Hibachi API

Welcome to Hibachi!

This is our API documentation.

Hibachi provides 2 types of API:

- REST API
- Websocket API

The REST API provides all functionality access with easy-to-use request and response models.

Websocket API provides lower latency access to real time trading and data.

Key Definitions

- **API Key**: An API key is an alphanumeric string that is used to identify and authenticate an application or user to interact with API/WS endpoints. This is required for almost ALL endpoints covered in this documentation except `/market` REST and `/ws/market` WS endpoints. This key is NOT used for signing; that is done using your Private/Secret key.
- **Private/Secret Key**: A private key is a cryptographic code that's used to sign transactions and to prove ownership of your Hibachi Account. NOTE: This is NOT your crypto wallet's private key. You will ONLY use this key to generate `signatures` for a **subset** of endpoints you are interacting with.
- **[For Gated Launch] Access Code:** An Access Code is an alphanumeric string that is used to gain access to Hibachi UI and `/market` REST and `/ws/market` WS endpoints **ONLY** During the Gated Launch Period.
- **Trustless Account**: A trustless account is a Hibachi Account you created via directly connecting to your crypto wallet. You will generate a signature with your `ECDSA Private Key` using the ECDSA method.
- **Exchange Managed Account**: An exchange managed account is a Hibachi Account you created via email or social login. You will generate a signature with `HMAC Secret Key` using the HMAC method.
- **Asset Quantities**:All quantities on Hibachi are represented internally as **64-bit** numbers. In order to enable trading fractional quantities (e.g. 0.01 BTC), each asset on Hibachi has a specific fixed number of decimals.
  - Example
    - *BTC* may be represented with 10 decimals
    - This means *1 BTC* is stored as *1 × 10^10 = 10,000,000* quantity on the exchange
  - When making trades, querying positions and similar, Hibachi will still expose and expect numbers as "real" float strings (e.g. "quantity: 1.0" for 1 BTC).
  - This way, users don't need to constantly convert numbers in order to be able to reason about them. However, this quantity notation is important when:
    - Signing any operations (trades, withdrawals, etc) since we want the traders to specify exactly what they want to trade and not leave room for misinterpretations.
    - Interacting with the Hibachi smart contracts
  - **The number of decimals is exposed for each contract in the `/market/exchange-info` endpoint response under the `underlyingDecimals` field**.
- **Asset Prices**: Like quantities, asset prices are also **64-bit** numbers that conform to decimals. Prices are used

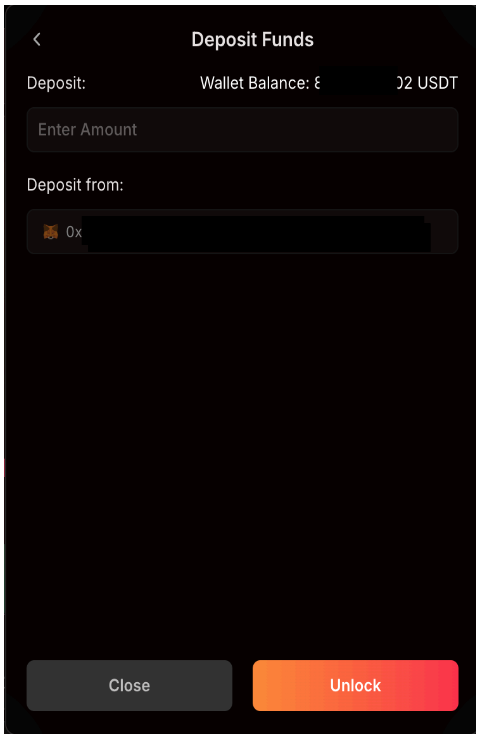
when converting between underlying and the settlement token in a market.

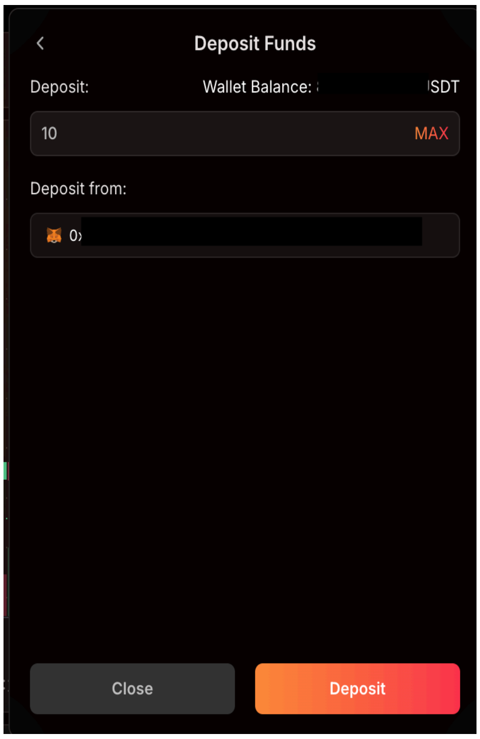
- For example, on BTC/USDT-P a price of 100,000 may be used in a limit order to indicate that somebody is willing to sell BTC for 100,000 USDT per BTC. Price is always denoted in units of the settlementAsset.
- Since both the underlying and settlement assets have unique quantity decimals, the price is stored in the difference between both decimals (settlementDecimals - underlyingDecimals).
- In order to support higher resolution of prices, an additional price multiplier of 2^32 is used to represent prices as fixed-point decimals with 32 bits after the decimal point.

Pre-requisites

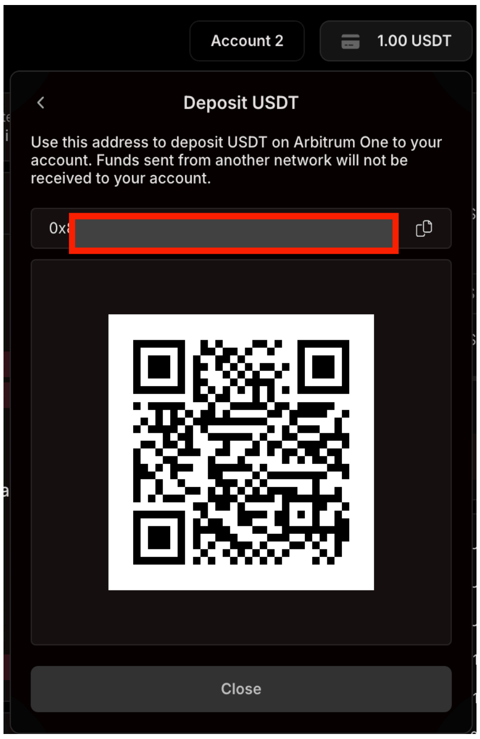
1. **[For Gated Launch]** You have been provided with Access Code to Hibachi UI (https://hibachi.xyz) and have created an account on the UI.
2. You **MUST** make one Deposit with **USDT** on **ARBITRUM ONE** to your account. (Your Account ID will not be generated until your first deposit; this ID is required for a lot of endpoints

- Trustless Wallet: Open the "Wallet Balance" Dropdown, and select "Deposit". First you need to `Unlock` your wallet with proper allowance, then you can enter the amount you would want to deposit to your Hibachi Account.





