Websockets

Stream real-time data directly to your applications with our websocket integration.

What are Websockets?

Websockets allow for two-way communication between a client and a server. Unlike traditional request-response models, Websockets keep a persistent connection open, enabling real-time data exchange. This is perfect for applications that require instant updates like chat apps, online games, trading bots, and marketplaces.

Helius supports all stable Solana Websockets. You can find a list of all these Websockets in the Solana documentation . You can use these with your Helius WSS URL:. Mainnet: wss://mainnet.helius-rpc.com/?api-key=

Devnet: wss://devnet.helius-rpc.com/?api-key=

Helius' Geyser Enhanced Websockets (beta)

In addition to supporting Solana's standard Websocket methods, Helius provides access to Geyser-enhanced Websockets. These Websockets boast faster response speeds when compared to the standard RPC Websocket methods. These Websockets are currently in beta. We would love your feedback on Discord!

Why Use Helius Websockets?

* Direct Streaming * : No more polling or periodic checks. Get data directly when changes happen. * Seamless Integration * : Easily integrate our websockets into your application at any layer. Websockets can be used directly in your frontend! * Cost-Effective * : Reduce the overhead and expenses that come with frequently polling data. * Built on top of Geyser * : Instantly react to live on-chain data changes by getting a direct feed into Geyser data. Read more about Geyser * here * . This feature is only available for paid users while still in beta.

Subscription Endpoints

Helius websockets are currently available in mainnet and devnet with the following URL's Mainnet:

wss://atlas-mainnet.helius-rpc.com?api-key= Devnet: wss://atlas-devnet.helius-rpc.com?api-key=

Transaction Subscribe

While Solana offers several websocket methods for various functionalities, it notably lacks a dedicated method for subscribing to transaction updates. To bridge this gap, Helius introduces the transactionSubscribe websocket method. This allows developers to tune in to real-time transaction events. In order to subscribe to transactions, you need to provide a TransactionSubscribeFilter and can optionally provide TransactionSubscribeOptions Example Payload: { "jsonrpc" :

"2.0" , "id" :

420 , "method" :

"transactionSubscribe" , "params" :

[{ "vote" : false, "failed" : false, "signature" :

"2dd5zTLrSs2udfNsegFRCnzSyQcPrM9svX6m1UbEM5bSdXXFj3XpqaodtKarLYFP2mTVUsV27sRDdZCgcKhjeD9S" , "accountInclude" :

["pqx3fvvh6b2eZBfLhTtQ5KxzU3CginmgGTmDCjk8TPP"] , "accountExclude" :

["FbfwE8ZmVdwUbbEXdq4ofhuUEiAxeSk5kaoYrJJekpnZ"] , "accountRequired" :

["As1XYY9RdGkjs62isDhLKG3yxMCMatnbanXrqU85XvXW"] } , { "commitment" :

"processed" , "encoding" :

"base64" , "transaction_details" :

"full" , "showRewards" : true, "maxSupportedTransactionVersion" :

0 }] } Example Notification: { "jsonrpc" :

"2.0" , "method" :

"transactionNotification" , "params" :

{ "subscription" :

4743323479349712 , "result" :

{ "transaction" :

{ "transaction" :

[
"Ae6zfSExLsJ/E1+q0jI+3ueAtSoW+6HnuDohmuFwagUo2BU4OpkSdUKYNI1dJfMOonWvjaumf4Vv1ghn9f3Avg0BAAEDGycH0OcYRpfnPNuu0DBQxTYPWpmwHdXPjb8y2P200JgK3hGiC2JyC9qjTd2In
, "base64"] , "meta" :

{ "err" : null, "status" :

{ "Ok" : null } , "fee" :

5000 , "preBalances" :

[28279852264 , 158122684 , 1] , "postBalances" :

[28279747264 , 158222684 , 1] , "innerInstructions" :

[] , "logMessages" :

["Program 11111111111111111111111111111111 invoke [1]" , "Program 11111111111111111111111111111111 success"] , "preTokenBalances" :

[] , "postTokenBalances" :

[] , "rewards" : null, "loadedAddresses" :

{ "writable" :

[] , "readonly" :

[] } , "computeUnitsConsumed" :

0 } } , "signature" :

"5moMXe6VW7L7aQZskcAkKGQ1y19qqUT1teQKBNAAmipzdxdqVLAdG47WrsByFYNJSAGa9TByv15oygnqYvP6Hn2p" } } }

TransactionSubscribeFilter:

vote: Option : A flag to include/exclude vote-related transactions. failed: Option : A flag to include/exclude transactions that failed. signature: Option: Filters updates to a specific transaction based on its signature. accountInclude: Option> : A list of accounts for which you want to receive transaction updates. This means that only one of the accounts must be included in the transaction updates (e.g., Account 1 OR Account 2). accountExclude: Option> : A list of accounts you want to exclude from transaction updates. accountRequired: Option> : Transactions must involve these specified accounts to be included in updates. This means that all of the accounts must be included in the transaction updates (e.g., Account 1 AND Account 2). You can include up to 50 000 addresses in the accountsInclude, accountExclude and accountRequired arrays.

Optional - TransactionSubscribeOptions:

commitment: Option : Specifies the commitment level for fetching data, dictating at what stage of the transactions lifecycle updates are sent. The possible values are * processed * confirmed * finalized encoding: Option : Sets the encoding format of the returned transaction data. The possible values are * base58 * base64 * base64+zstd * jsonParsed transactionDetails: OptionDetermines the level of detail for the returned transaction data. The possible values are * full * signatures * accounts * none showRewards: Option : A boolean flag indicating if reward data should be included in the transaction updates. maxSupportedTransactionVersion: Option : Specifies the highest version of transactions you want to receive updates for.

