Lesson 12 - Program development and decentralised storage

Cross Program Invocation

Cross Program Invocations enable for programs to call instructions on other programs, allowing for the composibility of Solana programs.
Calling between programs is achieved by one program invoking an instruction of the other.
Note

- Program Derived Addresess (PDAs) can be used by one program to sign for Cross Program Invocations of instructions on another program.
- CPI calls are currently limited to a depth of 4

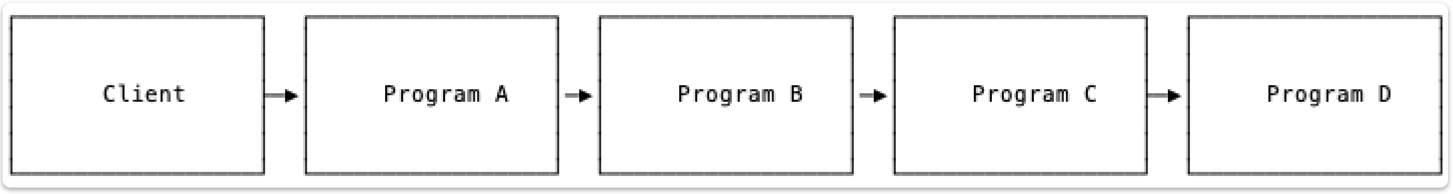
When a user interacts with the Solana blockchain, they can push several instructions in an array and send all of them as a single transaction. The benefit of this is that transactions are atomic, meaning that if any of the instructions fail, the entire operation rolls back and it's like nothing ever happened.

Instructions can also delegate to other instructions either within the same program or outside of the current program. The latter is called Cross-Program Invocations (CPI) and the signers of the current instruction are automatically passed along to the nested instructions.

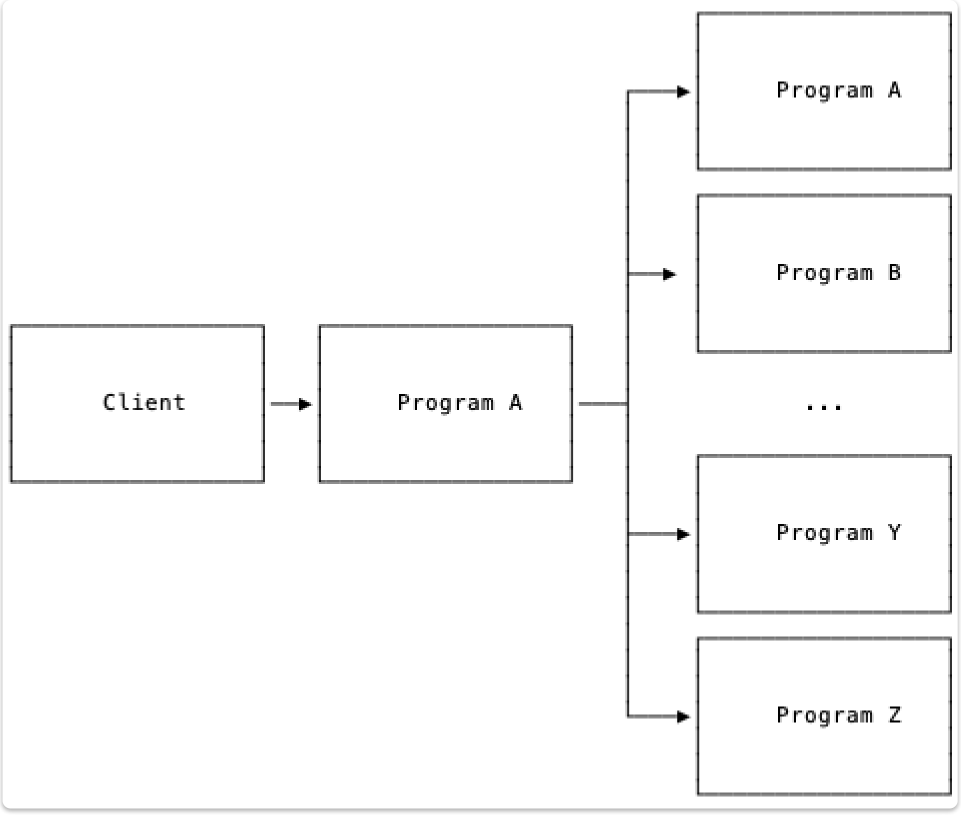
No matter how many instructions and nested instructions exists inside a transaction, it will always be atomic — i.e. it's all or nothing.

When including a signed account in a program call, in all CPIs including that account made by that program inside the current instruction, the account will also be signed, i.e. the signature is extended to the CPIs.
When a program calls invoke_signed, the runtime uses the given seeds and the program id of the calling program to recreate the PDA and if it matches one of the given accounts inside invoke_signed's arguments, that account's signed property will be set to true.

Due to the depth limitations logic has to be designed in such way that if depth execeed 4, it has to be split and intermediary state stored between seperate client calls.



The limit to how many separate programs can be invoked none sequentially, is limited by the cost of computation.


Client
Program A
Program A
Program B
...
Program Y
Program Z

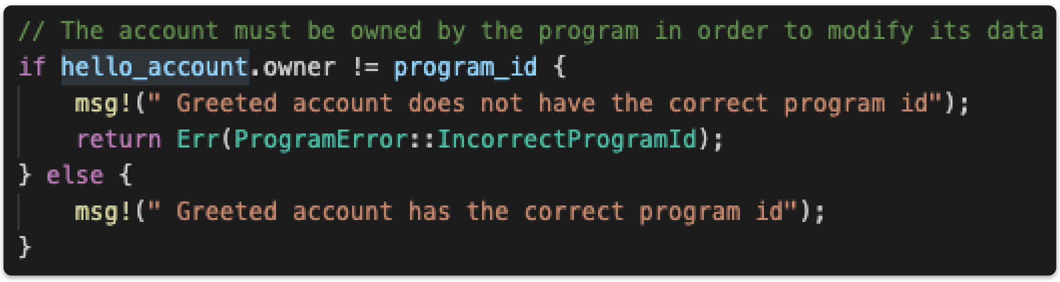
Authority vs Ownership

Ownership

This ownership doesn't refer to Rust ownership, but rather it is an internal relationship between Solana's accounts.

On Solana only the owner can modify the state. Cloudbreak, Solana's account database maintains mapping between public keys and accounts which they own.
The System Program assigns ownership and initialises account data.

This is what this check does. It looks up in Cloudbreak whether its account address maps to the one of the provided addresses:



```rust
// The account must be owned by the program in order to modify its data
if hello_account.owner != program_id {
    msg!(" Greeted account does not have the correct program id");
    return Err(ProgramError::IncorrectProgramId);
} else {
    msg!(" Greeted account has the correct program id");
}
```

This does not mean that the program necessarily would modify an account. Additional checks are applied to ensure client requests are valid.

Authority

User space access control can be implemented by checking provided signatures against additional filters.

A Public key (or a group of them) can specified in the account to hold additional privileges. These keys with extended functionality are referred to as the authority, admin, manager and sometimes confusingly as owner. Multi signatures as well as layered and finely grained access can likewise be achieved

An account number that stores a `u64` value can check whether the signature of the caller matches the one on the provided account.

```
struct Number{
        authority: Pubkey
        value: u64
}
```

In code the check could be implemented thus:

```
...

let accounts_iter = &mut accounts.iter();
let number_account = next_account_info(accounts_iter)?;
if caller.is_signer && *caller.key == number_account.authority{
        number.value = new_value; // modify state
}

...
```

Interestingly if the account is a PDA and is to have only a single account as the authority, an alternative verification scheme can be employed.

In the program, the PDA can be re derived from the callers public key to see if it matches the account provided.

A simplified account would look like this:

```
struct Number{
        value: u64
}
```

Whilst in the program logic:

```
...

let accounts_iter = &mut accounts.iter();
let number_account = next_account_info(accounts_iter)?;

let number_pda = Pubkey::create_with_seed(
caller.key,
&seed,
&program_id)
.unwrap();
```

```
if caller.is_signer && *number_account.key == number_pda{
        number.value = new_value; // modify state
}

...
```

This tradeoff would result in lower cost per account but a slightly higher cost of computation

---

Examples Program Flow

The usual flow from receiving the instruction to reporting `Ok(())`.

1. Program entry
2. Extracting instruction
3. Access checks
4. State change

Most of the high level logic happens in the `processor.rs`.

Here is an example of how a call to mint token amount would get handled on the SPL token program. source code

Program entry

All the parameters (accounts, instruction data, signatures) for this program call are passed to the entry point.

The processor is where the main logic occurs, though much of it is implemented in other modules.

It is called immediately after the program is invoked.

