---

title: Using WebSockets description: Guide to using WebSockets and Alchemy to make JSON-RPC requests and subscribe to events. author: "Elan Halpern" lang: en tags: ["alchemy", "websockets", "querying", "javascript"] skill: beginner source: Alchemy docs sourceUrl: https://docs.alchemyapi.io/guides/using-websockets published: 2020-12-01

This is an entry level guide to using WebSockets and Alchemy to make requests to the Ethereum blockchain.

WebSockets vs. HTTP {#websockets-vs-http}

Unlike HTTP, with WebSockets, you don't need to continuously make requests when you want specific information. WebSockets maintain a network connection for you (if done right) and listen for changes.

As with any network connection, you should not assume that a WebSocket will remain open forever without interruption, but correctly handling dropped connections and reconnection by hand can be challenging to get right. Another downside of WebSockets is that you do not get HTTP status codes in the response, but only the error message.

Alchemy Web3 automatically adds handling for WebSocket failures and retries with no configuration necessary.

Try it out {#try-it-out}

The easiest way to test out WebSockets is to install a command line tool for making WebSocket requests such as wscat. Using wscat, you can send requests as follows:

Note: if you have an Alchemy account you can replace `demo` with your own API key. Sign up for a free Alchemy account here!

``` wscat -c wss://eth-mainnet.ws.alchemyapi.io/ws/demo

> {"jsonrpc": "2.0", "id": 0, "method": "eth_gasPrice"}

< {"jsonrpc": "2.0", "result": "0xb2d05e00", "id": 0}

```

How to use WebSockets {#how-to-use-websockets}

To begin, open a WebSocket using the WebSocket URL for your app. You can find your app's WebSocket URL by opening the app's page in your dashboard and clicking "View Key". Note that your app's URL for WebSockets is different from its URL for HTTP requests, but both can be found by clicking "View Key".

Any of the APIs listed in the Alchemy API Reference can be used via WebSocket. To do so, use the same payload that would be sent as the body of a HTTP POST request, but instead send that payload through the WebSocket.

With Web3 {#with-web3}

Transitioning to WebSockets while using a client library like Web3 is simple. Simply pass the WebSocket URL instead of the HTTP one when instantiating your Web3 client. For example:

```js const web3 = new Web3("wss://eth-mainnet.ws.alchemyapi.io/ws/your-api-key")

web3.eth.getBlockNumber().then(console.log) // -> 7946893 ```

## Subscription API {#subscription-api}

When connected through a WebSocket, you may use two additional methods: `eth_subscribe` and `eth_unsubscribe`. These methods will allow you to listen for particular events and be notified immediately.

### `eth_subscribe` {#eth-subscribe}

Creates a new subscription for specified events. Learn more about `eth_subscribe`.

#### Parameters {#parameters}

1. Subscription types
2. Optional params

The first argument specifies the type of event for which to listen. The second argument contains additional options which depend on the first argument. The different description types, their options, and their event payloads are described below.

#### Returns {#returns}

The subscription ID: This ID will be attached to any received events, and can also be used to cancel the subscription using `eth_unsubscribe`.

#### Subscription events {#subscription-events}

While the subscription is active, you will receive events which are objects with the following fields:

- `jsonrpc`: Always "2.0"
- `method`: Always "eth_subscription"
- `params`: An object with the following fields:
- `subscription`: The subscription ID returned by the `eth_subscribe` call which created this subscription.
- `result`: An object whose contents vary depending on the type of subscription.

#### Subscription types {#subscription-types}

1. `alchemy_newFullPendingTransactions`

Returns the transaction information for all transactions that are added to the pending state. This subscription type subscribes to pending transactions, similar to the standard Web3 call `web3.eth.subscribe("pendingTransactions")`, but differs in that it emits *full transaction information* rather than just transaction hashes.

Example:

```json

> {"jsonrpc": "2.0", "id": 1, "method": "eth_subscribe", "params": ["alchemy_newFullPendingTransactions"]}

< {"id":1,"result":"0x9a52eeddc2b289f985c0e23a7d8427c8","jsonrpc":"2.0"} < { "jsonrpc":"2.0", "method":"eth_subscription", "params":{ "result":{ "blockHash":null, "blockNumber":null, "from":"0xa36452fc31f6f482ad823cd1cf5515177d57667f", "gas":"0x1adb0", "gasPrice":"0x7735c4d40", "hash":"0x50bff0736c713458c92dd1848d12f3354149be1363123dae35e94e0f2a9d56bf",

