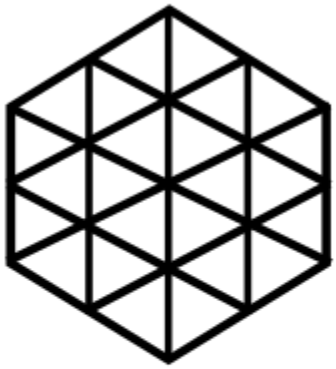


Codex

Codex: A Data Durability Engine

October 2022

V0.1

Table of Content

Codex

Introduction

Codex is a decentralized storage platform that aims to be the state-of-the-art solution for all the Web3 applications seeking for storing large amounts of data without relying on a centralized institution that could be viewed as a single point of failure [5]. For this, it is necessary to use highly efficient reliability techniques that guarantee fault tolerance in a distributed system in which node and network failures are frequent. In addition, it is important to design a system that tolerates Byzantines failures, such as for example malicious nodes that do not fulfill their data storage contracts properly. The technical challenges to implement such a platform without incurring too much overhead on the system are important. This document explains the requirements and design choices the Codex project targets in order to reach the objective of offering the highest data durability guarantees in the ecosystem.

Requirements

Any storage system that aims to be a valuable piece of the future Web3 has to respect specific requirements to contribute to the ecosystem. Codex's vision of decentralized storage requires guaranteeing the following properties:

- ***Decentralization****:* The objective of the Codex project is to help decentralized applications have the infrastructure needed to offer unstoppable and uncensorable services. For this, it is essential to have a completely decentralized underlying system. Thus, Codex aims to be as decentralized as possible, by avoiding relying on big storage providers and trying to implement protocol constructs that guarantee equal market opportunities for small/solo storage providers.
- ***Durability****:* The primary intent of Codex is to provide the highest long-term data durability guarantees in the market, even higher than centralized solutions. For this reason, Codex leverages cutting edge technology related to Remote Auditing. In addition, multiple analytical models and simulators have guided the research and development (R&D) of Codex, in order to provide a highly optimized solution.
- ***Availability****:* In addition to long-term data durability, Web3 apps need to rely on low latency data access to provide a coherent transition for users from Web2-like user-interactions and experiences to their Web3 equivalent. This makes data availability one of the principal requirements for decentralized storage. To reach that goal, Codex implements strong incentives and a robust data dispersion scheme that allows for extreme flexibility at the moment of accessing a dataset.
- ***Reliability****:* Decentralized storage systems rely on thousands of independent nodes to provide storage, and many of those nodes might not have the most reliable hardware/network. Thus, it is paramount for any decentralized storage solution to make sure the system is resilient to frequent failures and high node churn rates. Codex

implements the strongest erasure coding techniques [6] to offer the best service level agreements (SLAs) in the market.

- ***High-performance****:* While performance is obviously an important aspect of every distributed system, Codex puts special emphasis on this. The Codex vision is that storage is one of the most important components of the future digital technology in which data is the most valuable resource. In particular, decentralized storage is critical to guarantee a censorship-free digital community. As a result, Codex expects to be the infrastructure support of thousands of Web3 enterprises that expect nothing but the best performance for their businesses.

Design

Codex uses the latest innovations in storage technology to provide the highest durability guarantees and the fastest performance. The main technology components used in Codex are described below.

Erasure Coding

Any cloud storage solution, decentralized or not, relies on thousands of storage nodes in order to deliver high performance and strong reliability. However, computer hardware is always subject to failures, and in a system with thousands of hardware components, failures are rather common. Therefore, it is important to use the most advanced techniques to guarantee that the system is resilient to frequent failures. Codex does not rely on data replication but on erasure coding. Most storage providers implement data replication due to its simplicity. However, data replication is extremely inefficient in terms of storage utilization and offers a low level of reliability [7]. In particular, Codex uses fast Reed-Solomon encoding to provide fast and robust resilience [12].

