Minted NFTs Indexer

info NEAR QueryAPI is currently under development. Users who want to test-drive this solution need to be added to the allowlist before creating or forking QueryAPI indexers.

You can request access throughthis link .

Overview

This tutorial creates a working NFT indexer usingNEAR QueryAPI , and builds aNEAR component that presents the data. The indexer is watching fornft_mint Events and captures some relevant data:

- receiptId
- of theReceipt
- where the mint has happened
- receiverId
- Marketplace
- Links to the transaction on NEAR Explorer

In this tutorial you'll learn how you can listen toEvents generated by smart contracts and how you can index them.

tip The indexer's source code can be found byfollowing this link .

NFT Events

NEAR Protocol supportsEvents . TheseEvents allow a contract developer to add standardized logs to theExecutionOutcomes thus allowing themselves or other developers to read those logs in more convenient manner via API or indexers. Events have a fieldstandard which aligns with NEPs. In this tutorial we'll be talking aboutNEP-171 Non-Fungible Token standard .

The indexer watches all the NFTs minted following theNEP-171 Events standard. It should detect every single NFT minted, and store a basic set of data like: in what Receipt it was minted, and which marketplace created it (for example,Paras ,ShardDog , andMintbase).

Defining the Database Schema

The first step to creating an indexer is to define the database schema. This is done by editing theschema.sql file in the code editor. The schema for this indexer looks like this:

```
CREATE

TABLE "nfts"

( "id"

SERIAL

NOT

NULL , "marketplace"

TEXT , "block_height"

BIGINT , "block_timestamp"

BIGINT , "receipt_id"

TEXT , "receiver_id"

TEXT , "nft_data"

TEXT , PRIMARY

KEY

( "id" ,

"block_height" ,
```

"block_timestamp")) ; This schema defines one table:nfts . The table has these columns:

- id
- : a unique identifier for each row in the table
- marketplace
- : the marketplace where the NFT was created
- block_height
- : the height of the block in which the NFT was created
- block_timestamp
- : the timestamp of the block in which the NFT was created
- receipt_id
- : the receipt ID of the transaction that created the NFT
- receiver_id
- : the receiver ID of the transaction that created the NFT
- nft_data
- : the content of the minted NFT

Defining the indexing logic

The next step is to define the indexing logic. This is done by editing theindexingLogic.js file in the code editor. The logic for this indexer can be divided into two parts:

1. Filtering blockchain transactions for a specific type of transaction
2. Saving the data from the filtered transactions to the database

Filtering Blockchain transactions

The first part of the logic is to filter blockchain transactions for a specific type of transaction, where theEvent is a NEP-171 nft_mint . This is done by using thegetBlock function. This function takes in a block and a context and returns a promise. Theblock is a Near Protocol block, and thecontext is a set of helper methods to retrieve and commit state. ThegetBlock function is called for every block on the blockchain.

ThegetBlock function for this NFT indexer looks like this:

```
async

function

getBlock ( block :

Block )

{ for

( let ev of block . events ( ) )

{ const r = block . actionByReceiptId ( ev . relatedReceiptId ) ; const createdOn = block . streamerMessage . block . header . timestamp ;

try

{ let event = ev . rawEvent ;

if

( event . standard

===

"nep171"

&& event . event

===

"nft_mint" )

{ console . log ( event ) ;

let marketplace =
```

"unknown" ; if

(r . receiverId . endsWith (".paras.near")) marketplace =

"Paras" ; else

if

(r . receiverId . endsWith (".sharddog.near")) marketplace =

"ShardDog" ; else

if

(r . receiverId . match (/ . mintbase \d + . near

/)) marketplace =

"Mintbase" ;

const nftMintData =

{ marketplace : marketplace , block_height : block . header ( ) . height , block_timestamp : createdOn , receipt_id : r . receiptId , receiver_id : r . receiverId , nft_data :

JSON . stringify (event . data) , } ;

await context . db . Nfts . insert (nftMintData) ;

console . log (NFT by { r . receiptId } has been added to the database) ; } }

catch

(e)

{ console . log (e) ; } } } This indexer filters Blocks that have Events of typenft_mint and standardnep171 . In addition, it stores the JSON event data and identifies the NFT marketplace.

Saving the data to the Database

The second part of the logic is to save the data from the filtered transactions to the database. This is solved easily by using the context.db.Nfts.insert helper method:

The logic for this looks like:

const nftMintData =

{ marketplace : marketplace , block_height : h , block_timestamp : createdOn , receipt_id : r . receiptId , receiver_id : r . receiverId , nft_data :

JSON . stringify (event . data) , } ;

// store result to the database await context . db . Nfts . insert (nftMintData) ;

NEAR Component

The final step is querying the indexer using GraphQL from a NEAR component with WebSockets.