- Exchange Managed Wallet:Open the "Wallet Balance" Dropdown, and select "Deposit" to reveal wallet deposit address, then send USDT from any of your crypto wallets that holds USDT to this deposit address



Use this address to deposit USDT on Arbitrum One to your account. Funds sent from another network will not be received to your account.

0x [redacted]

Getting Started

1. Create an API Key

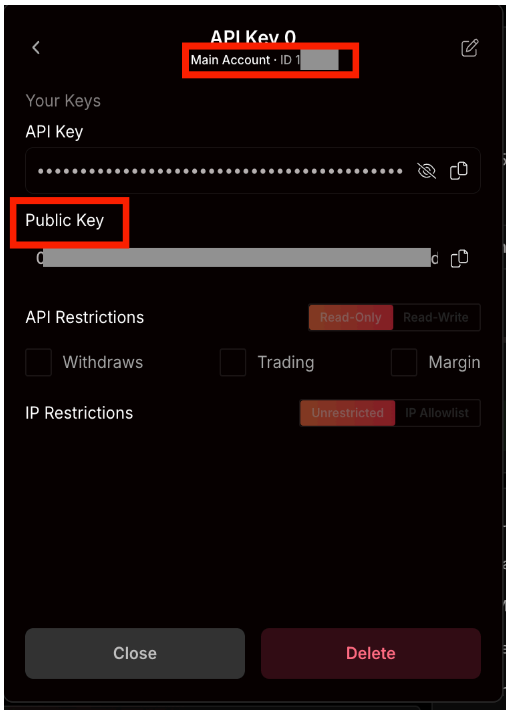
- After you login to your Hibachi Account from the UI, click on the gear button on the top right corner, then select **API Keys**
- Click on **Create API Key**, then follow below definition to choose the correct setting for your API Key, and click on **Generate**
    - **Select Account:** This is to choose the account you want to associate with this API Key. Note you cannot use an API Key you created to access another account you own

use an API Key you created to access another account you own.

- **Name of the API Key:** a name you want to associate with this API Key

- **API Restrictions:** you can choose if this API Key will have `read-only` or `read-write` permissions. **IMPORTANT**, by default a `read-only` permission is selected
  - `read-write` **permissions:**
    - **Trading:** This API Key will be place orders
    - **Withdraws:** This API Key will be able to withdraw funds from the exchange to outside destinations
    - Transfers: This API Key will be able to transfer funds to other sub-accounts related to the main account. This will not allow transferring to other users OR withdrawing unless the withdraw permission is present

- On the next page, **save the API Key and Private Key to a safe vault** (e.g password manager etc). NOTE: You will ONLY be able to see the Private Key during generation, so make sure you save it before exiting the page
- Click on **Next**, save below information that will be needed for API Endpoints interaction together with your API Key
  - Account ID: you will see `ID: XXX` below your API Key Name
  - Public Key



API Key 0
Main Account · ID 1[redacted]

Your Keys
API Key
••
Public Key
0[redacted]d

API Restrictions — Read-Only | Read-Write
Withdraws | Trading | Margin
IP Restrictions — Unrestricted | IP Allowlist

Close | Delete

2. API Authorization

- In **every** API Request Header (**except** `/market` **or** `ws/market`), you supply the `API Key` with `--header 'Authorization: '`

3. [For Gated Launch] Access Code Authorization

- In **every** Market REST API Request **Header** (`/market`), you supply the `Access code` with `--header ‘access-user: ’` AND `--header ‘access-token: ’`
- For Market WS API connection (`ws/market`), you supply the `Access code` in parameters. `/ws/market?accessUser=&accessToken=`
  - Note: if your accessUser is an email, you need to use URL Encoding to encode your email address and then suppy in the query parameter. Here is a link to easily encode your email with URL Encoding. https://www.urlencoder.org/

4. Security Headers in request

- User-Agent: None null value
- Accept
- Content-Type

Signing

- Signing is **ONLY** required for the **write-operations listed below**. GET requests do not require a signature. Signing needs to be done using a very specific payload structure for each operation. If you do not use the exact order and padding below your requests will fail signature verification. Please refer to this public js repo for examples.
  - **Capital: Withdraw and Transfer**
    - **Withdraw**
      - You must sign a payload comprising the following fields, padded to their respective sizes, and in the following order. This signed payload needs to be 32 bytes and be passed that in the `signature` field
        - `asset_id`: 4 bytes
        - `quantity`: 8 bytes
        - `max_fees`: 8 bytes
        - `withdrawal_address`: 20 bytes
    - **Transfer**
      - You must sign a payload comprising the following fields and pass that in the `signature` field
        - `nonce`: 8 bytes
        - `asset_id`: 4 bytes
        - `quantity`: 8 bytes
        - `dst_account_public_key`: 64 bytes
        - `max_fees_percent`: 8 bytes
  - **Trade**
    - All write operations (place orders, edit orders, cancel orders requests) require signatures.
      - **Trustless accounts**: these signatures use ECDSA with the private key provided per API Key.This is a string of **65 bytes including the recovery ID**. Depending on what library you're using, the ECDSA signature should give you a recovery ID byte as well. You then concatenate that bit as the 65th byte.
      - **Exchange-managed accounts**,:these signatures use HMAC with the private key provided per API Key.
      - Each operation has a specific method for generating the signature digest that needs to be signed. Each operation is detailed below. A**ll numbers are formatted in big endian.**
        - Place/Edit Order
          - **Structure**: The payload for placing or editing orders consists of the following. For editing an order, you can use either the `orderId` or the `nonce` field to refer to the order.
          - **Nonce: 8 bytes**
            - Current timestamp serving as a unique identifier for your order
            - Nonce needs to be a milli-second or micro-second unix timestamp within +- 15 seconds of the current time
            - Nonce needs to be unique per account (if you send two orders with the

