

WIFI: Internet Backup - B
Pass: ShareWithIPFS

IPFS Cluster

The Workshop

Hector Sanjuan	Michelle Hertzfeld	Adrian Lanzafame
		
@hsanjuan	@meiqimichelle	@lanzafame

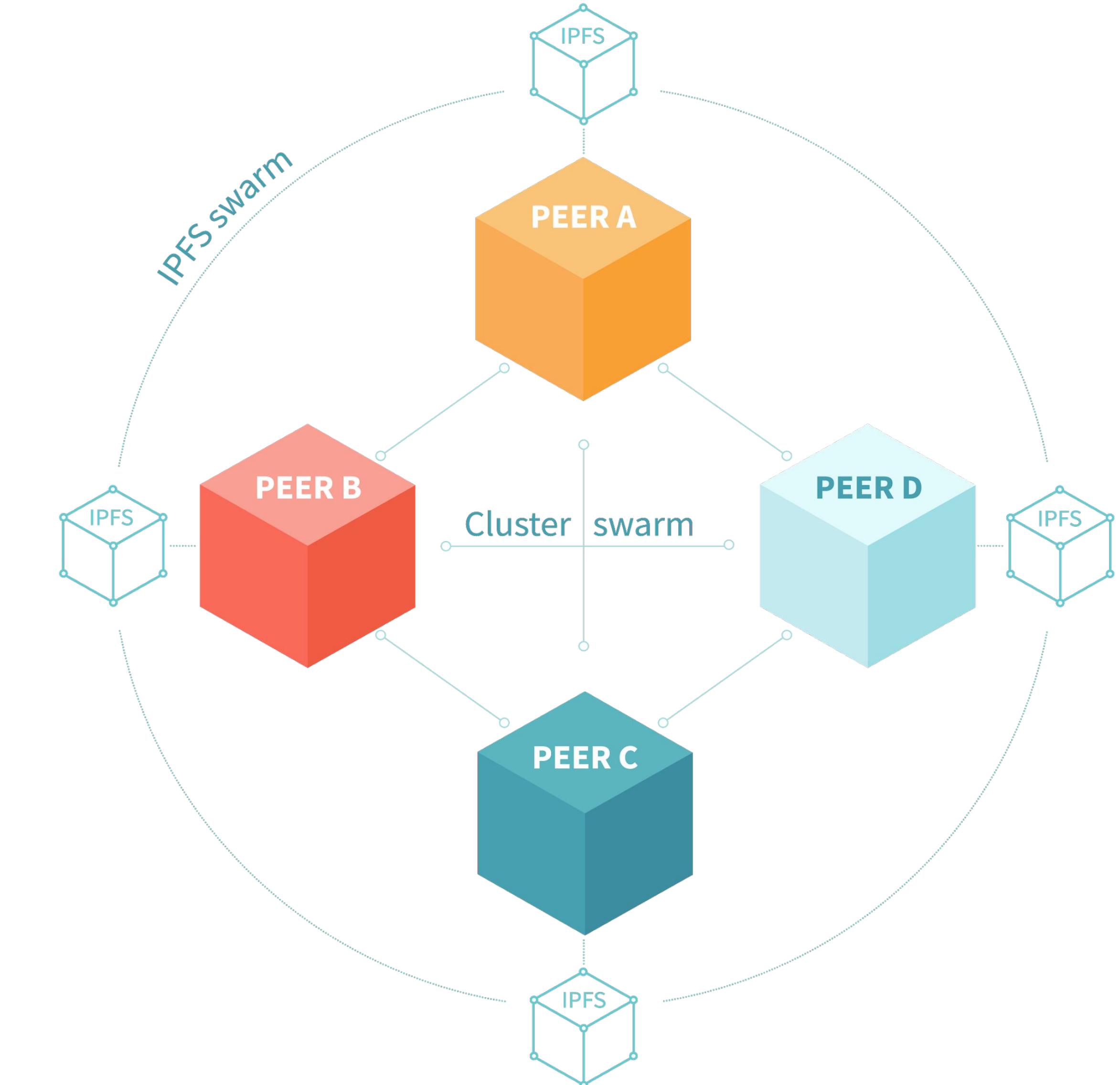
These slides are provided for historical reasons, but parts are outdated for IPFS Cluster versions newer than 0.11.0-rc5.



Welcome!

