

Litepaper v2

November 2026

[ARKIV]

arkiv.network



CONTENTS

Data Autonomy: A New Path Forward 04

What is Arkiv 05

How Arkiv Works 06

The Data Lifecycle 08

The SDK: A Technical Teaser 10

Golem DB 12

Embracing Ethereum 13

GLM and Storage Economics 14

The Practical Difference 15

What You can Build 16

An Ecosystem for Everyone 18

DATA AUTONOMY: A NEW PATH FORWARD

The way people use the Internet is currently split into two realms that mostly exist in parallel: the evolving, blockchain-driven Web3 and the more traditional Internet, sometimes called Web2. While both approaches have distinct strengths and are continuously expanding in scope and impact, their separation makes it difficult for users to navigate and fully use their combined potential.

In Web2, managing data has historically meant setting up your own infrastructure and running it. Today that complexity can be avoided by renting hardware, subscribing to cloud services or using specialized web applications. These services are convenient, and the trade-offs are familiar: centralization and censorship risk, account management burdens, vendor lock-in. Behind all of them, data ownership remains uncertain when the data is stored on someone else's server.

Web3 took a different route, shifting trust from central entities to decentralized protocols. So far most of what runs on it stores and transfers value, through tokens, and data management is still largely untapped. The apps are decentralized; their data often is not. A dapp's leaderboards, agent memory and historical data commonly sit in a database one operator can throttle, censor or switch off. Reading on-chain data beyond a simple lookup means running your own indexer or trusting someone else's server.

Most guarantees that data has today come from the organization that hosts it, not from the data itself: who wrote it, whether it is intact, who may change it, who may reach it, how it can be found, and how long it is kept. To trust any of them you must trust the host, and when the data moves elsewhere each guarantee has to be established anew. As long as these guarantees stay bound to the host that grants them, an owner controls their data only on the host's terms.

What is missing is a way to build such guarantees into the data itself. That rules out infrastructure controlled by one party, since the guarantees would then depend on that party as they depend on a host today. It equally rules out gatekeeping over what may be written or by whom. The infrastructure must therefore be shared, agnostic about the data's contents and authors, and open to anyone to write and query. On such infrastructure each owner controls their own data, which is tamper-evident and can carry rules of its own, such as a lifetime or terms for who may extend or change it.

This is the commitment of Arkiv. Data Autonomy should be the norm, not the exception: guarantees that stay with the data, not with the host, so that the owner sets the terms. The goal is to give Everyone that control without adding unnecessary complexity.

WHAT IS ARKIV

Arkiv is an Ethereum-aligned Web3 database: the protocol for owned, queryable data with Web3 guarantees. The protocol, run by a decentralized network rather than by a host, enforces who owns each record, who may change it and how long it lives.

Arkiv enables Data Autonomy for Everyone by combining three things:

- **From Web2 databases it keeps:** structured records, indexing, a query language. A record holds typed cells. Cells declared as attributes are indexed by the chain and can be queried; the other cells are simply stored. Queries filter, select and paginate over the attributes, as they would in any database.
- **From Web3 it takes:** ownership, tamper-evidence, permissionless access and decentralized design goals. A record belongs to a wallet address, and only its owner may change it. Every write is a signed transaction and the chain's state is verifiable, so an altered record shows up as altered.
- **It adds:** rules the data carries itself, enforced by the chain. Every record carries a few reserved cells, the protocol fields, that make up its rules: who owns it, when it expires, and flags such as *immutable* and *extendable by anyone*. The record stays queryable until it expires unless it is renewed, so temporary data is paid for only as long as it is needed. An immutable record cannot be changed or deleted by anyone, its owner included, while it lives. A record flagged *extendable by anyone* can be kept alive by strangers who pay for the extension, without their gaining any control over it.