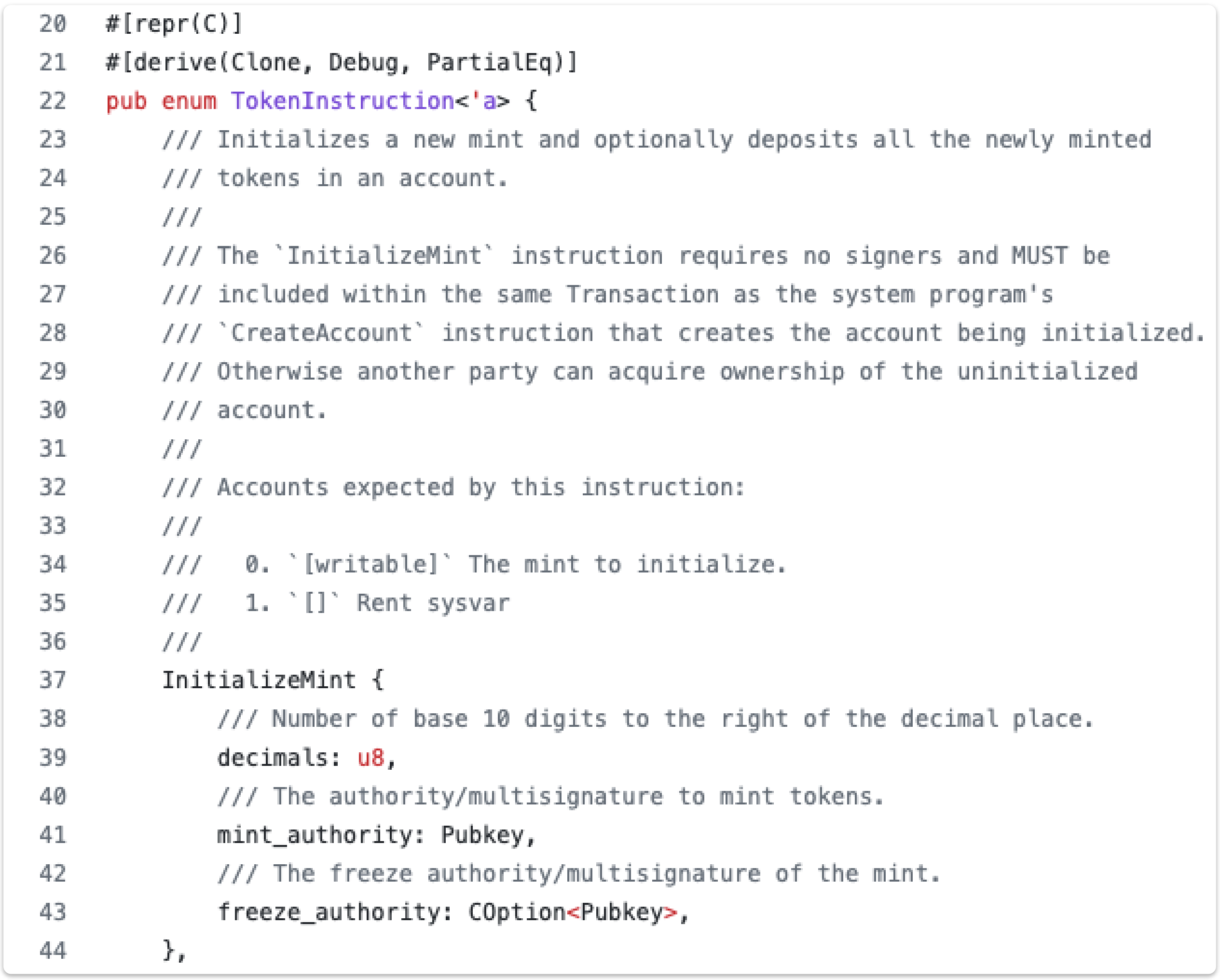
```rust
entrypoint!(process_instruction);
fn process_instruction(
    program_id: &Pubkey,
    accounts: &[AccountInfo],
    instruction_data: &[u8],
) -> ProgramResult {
    if let Err(error) = Processor::process(program_id, accounts, instruction_data) {
        // catch the error so we can print it
        error.print::<TokenError>();
        return Err(error);
    }
    Ok(())
}
```

The first thing the processor does is unpacking the instruction and matching it against associated function. On line 841 of `processor.rs` is the function `process`:

```rust
840     /// Processes an [Instruction](enum.Instruction.html).
841     pub fn process(program_id: &Pubkey, accounts: &[AccountInfo], input: &[u8]) -> ProgramResult {
842         let instruction = TokenInstruction::unpack(input)?;
843 
844         match instruction {
845             TokenInstruction::InitializeMint {
846                 decimals,
847                 mint_authority,
848                 freeze_authority,
849             } => {
850                 msg!("Instruction: InitializeMint");
851                 Self::process_initialize_mint(accounts, decimals, mint_authority, freeze_authority)
852             }
```

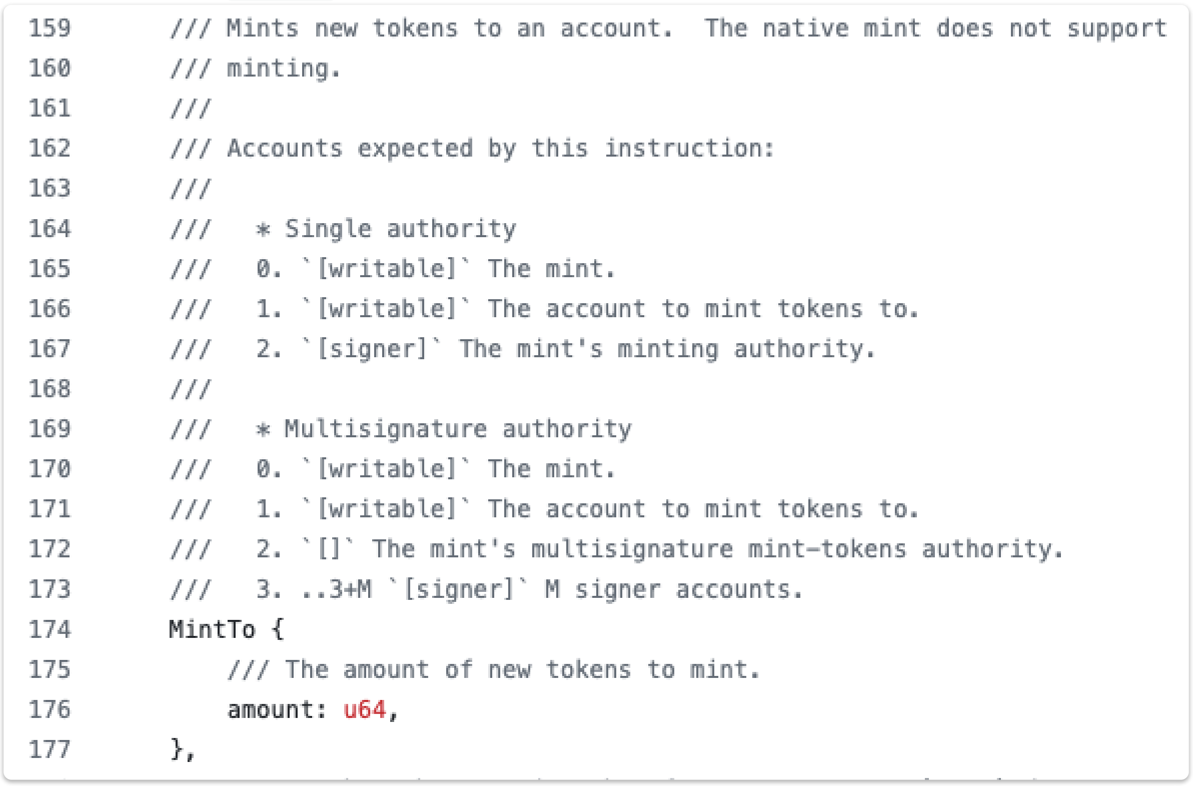
Extracting instruction

In file `instruction.rs` at line 22 is the start of the struct describing all of the possible instructions that implement SPL token functionality.



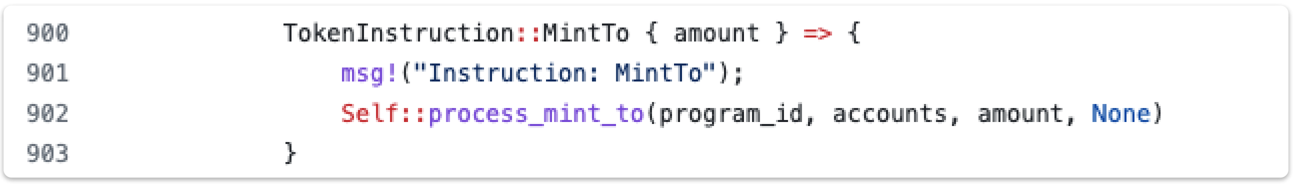
```
20  #[repr(C)]
21  #[derive(Clone, Debug, PartialEq)]
22  pub enum TokenInstruction<'a> {
23      /// Initializes a new mint and optionally deposits all the newly minted
24      /// tokens in an account.
25      ///
26      /// The `InitializeMint` instruction requires no signers and MUST be
27      /// included within the same Transaction as the system program's
28      /// `CreateAccount` instruction that creates the account being initialized.
29      /// Otherwise another party can acquire ownership of the uninitialized
30      /// account.
31      ///
32      /// Accounts expected by this instruction:
33      ///
34      ///   0. `[writable]` The mint to initialize.
35      ///   1. `[]` Rent sysvar
36      ///
37      InitializeMint {
38          /// Number of base 10 digits to the right of the decimal place.
39          decimals: u8,
40          /// The authority/multisignature to mint tokens.
41          mint_authority: Pubkey,
42          /// The freeze authority/multisignature of the mint.
43          freeze_authority: COption<Pubkey>,
44      },
```