Remote Auditing

As discussed in the previous section, data protection schemes are essential to guarantee durability. Data protection schemes, however, need to be complemented with a fast failure detection mechanism for any data storage network to recover from failures effectively. Data loss detection is not a trivial problem in Byzantine decentralized systems. Malicious storage nodes might try to implement a wide range of strategies to fakely pretend to store data in order to save storage and bandwidth costs. To solve this issue, a large number of techniques have been proposed in the academic literature, such as proof of custody [8], and proof of space-time [1], among many others. Most of them rely on a frequent random sampling of data blocks across the whole dataset. During this process, storage nodes have to provide clear evidence that they hold the data they say they hold. While these mechanisms are widely understood today, the real challenge is how to implement them efficiently. Indeed, storage proofs can take significant space and they can also consume significant network bandwidth. Codex aims to minimize proofs' storage cost and network bandwidth using succinct non-interactive arguments of knowledge (SNARKS) [9]. While the concept of SNARKS has

been in the literature for several years now, their efficient implementation has been out of reach until very recently. Thus, Codex plans to leverage the latest, most optimized SNARK techniques to implement the most efficient, low-cost, and highly reliable data loss detection system for decentralized storage.

Lazy Repair

A good failure detection system needs to be complemented with a fast and efficient recovery mechanism. It is critical to recover any piece of data missing in the system to guarantee data durability. Codex implements a strong Reed-Solomon (RS) scheme in which multiple data blocks from a dataset can be lost before the dataset becomes irretrievable. The margin of lost blocks depends on the dataset and the specific parameters asked by the user, but in all cases, Codex can tolerate multiple missing blocks and still be able to quickly reconstruct the whole dataset. Depending on the node churn rate in the system, it might be bandwidth intensive and inefficient to repair every block as soon as the missing block is detected since this requires downloading a certain number of blocks for the RS algorithm to decode and produce the original data. Therefore, Codex implements a bandwidth-efficient lazy recovery technique [10] that still guarantees high reliability and minimizes the overhead on the platform. By reducing the congestion on the network, Codex aims to provide the fastest, high-performance decentralized storage solution in the ecosystem.

Marketplace

The marketplace of a decentralized storage platform is an important piece of the system and it has a profound impact on several of the requirements expressed above. For instance, if the marketplace just implements a naive first-come-first-served strategy, large storage providers with low-latency and high-performance nodes will get most of the storage deals, generating some sort of partial centralization. Another aspect related to the marketplace is the cost of data repairs. The rules of the marketplace can drive the system towards more (or less) homogeneity in terms of data distribution. In systems where the datasets are not homogeneously distributed across all storage nodes, but instead accumulated on a few big nodes, the data repair is more time-consuming because only a few nodes are involved in the recovery process and they need to dedicate significant bandwidth and hardware resources for the data repair to be successful. Codex implements an advanced marketplace that has been modeled and simulated to guarantee the highest levels of decentralization and data distribution.

Decentralized storage comparison

There are several decentralized storage solutions with different features and technologies. The most popular ones today are Filecoin [1], Storj [2], Arweave [3], and Sia [4]. However, none of them incorporates all the features that we estimate are necessary to create a scalable, robust, and high-performance decentralized solution for Web3 applications. The following table summarizes the differences between these storage solutions.

Feature	Filecoin	Storj	Arweave	Sia	Codex
Decentralized storage network	✓	✓	✓	✓	✓
High-performance erasure coding	✗	✓	✗	✗	✓
SNARK-based proof of retrievability	✓	✗	✗	✗	✓
Bandwidth-efficient lazy repair	✗	✓	✗	✗	✓
Advanced marketplace protocol	✗	✗	✗	✗	✓

Interface

To try Codex, one can simply clone the repository [11]:

```
git clone https://github.com/status-im/nim-codex.git
```

To build the project, and run:

```
make update && make exec
```

The executable will be placed under the build directory under the project root. You can run the client with:

```
./build/codex
```

The client exposes a REST API that can be used to interact with it. These commands could be invoked with any HTTP client, however, the following endpoints assume the use of the `curl` command.