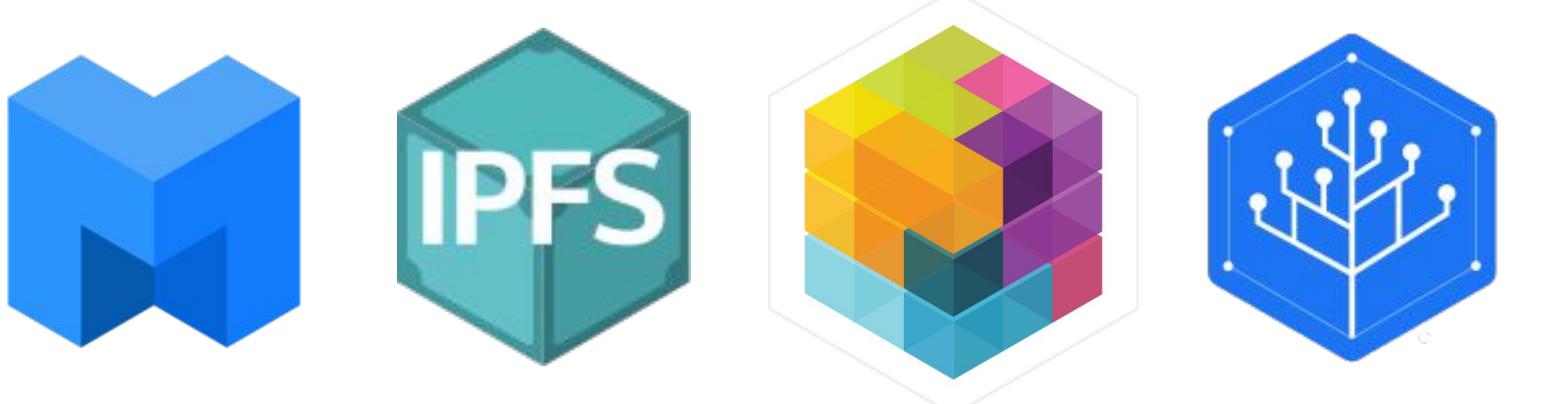
IPFS Cluster

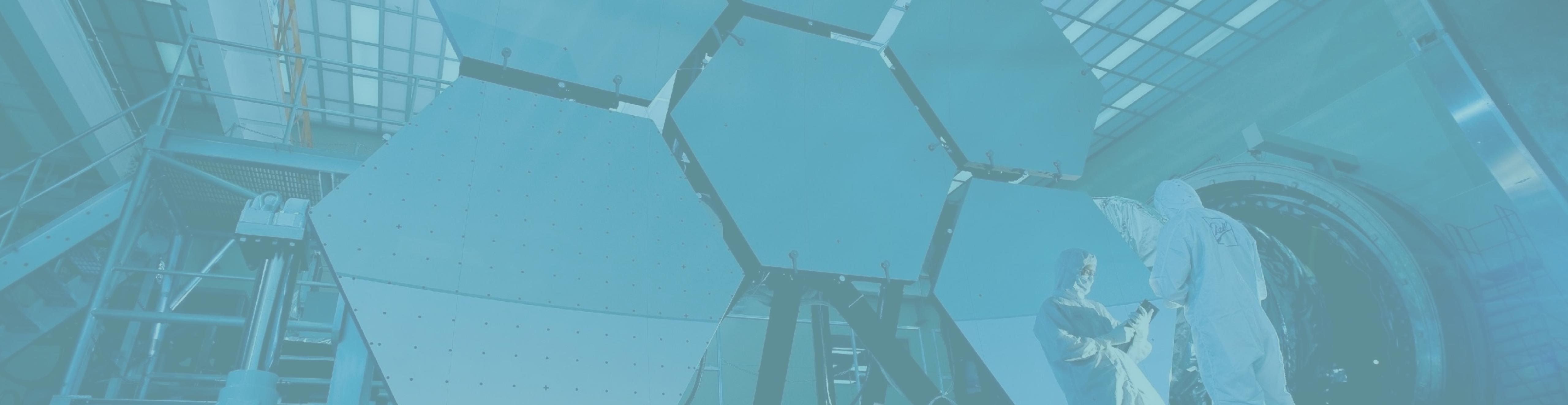
- Manage a global pinset across multiple IPFS daemons
- Track the status of every pin on every peer
- Setup flexible collaborative Clusters to replicate content