And at line 174 `MintTo` is described.



```
159     /// Mints new tokens to an account.  The native mint does not support
160     /// minting.
161     ///
162     /// Accounts expected by this instruction:
163     ///
164     ///   * Single authority
165     ///   0. `[writable]` The mint.
166     ///   1. `[writable]` The account to mint tokens to.
167     ///   2. `[signer]` The mint's minting authority.
168     ///
169     ///   * Multisignature authority
170     ///   0. `[writable]` The mint.
171     ///   1. `[writable]` The account to mint tokens to.
172     ///   2. `[]` The mint's multisignature mint-tokens authority.
173     ///   3. ..3+M `[signer]` M signer accounts.
174     MintTo {
175         /// The amount of new tokens to mint.
176         amount: u64,
177     },
```

Based on the unpacking of the instruction, data processor knows which function to call.



```rust
            TokenInstruction::MintTo { amount } => {
                msg!("Instruction: MintTo");
                Self::process_mint_to(program_id, accounts, amount, None)
            }
```

This is the function in question and the first step is the sequential unpacking of the accounts.

```rust
    /// Processes a [MintTo](enum.TokenInstruction.html) instruction.
    pub fn process_mint_to(
        program_id: &Pubkey,
        accounts: &[AccountInfo],
        amount: u64,
        expected_decimals: Option<u8>,
    ) -> ProgramResult {
        let account_info_iter = &mut accounts.iter();
        let mint_info = next_account_info(account_info_iter)?;
        let destination_account_info = next_account_info(account_info_iter)?;
        let owner_info = next_account_info(account_info_iter)?;
```

Followed by several sanity checks.

```rust
        if destination_account.is_native() {
            return Err(TokenError::NativeNotSupported.into());
        }
        if !Self::cmp_pubkeys(mint_info.key, &destination_account.mint) {
            return Err(TokenError::MintMismatch.into());
        }

        let mut mint = Mint::unpack(&mint_info.data.borrow())?;
        if let Some(expected_decimals) = expected_decimals {
            if expected_decimals != mint.decimals {
                return Err(TokenError::MintDecimalsMismatch.into());
            }
        }
```

Access checks

On line 546 of the `processor.rs` struct method `validate_owner` is called to check whether the provided signatures gain access to restricted area.

```
546         match mint.mint_authority {
547             COption::Some(mint_authority) => Self::validate_owner(
548                 program_id,
549                 &mint_authority,
550                 owner_info,
551                 account_info_iter.as_slice(),
552             )?,
553             COption::None => return Err(TokenError::FixedSupply.into()),
554         }
555
```

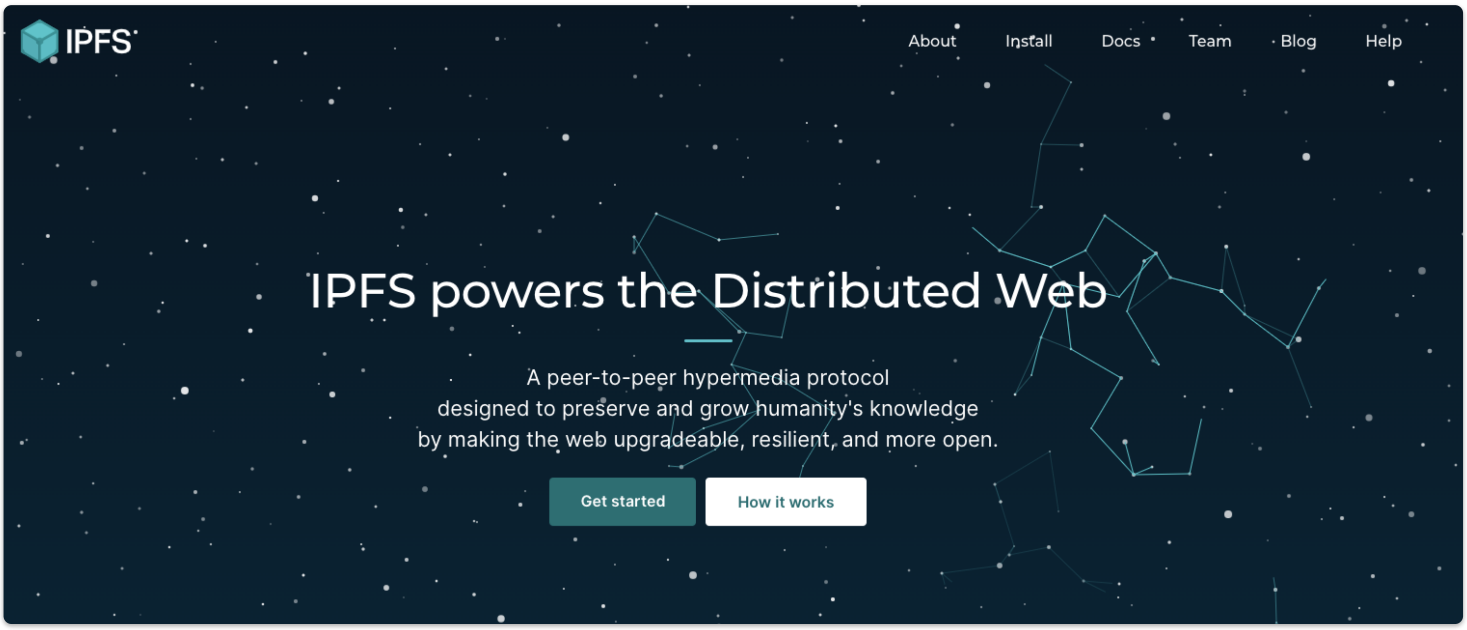
State change

At the end of the `process_to_mint()` the state of the `Mint` account and `Account` account is changed.

```
571         Account::pack(
572             destination_account,
573             &mut destination_account_info.data.borrow_mut(),
574         )?;
575         Mint::pack(mint, &mut mint_info.data.borrow_mut())?;
576
577         Ok(())
578     }
```

Decentralised Storage

IPFS



IPFS powers the Distributed Web

A peer-to-peer hypermedia protocol
designed to preserve and grow humanity's knowledge
by making the web upgradeable, resilient, and more open.

IPFS is a distributed system for storing and accessing files, websites, applications, and data.

Concepts

IPFS is based on 3 concepts

1. Unique identification via content addressing
2. Content linking via directed acyclic graphs (DAGs)
3. Content discovery via distributed hash tables (DHTs)

Content addressing

Instead of being location-based, IPFS addresses a file by *what's in it*, or by its *content*.
The content identifier above is a *cryptographic hash* of the content at that address. The hash is unique to the content that it came from.
Because the address of a file in IPFS is created from the content itself, links in IPFS can't be changed.

A *content identifier*, or CID, is a label used to point to material in IPFS. It doesn't indicate *where* the content is stored, but it forms a kind of address based on the content itself. CIDs are short, regardless of the size of their underlying content.
CIDs are based on the content’s cryptographic hash.
That means:

- Any difference in the content will produce a different CID and
- The same content added to two different IPFS nodes using the same settings will produce the same CID.

IPFS uses the `sha-256` hashing algorithm by default, but there is support for many other algorithms.
IPFS uses multi hashes (self describing hashes) to create CIDs

Directed Acyclic Graph

IPFS and many other distributed systems take advantage of a data structure called directed acyclic graphs.

See this article for more details
![

Distributed Hash Table

Nodes can store & share data without central coordination and use DHTs as a means to find each other.
A distributed hash table provides a lookup service similar to a hash table: key-value pairs are stored in a DHT, and any participating node can efficiently retrieve the value associated with a given key.
The main advantage of a DHT is that nodes can be added or removed with minimum work around re-distributing keys.
The libp2p projectis the part of the IPFS ecosystem that provides the DHT and handles peers connecting and talking to each other.

IPFS Network Properties

- Autonomy and decentralisation: the nodes collectively form the system without any central coordination.
- Fault tolerance: the system should be reliable (in some sense) even with nodes continuously joining, leaving, and failing.
- Scalability: the system should function efficiently even with thousands or millions of nodes.

The DHT is used in IPFS for routing, in other words:

1. to announce added data to the network
2. and help locate data that is requested by any node.

The white paper states:

Small values (equal to or less than 1KB) are stored directly on the DHT. For values larger, the DHT stores references, which are the NodeIds of peers who can serve the block.

Block Exchange - BitSwap Protocol

In IPFS, data distribution happens by exchanging blocks with peers using a BitTorrent inspired protocol: BitSwap.

Like BitTorrent, BitSwap peers are looking to acquire a set of blocks (want_list), and have another set of blocks to offer in exchange (have_list). Unlike BitTorrent, BitSwap is not limited to the blocks in one torrent.

BitSwap operates as a persistent marketplace where node can acquire the blocks they need, regardless of what files those blocks are part of. The blocks could come from completely unrelated files in the filesystem. Nodes come together to barter in the marketplace.

While the notion of a barter system implies a virtual currency could be created, this would require a global ledger to track ownership and transfer of the currency. This can be implemented as a BitSwap Strategy

IPNS

IPNS gives us

- human readable links
  E.g. `/ipns/extropy` instead of

  `QmPrPmbbUKA3ZodhzPWZnpFgcPMFWF4QsxXbkWfEptTBKl`

- A link to the latest version of content

IPFS based websites

- websites that are completely distributed
- websites that have no origin server
- websites that can run entirely on client side browsers

See this guide on how to host a website on IPFS

Structures used in IPFS

1. IPFS object
   a data structure with two fields:
   Data — a blob of unstructured binary data of size < 256 kB.
   Links — an array of Link structures, these are links to other IPFS objects.

2. IPFS Links

   A Link structure has three data fields:

   - Name — the name of the Link.
   - Hash — the hash of the linked IPFS object.
   - Size — the cumulative size of the linked IPFS object, including following its links.