PeerId

Connect to a peer identified by its ID. Takes an optional `addrs` parameter with a list of valid multiaddresses. If `addrs` is absent, the peer will be discovered over the DHT. Example:

```
curl "127.0.0.1:8080/api/codex/v1/connect/<peer id>?addrs=<multiaddress>"
```

Download

Download data identified by a CID. Example:

```
curl -vvv "127.0.0.1:8080/api/codex/v1/download/<CID of the content>"
--output <name of output file>
```

Upload

Upload a file, upon success returns the CID of the uploaded file. Example:

```
curl -vvv -H "content-type: application/octet-stream" -H Expect: -T "<path
to file>" "127.0.0.1:8080/api/codex/v1/upload" -X POST
```

Info

Get useful node info such as its peer id, address and SPR. Example:

```
curl -vvv "127.0.0.1:8080/api/codex/v1/info"
```

Tutorial

This short tutorial explains how to run two Codex clients. First, clone the Codex repository:

```
git clone https://github.com/status-im/nim-codex.git
```

```
cd nim-codex
```

To build the project run:

```
make update && make exec
```

Run the client with:

```
build/codex --data-dir="$(pwd)/Codex1" -i=127.0.0.1
```

This will start codex with a data directory pointing to Codex1 under the current execution directory and announce itself on the DHT under 127.0.0.1.

To run a second client that automatically discovers nodes on the network, we need to get the Signed Peer Record (SPR) of the first client, Client1. We can do this by querying the /info endpoint of the node's REST API.

```
curl http://127.0.0.1:8080/api/codex/v1/info
```

This should output information about Client1, including its PeerID, TCP/UDP addresses, data directory, and SPR:

```
{
  "id": "16Uiu2HAm92LGXYTuhtLaZzkFnsCx6FFJsNmswK6o9oPXFbSKHQEa",
  "addrs": [
    "/ip4/0.0.0.0/udp/8090",
    "/ip4/0.0.0.0/tcp/49336"
  ],
  "repo": "/repos/status-im/nim-codex/Codex1",
  "spr":
"spr:CiUIAhIhAmqg5fVU2yxPStLdUOWgwrkWZMHW2MHf6i6l8IjA4tssEgIDARpICicAJQgCEi
ECaqDl9VTbLE9K0t1Q5aDCuRZkwdbYwd_qLqXwiMDi2ywQ5v2VlAYaCwoJBH8AAAGRAh-aGgoKC
AR_AAABBts3KkcwRQIhAPOKl38CviplVbMVnA_9q3N1K_nk5oGuNp7DWeOqiJzzAiATQ2acPyQv
PxLU9YS-TiVo4RUXndRcwMFMX2Yjhw8k3A"
}
```

Now, let's start a second client, Client2. Because we're already using the default ports TCP (:8080) and UDP (:8090) for the first client, we have to specify new ports to avoid a collision. Additionally, we can specify the SPR from Client1 as the bootstrap node for discovery purposes, allowing Client2 to determine where content is located in the network.

```
build/codex --data-dir="$(pwd)/Codex2" -i=127.0.0.1 --api-port=8081
--udp-port=8091
--bootstrap-node=spr:CiUIAhIhAmqg5fVU2yxPStLdUOWgwrkWZMHW2MHf6i6l8IjA4tssEg
IDARpICicAJQgCEiECaqDl9VTbLE9K0t1Q5aDCuRZkwdbYwd_qLqXwiMDi2ywQ5v2VlAYaCwoJB
H8AAAGRAh-aGgoKCAR_AAABBts3KkcwRQIhAPOKl38CviplVbMVnA_9q3N1K_nk5oGuNp7DWeOq
iJzzAiATQ2acPyQvPxLU9YS-TiVo4RUXndRcwMFMX2Yjhw8k3A
```

There are now two clients running. We could upload a file to Client1 and download that file (given its CID) using Client2, by using the clients' REST API.