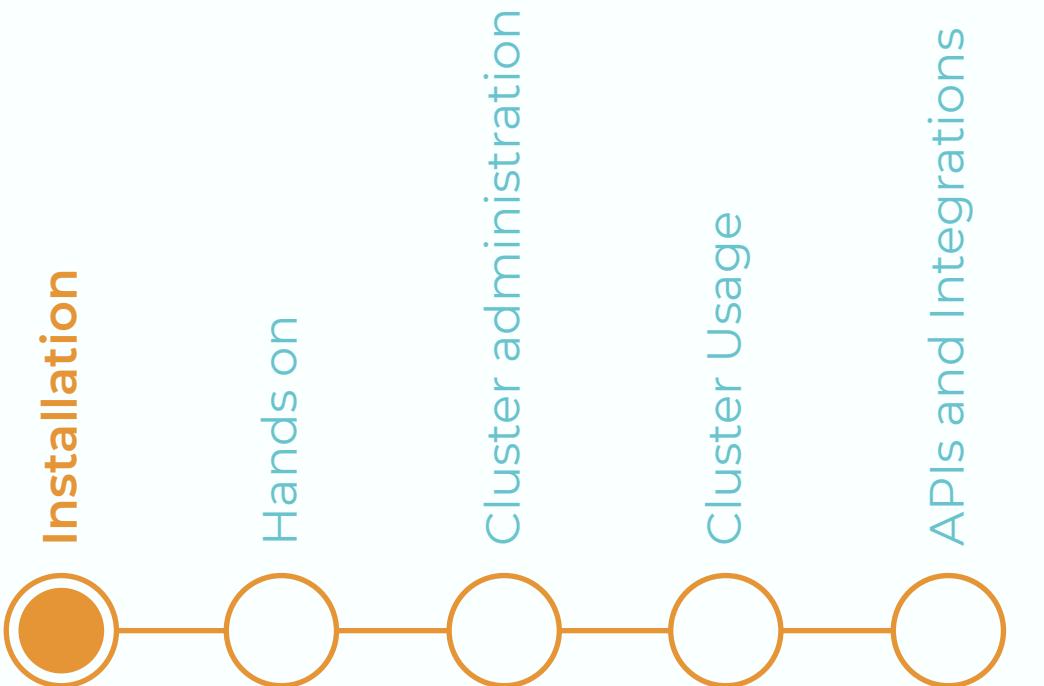
Agenda

- 1. Installation
- 2. Hands-on
- 3. Cluster administration:
 - ... using Raft
 - ... using Crdt
- 4. Cluster usage
- 5. Cluster APIs and Integrations





Installation



Installation

1. Go to: <https://github.com/ipfs/camp/>
2. Click on CORE_AND_ELECTIVE_COURSES
3. Click on ELECTIVE_COURSE_B
4. And scroll to the **#installation-instructions** in the README.
5. Follow them very closely

Installation check

1. "secret" is set to "" in service.json
2. "trusted_peers" is set to ["*"] in service.json





Hands on IPFS Cluster



Start your cluster peer

```
$ ./ipfs-cluster-service daemon --consensus crdt --bootstrap /dnsaddr/workshop.cluster.ipfs.io
```

ipfs-cluster-ctl

```
$ ./ipfs-cluster-ctl --help daemon
$ ./ipfs-cluster-ctl [--enc=json] id
$ ./ipfs-cluster-ctl peers ls
$ ./ipfs-cluster-ctl pin ls
$ ./ipfs-cluster-ctl status
$ ./ipfs-cluster-ctl status --filter pinning
$ ./ipfs-cluster-ctl status $cid [--local]
$ ./ipfs-cluster-ctl health metrics freespace
$ ./ipfs-cluster-ctl pin add <hash>
$ ./ipfs-cluster-ctl pin rm <hash>
$ ./ipfs-cluster-ctl add somefile.txt
```