IPFS Gateways

IPFS deployment seeks to include native support of IPFS in all popular browsers and tools.
Gateways provide workarounds for applications that do not yet support IPFS natively.
The URL will be of the form
`ipfs://{CID}/{optional path to resource}`

Public gateway operators include:

- Protocol Labs, which deploys the public gateway `https://ipfs.io`.
- Third-party public gateways. E.g., `https://cf-ipfs.com`.

For example you can explore objects
`https://explore.ipld.io/#/explore/QmSnuWmxptJZdLJpKRarxBMS2Ju2oANVrgbr2xWbie9b2D`

IPFS and NFTs

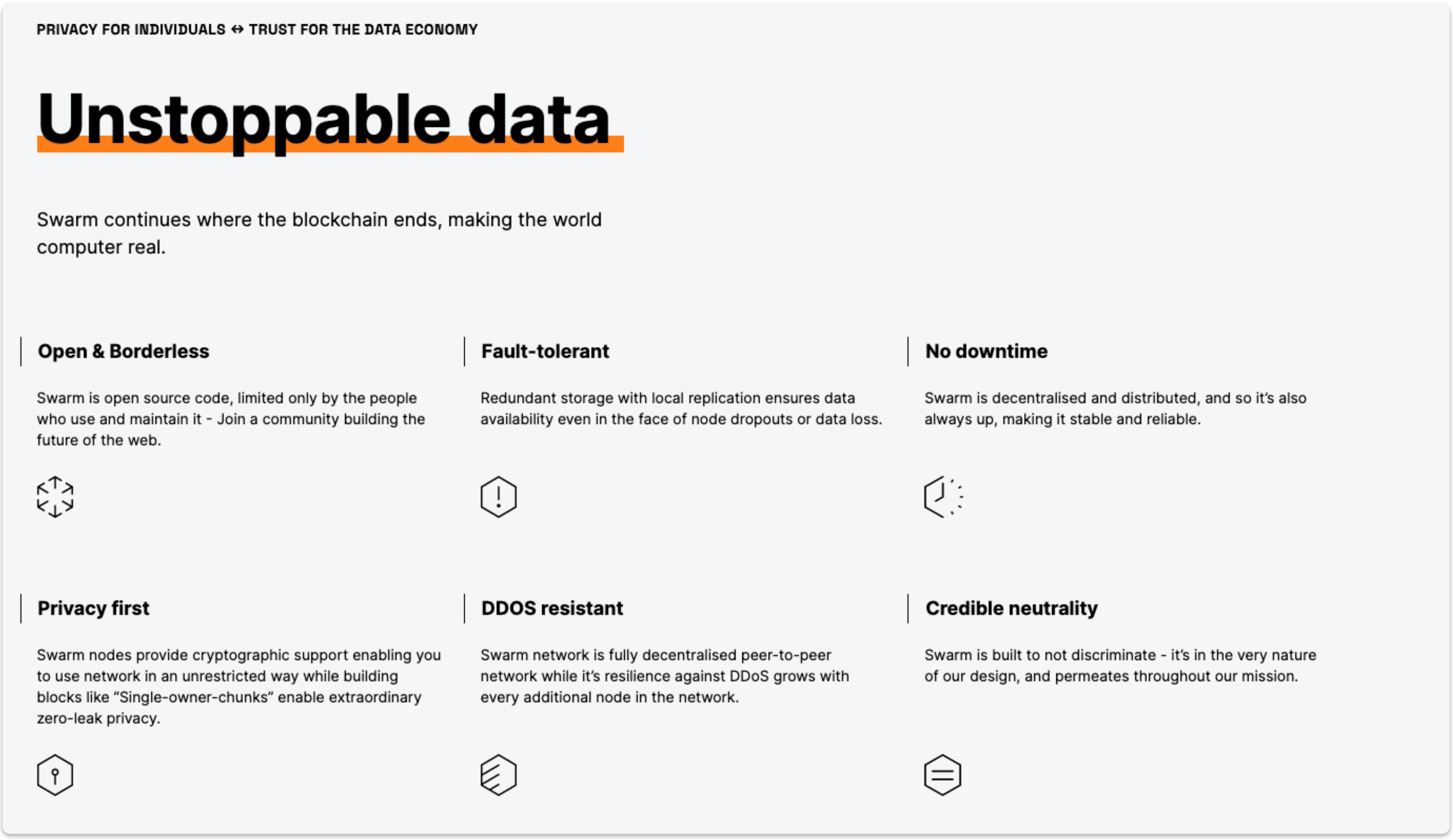
See this guide and these best practices
IPFS provide a command line Minty to help create NFTs.

Alternatives to IPFS

Swarm

See white paper

Swarm is a peer-to-peer network of nodes that collectively provide a decentralised storage and communication service.
This system is economically self-sustaining due to a built-in incentive system which is enforced through smart contracts on the Ethereum blockchain and powered by the BZZ token.



PRIVACY FOR INDIVIDUALS ↔ TRUST FOR THE DATA ECONOMY

Unstoppable data

Swarm continues where the blockchain ends, making the world computer real.

Open & Borderless

Swarm is open source code, limited only by the people who use and maintain it - Join a community building the future of the web.

Fault-tolerant

Redundant storage with local replication ensures data availability even in the face of node dropouts or data loss.

No downtime

Swarm is decentralised and distributed, and so it's also always up, making it stable and reliable.

Privacy first

Swarm nodes provide cryptographic support enabling you to use network in an unrestricted way while building blocks like "Single-owner-chunks" enable extraordinary zero-leak privacy.

DDOS resistant

Swarm network is fully decentralised peer-to-peer network while it's resilience against DDoS grows with every additional node in the network.

Credible neutrality

Swarm is built to not discriminate - it's in the very nature of our design, and permeates throughout our mission.

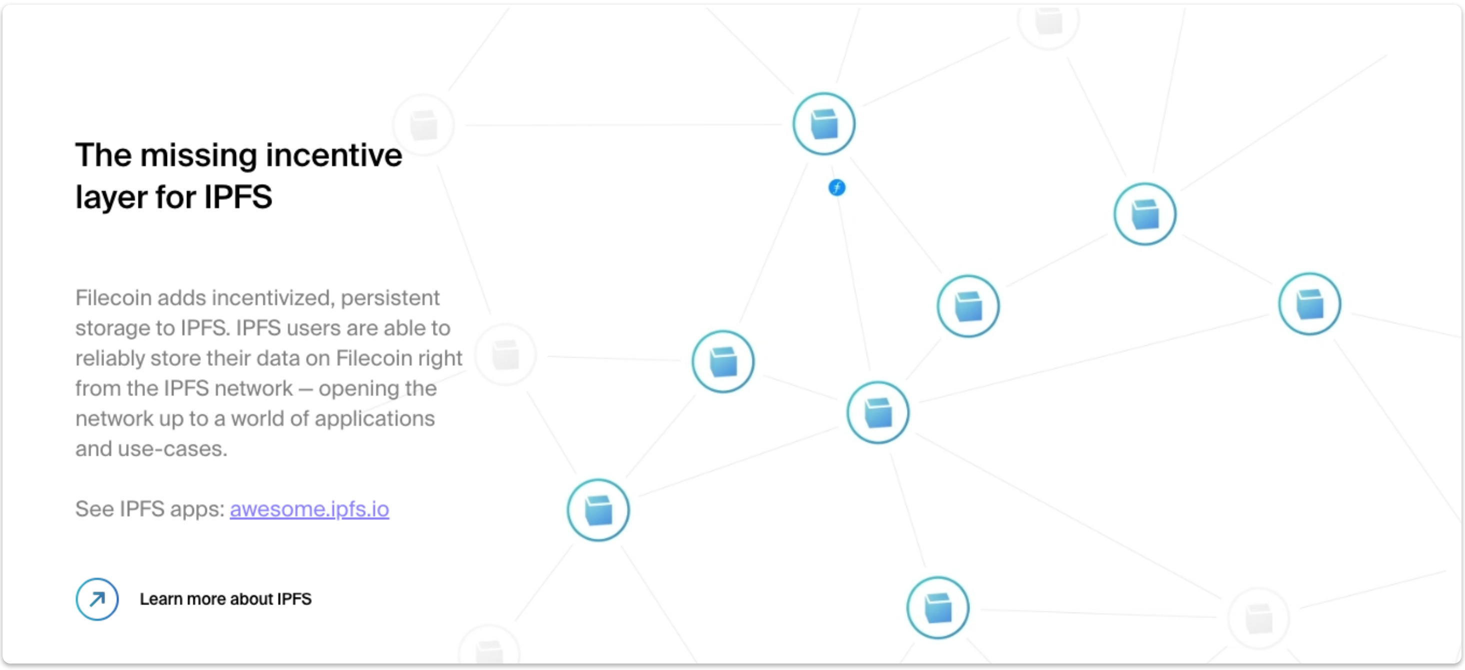
For Swarm to properly function as a decentralised p2p storage and communication infrastructure, on very basic terms there must be network participants who:

- contribute bandwidth for incoming and outgoing requests
- provide storage for users to upload and retrieve data
- forward incoming requests to peers who can fulfil them if they can not serve the request themselves

Swarm introduces its own incentives system for ensuring correct network behaviour by rewarding nodes for serving these functions.