        - Nonce needs to be unique per account (if you send two orders with the same nonce from the same account, they will be rejected by our API)
    - **Contract ID: 4 bytes**
    - **Quantity: 8 bytes -** See section on asset quantities for how these are computed
    - **Side: 4 bytes - Ask: 0 ; Bid: 1**
    - **Price (*Optional*): 8 bytes**
        - Only applicable for limit order
        - See section on asset prices for how these are computed
    - **Max Fees Percent: 8 bytes**
        - Maximum fees allowed for order operation
        - Denoted in basis points (e.g. 20 ⇒ 0.2% fee)
        - We can get fee information from /market/exchange-info.
    - **Creation Deadline (*Optional*)**
        - If our system can not place the order by the deadline, the order will be rejected
        - This is useful if you are concerned about high network or system latency.
        - The `creationDeadline` field should have the number of seconds since the Unix epoch as a floating point number
        - For trigger orders, the creation deadline is applied to the order when it is triggered.
    - **Trigger Price (*Optional*)**
        - **If present, indicates that this is a trigger order (also called stop order)** that should be sent when the market price reaches the trigger price
        - When modifying trigger orders:
            - If order has been triggered, you should not set the `updatedTriggerPrice` field. You can modify the order like a regular limit order
            - If the order has not been triggered yet, you should set the `updatedTriggerPrice` field with the new trigger price
            - If the order has not been triggered yet, you can modify market orders to limit orders and vice-versa
                - If updatedPrice is present, the order will be modified to a limit order with that price
                - If updatedPrice is absent, the order will be modified to a market order
            - When triggered, the order ends up at the back of the queue for the price level
    - Example
        - Let's assume we want to place the following order and contract information from `/exchange-info`

json

```json
//order
{
    "symbol": "BTC/USDT-P",
    "side": "ASK",
    "nonce": 1714701600000000,
    "quantity": "1 BTC",
    "price": "100,000 USDT",

    "max_fees": "0.0005"    //5 basis points
```

We can now compute quantity and price:

- Quantity: `1 BTC x 10^10 = 10,000,000,000` since `underlyingDecimals`: 10
- **Price:** `100,000 USDT x 2^32 x 10^(10 - 6 = -4) = 42,949,672,960` since `priceMultiplier`: `2^32` and `underlyingDecimals` - `settlementDecimals` = `10 - 6 = -4`
- Using these numbers, we can compute the elements of the buffer as**:**
  - Nonce: `1714701600000000` ⇒ `0x0006178313c38800`
  - ContractId: `2` ⇒ `0x00000002`
  - Quantity: `10000000000` ⇒ `0x00000002540be400`
  - Side: `ASK` ⇒ `0x00000000`
  - Price: `42949672960` ⇒ `0x0000000a00000000`
  - Fees: For fixed fees like withdraw/deposit, we should use fees * 10^6, one example: 1.23⇒1230000⇒0×000000000012C4B00. For rate fees, we should use rate fees * 10^8, one example: 0.0005⇒5000⇒0×0000000000001388
- Concatenating these elements gives us a buffer of: `0x0006178313c3880000000002000000002540be400000000000000000000000a000000000000000000001388`
- If exchange managed, we would sign this order using HMAC

```javascript
const hmacSignature = crypto
   .createHmac('sha256', 'YOUR-SECRET-KEY')
   .update(buffer)
   .digest('hex');
```

- If trustless account, we would sign using ECDSA

```javascript
const msgHash = ethers.sha256(buffer).slice(2);
const ecdsaSignature = ec.sign(msgHash, 'hex', { canonical: true });
```

- **Cancel Order:** For order cancellations, you simply need to sign the 64-bit orderId or nonce for the original order. You can use either the order ID or nonce to reference the order.
  - For example, the place order operation above could result in a response payload like `{"order_id":"579183763093760000"}`
  - If we wanted to cancel this order, we'd need to sign 579183763093760000 as 8 bytes, which would map to 0×0809ac905ae0a800
  - Canceling all orders: You can also cancel all orders by sending an HTTP delete to the /trade/orders endpoint with the account ID, a nonce, and signature signed on the nonce

REST API

You can find documentation for all Hibachi REST API on this page.

We have 2 different categories of API:

- Account API: for account specific query and action
- Market API: for market data and metadata

Websocket API

For Websocket API, please use this link to access our public Postman workspace.

You can find the detailed documentation this way:

1. Select "Websockets API" and choose one of the API endpoint.
2. Click the "Documentation" button on top right.

Quick Links

1. Hibachi UI: https://hibachi.xyz
2. Use Exchange Info API to get the list of future contracts
3. Use Trade Order API to place orders
4. Use Account Info API to get account status

Account API

You can use Account API to:

- Query and move funds: `/capital`
- Trade and view orders: `/trade`

AUTHORIZATION API Key

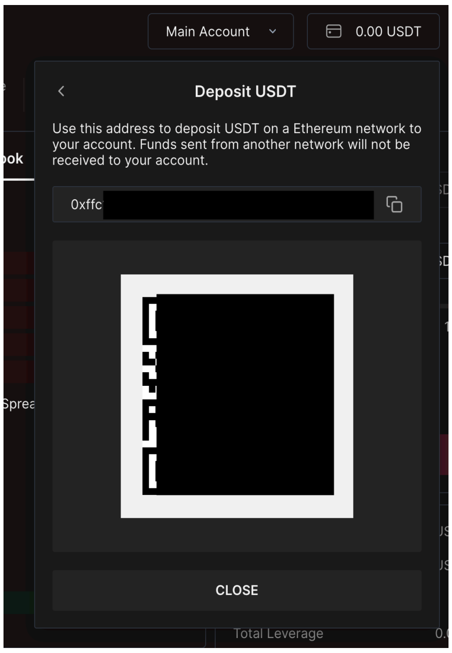
Key	Authorization
Value	<value>

Capital

Deposit USDT

You **MUST** make one Deposit with **USDT** on **ARBITRUM ONE** to your account. (Your Account ID will not be generated until your first deposit; this ID is required for a lot of endpoints

Open the "Wallet Balance" Dropdown, and select "Deposit" to reveal wallet deposit address, then send USDT from any of your crypto wallets that holds USDT to this deposit address


Main Account
0.00 USDT
Deposit USDT
Use this address to deposit USDT on a Ethereum network to your account. Funds sent from another network will not be received to your account.
0xffc[redacted]
[redacted]
CLOSE

Wallet Deposit Address Screen

Withdraw

To withdraw funds, you can use the `/capital/withdraw` endpoint.