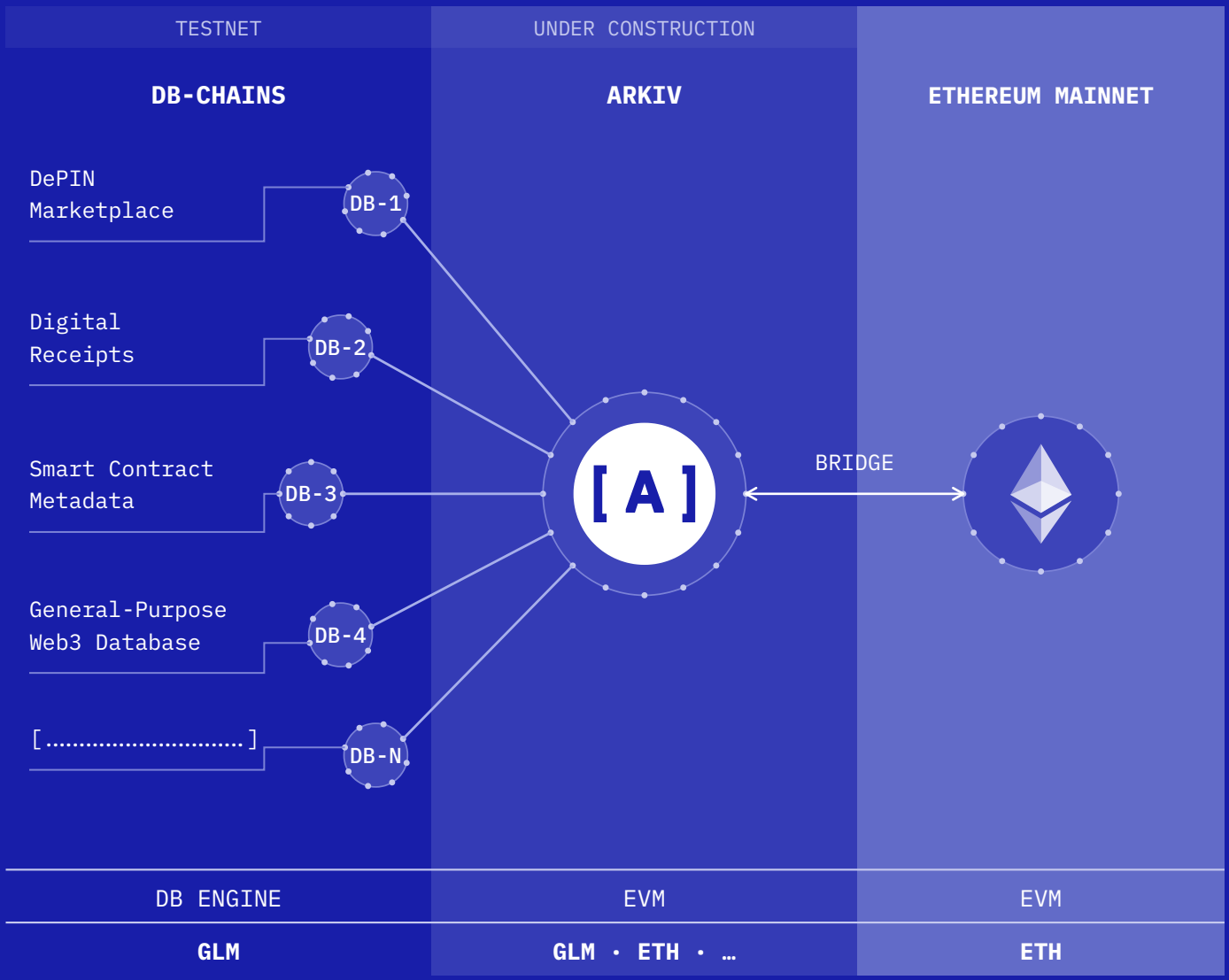
Each guarantee can be checked against what the network committed, so nobody has to take a host's word. Everything on Arkiv's public chains is readable by anyone: the data is not confidential, and no host decides who may reach it. Data that must stay private is encrypted by its owner before it is written; privacy features in the protocol itself may come later. The protocol vouches for who wrote a record and that nobody changed it; it cannot vouch for whether the record is true.

You use Arkiv like a database, over familiar RPC endpoints, a convenient SDK, or an MCP server that lets AI agents do the same. Writes are paid in GLM, the Golem Network's token; reads and queries carry no protocol fee, though an RPC provider may charge for serving them.

Today Arkiv runs as public testnets; the decentralized network described here is planned for 2027.

HOW ARKIV WORKS

Arkiv is designed in three layers: the Arkiv coordination chain, bridged to Ethereum; the DB-Chains that attach to it; and the database engine inside each DB-Chain. Arkiv is a sidechain, not a rollup.



