Introduction

This page will explain what The Graph is and how you can get started.

What The Graph Is

The Graph is a decentralized protocol for indexing and querying data from blockchains, starting with Ethereum. It makes it possible to query data that is difficult to query directly.

Projects with complex smart contracts like Uniswap and NFTs initiatives like Bored Ape Yacht Club store data on the Ethereum blockchain, making it really difficult to read anything other than basic data directly from the blockchain.

In the case of Bored Ape Yacht Club, we can perform basic read operations on the contract like getting the owner of a certain Ape, getting the content URI of an Ape based on their ID, or the total supply, as these read operations are programmed directly into the smart contract, but more advanced real-world queries and operations like aggregation, search, relationships, and non-trivial filtering are not possible. For example, if we wanted to query for apes that are owned by a certain address, and filter by one of its characteristics, we would not be able to get that information by interacting directly with the contract itself.

To get this data, you would have to process every single `transfer` event ever emitted, read the metadata from IPFS using the Token ID and IPFS hash, and then aggregate it. Even for these types of relatively simple questions, it would take **hours or even days** for a decentralized application (dapp) running in a browser to get an answer.

You could also build out your own server, process the transactions there, save them to a database, and build an API endpoint on top of it all in order to query the data. However, this option is resource intensive, needs maintenance, presents a single point of failure, and breaks important security properties required for decentralization.

Indexing blockchain data is really, really hard.

Blockchain properties like finality, chain reorganizations, or uncled blocks complicate this process further, and make it not just time consuming but conceptually hard to retrieve

correct query results from blockchain data.

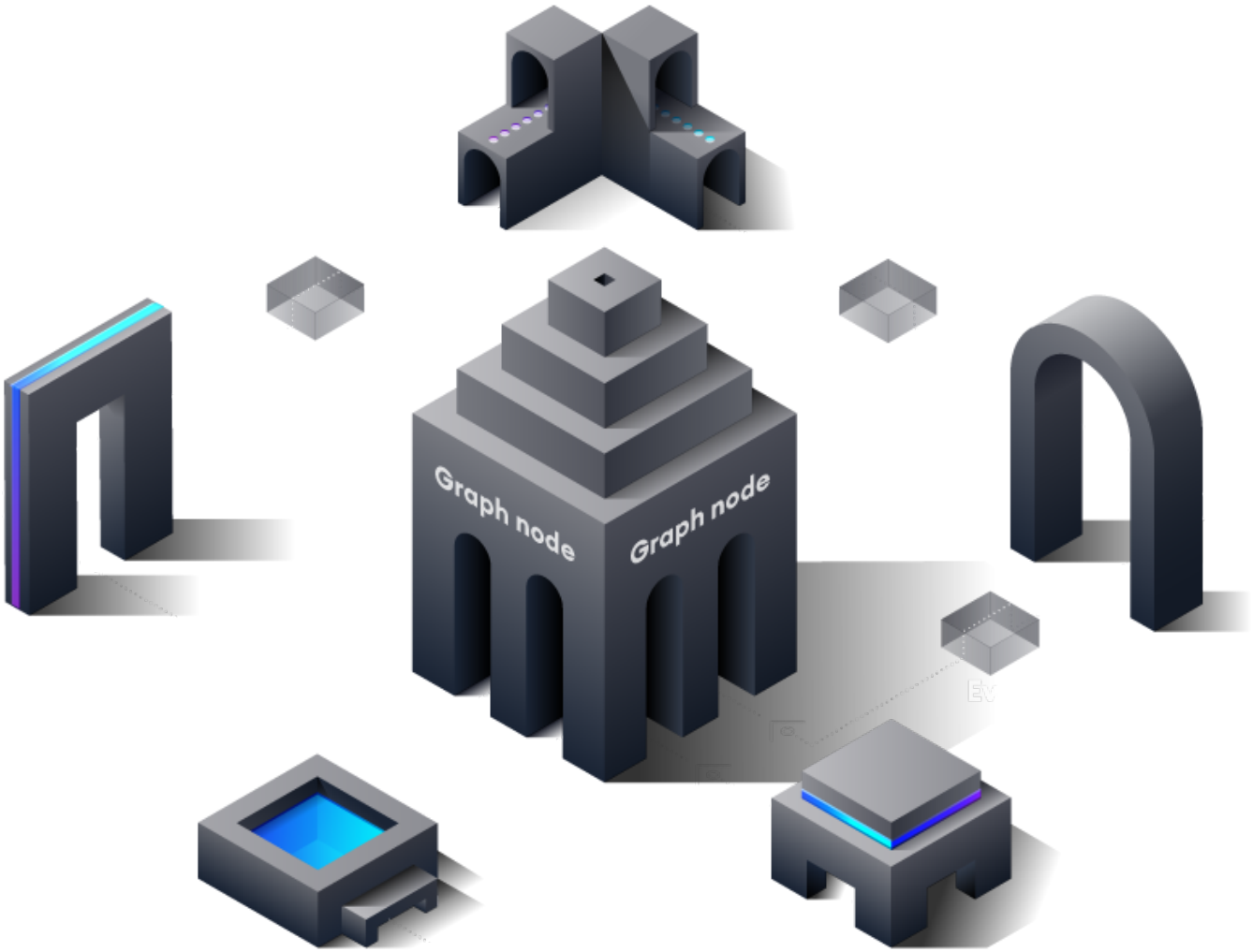
The Graph solves this with a decentralized protocol that indexes and enables the performant and efficient querying of blockchain data. These APIs (indexed "subgraphs") can then be queried with a standard GraphQL API. Today, there is a hosted service as well as a decentralized protocol with the same capabilities. Both are backed by the open source implementation of Graph Node.

How The Graph Works

The Graph learns what and how to index Ethereum data based on subgraph descriptions, known as the subgraph manifest. The subgraph description defines the smart contracts of interest for a subgraph, the events in those contracts to pay attention to, and how to map event data to data that The Graph will store in its database.

Once you have written a `subgraph manifest`, you use the Graph CLI to store the definition in IPFS and tell the indexer to start indexing data for that subgraph.

This diagram gives more detail about the flow of data once a subgraph manifest has been deployed, dealing with Ethereum transactions:



The flow follows these steps:

1. A decentralized application adds data to Ethereum through a transaction on a smart contract.
2. The smart contract emits one or more events while processing the transaction.
3. Graph Node continually scans Ethereum for new blocks and the data for your subgraph they may contain.
4. Graph Node finds Ethereum events for your subgraph in these blocks and runs the mapping handlers you provided. The mapping is a WASM module that creates or updates the data entities that Graph Node stores in response to Ethereum events.

5. The decentralized application queries the Graph Node for data indexed from the blockchain, using the node's GraphQL endpoint. The Graph Node in turn translates the GraphQL queries into queries for its underlying data store in order to fetch this data, making use of the store's indexing capabilities. The decentralized application displays this data in a rich UI for end-users, which they use to issue new transactions on Ethereum. The cycle repeats.

Next Steps

In the following sections we will go into more detail on how to define subgraphs, how to deploy them, and how to query data from the indexes that Graph Node builds.

Before you start writing your own subgraph, you might want to have a look at the Graph Explorer and explore some of the subgraphs that have already been deployed. The page for each subgraph contains a playground that lets you query that subgraph's data with GraphQL.