- Triggers a withdraw request for the exchange to make funds available for deposit.
- If accepted, the withdraw request will be submitted in the next ZK proof batch (10-30min) at which point the funds will become available for claiming.
- For self-withdrawals (zero-fee), the user will need to claim the funds themselves.
- For auto-withdrawals (gas fee excluded) the exchange will trigger the blockchain transaction for the funds to get claimed to the withdraw_address provided in the original request.

You must sign a payload comprising the following fields and pass that in the `signature` field

1. `assetId`
2. `quantity`
3. `maxFees`
4. `withdrawalAddress`

Transfer

To transfer funds, you can use the `/capital/transfer` endpoint.

- Tranfers funds between accounts within the system (no withdrawals).
- Transfers are instant and can be used right away by the receiving account.

You must sign a payload comprising the following fields and pass that in the `signature` field

1. `nonce`
2. `assetId`
3. `quantity`
4. `dstPublicKey`

5. `maxFees`

AUTHORIZATION API Key

This folder is using API Key from folder Account API

GET /capital/balance

```
https://api.hibachi.xyz/capital/balance?accountId=<accountId>
```

Usage

Get the balance of your account.

The returned balance is your net equity which includes unrealized PnL.

Required Request Parameter:

accountId: The account ID can be found on the UI details page of each API Key. Your Account ID will not be generated until your first deposit; this ID is required for a lot of endpoints

AUTHORIZATION API Key

This request is using API Key from folder Account API

PARAMS

accountId	<accountId>

GET /capital/history

```
https://api.hibachi.xyz/capital/history?accountId=<accountId>
```

Usage

Get the deposit and withdraw history of your account.

It will return most recent up to 100 deposit and 100 withdraw.

Required Request Parameter:

`accountId` : The account ID can be found on the UI details page of each API Key. Your Account ID will not be generated until your first deposit; this ID is required for a lot of endpoints

AUTHORIZATION API Key

This request is using API Key from folder Account API

PARAMS

accountId	<accountId>

POST /capital/withdraw

https://api.hibachi.xyz/capital/withdraw

Usage

Submit a withdraw request.

Signing

Use below field to sign the message and pass it in to `signature` field (65 bytes). Refer to the Signing section for more context

- `assetId` (4 bytes)
- `quantity` (8 bytes): should be no more than `maximalWithdraw` returned by `/trade/account/info` endpoint, otherwise it will be rejected.
- `maxFees` (8 bytes): should be at least returned by /market/exchange-info endpoint, otherwise it will be rejected.
- `withdrawalAddress` (20 bytes)

Request Body Payload

Required Fields

- `accountId`
- `coin`: "USDT"
- `withdrawAddress`
- `quantity`
- `signature`

NOTE: Self Claim Option for trustless account: You can replace `withdrawalAddress` with `selfWithdrawal` , but you will need to follow our ABI to make direct contract interaction for claim the withdraw.

AUTHORIZATION API Key

This request is using API Key from folder Account API

Body raw (json)

json

```
{
```

```
{
    "accountId": 140,
    "coin": "USDT",

    "withdrawAddress": "0000000000000000000000000000000000000000",
    "network": "arbitrum",
    "quantity": "2.0113592",
    "maxFees": "0.0113592",
    "signature": "00000000000000000000000000000000000000000000000000000000000000000000000
}
```

POST /capital/transfer

https://api.hibachi.xyz/capital/transfer

Usage

Submit a withdraw request.

Signing

Use below field to sign the message and pass it in to `signature` field (65 bytes). Refer to the Signing section for more context

- `nonce` (8 bytes)
- `assetId` (4 bytes)
- `quantity` (8 bytes)
- `dstPublicKey` (20 bytes)
- maxFeesPercent`(8 bytes), should be at least returned value at /market/exchange-info endpoint, otherwise, it will be rejected.`

Required Request Body

- `accountId`
- `coin`: USDT
- `fees`
- `nonce`
- `quantity`
- `receivingAddress`
- `signature`

AUTHORIZATION API Key

This request is using API Key from folder Account API

Body raw (json)

json

```
{
    "accountId": 140,

    "coin": "USDT",
    "fees": "0",
    "nonce": 1728515344533,
    "quantity": "1",
    "dstPublicKey": "000000000000000000000000000000000000000000000000000000000000000000
    "signature": "0000000000000000000000000000000000000000000000000000000000000000000000
}
```

GET /capital/deposit-info

https://api.hibachi.xyz/capital/deposit-info?accountId=<accountId>&publicKey=0x00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

Usage

Get deposit address

Required Request Parameter:

`publicKey:`The public key of the account

AUTHORIZATION API Key

This request is using API Key from folder Account API

PARAMS

accountId	<accountId>
publicKey	0x0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

Authentication

You will need to use API keys for accessing Hibachi APIs

Generate API key from settings dropdown, create one with all restrictions enabled with `read-write` access.

API key generation

After generation, you can view the details of the API key

API key details

Click `Copy` on API Key. It will be used in authorization header for API access.

If you provisioned an exchange-managed account, you need to use `Private Key` for `HMAC` signing of your request on signature required endpoints.

AUTHORIZATION API Key

This folder is using API Key from folder Account API

Trade

Signatures

All write operations (place orders, edit orders, cancel orders, withdraw requests) require signatures.

- **For self-managed accounts**, these signatures use ECDSA with private keys derived from the user's seed phrase.
- **For exchange-managed accounts**, these signatures use HMAC with the secret created for a specific API key.

Each operation has a specific method for generating the signature digest that needs to be signed. Each operation is detailed below. All numbers are formatted in big endian.

Place / Edit Order

Structure

The payload for placing or editing orders consists of the following. For editing and querying an order, you can use either the `orderId` or the `nonce` field to refer to the order.