This litepaper is a snapshot in time, capturing the current design and decisions. What the diagram above and the following pages describe is the design as it stands today, and it is expected to evolve as development continues.

The Arkiv chain

The Arkiv chain is the gateway to the ecosystem of DB-Chains: it keeps their registry and runs the logic they share.

DB-Chains

A DB-Chain is a blockchain built around data. Where Ethereum and the Arkiv chain use the EVM, Ethereum's virtual machine, to process state, a DB-Chain's execution client delegates state processing to a database engine. Every write is signed, and the chain's history records its provenance.

Arkiv's approach defines a new concept, the Ethereum Domain Specific Engine (EDSE). An EDSE does not have to rely on the Ethereum Virtual Machine to process transactions and state. Instead it can use a specialized engine tailored to a specific use case, akin to a domain-specific language, which is narrower in scope but can offer greater efficiency and performance than the general-purpose EVM. Arkiv is one application of the idea, a data-centric one.

Composable by design

Each DB-Chain chooses its own engine, token economics and application logic. It can be permissionless, open to any writer who pays the fee, or permissioned, admitting only known participants. It can implement expiration logic, as DB-Chain-1 does, or store records indefinitely. It can be tailored to a specific use case or be a general-purpose store. The ecosystem can therefore hold specialized chains alongside general ones. Anyone may register a DB-Chain on the Arkiv chain, of either kind. The first deployed DB-Chain, DB-Chain-1, is open to any writer.

Database engines

The database engine runs inside each DB-Chain's nodes. It provides structured storage, indexes and queries. It must be deterministic.

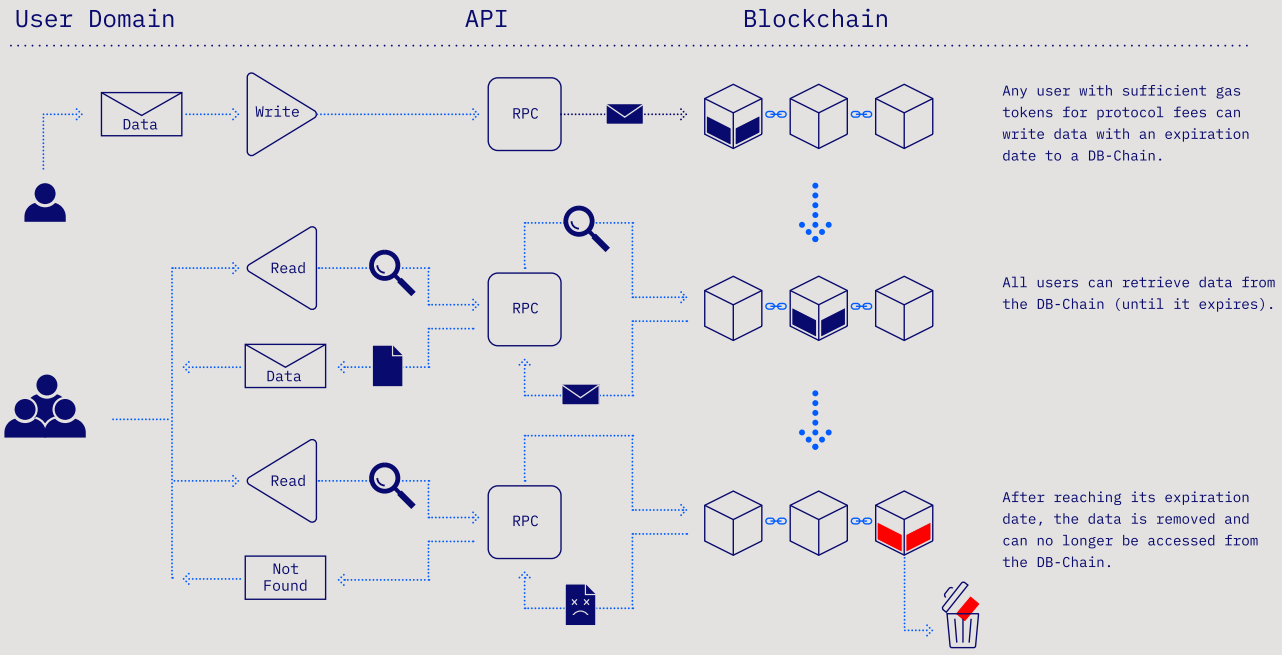
For DB-Chain-1, a database called Golem DB was designed for this role. It meets these requirements, keeps the data store fully abstracted from the chain's execution logic, and leaves room to tune or replace the engine without changing the chain. We built the chain's execution client as a fork of reth, the Rust Ethereum client, changing as little as possible: Ethereum's state and the EVM are swapped out, and Golem DB takes their place. Section 6 describes Golem DB in detail.