Cluster administration

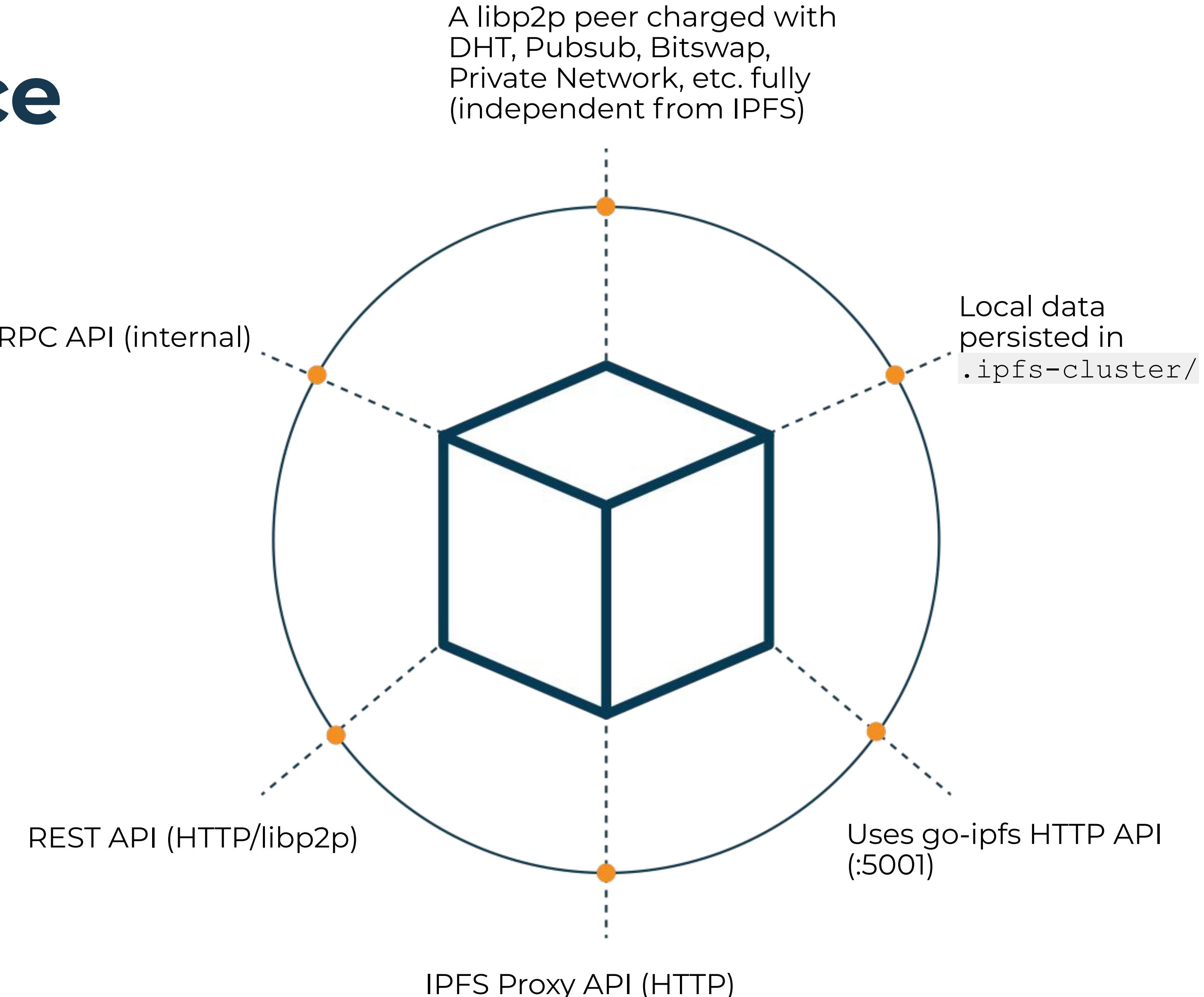


ipfs-cluster-service init

- Creates:
 - `~/.ipfs-cluster/service.json`: the main config.
Should usually be the SAME in every peer.
 - `~/.ipfs-cluster/identity.json`: peer ID and private key.
Should be different in every peer.

The Cluster Peer: ipfs-cluster-service

- ipfs-cluster-service **init**
- ipfs-cluster-service **daemon**
 - --consensus {crdt|raft}
- ipfs-cluster-service **state**
 - **export** --consensus {crdt|raft}
 - **import** --consensus {crdt|raft}
 - **cleanup** --consensus {crdt|raft}



The configuration file

```
{  
  "cluster": {  
    "secret": "THE SAME SECRET FOR EVERYONE",  
    <other options>...  
  },  
  "component_type": {  
    "component_implementation": {  
      <options>  
    }  
  }  
}
```

- Multiple APIs used. Only one pin_tracker, consensus, informer...
- Actual implementation can be selected with flags to ipfs-cluster daemon

```
{  
  "cluster": {...},  
  "consensus": {  
    "crdt": {...}  
    "raft": {...},  
  },  
  "api": {  
    "ipfsproxy": {...},  
    "restapi": {...}  
  },  
  "ipfs_connector": {  
    "ipfshttp": {...}  
  },  
  "pin_tracker": {  
    "maptracker": {...},  
    "stateless": {...}  
  },  
  "monitor": {  
    "pubsubmon": {...}  
  },  
  "informer": {  
    "disk": {...},  
    "numpin": {...}  
  },  
  "observations": {  
    "metrics": {...},  
    "tracing": {...}  
  }  
}
```