- *Nonce*: 8 bytes
  - Current timestamp serving as a unique identifier for your order
  - Nonce needs to be a milli-second or micro-second unix timestamp within +- 15 seconds of the current time
  - Nonce needs to be unique per account (if you send two orders with the same nonce from the same account, they will be rejected by our API)
- *Contract ID*: 4 bytes
- *Quantity*: 8 bytes
  - See section on asset quantities for how these are computed
- *Side*: 4 bytes
  - Ask: 0
  - Bid: 1
- *Price* (***Optional***): 8 bytes
  - Only applicable for limit orders
  - See section on asset prices for how these are computed
- *Max Fees*: 8 bytes
  - Maximum fees allowed for order operation
  - Denoted in basis points (e.g. `20 => 0.2% fee`)
  - For the current test phase, fees can be set to `0`. Once we finalize fee models, this will change
- *Creation Deadline* **(Optional):**
  - If our system can not place the order by the deadline, the order will be rejected.
  - This is useful if you are concerned about high network or system latency.
  - The `creationDeadline` field should have the number of seconds since the Unix epoch as a floating point number.
  - For trigger orders, the creation deadline is applied to the order when it is triggered.
- *Trigger Price* **(Optional)**:

    - If present, indicates that this is a trigger order (also called stop order) that should be sent when the market price reaches the trigger price.
    - When modifying trigger orders:
        - If order has been triggered, you should not set the `updatedTriggerPrice` field. You can modify the order like a regular limit order.
        - If the order has not been triggered yet, you should set the `updatedTriggerPrice` field with the new trigger price.
        - If the order has not been triggered yet, you can modify market orders to limit orders and vice-versa.
            - If updatedPrice is present, the order will be modified to a limit order with that price.
            - If updatedPrice is absent, the order will be modified to a market order.
    - When triggered, the order ends up at the back of the queue for the price level.
- *TWAP Duration* **(Optional)**
    - If present, `twapDurationMinutes` is the duration of the TWAP (trade-weighted average price) order in minutes.
    - This must be an integer between 1 and 60, inclusive.
    - A child order is sent every 30 seconds during the duration.
- TWAP Quantity Mode **(Optional)**
    - If present, `twapQuantityMode` configures the quantities of the child orders:
        - If set to "RANDOM", the quantity is randomized.
        - If set to "FIXED", the total quantity of the TWAP order is evenly divided across child orders.
    - If a child order is not fully filled, the unfilled quantity is rolled over into subsequent child orders.

TWAP orders

Querying TWAP orders yields the "SCHEDULED_TWAP" order status, which has the following additional fields:

- `numOrdersRemaining` is the number of child orders that will be sent in the future.
- `numOrdersTotal` is the total number of child orders (including past and future orders).
- `quantityMode` is the TWAP quantity mode.

You can cancel TWAP orders, but you can not edit them. For TWAP orders, child orders will execute within 5% of the price at the time when the original TWAP order was sent.

Example

Let's assume we want to place the following order

json

```json
{
  "symbol": "BTC/USDT-P",
  "side": "ASK",
  "nonce": 1714701600000000,
  "quantity": "1 BTC",
  "price": "100,000 USDT",
  "maxFees": 0,
}
```

and that `/exchange-info` has returned this contract information

```json
{
  "symbol": "BTC/USDT-P",
  "id": 2,
  "underlyingDecimals": 10,
  "underlyingSymbol": "BTC",
  "settlementDecimals": 6,
  "settlementSymbol": "USDT",
}
```

Given the above, we can compute quantity and price:

- **Quantity**: `1 BTC x 10^10 = 10,000,000,000` since `underlyingDecimals: 10`
- **Price**: `100,000 USDT x 2^32 x 10^(10 - 6 = -4) = 42,949,672,960` since `priceMultiplier: 2^32` and `underlyingDecimals - settlementDecimals = 10 - 6 = -4`

Using these numbers, we can compute the elements of the buffer as:

- Nonce `1714701600000000 => 0x0006178313c38800`
- ContractId: `2 => 0x00000002`
- Quantity: `10000000000 => 0x00000002540be400`
- Side: `ASK => 0x00000000`
- Price: `42949672960 => 0x0000000a00000000`
- Fees: `0 bps => 0x0000000000000000`

Concatenating these elements gives us a buffer of:
`0x0006178313c3880000000002000000002540be400000000000000000a000000000000000000000000`

If exchange managed, we would sign this order using HMAC

```javascript
const hmacSignature = crypto
        .createHmac('sha256', 'YOUR-SECRET-KEY')
        .update(buffer)
        .digest('hex');
```

If self-managed, we would sign using ECDSA

```javascript
const msgHash = ethers.sha256(buffer).slice(2);
const ecdsaSignature = ec.sign(msgHash, 'hex', { canonical: true });
```

This results in the final order payload:

```json
{
  "accountId": "<AccountId>",
  "symbol":"BTC/USDT-P",
  "side":"ASK",
  "orderType":"LIMIT",
  "quantity":"1",

  "price":"100000"
  "signature": "<HMAC/ECDSA signature of 0x00061..200000002540be40..0000>"
}
```

Cancel Order

For order cancellations, you simply need to sign the 64-bit `orderId` or `nonce` for the original order. You can use either the order ID or nonce to reference the order.

For example, the place order operation above could result in a response payload like

json

```
{
  "order_id":"579183763093760000"
}
```

If we wanted to cancel this order, we'd need to sign `579183763093760000` as 8 bytes, which would map to `0x0809ac905ae0a800`

Cancelling all orders

You can also cancel all orders by sending an HTTP delete to the /trade/orders endpoint with the account ID, a nonce, and signature signed on the nonce.

Rate Limits

The following operations are subject to a rate limit of 50 requests / second and 500 requests / minute:

- Placing orders.
- Modifying orders.
- Cancelling orders.
- Cancelling all orders. (This is considered as one operation regardless of the number of orders being cancelled)

If you exceed the rate limit, you will receive status code 429 in the REST and WebSocket APIs and information on which rate limit was exceeded. For example, if you exceed the rate limit you'd a response like the following in the WebSocket API:

json

```
{
  "error": "{\"count\": 51, \"errorCode\": 5, \"limit\": 50, \"name\": \"PLACE\", \"status\
  "id": 123,
  "status": 429
}
```

AUTHORIZATION API Key

This folder is using API Key from folder Account API

GET /trade/account/info

https://api.hibachi.xyz/trade/account/info?accountId=<accountId>

Usage

Get account information/details

Required Query Parameter

- accountId

Key Response Fields

- `balance` (net equity) and `maximalWithdraw`
- (collateral) `assets`
- `positions`
- risk related summary: `totalOrderNotional`, `totalPositionNotional`, `totalUnrealizedPnl`
- The `entryNotional` in `positions` is: entry price multiply by quantity.
- `totalOrderNotional` does not count orders that will reduce your position.