DB-Chain-1 and the wider ecosystem

DB-Chain-1 is the first DB-Chain, a reference implementation intended to make the design usable. It produces a block every two seconds. Its data model and configuration are not the limit of the ecosystem. Other DB-Chains may run different profiles, their own configurations of engine, rules and economics, and must state the guarantees they provide.

> **Status, November 2026.** Arkiv is not yet deployed as the decentralized network described here. Today DB-Chains run as public testnets, with more features being added; the Arkiv chain that will link them to Ethereum is under construction. Testnet data may be purged before mainnet. The mainnet, decentralized from launch, is planned for 2027. This document describes the design. The current state is at hub.arkiv.network.

THE DATA LIFECYCLE



On an expiring DB-Chain, the owner writes records and anyone may query them while they are live. When a record expires it leaves the query surface; retained history and copies that readers made are separate.

Write

A write submits a record through the SDK or RPC interface as a transaction signed by the writer's wallet. It sets the record's cells, declares which of them are attributes, and gives the record an expiration. It can also set the record's rules: whether it is immutable, and whether anyone may extend its lifetime.

Own

The record belongs to the wallet that wrote it. Today that owner alone may change, transfer or delete the record, within the rules the record carries; an immutable record admits none of these while it lives. Through Arkiv Improvement Proposals, permissions will evolve to support delegation, such as named editors, and multi-party ownership. Delegation could be scoped to particular records and limited in time: a session key kept in the browser would then write on the user's behalf, so that a diagram tool saving to Arkiv, or a chat using Arkiv to carry its messages, does not require the user to sign every transaction manually with a wallet.

Query

A record can be fetched by its key or found by querying over attributes, with filters, sorting and cursor paging. A query names the cells it wants back through a projection, so an application listing posts can ask for each title and author and skip the body; any cell can be returned this way, and attributes can be used for filtering and sorting. A query can count its matches instead of returning them, and it can run as of an earlier block inside the chain's retention window, the span of recent history every node keeps, which each chain sets for itself. Reads and queries are not transactions.

Update

While a record lives, its owner can update it: change the value of a cell, add or remove cells, or turn a stored cell into an attribute and back. An update is a signed, paid transaction like the original write. It cannot shorten the record's lifetime; expiration only ever moves later, through extension.

Extend or expire

A record stays on the query surface, the set of records that queries can see, until it expires or is deleted under its rules. While it is still needed, anyone the record authorizes can extend its lifetime. Not all data needs to live forever. A listing, a message or a day's sensor readings can be given a lifetime of a day or a week and paid for only for that time.

Expiration removes a record from the query surface. It does not erase history: retained history and archival services have their own availability limits, and a reader who copied public data keeps that copy.

Whether records expire at all is set for a DB-Chain as a whole. DB-Chain-1 enables it; another chain may not. On an expiring chain, a long-lived record lives as long as someone keeps renewing it.

> Each step of this lifecycle emits an event: created, updated, deleted, extended, owner changed. Applications can subscribe to them as they happen, instead of polling for changes.

THE SDK: A TECHNICAL TEASER

The TypeScript SDK wraps the lifecycle in a small API. The following example defines an entity and its schema, then writes, extends, queries, updates and deletes it:

```
npm i @arkiv-network/sdk
```

Define

```
const Posts = defineSchema("posts", {
  author: addr().indexed(),
  title: str().indexed(),
  body: bytes(),
  publishedAt: u64().indexed(),
  tag: str().indexed().optional(),
});
```

Write

```
const { entityKey: helloKey } = await walletClient.createEntity(
  Posts,
  {
    author: addr(account.address),
    title: "Hello from Arkiv",
    body: stringToBytes("Arkiv chain wishes you all the best!"),
    publishedAt: u64(Math.floor(Date.now() / 1000)),
  },
  { expiration: ExpirationTime.fromDays(30) },
);
```