Filecoin

See Docs
See Primer



The missing incentive layer for IPFS

Filecoin adds incentivized, persistent storage to IPFS. IPFS users are able to reliably store their data on Filecoin right from the IPFS network — opening the network up to a world of applications and use-cases.

See IPFS apps: awesome.ipfs.io

Filecoin is a digital storage and data retrieval method, made by Protocol Labs and builds on top of InterPlanetary File System, allowing users to rent unused hard drive space.

The project was launched in August 2017 and raised over $200 million within 30 minutes.

The Filecoin Network is made with storage providers and clients. They make deals and contribute to maintaining the Filecoin blockchain, obtaining storage services, and receiving rewards in the process.

Filecoin Networks

Peers communicate over secure channels that they use to distribute information to the network (gossiping), to transfer data among themselves, and to discover other peers, maintaining a well-connected swarm in which information like blocks and messages flows swiftly even when many thousands of peers participate.

- Mainnet, the only production Filecoin network.
- Calibration, the primary testing network for Filecoin.
- Wallaby, an early testing network for bleeding edge Filecoin Virtual Machine deployments.

Filecoin Nodes

Filecoin Nodes or *Filecoin clients* are peers that sync the Filecoin blockchain and validate the messages in every block, which, once applied, provide a global state.

Filecoin Nodes can also publish different types of *messages* to the network by broadcasting them. For example, a client can publish a message to send FIL from one address to a different one. Nodes can propose storage and retrieval deals to Filecoin storage providers and pay for them as they are executed.

Filecoin Storage Providers

The storage providers provide services to the network by executing different types of deals and appending new blocks to the chain (every 30 seconds), for which they collect FIL rewards.

Filecoin Deals

There are two main types of deals in Filecoin: *storage deals* and *retrieval deals*.

Storage deals are agreements between clients and *storage providers* to store some data in the network. Once a deal is initiated, and the storage provider has received the data to store, it will repeatedly prove to the chain that it is still storing the data per the agreement so that it can collect rewards.
If not, the storage provider will be slashed and they will lose FIL.

Retrieval deals are agreements between clients and *retrieval providers* (which may or not be also storage providers) to extract data that is stored in the network .
Unlike storage deals, these deals are fulfilled off-chain, using *payment channels* to incrementally pay for the data received.

Proofs

See article about the proof system.
Storage providers must prove that:

- They store all the data submitted by the client
- They store it during the whole lifetime of the deal

Proof of space

A proof-of-space is a piece of data that a prover sends to a verifier to prove that the prover has reserved a certain amount of space. For practicality, the verification process needs to be efficient, namely, consume a small amount of space and time. For soundness, it should be hard for the prover to pass the verification if it does not actually reserve the claimed amount of space.
One way of implementing PoSpace is by using hard-to-pebble graphs.
The verifier asks the prover to build a labelling of a hard-to-pebble graph. The prover commits to the labelling.
The verifier then asks the prover to open several random locations in the commitment.

Proof of spacetime

Proof-of-spacetime differs from proof-of-capacity in that PoST allows network participants to prove that they have spent a "spacetime" resource, meaning that they have allocated storage capacity to the network over a period of time.

Gas Fees

Executing messages, for example by including transactions or proofs in the chain, consumes both computation and storage resources on the network.
The gas consumed by a message directly affects the cost that the sender has to pay for it to be included in a new block by a storage provider.

An amount of the fees is burned (sent to an irrecoverable address) to compensate for the network expenditure of resources, since all nodes need to validate the messages.

How does Filecoin relate to IPFS ?

While interacting with IPFS does not require using Filecoin, all Filecoin nodes *are* IPFS nodes under the hood, and can connect to and fetch IPLD-formatted data from other IPFS nodes using libp2p. However, Filecoin nodes don’t join or participate in the public IPFS DHT.

Filecoin tutorial

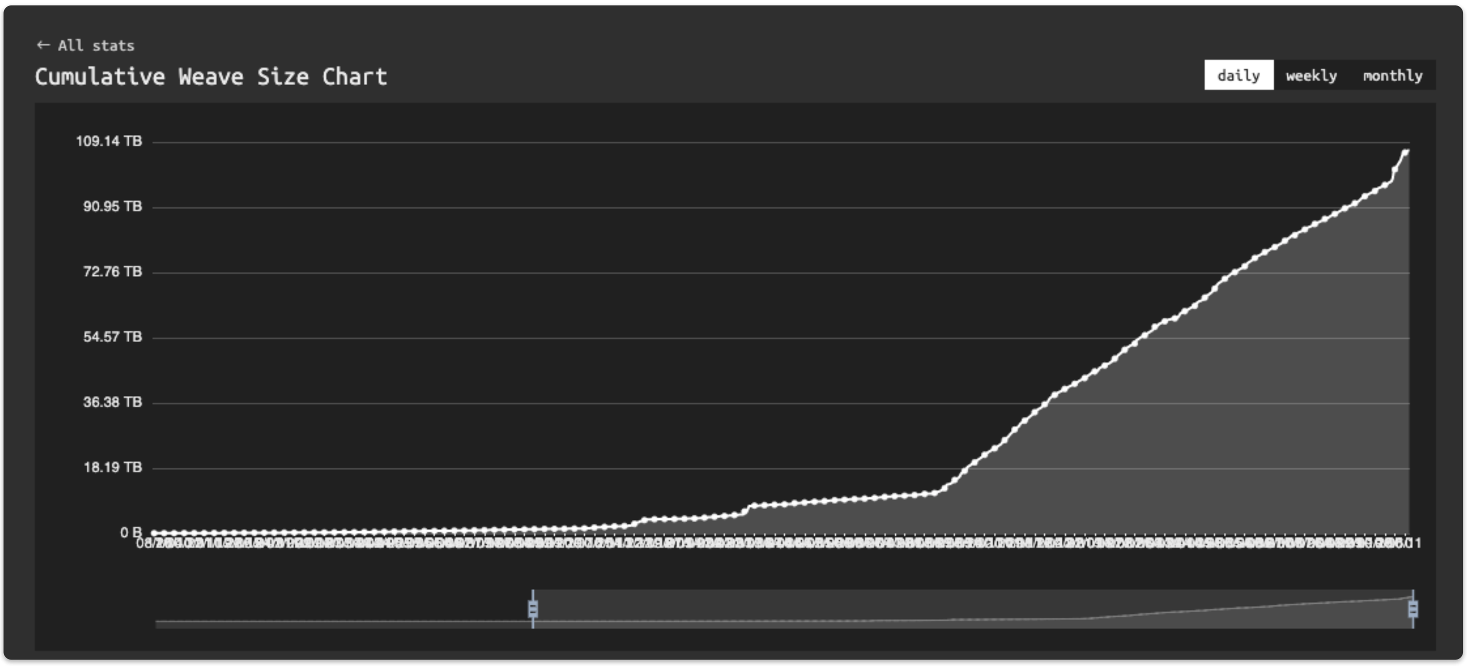
Protoschool offers a tutorial about Filecoin and storage.

Arweave

Features

- Decentralised storage built on top of the Arweave protocol
- Permanent and resilient
- Proof of Access mechanism (similar to Filecoin’s Proof of Space and Proof of Replication)
- Incentivises quick retrieval
- Data replicated and distributed across the network

Currently adding approx 6 TB per month



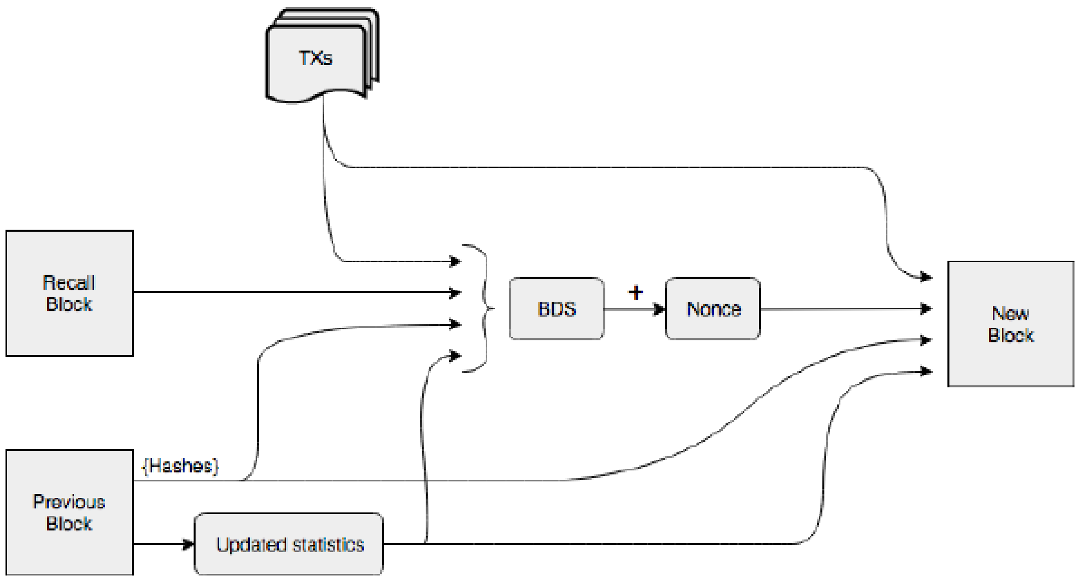
Storage details

- Mechanism to distribute tokens to sustainably incentivise perpetual storage or arbitrary quantities of data.
- Based on a logarithmically decreasing $/GB-h.
- Storage media assumed to take 434 yrs to reach max theoretical data density limit of 1.53x10^67 bits/cm^3 (currently 1.66x10^12)
- Storage medium reliability increases.
- Incentive for cheap storage grows as humanity's demand for data is growing.
- Predictions for future safekeeping of data based on ultra-low cost of storage medium.
- Arweave expected to be 'nested' inside future storage systems.