AUTHORIZATION API Key

This request is using API Key from folder Account API

HEADERS

Authorization

PARAMS

accountId	<accountId>

GET /trade/account/trades

https://api.hibachi.xyz/trade/account/trades?accountId=<accountId>

Usage

Get the trades history of your account.

It will return most recent up to 100 records.

Required Query Parameter

- accountId

Optional Query Parameter

- `page` : this is optional, if not supply, it will default to 0 and return 100 trades. If suppy 100 trades in each page

AUTHORIZATION API Key

This request is using API Key from folder Account API

PARAMS

accountId	<accountId>

GET /trade/account/settlements_history

```
https://api.hibachi.xyz/trade/account/settlements_history?accountId=<accountId>
```

Usage

You can obtain the history of settled trades using the `/trade/account/settlements_history` endpoint.

Required Query Parameter

- accountId

AUTHORIZATION API Key

This request is using API Key from folder Account API

PARAMS

accountId	<accountId>

GET /trade/orders

```
https://api.hibachi.xyz/trade/orders?accountId=<accountId>
```

Usage

Get the pending orders of your account.

Required Query Parameter

- accountId

Response Key Fields

- `total_quantity` is the quantity you specify when you submit the order.
- `available_quantity` is the quanaity that is remaining for match. It is `total_quantity` minus matched quantity.

AUTHORIZATION API Key

This request is using API Key from folder Account API

PARAMS

accountId	<accountId>

GET /trade/order

```
https://api.hibachi.xyz/trade/order?orderId=580604712937324546&accountId=<accountId>
```

Usage

Get the details about your **one particular order**.

Required Query Parameter

- accountId
- exactly one of orderId or nonce

AUTHORIZATION API Key

This request is using API Key from folder Account API

PARAMS

orderId	580604712937324546
accountId	<accountId>

POST /trade/order

```
https://api.hibachi.xyz/trade/order
```

Usage

Submit a new order.
In the response, it will return the `order_id` for this new order.

Signing

Use below field to sign the message and pass it in to `signature` field (65 bytes) Refer to the Signing section for more context

- `nonce`
- `contractId`
- `quantity`
- `side`
- `price`
- `maxFeesPercent, should be at least the returned value of /market/exchange-info endpoint. Otherwise, it will be rejected.`
- `creationDeadline (Optional)`
- `triggerPrice (Optional)`

Required Request Body Fields:

- `accountId` :should be one of: `LIMIT, MARKET`
- `symbol:` should be one of the symbol from one of the `futureContracts` returned by `/market/exchange-info` API
- `nonce:` should be a unix timestamp either ms or us unique to this order
- `side:` should be one of: `ASK`, `BID`
- `orderType`
- `quantity`
- `price`
- `siganture`
- `maxFeesPercent`

Optional Request Body Fields

- `creationDeadline`
- `triggerPrice`
- `twapConfig`

AUTHORIZATION API Key

This request is using API Key from folder Account API

Body raw (json)

json

```
{
    "accountId": "<accountId>",
    "symbol":"BTC/USDT-P",
    "nonce": 1722449703124,

    "side":"ASK",
    "orderType":"LIMIT",
    "quantity":"0.0001",
    "price":"100000",
    "signature": "000000000000000000000000000000000000000000000000000000000000000000000000",
    "maxFeesPercent": "0.00045"
}
```

POST /trade/orders

https://api.hibachi.xyz/trade/orders

Usage

Submit multiple new order requests for an account. Supports `place`, and `modify`.

The format is identical to POST `/trade/order`, the only differences are:

- `accountId` is specified in the top-level only once rather than for each order
- each `order` element needs to have an `action` field which can be either `place`, or `modify`

It will return a list of responses – one for each `order` in the input.

Signing

Use below field to sign the message and pass it in to `signature` field (65 bytes). Refer to the Signing section for more context.

- `nonce`
- `contractId`
- `quantity`
- `side`
- `price`
- `maxFeesPercent, should be at least the value returned from /market/exchange-info endpoint, otherwise, it will be rejected.`
- `creationDeadline (Optional)`
- `triggerPrice (Optional)`

Request Body Required Fields:

- `accountId` :should be one of: `LIMIT, MARKET`
- `symbol:`should be one of the symbol from one of the `futureContracts` returned by `/market/exchange-info` API
- `nonce:` should be a unix timestamp either ms or us unique to this order
- `side:` should be one of: `ASK`, `BID`
- `orderType`
- `quantity`

- `price`
- `signature`

- `action`: should be `place`,or `modify`
- maxFeesPercent

Optional Request Body Fields:

- `creationDeadline`
- `triggerPrice`

AUTHORIZATION API Key

This request is using API Key from folder Account API

Body raw (json)

json

```json
{
    "accountId": "<accountId>",
    "orders": [
        {
            "action": "place",
            "nonce": 1714701600000000,
            "symbol": "ETH/USDT-P",
            "orderType": "LIMIT",
            "side": "BID",
            "quantity": "1.01",
            "price": "3500.1",
            "maxFeesPercent": "0.00045",
            "signature": "0000000000000000000000000000000000000000000000000000000000000000"
        },
        {
            "action": "modify",
            "orderId": "578721673790138369",
            "nonce": 1714701600000002,
            "updatedQuantity": "0.1",
            "updatedPrice": "3500.1",
            "maxFeesPercent": "0.00045",
            "signature": "0000000000000000000000000000000000000000000000000000000000000000"
        },
        {
            "action": "cancel",
            "orderId": "578721673790138370",
            "signature": "0000000000000000000000000000000000000000000000000000000000000000"
        }
    ]
}
```

PUT /trade/order

https://api.hibachi.xyz/trade/order

Usage

Modify an existing order.

Signing

Refer to the Signing section for more context

Request Body Required Fields:

- `orderId` : should be the `order_id` returned by `/trade/order` or `/trade/orders` API. Alternatively you can use the nonce to specify which order to modify.
- `nonce` : Needs to be either a new unique nonce timestamp (same as if you were placing a new order) or the nonce of the order being modified.
- You can only update `quantity` and `price`. The symbol must remain the same.
- `quantity` :should be no less than the matched quantity, otherwise it will be rejected.
- `signature`
- maxFeesPercent: should be at least the value returned by the /market/exchange-info endpoint, otherwise, it will be rejected.