Write immutable

```
const { entityKey: termsKey } = await walletClient.createEntity(
  Posts,
  {
    author: addr(account.address),
    title: "Terms of service v1",
    body: stringToBytes("These terms never change."),
    publishedAt: u64(Math.floor(Date.now() / 1000)),
  },
  {
    expiration: ExpirationTime.fromDays(90),
    flags: { readonly: true },
  },
);
```

Extend

```
await walletClient.extendEntity({
  entityKey: helloKey,
  expiration: ExpirationTime.fromDays(120),
});
```

Build with agents

Use our MCP and teach your agents Arkiv best practices.

```
npx skills add arkiv-network/skills
```

Query

```
const results = await publicClient
  .select({ author: true, title: true, body: true })
  .from(Posts)
  .where(eq("title", "Hello from Arkiv"))
  .ownedBy(account.address)
  .fetch();
```

Update

```
await walletClient.patchEntity(Posts, helloKey, {
  set: {
    body: stringToBytes("Edited: Arkiv chain wishes you all the best!"),
    tag: "announcement",
  },
});
```

Update: attempting to edit a read-only entity results in an error

```
try {
  await walletClient.patchEntity(Posts, termsKey, {
    set: { body: stringToBytes("Changed my mind.") },
  });
} catch (error) {
  console.log(
    "Patching a readonly entity is rejected:",
    (error as Error).message,
  );
}
```

Delete

```
const deleted = await walletClient.deleteEntity({
  entityKey: helloKey,
});
```

GOLEM DB

Golem DB is the database engine behind the reference implementation of a DB-Chain and Arkiv's first stable testnets. We built it as a general-purpose database that can run under any blockchain, or as the database behind any other application, Web2 included.

A DB-Chain has two parts. The blockchain handles consensus and history and, through its execution client, checks and prices each write; the database engine applies the state changes themselves. Arkiv treats them as separate concerns with a single interface between them. For that to work, the database has to do several things a chain depends on and that databases do not normally do together. It has to execute deterministically so every node computes the same state, commit to that state cryptographically and prove things against the commitment, and meter the work it does so the chain can charge for it. We looked at existing databases: each of these features turned up somewhere, but none had them all. So we wrote the requirements down as a specification and built Golem DB against it. The same specification is the reference for anyone building another engine later, or adapting an existing database to work with Arkiv.

A database first. Seen from an application, Golem DB is an ordinary record store with secondary indexes and a query language. A record has a key and a set of named, typed cells. An index exists from the first time an attribute is written, and the engine itself needs no schema. Queries filter records with boolean logic over their attributes, and results can be sorted and paged.

Determinism. The same operations on the same state give byte-identical results, ordering, costs and commitment on every node. Every result set has one fully specified order, or nodes would disagree about which records fall inside a page. If writes varied between nodes, the chain would fork; if queries varied, two nodes would answer the same question differently.

Commitment and proofs. After every block the engine produces one state commitment, a short fingerprint of its entire state, which for a DB-Chain plays the part of Ethereum's state root. Against it, anyone who holds only that commitment can check a record's cells, an index entry, or the absence of either.

Branches and budgets. A block is executed in a branch over the committed state. A transaction that fails is rolled back inside it, and a cost estimate is a run in a branch that is then discarded, as with eth_estimateGas. Every operation is metered in units of work and returns a receipt, and the chain charges for writes; a call may carry a budget and aborts when it runs out.

The boundary with Arkiv. The database engine's interface contains no blockchain vocabulary: no blocks, no transactions, no gas, no address semantics and no owner field. Arkiv's protocol fields, ownership among them, are stored as ordinary cells; to the engine the owner is a cell holding an Ethereum address, queryable like any other. Authorization happens in Arkiv's execution client before anything reaches the engine; the engine meters the work and returns a receipt, and Arkiv turns that receipt into the fee.

EMBRACING ETHEREUM

Arkiv positions itself within the Ethereum ecosystem and aligns with its values. Ethereum is the most widely used and battle-tested programmable blockchain, with the largest developer community in the space and a culture of open standards, credible neutrality and decentralization that Arkiv shares. The Arkiv chain is bridged to Ethereum, accounts are Ethereum addresses, users sign with their usual wallets, and the chains expose Ethereum-style JSON-RPC, so the tooling developers already have carries over. In time Arkiv and its DB-Chains intend to follow Ethereum's multi-client approach as well, the client diversity that keeps a network resilient against a bug in any one implementation.