The configuration file

```
{  
  "cluster": {  
    "secret": "THE SAME SECRET FOR EVERYONE",  
    <other options>...  
  },  
  "component_type": {  
    "component_implementation": {  
      <options>  
    }  
  }  
}
```

ipfs-cluster-service daemon --consensus raft

- **Peer** additions and removals require a raft commit
- **Pin** additions and removals require a raft commit
- Raft commits requires quorum and are redirected to raft leader which executes them (slow pin ingest)
- Must trust all peers! (every peer can do everything)
- Suitable for stable clusters, long-lived peers. Raft is very mature (same Raft implementation as Consul)

ipfs-cluster-service daemon --consensus crdt

- Peers can join and leave freely
- “trusted_peers”: only operations submitted by trusted peers are applied
- Pubsub to distribute the new heads of an always growing DAG
- Experimental, super flexible, fast ingestion
- Suited for collaborative clusters (untrusted peers cannot affect your peer). “Trusted peers” vs “followers”

Starting and Bootstrapping

- IPFS runs public bootstrappers for the IPFS Network
- Each cluster is its own Cluster swarm, therefore at least one of the cluster peers needs to act as “bootstrapper” for others
- Bootstrap is necessary on the first run of a cluster peer
- Subsequent runs remember “peers” in the cluster

Bootstrapping (Raft)

- Any new peer **must** start with:

```
ipfs-cluster-service daemon --bootstrap <existing peer>
```

- “Hello <existing peer>, please add me to the Raft peerset”.
- Subsequent starts should just use `ipfs-cluster-service daemon`
- The **peerstore** file with the raft peers addresses is automatically saved and the peer IDs in the cluster are part of the Raft internal state, also persisted. The **peerstore** can be however pre-filled by the user with known peer addresses.

Bootstrapping (CRDT)

- You can either start with:

```
ipfs-cluster-service daemon \  
  --bootstrap <existing peer> --consensus crdt
```

- or fill-in the `.ipfs-cluster/peerstore` file with some known peer multiaddresses
- When the peer is stopped known peer addresses will be added to the peerstore file for future usage
- Trusted peers are prioritized when bootstrapping

A healthy Cluster peer...

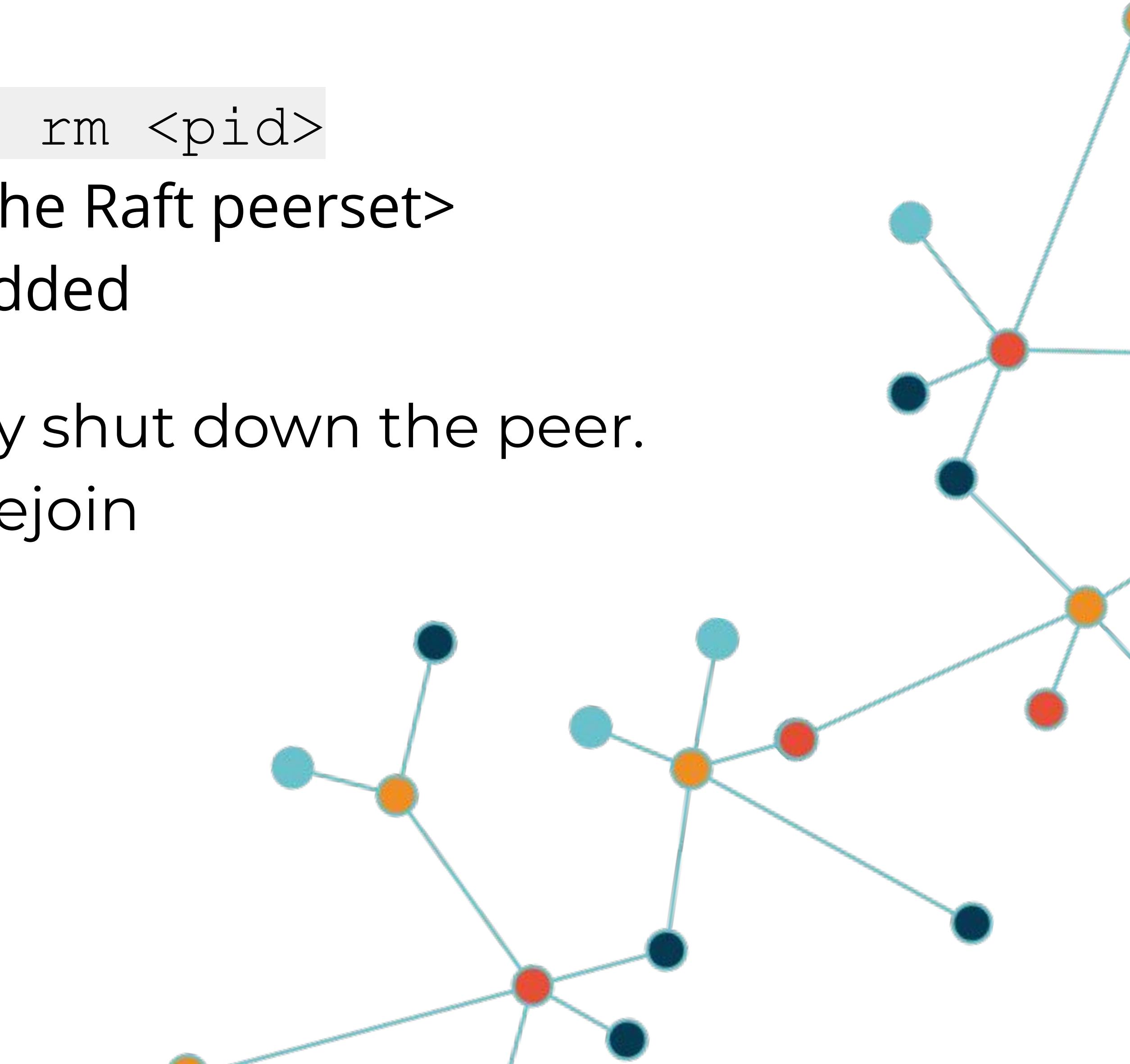
- Has a working ipfs daemon by its side
- Can connect to other peers in the Cluster:
 - Knows their Peer IDs and can learn their addresses via peerstore file or DHT query
- In Raft: knows a Raft leader or the cluster can elect one (>50% are online)
- In CRDT: receives pubsub updates from trusted peers (either by being connected to them or by having the relayed)

