Solana Dev 101 - Using DAS API For Fetching all NFTs in a Collection

Overview

TheDigital Asset Standard

(DAS) API is a newly released interface that unifies regular and compressed assets on Solana (tokens, NFTs, etc.). With the introduction of compressed assets, Solana developers can now retrieve all assets associated with a wallet, collection, or authority more efficiently, eliminating the need to use multiple endpoints. The DAS API is also indexed behind the scenes, resulting in the most performant calls for you as a developer. With DAS available, you can streamline the process of retrieving information by eliminating lengthy gPA calls. In the getAssetsByOwner endpoint, you can access metadata and off-chain information for all assets belonging to a specific collection using its on-chain collection ID.

In this tutorial, we will demonstrate how to utilize the DAS API to retrieve asset information from the Mad Lads collection. To follow along with our current code base, you can check out the GitHub repository here. You can also review our expansive DAS API docs more here.

Prerequisites

- Node.js installed (v18.0 or higher for using the built-in fetch).
- Basic understanding of JavaScript.

Setting up your environment

1. Create a folder for this project called collection.
2. Inside the collection folder

, create a file named assetList.js

. We will write our function in this file.

1. Create an API Key at our [Developer Portal

](https://dev.helius.xyz/dashboard/app). Navigate to RPCs and copy the Mainnet RPC link, which will be used in this tutorial as the URL variable.

1. Obtain a Certified Collection ID for a demo collection to test. In this case, we will use Mad Lads, which has a collection ID of J1S9H3QjnRtBbbuD4HjPV6RpRhwuk4zKbxsnCHuTgh9w

. You can find the on-chain collection address on a marketplace such as Magic Eden when viewing a specific NFT.

Please note that if there is no on-chain collection ID, you will need to use an alternative DAS method to retrieve the results.

Steps to Follow

1. Create getAssetsByGroup Function

First, let's create a function to retrieve all assets related to a collection. We will nest our POST request to the DAS API inside this function.

Begin by establishing an asynchronous function:

In this section, we have imported the fs

module for handling file system operations, identified the RPC URL, and declared the getAssetsByGroup

function.

Ensure you substitute

with your API Key from the Developer Portal.

2. Creating POST Request to DAS

Let's define the getAssetsByGroup

function and specify the starting page and request return parameters. We will use the fetch function to align with our [method documentation

](https://docs.helius.xyz/solana-rpc-nodes/digital-asset-standard-api/get-assets-by-group).

We start a timer with console.time('getAssetsByGroup')

and initialize variables for the current page and an empty array to store the fetched assets.

We use fetch

with await

to send an asynchronous POST request to the specified url endpoint:

Next, we're entering a while

loop that will continue fetching data from the API as long as the page

variable is not false.

We then use fetch

with await

, which is an asynchronous operation used to send HTTP requests. We specify the url

of the API endpoint and set our method to 'POST'. This means we're sending data to the server in the body of the request.

In the headers of our request, we're setting 'Content-Type' to 'application/json'. This tells the server that we're sending JSON data.

Following this, we configure the body of our request, a JSON object that we stringify into a format that can be sent to our endpoint. This is where we define our groupKey (this will be “collection”), and our groupValue (this will represent the on-chain collection ID).

Now, we initiate an error to catch if the server response is not affirmative. In case of a successful request, it will render the response in a JSON format.

You may experience an error if you do not have a valid API key set in the url.

3. Append New Assets to the List

In the previous segment, we originally directed getAssetsByGroup

to operate when the page is set to 1. However, it is not yet configured to traverse through all possible pages of results. Let's establish that next:

This code adds the items from the response to the assetList

array. If the total number of results is not equal to the limit of 1,000, we set page

to false

to exit the loop.

4. Record Assets to a File

To save the retrieved asset information to an external JSON file, add the following code:

This code constructs a resultData

object consisting of the total result count and the assetList

array. We apply fs.writeFile

to inscribe the data to a JSON file named results.json

. Finally, we log a confirmation message and conclude the timer with console.timeEnd

.

5. Implement Error Handling

Now, we need to design a safety net to address potential server request failures. This can be achieved with the following setup:

This code block will log an error message in our console if there's a problem during the execution of our request.

You may experience an error requesting if you are not inserting a valid on-chain collection ID.

Final Code

Your assetList.js

file should resemble the following code snippet.

Result

Once your file resembles the code above, you can run it using the node assetList.js

command in your terminal to start the request. This will generate a results.json

file.

Upon completion, the console will indicate that the results have been saved to the results.json

file and log the time taken to fetch the assets. In our case, using Node.js to retrieve asset information for the Mad Lads on-chain collection, the process took an average of 9.27 seconds

.

When you open the results.json

file, you will see the total number of results returned, along with the asset details. These will represent the individual NFTs belonging to the collection you queried.

To further customize the returned data, you can extract specific information such as the image, owner, and other metadata that is deemed valuable.

Please note that the collection may show a total count of 9967 instead of 10,000 due to accounting for burned and off-chain assets

results.json

This will show the entirety of the assets returned. Now, you can break this down even further to return only the token address, owner, and various other metadata information.

You will notice the collection shows 9967 instead of 10,000. That is due to reflecting the amount burned and no longer on-chain.

Conclusion

Congratulations! You have successfully retrieved all assets for a 10k-sized collection using the newly released Digital Asset Standard (DAS) API. In summary:

- The DAS API provides a streamlined approach to asset fetching for Solana dApps.
- The method works for both regular and compressed collections.
- By utilizing the DAS API, you can access valuable metadata and ownership information in under 15 seconds.

By using the DAS API, we can streamline asset fetching for dApps on Solana. Instead of making multiple API calls to gather information, we only need to use a single endpoint.

We will touch on a few other streamlined options for returning assets for touching assets coming up in future tutorials.

Feel free to join our Discord and post any questions you have!