Build Swap Transaction

The Swap API is one of the ways for you to interact with the Jupiter Swap Aggregator program. Before you send a transaction to the network, you will need to build the transaction that defines the instructions to execute and accounts to read/write to.

It can be complex to handle this yourself, but good news! Most of our APIs and SDKs just handles it for you, so you get a response with the transaction to be prepared and sent to the network.

> **USE SWAP API TO HANDLE IT FOR YOU OR ...**
>
> If you are looking to interact with the Jupiter Swap Aggregator program in a different way, check out the other guides:
>
> **Swap Instructions**
>
> To compose with instructions and build your own transaction, read how to use the `/swap-instructions` in this section.
>
> **Flash Fill or Cross Program Invocation (CPI)**
>
> To interact with your own Solana program, read how to use the **Flash Fill method** or **CPI** in this section.

Let's Get Started

In this guide, we will pick up from where **Get Quote** guide has left off.

If you have not set up your environment to use the necessary libraries, the RPC connection to the network and successfully get a quote from the Quote API, please start at Environment Setup or get quote.

> **API REFERENCE**

> To fully utilize the Swap API, check out the Swap API or Swap Instructions Reference.

Swap API

> **NOTE**
>
> Base URL: `https://lite-api.jup.ag/swap/v1/swap`
>
> For higher rate limits, refer to the API Key Setup doc.

From the previous guide on getting a quote, now using the quote response and your wallet, you can receive a **serialized swap transaction** that needs to be prepared and signed before sending to the network.

Get Serialized Transaction

Using the root URL and parameters to pass in, it is as simple as the example code below!

> **OPTIMIZING FOR TRANSACTION LANDING IS SUPER SUPER IMPORTANT!**
>
> This code block includes additional parameters that our Swap API supports, such as estimating compute units, priority fees and slippage, to optimize for transaction landing.
>
> To understand how these parameters help, the next step, Send Swap Transaction guide will discuss them.

```
const swapResponse = await (
await fetch('https://lite-api.jup.ag/swap/v1/swap', {
    method: 'POST',
    headers: {
    'Content-Type': 'application/json',
    },
    body: JSON.stringify({
    quoteResponse,
    userPublicKey: wallet.publicKey.toString(),

    // ADDITIONAL PARAMETERS TO OPTIMIZE FOR TRANSACTION LANDING
```

```
    // See next guide to optimize for transaction landing
    dynamicComputeUnitLimit: true,
    dynamicSlippage: true,
    prioritizationFeeLamports: {
          priorityLevelWithMaxLamports: {
            maxLamports: 1000000,
            priorityLevel: "veryHigh"
          }
        }
    })
})
).json();

console.log(swapResponse);
```

From the above example, you should see this response.

```
{
    swapTransaction:
'AQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
    lastValidBlockHeight: 279632475,
    prioritizationFeeLamports: 9999,
    computeUnitLimit: 388876,
    prioritizationType: {
        computeBudget: {
            microLamports: 25715,
            estimatedMicroLamports: 785154
        }
    },
    dynamicSlippageReport: {
        slippageBps: 50,
        otherAmount: 20612318,
        simulatedIncurredSlippageBps: -18,
        amplificationRatio: '1.5',
        categoryName: 'lst',
        heuristicMaxSlippageBps: 100
    },
    simulationError: null
}
```

What’s Next

Now, you are able to get a quote and use our Swap API to build the swap transaction for you. Next steps is to proceed to prepare and sign the transaction and send the signed transaction to the network.

Let's go sign and send!

Additional Resources

Build Your Own Transaction With Instructions

If you prefer to compose with instructions instead of the provided transaction that is returned from the `/swap` endpoint (like the above example). You can post to `/swap-instructions` instead, it takes the same parameters as the `/swap` endpoint but returns you the instructions rather than the serialized transaction.

> **NOTE**
>
> In some cases, you may add more accounts to the transaction, which may exceed the transaction size limits. To work around this, you can use the `maxAccounts` parameter in `/quote` endpoint to limit the number of accounts in the transaction.
>
> Refer to the GET /quote's `maxAccounts` guide for more details.

▸ **/swap-instructions code snippet**

Build Your Own Transaction With Flash Fill Or CPI

If you prefer to interact with the Jupiter Swap Aggregator program with your own on-chain program. There are 2 ways to do it, typically on-chain program call **Cross Program Invocation (CPI)** to interact with each other, we also have another method called **Flash Fill** built by Jupiter (due to limitations of CPI in the past).

> **CPI IS NOW RECOMMENDED!**
>
> As of January 2025, Jupiter Swap via CPI is recommended for most users.
>
> The `Loosen CPI restriction` feature has been deployed on Solana, you can read more here.

WHY FLASH FILL?

With Jupiter's complex routing, best prices comes at a cost. It often means more compute resources and accounts are required as it would route across multiple DEXes in one transaction.

Solana transactions are limited to 1232 bytes, Jupiter is using Address Lookup Tables (ALTs) to include more accounts in one transaction. However, the CPI method cannot use ALTs, which means when you add more accounts to a Jupiter Swap transaction, it will likely fail if it exceeds the transaction size limits.

Flash Fill allows the use of Versioned Transaction and ALTs, hence, reducing the total accounts used for a Jupiter Swap transaction.

▸ **CPI References**

▸ **Flash Fill References**

Edit this page