Adding more peers...

- Same as “bootstrapping”
- Re-using a peer for a different cluster:
 - In Raft: state is “cleaned” when bootstrapping (it will be synced from the Cluster)
 - In CRDTs: you probably want to run `ipfs-cluster-service state cleanup --consensus crdt` when joining a “new” cluster.

Removing peers...

- In Raft: `ipfs-cluster-ctl peers rm <pid>`
 - <Please remove this Peer ID from the Raft peerset>
 - Peer needs to bootstrap to be re-added
- In CRDT: remove is a no-op. Simply shut down the peer.
 - Start the peer normally for it to rejoin



Disaster recovery

- Shutdown all peers and choose one
 - ipfs-cluster-service **state export** \
 -f state.json --consensus <crdt/raft>
 - ipfs-cluster-service **state cleanup** \
 --consensus <crdt/raft>
 - ipfs-cluster-service **state import** state.json \
 --consensus <crdt/raft>
- Launch this single peer with the imported state.
- Cleanup and bootstrap the rest of peers to it.

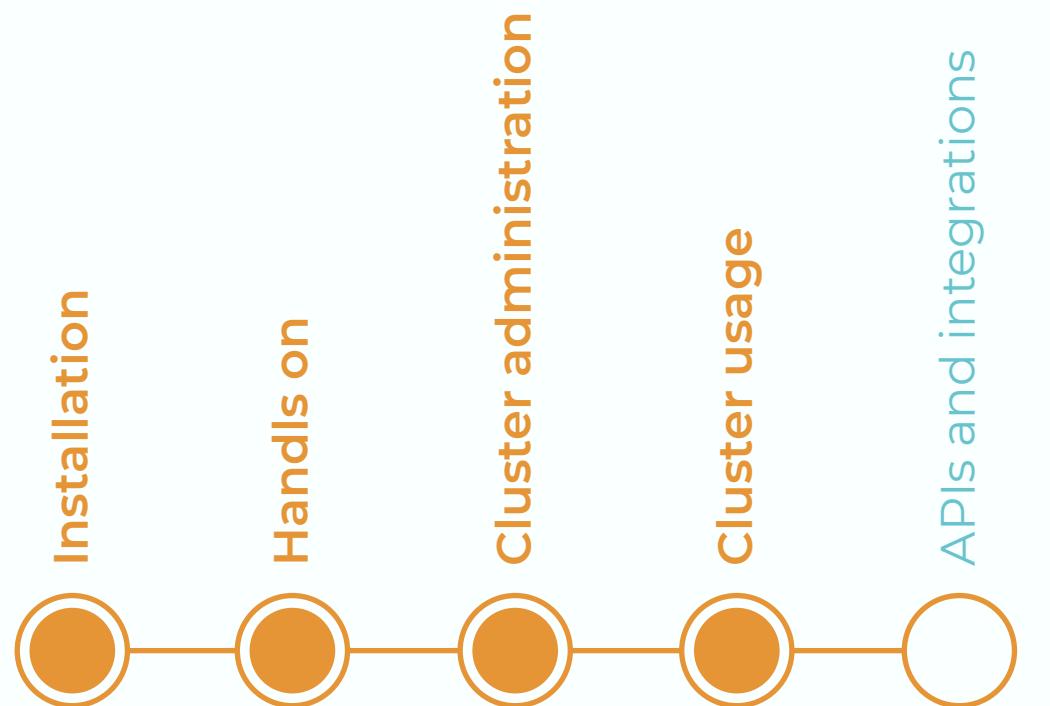
Deployment automations

- Kubernetes:
 - With Kustomize
 - Most useful for AWS, GCP, Azure
- Ansible roles:
 - Most useful for regular cloud providers: Hetzner, Digital Ocean...
- Docker containers / compose
 - Manual setups and others