Arkiv is a sidechain: each chain has its own validators. On the Arkiv chain and DB-Chain-1, security is expected to come from staking GLM, permissionless and non-custodial, done in the protocol itself. Blocks are final as soon as the validators agree on them, which holds as long as more than two thirds of the staked GLM is in honest hands. The current target is a consensus algorithm with single-slot finality (e.g. CometBFT). The Arkiv chain may also post its state commitment to Ethereum, which records it without verifying it. A DB-Chain may post its own state commitment to the Arkiv chain in the same way. Arkiv connects to Ethereum; its security comes from its own validators.

Benefits

For the Ethereum Ecosystem, Arkiv grows Ethereum's utility by bringing a new way of storing, processing and interacting with data, instead of one more chain reproducing the EVM. Its database engine is narrower in scope than a general-purpose virtual machine and better suited to data, and it makes reads more expressive, since a node can answer database-style queries rather than single lookups.

For Web2 Teams, Arkiv is a database designed so that no one party operates its public DB-Chains and no vendor controls the data, with a decade of blockchain practice built in: signed, attributable writes, payments settled on chain, and state that every participant can check.

For Web3 Builders, Arkiv widens what a chain can hold. Blockchains have so far created value mostly by moving assets and tracking state transitions in a ledger; application data that now sits in an operator's database can instead live on a chain its users can query and verify, which improves interoperability across the ecosystem and the utility of the chains already in it.

For DePIN, Arkiv offers a data layer built for marketplaces. Decentralized physical infrastructure networks need resource discovery, settlement and rich data exchange, and existing chains are often a poor fit for that load, which pushes projects toward centralized or peer-to-peer components. Arkiv's queryable records suit marketplace data, and a DB-Chain can be configured to a particular network's needs.

GLM AND STORAGE ECONOMICS

GLM, the token used for protocol fees on the Arkiv chain and on the DB-Chains Arkiv itself deploys, is an ERC-20 token on Ethereum; a DB-Chain launched by a third party may charge its fees in a different token. On the Golem Network, GLM supports decentralized compute: requestors use it in exchange for processing power from providers who contribute their machines. Arkiv was conceived and funded within the Golem ecosystem and extends GLM to Web3 data, where it pays for writes and, at mainnet, secures the protocol. GLM enters the Arkiv chain over the Ethereum bridge and reaches a DB-Chain only over a bridge from the Arkiv chain.

The price of a write reflects the resources it uses: the amount of data stored, the lifetime purchased, indexing work, computation during state processing and current storage load. Like gas on Ethereum, it also moves with demand for block space, through a base fee that rises when blocks fill up. The exact fee schedule is set in the chain's rules, and a cost estimate is available before a transaction is signed.

Expiration is part of this economic model. A temporary listing, event record or short-lived application state is paid for over the period it is needed and renewed if it is still wanted. Each chain sets a cap on how far ahead a lifetime can be bought; renewal is unlimited.

Reads and queries carry no protocol fee on DB-Chain-1. RPC providers may limit or charge for the service of answering them.

GLM is the preferred token across the Arkiv ecosystem. Support for paying in other tokens is planned as well, through account abstraction, a swap when assets are bridged from the Arkiv chain to a DB-Chain, or another mechanism; the design is still pending. Whichever is chosen, the protocol fee itself stays in one token per chain.

Other DB-Chains set their own economics. A chain charges its fees in GLM or in a single native token of its own.

THE PRACTICAL DIFFERENCE

Arkiv is designed to enable cost-efficient data management within Web3 while maintaining the familiar Web2 database interaction standards, such as query languages.

	Web2	Web3	Arkiv
Experience	Battle tested DB UX	UX tailored for transactional logic and asset transfer	Data-centric UX in a Web3 setting
Access	Centralized	Permissionless	Permissionless
Control	Control depends on host	Wallet-owned	Wallet-owned
Queries	Queryable data	Data not easily queryable	Queryable data
Cost	Subscription or usage billing	Gas per transaction; storage kept forever on most chains	Gas per transaction, priced by size and lifetime; renew to keep

WHAT YOU CAN BUILD

The patterns below are some of the many possible building blocks; the sample applications that follow show them in use.