Here's a simple GraphQL query that gets the last{LIMIT} minted NFTs:

IndexerQuery

{ bucanero_near_nft_v4_nfts (order_by :

{ block_timestamp :

desc } ,

limit :

{ LIMIT })

{ block_height block_timestamp id marketplace nft_data receipt_id receiver_id }

Setup

Here's a code snippet that subscribes and processes the most recent activity (last 10 NFTs) from theNFT indexer :

tip The code below is only a snippet. If you want the full source code to play around with the component, you can fork theNFT Activity Feed source code and build your own NEAR component. const

GRAPHQL_ENDPOINT

=

"near-queryapi.api.pagoda.co" ;

const

LIMIT

=

10 ; const accountId = props . accountId

||

"bucanero.near"

|| context . accountId ;

State . init ({ widgetActivities :

[ ] , widgetActivityCount :

0 , startWebSocketWidgetActivity :

null , initialFetch :

false , }) ;

const widgetActivitySubscription =

subscription IndexerQuery { bucanero_near_nft_v4_nfts(order_by: {block_timestamp: desc}, limit: { LIMIT }) { block_height block_timestamp id marketplace nft_data receipt_id receiver_id } } ;

const subscriptionWidgetActivity =

{ type :

"start" , id :

"widgetNftActivity" ,

// You can use any unique identifier payload :

{ operationName :

"IndexerQuery" , query : widgetActivitySubscription , variables :

{ } , } , } ; function

processWidgetActivity (activity)

{ return

{

... activity } ; } function

startWebSocketWidgetActivity (processWidgetActivities)

{ let ws =

State . get ( ) . ws_widgetActivity ;

if

(ws)

{ ws . close ( ) ; return ; }

ws

new

WebSocket (wss:// { GRAPHQL_ENDPOINT } /v1/graphql ,

"graphql-ws") ;

ws . onopen

=

( )

=>

{ console . log (Connection to WS has been established) ; ws . send (JSON . stringify ({ type :

"connection_init" , payload :

{ headers :

{ "Content-Type" :

"application/json" , "Hasura-Client-Name" :

"hasura-console" , "x-hasura-role" :

"bucanero_near" , } , lazy :

true , } , })) ;

setTimeout (( )

=> ws . send (JSON . stringify (subscriptionWidgetActivity)) ,

50) ; } ;

ws . onclose

=

( )

=>

{ State . update ({

ws_widgetActivity :

null

}) ; console . log (WS Connection has been closed) ; } ;

ws . onmessage

=

(e)

=>

{ const data =

JSON . parse (e . data) ; console . log ("received data" , data) ; if

(data . type

===

"data"

&& data . id

===

"widgetNftActivity")

{ processWidgetActivities (data . payload . data) ; } } ;

ws . onerror

=

(err)

=>

{ State . update ({

ws_widgetActivity :

null

}) ; console . log ("WebSocket error" , err) ; } ;

State . update ({

ws_widgetActivity : ws }) ; } info Pay attention to thewidgetActivitySubscription GraphQL query and thesubscriptionWidgetActivity JSON payload.

Processing

This is the JS function that process the incoming widget activities generated by the QueryAPI indexer, allowing the NEAR component to create a feed based on the blockchain's widget activity:

tip You can fork the[NFT Activity Feed source code] and build your own NEAR component. function

processWidgetActivities (incoming_data)

{ let incoming_widgetActivities = incoming_data . bucanero_near_nft_v4_nfts . flatMap (processWidgetActivity) ; const newActivities =

[... incoming_widgetActivities . filter ((activity)

=>

{ return

(state . widgetActivities . length

==

0

|| activity . block_timestamp

    state . widgetActivities [0] . block_timestamp) ; }) ,] ; if

(newActivities . length

0

&& state . widgetActivities . length

0)

{ } const prevActivities = state . prevActivities

||

[ ] ; State . update ({

widgetActivities :

```
[ ... newActivities ,

... prevActivities ]

} ) ; }

if

( state . ws_widgetActivity

===

undefined )

{ State . update ( { startWebSocketWidgetActivity : startWebSocketWidgetActivity , } ) ; state . startWebSocketWidgetActivity
( processWidgetActivities ) ; }
```

Rendering

Finally, rendering the activity feed on the NEAR component is straight-forward, by iterating through thestate.widgetActivities map:

```
return

( < div

    < Title

    NFT

Minting

Activity

Feed { " " } < TextLink href = "https://near.org/dataplatform.near/widget/QueryApi.App"

    { " " } Powered

By

QueryAPI { " " } < / TextLink

    < / Title

    < RowContainer

    { state . widgetActivities . map ( ( activity , i )

=>

( < Card

    < div

    < Widget src = "mob.near/widget/TimeAgo" props = { {

blockHeight : activity . block_height

} } /

    { " " } ago < / div

    < CardBody

    < div key = { i }

    < Text bold

    NFT

Marketplace :

{ activity . marketplace } < / Text
```

```
        < TextLink href = {https://nearblocks.io/address/ { activity . receiver_id }  }
        { activity . receiver_id } < / TextLink
        < Text bold
        Receipt
ID :
{ activity . receipt_id } < / Text
        < / div
        < / CardBody
        < CardFooter
        < TextLink href = {https://legacy.nearblocks.io/?query= { activity . receipt_id } }
        View details on NEAR
Explorer < / TextLink
        < / CardFooter
        < / Card
        ) ) } < / RowContainer
        < / div
        ) ;
```
Edit this page Last updatedonJan 31, 2024 bygagdiez Was this page helpful? Yes No

Previous Hype Indexer Next Social Feed Indexer