AUTHORIZATION API Key

This request is using API Key from folder Account API

Body raw (json)

json

```json
{
    "orderId": "578721673790138369",
    "accountId": "<accountId>",
    "nonce": 1714701600000002,
    "updatedQuantity": "0.1",
    "updatedPrice": "3500.1",
    "maxFeesPercent": "0.00045",
    "signature": "0000000000000000000000000000000000000000000000000000000000000000"
}
```

DELETE /trade/order

https://api.hibachi.xyz/trade/order

Usage

Remove an existing order.

Signing

Use below field to sign the message and pass it in to `signature` field (65 bytes). Refer to the Signing section for more context

- `accountId`
- `orderId` **OR** `nonce`
  - should be the `order_id` returned by `/trade/order` or `/trade/orders` API.
  - `nonce` should be the nonce used for create the order

Request Body Required Fields:

- `accountId`
- `orderId` **OR** `nonce`
  - `orderId` should be the `orderId` returned by `/trade/order` or `/trade/orders` API.
  - `nonce` should be the nonce used for create the order

AUTHORIZATION API Key

This request is using API Key from folder Account API

Body raw (json)

json

```json
{
    "accountId": "<accountId>",
    "orderId": "578721673790138368",
    "signature": "0000000000000000000000000000000000000000000000000000000000000000"
}
```

DELETE /trade/orders

https://api.hibachi.xyz/trade/orders

Usage

Allows you to batch cancel all open orders for an account.

Signing:

Refer to the Signing section for more context

- `nonce` needs to be a unique valid timestamp nonce and needs to be used as the payload for signature

Request Body Required Fields

- `accountId`
- `nonce`

- `signature`

Optional Request Body Fields:

- `contractId`

AUTHORIZATION API Key

This request is using API Key from folder Account API

Body raw (json)

json

```
{
    "accountId": "<accountId>",
    "nonce": 1714701600000000,
    "signature": "0000000000000000000000000000000000000000000000000000000000000000"
}
```

Market API

You can use Market API to query:

- Exchange Listings: `/market/exchange-info` (and optionally `/market/inventory`)
- Market Data: `/market/data`

Exchange Listings

The exchange listings endpoints expose all markets that are currently listed on Hibachi with their relevant market info.

The primary API endpoint is `/market/exchange-info` which exposes the raw markets and their trading configuration and also fees information associated with the Hibachi exchange.

.

There is an additional `/market/inventory` endpoint which is aimed primarily for client side usage and exposes some loose market stats (like 24h percentage change) in addition to the raw configs.

We recommend using `/market/exchange-info` for any automated trading activity, since its latency does not get affected by supplemental stats.

If you're looking to get specific stats on markets, the Market Data endpoints are much more comprehensive and better suited for such a use case.

Asset Quantities

All quantities on Hibachi are represented internally as **64-bit numbers**.

In order to enable trading fractional quantities (e.g. `0.01 BTC`), each asset on Hibachi as a specific fixed number of decimals.

Example:

- `BTC` may be represented with 10 decimals
- This means `1 BTC` is stored as `1 x 10^10 = 10,000,000` quantity on the exchange.

When making trades, querying positions and similar, Hibachi will still expose and expect numbers as "real" float strings (e.g. `"quantity: 1.0"` for `1 BTC`).

This way, users don't need to constantly convert numbers in order to be able to reason about them. However, this `quantity` notation is important when:

1. Signing any operations (trades, withdrawals, etc) since we want the traders to specify exactly what they want to trade and not leave room for misinterpretations. Please make sure to use correct fees when signinging , otherwise, the request will be rejected.
2. Interacting with the Hibachi smart contracts

The number of decimals is exposed for each contract in the `/market/exchange-info` endpoint response under the `underlyingDecimals` field.

Asset Prices

Like quantities, asset prices are also **64-bit numbers** that conform to decimals. Prices are used when converting between underlying and settlement token in a market.

For example, on `BTC/USDT-P` a price of `100,000` may be used in a limit order to indicate that somebody is wiling to sell `BTC` for `100,000 USDT` per `BTC`. Price is always denoted in units of the `settlementAsset`.

Since both the underlying and settlement assets have unique quantity decimals, the price is stored in the difference between both decimals (`settlementDecimals - underlyingDecimals`).

In order to support higher resolution of prices, an additional **price multiplier of** `2^32` is used to represent prices as fixed-point decimals with 32 bits after the decimal point.

Example:

- Let's assume somebody wants to sell `BTC/USDT-P` for `100,000 USDT per BTC`
- In this case, `/exchange-info` endpoint for this market has returned

json

```json
{
  "symbol": "BTC/USDT-P",
  "id": 2,
  "underlyingDecimals": 10,
  "underlyingSymbol": "BTC",
  "settlementDecimals": 6,
  "settlementSymbol": "USDT",
}
```

- This means that `settlementDecimals - underlyingDecimals = 6 - 10 = -4`
- Converting our price of `100,000 USDT` can be done using

javascript

```javascript
price_hibachi = price_raw x 10^(settlement_decimals - underlying_decimals) x price_multipli
pirce_hibachi = 100,000 x 10^(6 - 10 = -4) x 2^32 = 42949672960
```

Fees

Hibachi charges fees for the following actions:

- **Trade**: Fees are based on the notional value of an execution. Makers and takers pay different rates; tradeMakerFeeRate in the returned JSON indicates the rate for makers and tradeTakerFeeRate indicates the rate for takers. Hibachi determines the maker/taker status based on which side is the sitting order in the match, the sitting order side would be maker. Rates from `/exchange-info` are in units of 1, so 0.0002 means 0.02%. When signing orders, clients should use the maximum of tradeTakerFeeRate and tradeMakerFeeRate. Hibachi will charge a corresponding rate based on the taker/maker side of the trade.
- **Transfer**: Fees are based on the notional value of a transfer, with the rate specified by transferFeeRate from `/exchange-info`. The unit is 1 , so 0.0001 means 1 basis point (0.01%).
- **Deposit** : Hibachi charges fees to cover the gas costs for relaying deposits. The value (depositFees) from `/exchange-info` is in units of 1 USDT. Self-managed accounts are exempt from this fee. These fees vary dynamically with current gas fees and ether prices. Client should use some value like 1.2 x depositFees as the gast cost could change, Hibachi will charge the actual cost.
- **Withdrawal**: Hibachi charges fees to cover the gas costs for relaying withdrawals. The value (withdrawal fees) from `/exchange-info` is in units of 1 USDT, so 1.369258 means 1.369258 USDT. These fees also vary dynamically with current gas fees and ether prices. Client should use some value like 1.2 x depositFees as the gast cost could change, Hibachi will charge the actual cost.
- **Instant withdrawal Fees**:
  1. We provide the instantWithdrawDstPublicKey in feeConfig, you need to use this for the instant withdrawal request.
  2. instantWithdrawalFees provides fee information. The instantWithdrawalFees has a list of [threshold, rates]. Clients should use the maxium one that withdrawed quantity >= threshold, for example, in the example below, if the withdraw quanity is 2000, we should use 0.005 as the fee; if the withdraw quantity is 500, we should use 0.01.