To upload the document `upImage.png` using Client1 do:

```
curl -vvv -H "content-type: application/octet-stream" -H Expect: -T
upImage.png "127.0.0.1:8080/api/codex/v1/upload" -X POST
```

You should see an output similar to the following one.

```
*   Trying 127.0.0.1:8080...
  % Total    % Received % Xferd  Average Speed   Time    Time     Time
Current
                                 Dload  Upload   Total   Spent    Left
Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--
0* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> POST /api/codex/v1/upload HTTP/1.1
> Host: 127.0.0.1:8080
> User-Agent: curl/7.81.0
> Accept: */*
> content-type: application/octet-stream
> Content-Length: 36862
>
} [36862 bytes data]
* We are completely uploaded and fine
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Server: nim-presto/0.0.3 (amd64/linux)
< Content-Length: 49
< Content-Type: text/text
< Date: Mon, 10 Oct 2022 19:18:35 GMT
< Connection: keep-alive
<
{ [49 bytes data]
100 36911  100    49  100 36862  16931  12.1M --:--:-- --:--:-- --:--:--
17.6M
* Connection #0 to host 127.0.0.1 left intact
zdj7WVjm5FiwQxzQVggtX56NPMmkPMwjGLASRQYEVP6kzDBsz
```

At this point, the document `upImage.png` has been uploaded to the Codex platform. The last line of the output (`zdj7WVjm5FiwQxzQVggtX56NPMmkPMwjGLASRQYEVP6kzDBsz`) represents the CID of the document, which you will need in order to retrieve the data.

Now, to download the document using the Client2 API and the document CID, do:

```
curl -vvv
"127.0.0.1:8081/api/codex/v1/download/zdj7WVjm5FiwQxzQVggtX56NPMmkPMwjGLASR
QYEVP6kzDBsz" --output downImage.png
```

This will generate a new document downImage.png. The output of the command should be similar to this.

```
*   Trying 127.0.0.1:8080...
  % Total    % Received % Xferd  Average Speed   Time    Time     Time
Current
                                 Dload  Upload   Total   Spent    Left
Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--
0* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> GET
/api/codex/v1/download/zdj7WVjm5FiwQxzQVggtX56NPMmkPMwjGLASRQYEVP6kzDBsz
HTTP/1.1
> Host: 127.0.0.1:8080
> User-Agent: curl/7.81.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Transfer-Encoding: chunked
< Server: nim-presto/0.0.3 (amd64/linux)
< Content-Type: application/octet-stream
< Date: Mon, 10 Oct 2022 19:20:17 GMT
< Connection: keep-alive
<
{ [6 bytes data]
100 36862    0 36862    0     0  4673k      0 --:--:-- --:--:-- --:--:--
5142k
* Connection #0 to host 127.0.0.1 left intact
```

You can now verify that both documents are binary identical.

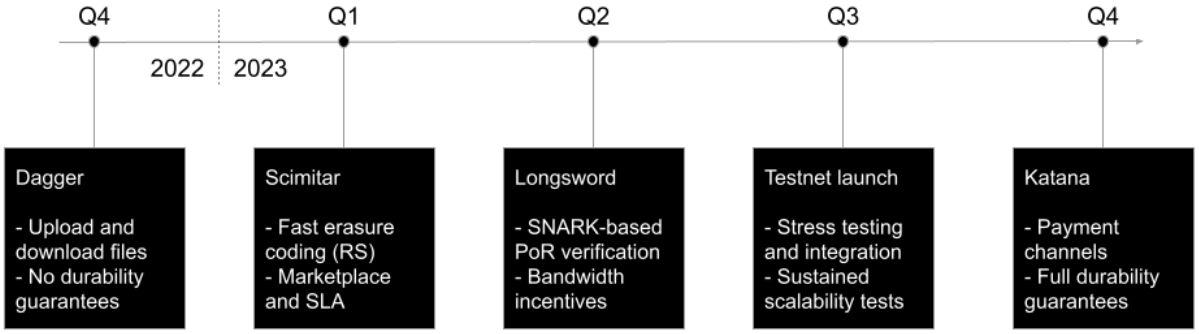
```
diff upImage.png downImage.png
```

This should produce no output at all. We hope this gives you an idea on how to use Codex for your projects.