"input":"0xa9059cbb000000000000000000000000d0707963952f2fba59dd06f2b425ace40b492fe00000000000000000000000000000000000000000000015b1111266cfca100000", "nonce":"0x0", "to":"0xea38eaa3c86c8f9b751533ba2e562deb9acded40", "transactionIndex":null, "value":"0x0", "v":"0x26", "r":"0x195c2c1ed126088e12d290aa93541677d3e3b1d10f137e11f86b1b9227f01e3b", "s":"0x60fc4edbf1527832a2a36dbc1e63ed6193a6eee654472fbebbf88ef1750b5344"}, "subscription":"0x9a52eeddc2b289f985c0e23a7d8427c8" } }

```

1. `newHeads`

Emits an event any time a new header is added to the chain, including during a chain reorganization.

When a chain reorganization occurs, this subscription will emit an event containing all new headers for the new chain. In particular, this means that you may see multiple headers emitted with the same height, and when this happens the later header should be taken as the correct one after a reorganization.

Example:

```json

{"jsonrpc": "2.0", "id": 1, "method": "eth_subscribe", "params": ["newHeads"]}

< {"jsonrpc":"2.0","id":2,"result":"0x9ce59a13059e417087c02d3236a0b1cc"} < { "jsonrpc": "2.0", "method": "eth_subscription", "params": { "result": { "extraData": "0xd983010305844765746887676f312e342e328777696e646f7773", "gasLimit": "0x47e7c4", "gasUsed": "0x38658", "logsBloom": "0x00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 "nonce": "0x084149998194cc5f", "number": "0x1348c9", "parentHash": "0x7736fab79e05dc611604d22470dadad26f56fe494421b5b333de816ce1f25701", "receiptRoot": "0x2fab35823ad00c7bb388595cb46652fe7886e00660a01e867824d3dceb1c8d36", "sha3Uncles": "0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d49347", "stateRoot": "0xb3346685172db67de536d8765c43c31009d0eb3bd9c501c9be3229203f15f378", "timestamp": "0x56ffeff8", "transactionsRoot": "0x0167ffa60e3ebc0b080cdb95f7c0087dd6c0e61413140e39d94d3468d7c9689f" }, "subscription": "0x9ce59a13059e417087c02d3236a0b1cc" } }

```

1. `logs`

Emits logs which are part of newly added blocks that match specified filter criteria.

When a chain reorganization occurs, logs which are part of blocks on the old chain will be emitted again with the property `removed` set to `true`. Further, logs which are part of the blocks on the new chain are emitted, meaning that it is possible to see logs for the same transaction multiple times in the case of a reorganization.

Parameters

1. An object with the following fields:
2. `address` (optional): either a string representing an address or an array of such strings.
   - Only logs created from one of these addresses will be emitted.
3. `topics`: an array of topic specifiers.
   - Each topic specifier is either `null`, a string representing a topic, or an array of strings.
   - Each position in the array which is not `null` restricts the emitted logs to only those who have one of the given topics in that position.

Some examples of topic specifications:

- `[]`: Any topics allowed.
- `[A]`: A in first position (and anything after).
- `[null, B]`: Anything in first position and B in second position (and anything after).
- `[A, B]`: A in first position and B in second position (and anything after).
- `[[A, B], [A, B]]`: (A or B) in first position and (A or B) in second position (and anything after).

Example:

```json

{"jsonrpc": "2.0", "id": 1, "method": "eth_subscribe", "params": ["logs", {"address": "0x8320fe7702b96808f7bbc0d4a888ed1468216cfd", "topics": ["0xd78a0cb8bb633d06981248b816e7bd33c2a35a6089241d099fa519e361cab902"]}]}

< {"jsonrpc":"2.0","id":2,"result":"0x4a8a4c0517381924f9838102c5a4dcb7"} < { "jsonrpc": "2.0", "method": "eth_subscription", "params": { "subscription": "0x4a8a4c0517381924f9838102c5a4dcb7", "result": { "address": "0x8320fe7702b96808f7bbc0d4a888ed1468216cfd", "blockHash": "0x61cdb2a09ab99abf791d474f20c2ea89bf8de2923a2d42bb49944c8c993cbf04", "blockNumber": "0x29e87", "data": "0x00000000000000000000000000000000000000000000000000000000000000010000000000000000000000000000000000000000000000000000000000000003", "logIndex":"0x0", "topics": ["0xd78a0cb8bb633d06981248b816e7bd33c2a35a6089241d099fa519e361cab902"], "transactionHash": "0xe044554a0a55067caafd07f8020ab9f2af60bdfe337e395ecd84b4877a3d1ab4", "transactionIndex": "0x0" } } }

```

`eth_unsubscribe` {#eth-unsubscribe}

Cancels an existing subscription so that no further events are sent.

Parameters

1. Subscription ID, as previously returned from an `eth_subscribe` call.

Returns

`true` if a subscription was successfully cancelled, or `false` if no subscription existed with the given ID.

Example:

Request

``` curl https://eth-mainnet.alchemyapi.io/v2/your-api-key -X POST -H "Content-Type: application/json" -d '{"id": 1, "method": "eth_unsubscribe", "params": ["0x9cef478923ff08bf67fde6c64013158d"]}'

```

Result

`json { "jsonrpc": "2.0", "id": 1, "result": true }`

Sign up with Alchemy for free, check out our documentation, and for the latest news, follow us on Twitter.
```