Chain Architecture

- Block constructed from previous block, recall block and transactions
- Recall bloc
    - A deterministic but unpredictable choice of block from the "weave's" history
    - Prevents data from being lost if it is rarely accessed
    - Lesser known blocks increase miner fees

Figure 6: Block construction from previous block, recall block, and transactions



Block Construction

- Waiting pool for all new transactions.
- Mining pool for tx's that are validated and likely known by other nodes.
- Transactions removed from mining pool once block is mined.
- Tx's can be reintroduced if there is a fork.
- Block Data Segment (BDS) serves as the 'puzzle' in the PoW mechanism is a hash of:
  - Independent hash of previous block
  - Entire contents of recall block
  - Transactions and other metadata for the candidate block

Creation process:

1. Assemble relevant metadata.
2. Rank and validate transactions.
3. Generate new BDS.
4. Find nonce (PoW).
5. Propagate new block in memoised form.

Network Mechanism Design

- Made up of miners and users.
- Users pay tokens (AR) to add data to the network.
- Miners receive tokens for mining new blocks.
- Each tx has optional recipient and data fields meaning they are very flexible.
- Token used for encoding data into system and rewarding miners (thus non-zero financial value).
- Maximum 66 million AR (55 million in genesis block, 11 million through block mining rewards).

Wildfire Mechanic Incentive Mechanism

- Type of Adaptive Interacting Incentive Agents (AIIA) game.
- Dominant Strategy Incentive Compatible (DSIC) nature of network ensures truth telling is a weakly dominant strategy.
- Peers are ranked on:
  - Generosity - sending new tx's and blocks.
  - Responsiveness - responding promptly to requests for information.
- Similar to Bittorrent's optimistic tit-for-tat algorithm.
- Scores are gossiped to higher-ranked peers allowing a node to rationalise its bandwidth allocation.
- Data/blocks/tx's are preferentially gossiped to more responsive peers in parallel, incentivising a stronger network.
  - Less strong peers are provided the data sequentially.
- Slow nodes are ultimately excluded from the network as peers remove them from their peer list.
  - Enforces pro-social behaviour.
  - Reduces wasted bandwidth.
  - Not assumed each node running same AIIA agent.

Fork Resistance and recovery

- Caused by:
  - Propagation delay
  - 'Edge' validation performed on data (to protect the network) before being propagated quickly.
  - Propagating blocks quickly will raise chance of receiving mining rewards.
- Forks are recovered to the majority fork immediately in a similar fashion to Nakamoto consensus-based blockchain protocols.
- A 'cumulative_difficulty' field in each block is used to resolve forks.

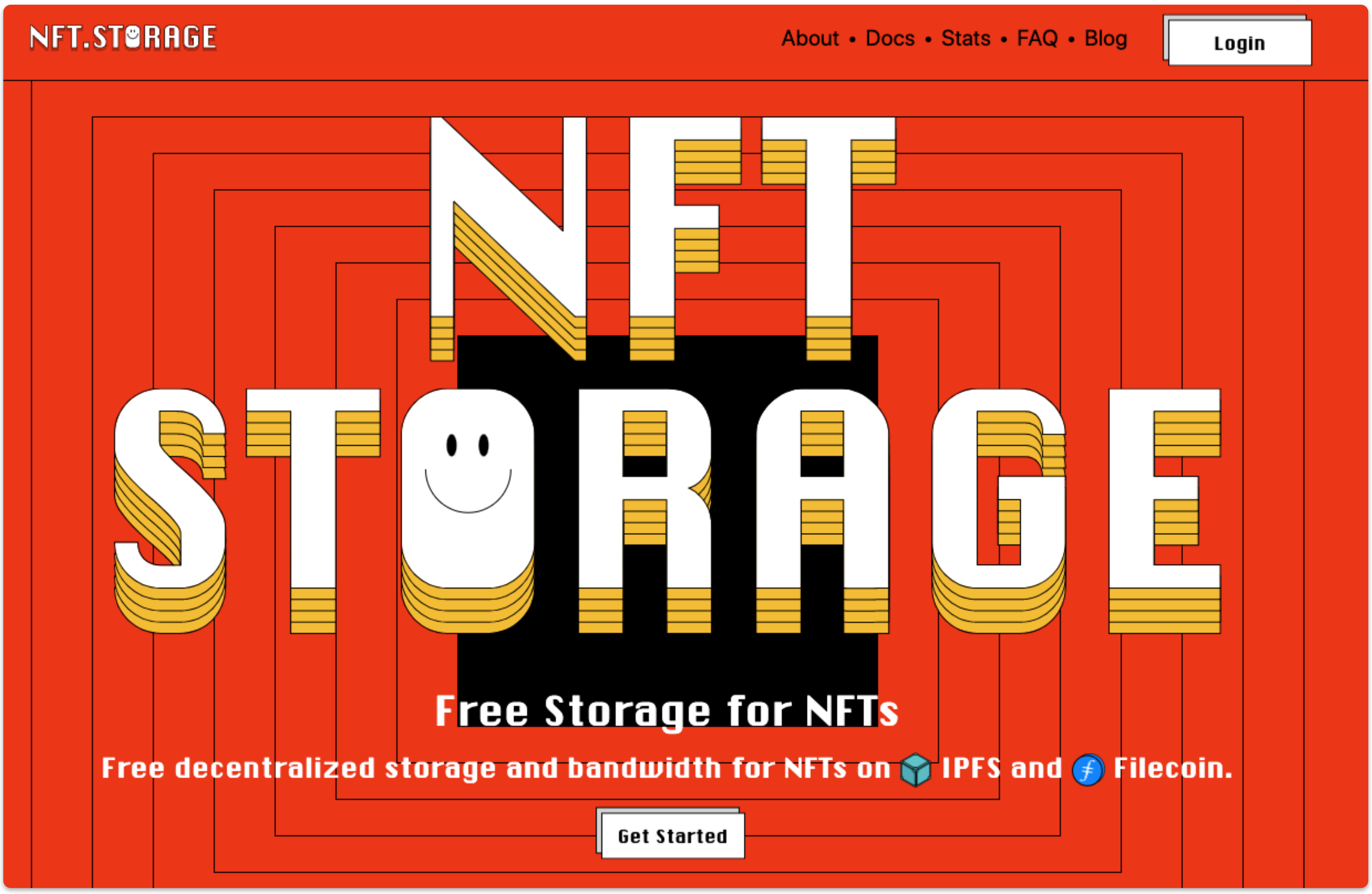
Content Policies

- Collective maintenance of the Arweave network necessitates a mechanism for miners to express their opinions on what content should and should not be hosted.
- Mechanisms used:
  - A democratic process of voting on content entry into the blockweave.
  - The individual ability of each node to choose what content to store on their machines.
  - The ability of gateway nodes to filter blockweave data that users are exposed to.
- Content matching protocols such as PhotoDNA (Microsoft) can be implemented.
- Differing content policies can cause forks.
- Gateways offer opportunity to access content with agreeable policy.

Use cases

- Providing a reliable archive of record
    - Once data is added to the blockweave, it cannot be removed or altered, either intentionally or unintentionally.
- Authenticity.
    - The blockweave offers ’proof of existence’ of a specific piece of data at a certain point in time, based on the associated transaction’s verifiable, reliable timestamp.
- Provenance.
    - Each transaction is linked permanently to the previous transaction from the same wallet, meaning that end-users can consistently verify the true origin of the data inside any transaction by wallet address, including that of decentralised applications hosted on the Arweave network.
- Decentralised application hosting.
    - The blockweave ensures more reliable access to applications than any centralised application-hosting platform, which commonly negatively impact application integrity.
- Incentivised data storing and serving.
    - Arweave’s unique mechanism design robustly encourages the rapid serving of data to all participants in the network, including end-users.

