0.0:5001/webui

Gateway (readonly) server listening on /ip4/127.0.0.1/tcp/8080

Daemon is ready

API port

Gateway port

Reading files:

via the HTTP Gateway

The HTTP Gateway works just like `ipfs cat`,
but with the added benefits of HTTP Caching,
directory listing, path addressing:

```
$ curl "http://{IP}:8080/ipfs/QmCID" > file.jpg
```

Gateway port 



IPFS Camp

Coding Challenge #4

reading files with js-ipfs



Coding Challenge #4:

Fetch avatar **image** by **CID**

```
> const data = await ipfs.cat(QmCID)
```

Relevant JS API docs:

github.com/ipfs/js-ipfs#files

Tracking updates by naming things

- Introduction
- Provisioning
- API
- Data
- Naming





Various ways of mapping:

static name → content path

Use your own:

?

→ /ipfs/{cid}

Built-in naming system:

IPNS: /ipns/{libp2p-key}

→ /ipfs/{cid}

Human-readable:

DNSLink: /ipns/{fqdn.tld}

→ /ipfs/{cid}

ENS: /ipns/{name}.eth.link

→ /ipfs/{cid}

IPNS 101:

IPNS: Naming with keys

Publishing API:

- `ipfs name publish {cid}` → `/ipns/{keyid}`
- `ipfs name resolve {keyid}` → `/ipfs/{cid}`

Multiple names:

- `ipfs key gen`
- `ipfs name publish --key {keyname}`

Poster Session & Deep Dives: IPNS,
Revocation/Rotating of IPNS Keys,
Fast IPNS, IPNS link rot



IPFS Camp

Coding Challenge #5

updating a mutable pointer
with IPNS in js-ipfs



Coding Challenge #5:

Publish data to IPNS

```
> await ipfs.name.publish(cid)
```

Relevant JS API docs:

github.com/ipfs/js-ipfs#name



IPFS Camp

Done!

Let's recap...



Let's recap:

What we've learned

- Using IPFS as a building block in your application
 - **High Level Patterns** (API+Service or Embedded)
 - Decision space: go-ipfs vs js-ipfs, local vs remote, API types
- Coding Challenges
 - Set up **js-ipfs-http-client** to connect to remote **go-ipfs**
 - Switch from js-ipfs-http-client to **embedded js-ipfs**
 - Work with **adding** and **getting** files
 - Publish **mutable pointer** to immutable content path

YOU HAVE COMPLETED 

DEVELOPING APPS WITH IPFS APIs

CORE COURSE C



IPFS Camp

That's all, folks.
Thank you!

Where to go tomorrow

Elective E: Deploying IPFS Infrastructure

Elective B: IPFS Cluster

Elective D: Building DApps with Textile

Deep Dives: IPNS*, Fast IPNS,
Revocation/Rotating of IPNS Keys



**Please make room
for the next group
:^(**