json

```json
"feeConfig": {
        "depositFees": "0.293585",
        "instantWithdrawDstPublicKey": "8888888888888888888888888888888888888888888888888888
        "instantWithdrawalFees": [
            [
                1000,
                0.005
            ],
            [
                500,
                0.01
```

GET /market/exchange-info

https://data-api.hibachi.xyz/market/exchange-info

Usage

Return exchange metadata, currently it will return all `futureContracts`.

Also returns a list of exchange maintenance windows in the "maintenanceWindow" field. For each window, the fields "begin" and "end" denote the beginning and end of the window, in seconds since the UNIX epoch. The field "note" contains a note.

The field "maintenanceStatus" can have the values "NORMAL", "UNSCHEDULED_MAINTENANCE", "SCHEDULED_MAINTENANCE". If the exchange is currently under scheduled maintenance, the field "currentMaintenanceWindow" displays information on the current maintenance window.

HEADERS

access-user	<access-user's email/wallet>
access-token	<access-token>

GET /market/inventory

https://data-api.hibachi.xyz/market/inventory

Usage

Similar to `/market/exchange-info`, in addition to the contract metadata we will return their latest price info.

HEADERS

access-user	<access-user's email/wallet>
access-token	<access-token>

Market Data

Order book

You can obtain order book data using the `/market/data/orderbook` endpoint.

The parameters are:

- `symbol` (string)
- `depth` (optional 32-bit unsigned integer representing the number of levels to query)
- `granularity` (optional string representing the precision of the levels returned)
  - For example, you can use a granularity of 0.01 to obtains levels separated by 0.01 in price.

For example, you can query `/market/data/orderbook?symbol=ETH/USDT-P&depth=3&granularity=0.01` and you will receive a response such as the following:

json

```json
{
    "ask": {
        "endPrice": "3060.85",
        "levels": [
            {
                "price": "3060.23",
                "quantity": "0.073006381"
            },
            {
                "price": "3060.53",
                "quantity": "0.179959398"
```

Open interest

You can obtain open interest data using the `/market/data/open-interest` endpoint.

Parameters:

- `symbol` (string)

For example, you can send an HTTP GET request to `/market/data/open-interest?symbol=ETH/USDT-P` and you will receive a response like

json

```json
{
    "total_quantity": "0.113622416"
}
```

GET /market/data/prices

https://data-api.hibachi.xyz/market/data/prices?symbol=ETH/USDT-P

Usage

Get the price and funding information for a future contract.

Required Query Parameter

- `symbol`: the symbol of the contrat you get in the response from `/market/exchange-info` endpoint

HEADERS

access-user	<access-user's email/wallet>
access-token	<access-token>

PARAMS

symbol	ETH/USDT-P

GET /market/data/stats

```
https://data-api.hibachi.xyz/market/data/stats?symbol=ETH/USDT-P
```

Usage

Get general trading stats like 24h high/low/volume for a future contract.

Required Query Parameter

- `symbol`: the symbol of the contrat you get in the response from `/market/exchange-info` endpoint

HEADERS

access-user	<access-user's email/wallet>
access-token	<access-token>

PARAMS

symbol	ETH/USDT-P

GET /market/data/trades

```
https://data-api.hibachi.xyz/market/data/trades?symbol=ETH/USDT-P
```

Usage

Get the most recent trades from all users for one future contract.

It will return most recent up to 100 records.

Required Query Parameter

- `symbol`: the symbol of the contrat you get in the response from `/market/exchange-info` endpoint

HEADERS

access-user	<access-user's email/wallet>
access-token	<access-token>

PARAMS

symbol	ETH/USDT-P

GET /market/data/klines

```
https://data-api.hibachi.xyz/market/data/klines?symbol=ETH/USDT-P&interval=1h&fromMs=&toMs=
```

Usage

Get the candlesticks for a future contract.

Required Query Parameters

- `symbol`: the symbol of the contrat you get in the response from `/market/exchange-info` endpoint
- `interval`: Supported intervals: `1min`, `5min`, `15min`, `1h`, `4h`, `1d`, `1w`.

Optional Query Parameters

- `fromMs`: the start time in Ms of the period youwant to retrieve candlesticks
- `toMs`: the end time in Ms of the period youwant to retrieve candlesticks

HEADERS

access-user	<access-user's email/wallet>
access-token	<access-token>

PARAMS

symbol	ETH/USDT-P
interval	1h
fromMs	Optional

toMs	Optional

GET /market/data/open-interest

```
https://data-api.hibachi.xyz/market/data/open-interest?symbol=ETH/USDT-P
```

Usage

Get the open interest for one future contract.

It will return the total open position from each side.

Required Query Parameter

- `symbol`: the symbol of the contrat you get in the response from `/market/exchange-info` endpoint

HEADERS

access-user	<access-user's email/wallet>
access-token	<access-token>

PARAMS

symbol	ETH/USDT-P

GET /market/data/orderbook

```
https://data-api.hibachi.xyz/market/data/orderbook?symbol=ETH/USDT-P&depth=3&granularity=0.01
```

Usage

Get the orderbook price levels.

It will return up to `depth` price levels on both side. The price level will be aggreated based on `granulairty`.

Required Request Query Parameters

- `symbol`: the symbol of the contrat you get in the response from `/market/exchange-info` endpoint

Optional Request Query Parameters

- `depth` should be between 1 and 100, inclusive.
- `granularity` should be one of the values in `orderbookGranularities` returned by `market/exchange-info`. Different contracts may have different `orderbookGranularities`.

HEADERS

access-user	<access-user's email/wallet>
access-token	<access-token>

PARAMS

symbol	ETH/USDT-P
depth	3
granularity	0.01