NFT Storage


NFT.STORAGE
About • Docs • Stats • FAQ • Blog
Login
NFT STORAGE
Free Storage for NFTs
Free decentralized storage and bandwidth for NFTs on IPFS and Filecoin.
Get Started

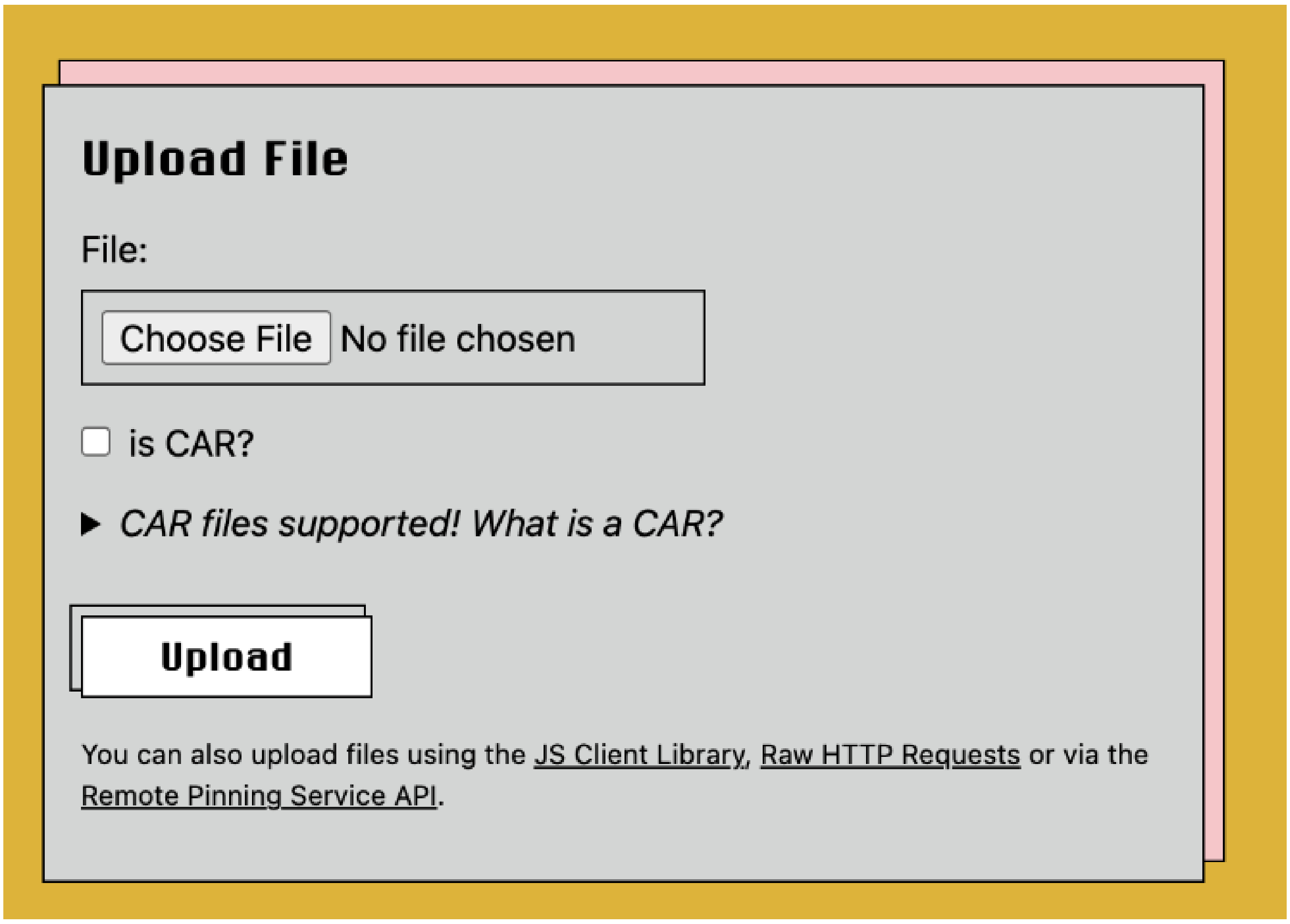
From Docs
NFT.Storage is a storage service that lets you upload off-chain NFT data for free, with the goal to store all NFT data as a public good.
The data is stored perpetually in the Filecoin decentralized storage network and made available over IPFS via its unique content ID.
You can upload as much data as you want as long as it's part of an NFT (e.g., metadata, images and other assets referenced in a token or its metadata), although there is currently a limit of 31GiB per individual upload.

Creating an account

Go to Login

Uploading files via the website

1. Click **Files** to go to your NFT.Storage file listing page.
2. Click the **Upload** button to go to the File upload page.
3. Click the **Choose File** button to select a file from your device:



You can also upload files using the JS Client Library, Raw HTTP Requests or via the Remote Pinning Service API.

You can alternatively use the NFT App

You can interact using the node package
Install with

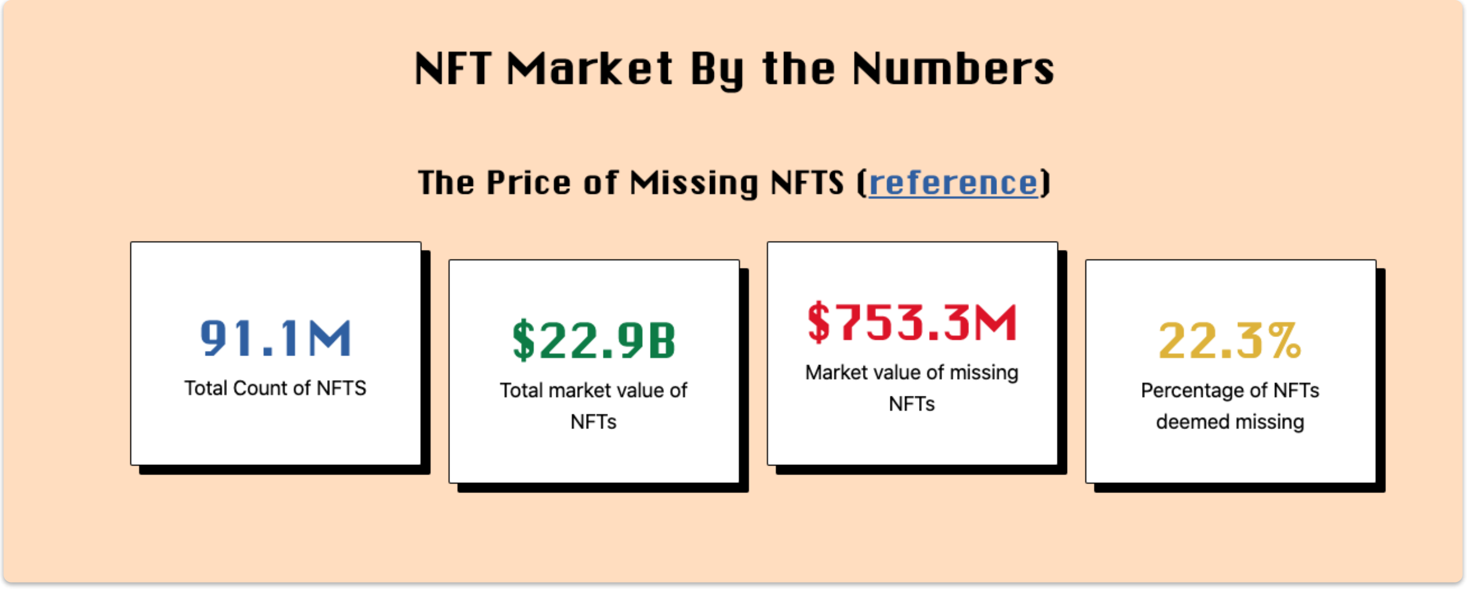
`npm install nft.storage`

there is some example client code here

NFT Storage Gateway

The project created a new HTTP gateway that uses existing public IPFS infrastructure and cloud-native caching strategies to provide a high-performance, CID-based HTTP retrieval solution that is NFT-focused.

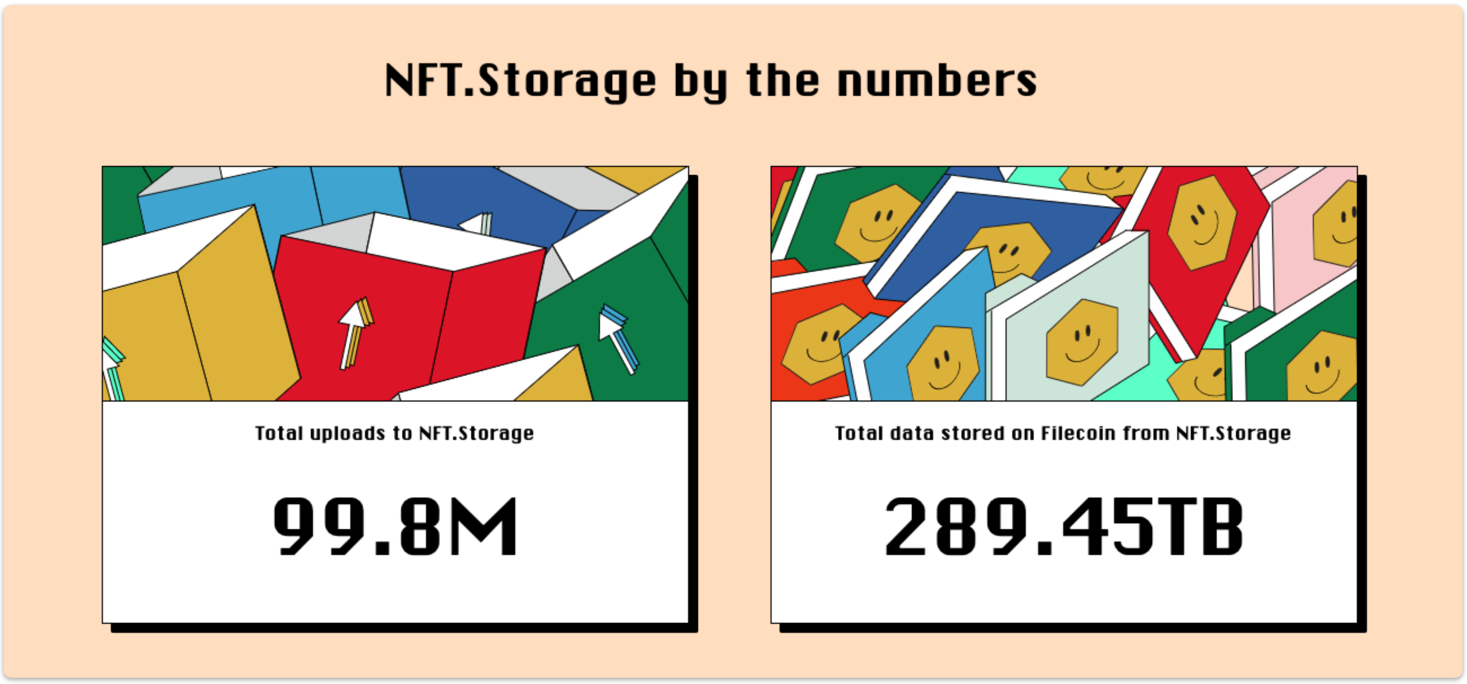
The NFT.Storage gateway is accessible at the URL `https://nftstorage.link`, and you can use it just by creating URLs with `nftstorage.link` as the host.