Patterns

A decentralized database	The general case. An application's records belong to the wallets that wrote them, any client can query them, and the data does not depend on one operator staying up. The SDK gives you a schema, writes and queries, as with any database.
Discovery	A peer, service or device publishes a record that describes itself, with endpoint, capabilities and price as attributes and a lifetime. Others query by attribute, the publisher renews the record while it is alive, and whatever stops renewing disappears from the results.
Message queue	A message is a record with topic and sender as attributes, the payload as a stored cell, and a lifetime of hours or days. Consumers query by topic and page with a cursor, or subscribe to the created event instead of polling. The queue empties itself as messages expire.
Transport for a decentralized protocol	Participants exchange signed records and the protocol logic stays in the clients. There is no relay server to run or to trust, and every message carries its author's signature. A chat application, for example, can use Arkiv as its message transport.
Semaphore and control plane	A record can hold a setting instead of content: a feature flag, a configuration, a lock. Services and devices read it, its owner changes it, and every change is signed, so it is always clear who set the current state. A short-lived record that a node must keep renewing doubles as a heartbeat for a fleet of services or IoT devices.

Applications

Marketplaces and DePIN	Discovery for compute, rendering, AI inference, network and physical infrastructure, with offers that expire when they no longer stand. One of the first projects to incorporate this approach will be the Golem Network Marketplace.
Agents	Memory, task queues and shared state for AI agents, written and read through the MCP server or the SDK.
Bulletin boards and public knowledge	A post belongs to its author and a thread is a query over a topic attribute. Reference material can be written *immutable* and *extendable by anyone*, so readers who value it keep it alive.
Websites, newsletters and status pages	Content as records that renewal keeps live, with no hosting provider in the way.
Data interchange between institutions	Grid managers, local governments and other institutions write and own their own records in one queryable system. Provenance is in the signature; sensitive data is encrypted before it is written.
Transparent public records	A city's policies, budgets and reports published as records that anyone can query, signed by the office that issued them. A change is a new signed write, so nothing is altered quietly, and a document can be made immutable for its lifetime.
Events, tickets, transport and assets	Event information, tickets, public transport data, digital receipts, descriptors for real-world assets and NFT metadata.
Smart contract metadata	Supplementary information for tokens and contracts, such as icons, team names and websites, written by the deployer's wallet and queryable by contract address.
Public commitments	An immutable record that cannot change while it lives. Renew it to keep it live.

AN ECOSYSTEM FOR EVERYONE

An open environment where community-driven proposals play an important role results in a healthy ecosystem. Arkiv aligns its development with real-world use cases, and the platform it is building is one where you, whether you come from Web2, Web3 or DePIN, shape the future of data.

Get Involved

Join us on the journey to Data Autonomy with Arkiv. Whether you're a developer, visionary or decentralization enthusiast, your participation is key to shaping a future of data.

Join the community on Discord, contribute to development, and bring your use case: tell us what you want to build and what gets in the way.

Current Status

As of November 2026, DB-Chains run as public testnets. The Arkiv coordination chain comes next, and work is under way on decentralized consensus and the bridges. Further protocol changes are made through Arkiv Improvement Proposals (AIPs), reviewed in public and adopted by network participants. Development is in the open at github.com/Arkiv-Network.

Start building

You already know the workflow: model your data, declare which cells are attributes, write a record through the SDK and query it. Wallet ownership and lifetime rules come with it.

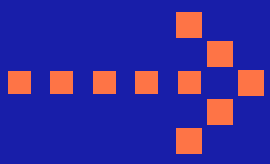
Build with the SDK, or let an agent build for you through the MCP server. The path to a first record in only a few minutes:

1. Create a wallet.
2. Go to hub.arkiv.network and get test funds from the faucet.
3. Write your first record with the SDK, or have an agent do it through MCP.
4. Get an API key on the hub when you need higher rate limits.

In an age where the decentralization ethos becomes increasingly important, collaboration becomes essential to ensure that the technological revolution benefits Everyone, not just a select few. Those who step forward now can help shape our digital future. This document is an invitation to be part of that journey.

See you at **arkiv.network**.

[ARKIV]



[A]