Cluster usage



ipfs-cluster-ctl

- ipfs-cluster-ctl is a CLI client to the HTTP REST API
- 99% feature-parity with the API
- ipfs-cluster-ctl **id**
- ipfs-cluster-ctl **peers ls**
- ipfs-cluster-ctl **pin add** --rmin X --rmax Y --name abc...
- ipfs-cluster-ctl **add** --recursive --name abc myfile.txt
- ipfs-cluster-ctl **health graph**
- ipfs-cluster-ctl **health metrics** freespace

ipfs-cluster-ctl pin ls [cid]

- Displays "allocations"
- aka the Cluster Pinset
- aka the global state
- Every peer keeps their copy of the global state which is the same everywhere.

```
{  
  "replication_factor_min": 1,  
  "replication_factor_max": 3,  
  "name": "a name",  
  "shard_size": 0,  
  "user_allocations": null,  
  "metadata": null,  
  "cid": {  
    "/": "QmT8n..."  
  },  
  "type": 2,  
  "allocations": [<peerID1,  
  peerID2],  
  "max_depth": -1,  
  "reference": null  
}
```

ipfs-cluster-ctl status [cid]

- Displays "pin status"
- aka the "local state"...
- ...by querying all peers for their pin statuses and merging them.
- A status query connects to all peers to retrieve the actual status of a pin in each of them.
- We call this a "broadcast" operation

```
{  
  "cid": {  
    "/": "QmcD..."  
  },  
  "peer_map": {  
    "<peerID1>": {  
      "cid": { "/" : "QmcD..." },  
      "peer": "<peerID1>",  
      "peername": "cluster2",  
      "status": "pinning",  
      "timestamp": "2019-06-10...",  
      "error": ""  
    },  
    "<peerID2>": {  
      "cid": { "/" : "QmcD..." },  
      "peer": "<peerID2>",  
      "peername": "cluster3",  
      "status": "pinned",  
      "timestamp": "2019-06-10T...",  
      "error": ""  
    }  
  }  
}
```

ipfs-cluster-ctl sync [cid]

- The "pin status" are cached for a few minutes.
- `ipfs-cluster-ctl sync` basically runs `ipfs pin ls` and updates the "pin state" to match that of ipfs if it has changed.
- Run regularly by cluster automatically.

```
{  
  "cid": {  
    "/": "QmcD..."  
  },  
  "peer_map": {  
    "<peerID1>": {  
      "cid": { "/" : "QmcD..." },  
      "peer": "<peerID1>",  
      "peername": "cluster2",  
      "status": "pinning",  
      "timestamp": "2019-06-10...",  
      "error": ""  
    },  
    "<peerID2>": {  
      "cid": { "/" : "QmcD..." },  
      "peer": "<peerID2>",  
      "peername": "cluster3",  
      "status": "pinned",  
      "timestamp": "2019-06-10T...",  
      "error": ""  
    }  
  }  
}
```

ipfs-cluster-ctl recover [cid]

- ipfs-cluster-ctl recover attempts to fix pins in "pin error" or "unpin error" status.
- Re-triggers the operation on IPFS

```
{  
  "cid": {  
    "/": "QmcD..."  
  },  
  "peer_map": {  
    "<peerID1>": {  
      "cid": { "/": "QmcD..." },  
      "peer": "<peerID1>",  
      "peername": "cluster2",  
      "status": "pin_error",  
      "timestamp": "2019-06-10...",  
      "error": "Connection refused"  
    }  
  }  
}
```

APIs and integrations



REST API

- Used by `ipfs-cluster-ctl`
- Runs on port :9094 by default
- Also runs as a libp2p-service (either as part of the cluster host or with an additional one)
 - Free encrypted channel!
- Supports custom CORS, HTTPs and Basic Authentication (out-of-the-box)



IPFS Proxy API

- Runs on port :9095 by default
- Behaves like IPFS. Forwards all requests to IPFS daemon except:
 - /api/v0/pin/add, /api/v0/pin/rm: trigger cluster pin/unpin with default options
 - /api/v0/pin/ls: lists cluster pins
 - /api/v0/pin/update: updates cluster pin
 - /api/v0/repo/stat: shows repo/stat aggregates for the full cluster
 - /api/v0/add: adds to cluster.
- Cluster as "drop in"

API Clients and integrations

- Go client (as part of IPFS Cluster, used by `ipfs-cluster-ctl`)
- Javascript client (by @vasa-develop)
- pinbot-irc: Uses go client to pin things on Cluster from an IRC channel
- ipfs-hubot: Uses the IPFS Proxy API and js-ipfs-api to pin things in Cluster
- [cluster-labs/horizon](#): A Web UI using the javascript client and REST API.