NFT Market By the Numbers

The Price of Missing NFTS (reference)

91.1M	$22.9B	$753.3M	22.3%
Total Count of NFTS	Total market value of NFTs	Market value of missing NFTs	Percentage of NFTs deemed missing



NFT.Storage by the numbers

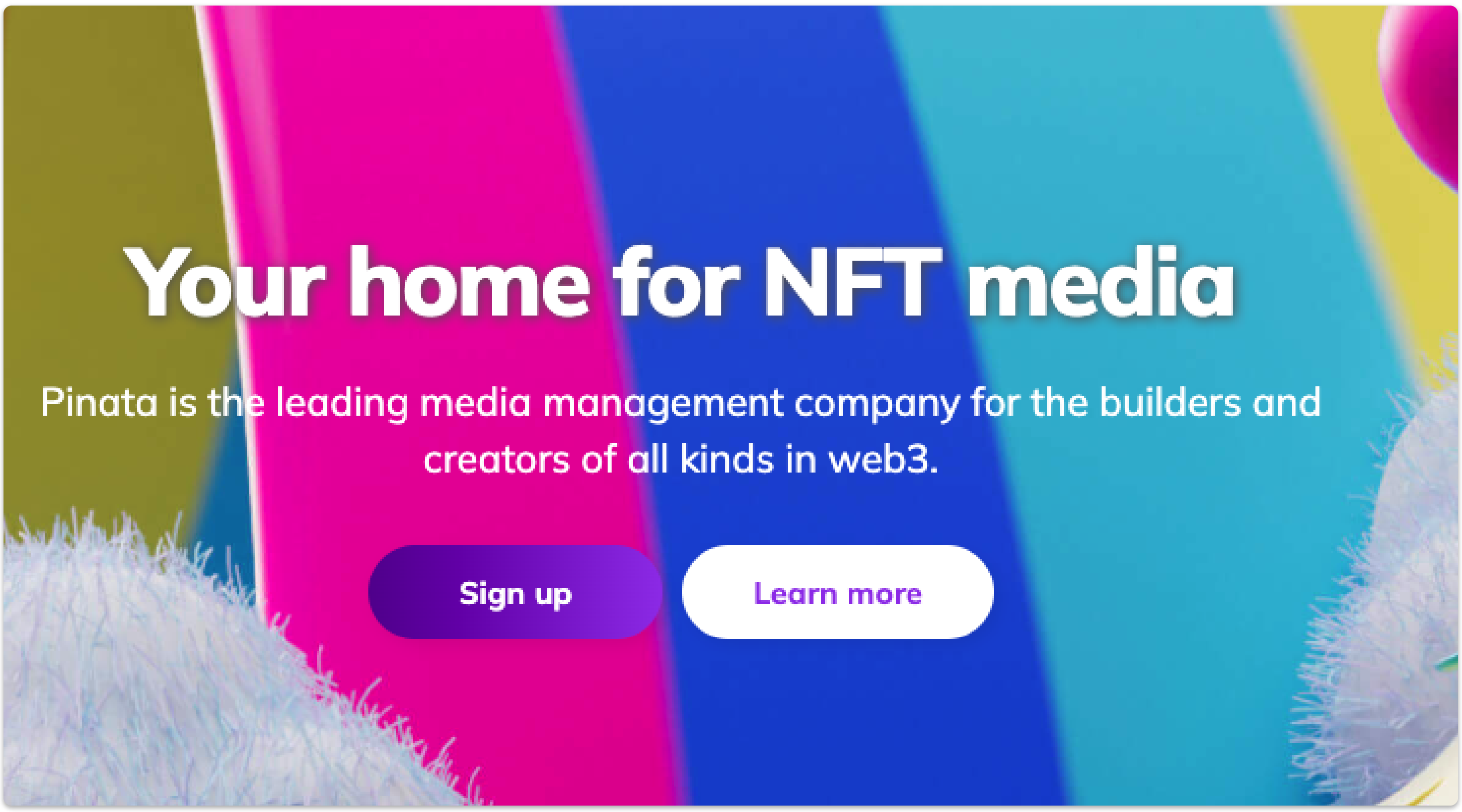
Total uploads to NFT.Storage	Total data stored on Filecoin from NFT.Storage
99.8M	289.45TB

NFT Storage and Metaplex

See Docs
We will see in the Metaplex lesson how to set up the storage

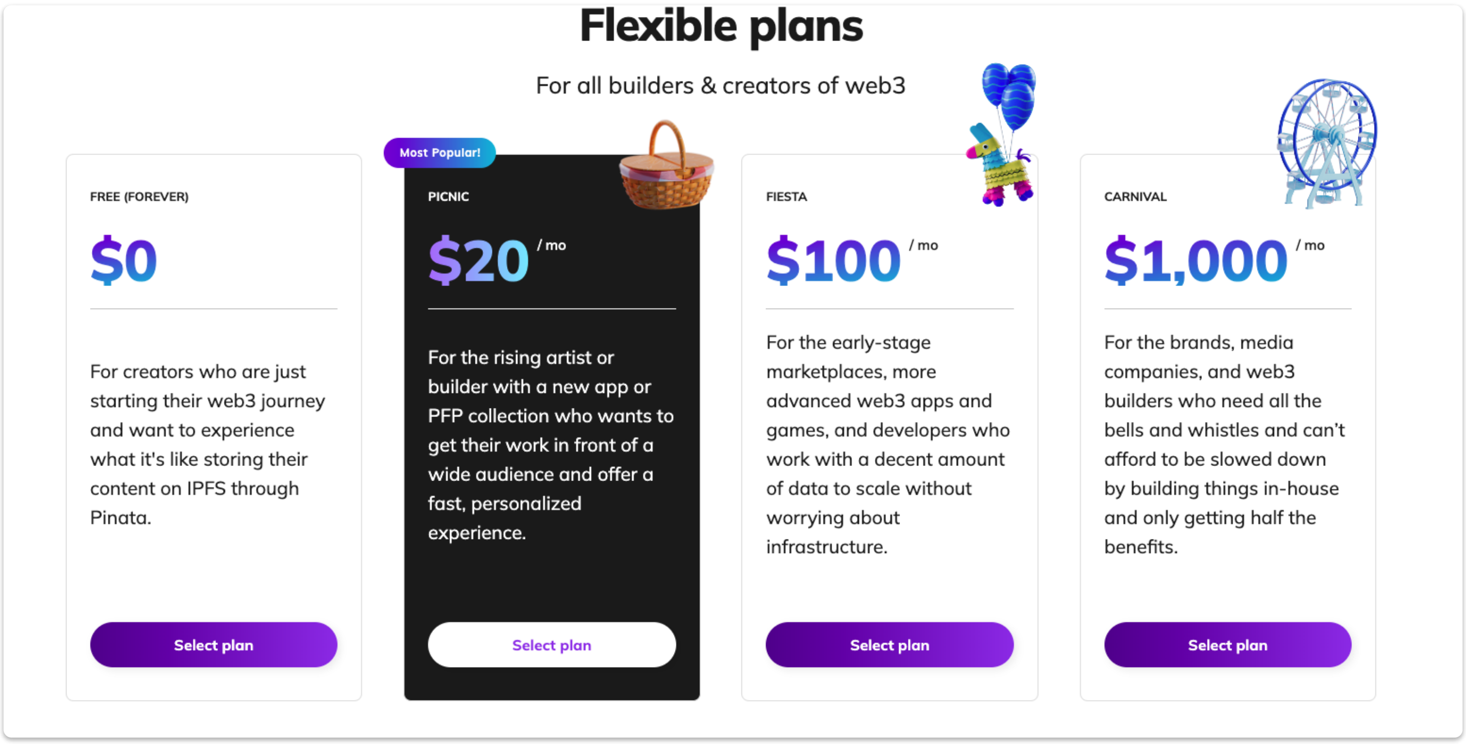
Pinata

See Docs



Your home for NFT media

Pinata is the leading media management company for the builders and creators of all kinds in web3.



Flexible plans

For all builders & creators of web3

FREE (FOREVER)	Most Popular! PICNIC	FIESTA	CARNIVAL
$0	$20 / mo	$100 / mo	$1,000 / mo
For creators who are just starting their web3 journey and want to experience what it's like storing their content on IPFS through Pinata.	For the rising artist or builder with a new app or PFP collection who wants to get their work in front of a wide audience and offer a fast, personalized experience.	For the early-stage marketplaces, more advanced web3 apps and games, and developers who work with a decent amount of data to scale without worrying about infrastructure.	For the brands, media companies, and web3 builders who need all the bells and whistles and can't afford to be slowed down by building things in-house and only getting half the benefits.
Select plan	Select plan	Select plan	Select plan

Features

- Dedicated gateway
- Built in URL shortener
- Built in CDN to optimise for video and images
- Allows sharing of unlockable content. (NFT verification / geo location)