Roadmap

As in most research and development projects, Codex is being developed in multiple stages, adding different features at each stage. At the end of each stage, a proof of concept (PoC) will be released and available to the community. In the first stage (PoC0) users will be able to upload, download and share files, in the same way, as the interplanetary file system (IPFS) allows it. Note that at this stage, data is not erasure coded and storage nodes are not being verified with any PoR mechanism, hence, there are no guarantees on data durability.

In the second stage (PoC1), datasets uploaded to the Codex platform will be systematically encoded using erasure coding. The specific RS encoding parameters should be selected by the user at the moment of uploading, depending on the specific durability requirements and the service level agreement (SLA) of the dataset in question. At this stage, the marketplace should be already in place and storage nodes can bid for storage contracts, although at this time there should be no real monetary transactions taking place in the system.



In the third stage (PoC2), storage nodes will submit proofs of the data they have using a SNARK-based PoR verification system that minimizes storage and bandwidth overhead. When data erasure is discovered in the system, the protocol follows the lazy repair strategy according to the RS encoding parameters and the conditions of the system. Additionally, at this stage the protocol already incorporates bandwidth incentives to maximize download performance.

Before the final release of the first official version of Codex, the Codex team will set up a testnet with the intention of stress-testing the platform and performing large-scale tests. Users will be able to use the testnet, search for vulnerabilities, and attempt to break the platform. The intention of this phase is to measure the robustness of the system under different conditions.

Finally, Codex v1.0 is expected to be released at the end of 2023, providing scalable payment channels with full durability guarantees for all users in the system.

References

[1] Filecoin: A Decentralized Storage Network, https://filecoin.io/filecoin.pdf, 2017

[2] Storj: A Decentralized Cloud Storage Network Framework, https://www.storj.io/storjv3.pdf, 2018

[3] Archain: An Open, Irrevocable, Unforgeable and Uncensorable Archive for the Internet, https://www.arweave.org/whitepaper.pdf, 2017

[4] Sia: Simple Decentralized Storage, https://sia.tech/sia.pdf, 2014

[5] Decentralizing Storage for Web3, https://blog.codex.storage/decentralizing-storage-for-web3/ 2022

[6] Wicker, Stephen B., and Vijay K. Bhargava, eds. *Reed-Solomon codes and their applications*. John Wiley & Sons, 1999.

[7] Weatherspoon, Hakim, and John D. Kubiatowicz. "Erasure coding vs. replication: A quantitative comparison." *International Workshop on Peer-to-Peer Systems*. Springer, Berlin, Heidelberg, 2002.

[8] Kravchenko, Pavel, and Vlad Zamfir. "Cryptographic proof of custody for incentivized file-sharing."
[9] Bitansky, Nir, et al. "Succinct non-interactive arguments via linear interactive proofs." *Theory of Cryptography Conference*. Springer, Berlin, Heidelberg, 2013.
[10] Giroire, Frédéric, Julian Monteiro, and Stéphane Pérennes. "Peer-to-peer storage systems: a practical guideline to be lazy." *2010 IEEE Global Telecommunications Conference GLOBECOM 2010*. IEEE, 2010.
[11] Codex, Decentralized Durability Engine, https://github.com/status-im/nim-codex
[12] Leopard-RS: MDS Reed-Solomon Erasure Correction Codes for Large Data in C, https://github.com/catid/leopard