Thank You!

Hector Sanjuan



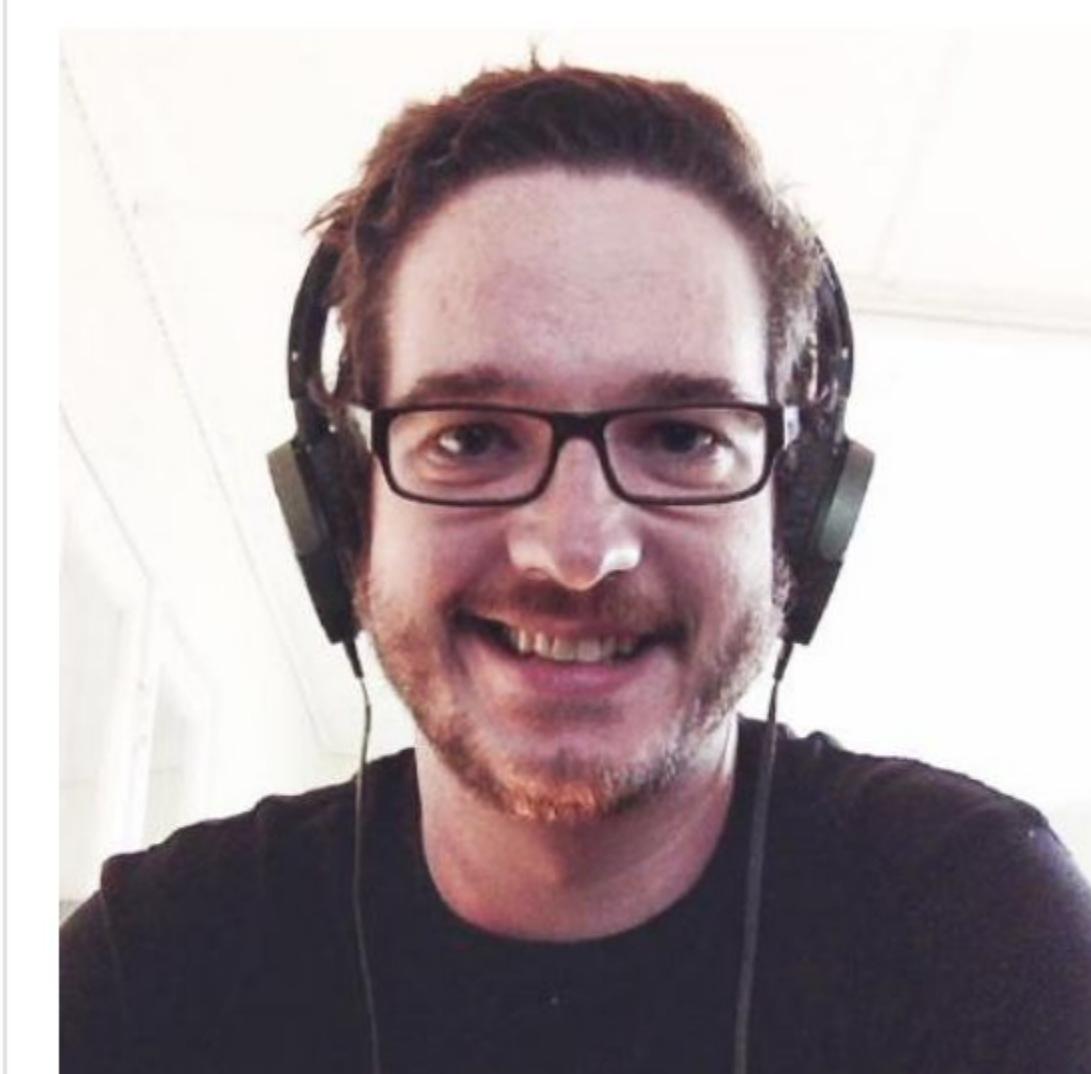
@hsanjuan

Michelle Hertzfeld



@meiqimichelle

Adrian Lanzafame



@lanzafame