Transaction Subscribe code example:

In this example we are subscribing to transactions that contain the account 8uDncGPHZJtFnyCktVSbdWmgt7xAiHgwq8ZmxKaK5ZJ1 . Whenever a transaction occurs that contains 8uDncGPHZJtFnyCktVSbdWmgt7xAiHgwq8ZmxKaK5ZJ1 in the accountKeys of the transaction, we will receive a websocket notification. Based on the subscription options, the transaction notification will be sent at the processed commitment level, with base64 encoding, full transaction details, and will show rewards. const WebSocket =

require ('ws'); // Create a WebSocket connection const ws =

new

WebSocket ('wss://atlas-mainnet.helius-rpc.com?api-key='); // Function to send a request to the WebSocket server function

sendRequest (ws)

{ const request =

{ jsonrpc :

"2.0" , id :

420 , method :

"transactionSubscribe" , params :

[{ accountInclude :

["8uDncGPHZJtFnyCktVSbdWmgt7xAiHgwq8ZmxKaK5ZJ1"] }, { commitment :

"processed" , encoding :

"base64" , transactionDetails :

"full" , showRewards :

true , maxSupportedTransactionVersion :

0 }] }; ws . send (JSON . stringify (request)); }   // Define WebSocket event handlers ws . on ('open' ,

function

open ()

{ console . log ('WebSocket is open'); sendRequest (ws);

// Send a request once the WebSocket is open }); ws . on ('message' ,

function

incoming (data)

{ const messageStr = data . toString ('utf8'); try

{ const messageObj =

JSON . parse (messageStr); console . log ('Received:' , messageObj); }

catch

(e)

{ console . error ('Failed to parse JSON:' , e); } }); ws . on ('error' ,

function

error (err)

{ console . error ('WebSocket error:' , err); }); ws . on ('close' ,

function

close ()

{ console . log ('WebSocket is closed'); });

Account Subscribe

Helius Websockets supports a method to subscribe to an account and receive notifications over the websocket connection when the lamports or data for a matching account public key changes. This method matches the Solana Websocket API accountSubscribe spec exactly. See the Solana docs for more: https://docs.solana.com/api/websocket#accountsubscribe Example Payload: { "jsonrpc" :

"2.0" , "id" :

420 , "method" :

"accountSubscribe" , "params" :

["SysvarC1ock11111111111111111111111111111111" , { "encoding" :

"jsonParsed" , "commitment" :

"finalized" }] } Example Notification: { 'jsonrpc' :

'2.0' , 'method' :

'accountNotification' , 'params' : { 'subscription' :

237508762798666 , 'result' : { 'context' :

{ 'slot' :

235781083 } , 'value' : { 'lamports' :

1169280 , 'data' :

'BvEhEb6hixL3QPn41gHcyi2CDGKt381jbNKFFCQr6XDTzCTXCuSUG9D' , 'owner' :

'Sysvar1111111111111111111111111111111111111' , 'executable' : False, 'rentEpoch' :

361 , 'space' :

40 } } } }

Parameters:

String : Account public key, sent in base58 format, required object: Option: Optional object used to pass in encoding for the data returned in the AccountNotification and commitment If nothing is passed into the object, the response will default to base58 encoding and a finalized commitment level. pub

struct

RpcAccountInfoConfig

{ pub encoding :

Option < UiAccountEncoding

,

// supported values: base58, base64, base64+zstd, jsonParsed pub commitment :

Option < CommitmentConfig

,

// supported values: finalized, confirmed, processed - defaults to finalized }

Account Subscribe code example:

In this example we are subscribing to account changes for the account SysvarC1ock11111111111111111111111111111111 .

Whenever a change occurs to the account data or the lamports for this account, we will see an update. This happens at a frequent interval for this specific account as the slot and unixTimestamp are both a part of the returned account data. // Create a WebSocket connection const ws =

new

WebSocket ('wss://atlas-mainnet.helius-rpc.com?api-key='); // Function to send a request to the WebSocket server function

sendRequest (ws)

{ const request =

{ jsonrpc :

"2.0" , id :

420 , method :

"accountSubscribe" , params :

["SysvarC1ock11111111111111111111111111111111" ,

// pubkey of account we want to subscribe to { encoding :

"jsonParsed" ,

// base58, base64, base65+zstd, jsonParsed commitment :

"confirmed" ,

// defaults to finalized if unset }] }; ws . send (JSON . stringify (request)); } // Define WebSocket event handlers ws . on ('open' ,

function

open ()

{ console . log ('WebSocket is open'); sendRequest (ws);

// Send a request once the WebSocket is open }); ws . on ('message' ,

function

incoming (data)

{ const messageStr = data . toString ('utf8'); try

{ const messageObj =

JSON . parse (messageStr); console . log ('Received:' , messageObj); }

catch

(e)

{ console . error ('Failed to parse JSON:' , e); } }); ws . on ('error' ,

function

error (err)

{ console . error ('WebSocket error:' , err); }); ws . on ('close' ,

function

close ()

{ console . log ('WebSocket is closed'); });Previous FAQ Next- Solana APIs Enhanced Transactions API Last modified1mo ago