

# **Permachive - Phase 1**

## **Summary of Discovery**

### **Archiver & Web Scraper Research**

07.03.2022



### **GG Capital**

**A Digital Asset Research Firm**





## Project Overview

Our first priority in determining how to set up archivers on [Arweave](#) for Permacheive was to search currently available, open-source programs that pull data from social media and news sites, then upload them to the Permaweb. Although we had multiple options for starting places, the TwittAR and ARticle repositories from [Bundlr Network's ARchivers](#) page provided multiple pieces of information we needed to begin properly. TAB originally took this displayed code and picked it apart from scratch to determine how the archiver operates.

For example, we used [TwittAR](#), a JavaScript (JS) Twitter library, to pull filtered streams of various tweets. We would then parse the data from these tweets into a JSON file, then upload the JSON file to Arweave using [Bundlr](#).

From here, our next goal was to create a visually cleaner display for content uploads. By using [Puppeteer](#), a JS library for generating screenshots of pages and other browser-based tools, we were able to simulate a browser and take screenshots of each post using the URL provided.

This led to our initial discovery of how an archiver could be architecturally coded:

- ~ Pull data using the filtered tweet stream for Twitter and NewsAPI for selected news sites
- ~ Separate the URL from acquired data
- ~ Pass the URL to Puppeteer to create a screenshot of the browser
- ~ Upload the image to Arweave using Bundlr - tweet/news content and other data would be used as the image's tag

The initial steps for this method require:

1. An Arweave wallet that the archiver would use to pay for archiving. This is what the script that runs each process would be connected to.
2. A dedicated server to host this script. (Arweave makes this simpler with their graphql server.)
3. A database to store the URL tags. Key point here is to create an adaptive search filter for easier access to content.
4. An additional server to host the frontend for Permacheive. used to pay for archiving that the script connects to, something to host that script, a database to store the URL tags, etc for easy search, and something to host the frontend

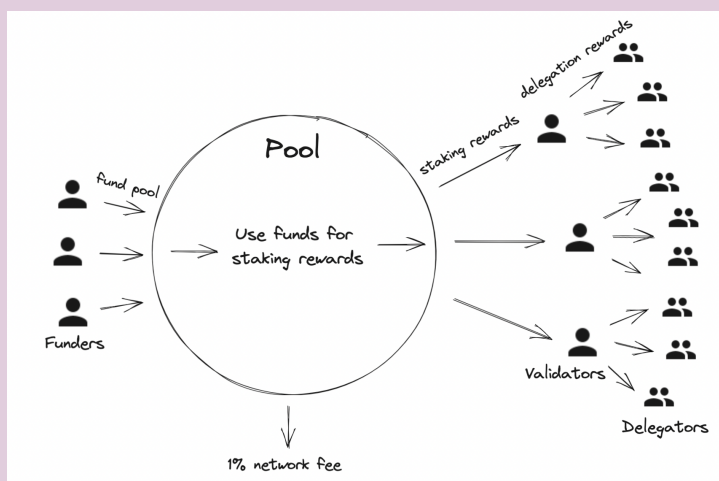
**Basic workflow: script constantly runs -> picks up posts/content -> uploads to arweave -> stores info in our database -> frontend pulls from database to show content.**



## Uploading Data to Arweave

When considering how to upload the filtered content that we collect in our database, we decided to continue working with our initial discussions around [Bundlr](#), reputed as the “fastest, easiest way to store data on web3”, and [Kyve](#), a fully efficient protocol built to “store any data stream, with full validation”. In this circumstance, we essentially have multiple goals to accomplish and additional metrics following product completion, including generating revenue streams and creating an accessible process for uploading uncensored data to Arweave or other data storage protocols.

If we were to focus on revenue stream generation, our ideal path would be to run a [Protocol Node](#) on Kyve, which would allow us to not only upload our database’s content, but receive rewards from the bundles that we would upload through our Protocol Node. This process



essentially operates with a Fund Pool, raised from Funders, that is then used to provide the staking reward to Validators that submit a valid bundle to the network. In turn, Kyve allows for larger bundles of data streams to be validated before uploading, which occurs within a [Bundle Proposal](#).

Our benefit here would be that Permacheive could establish part of its protocol through these various bundling solutions with the hopes of acquiring multiple sources of revenue while always optimizing for the best

method to upload our databases. However, this project is not in mainnet, only just starting its testnet launch recently. There’s still an additional opportunity here since Kyve is in their \*incentivized\* testnet, meaning setting up a Protocol Node prior to mainnet launch could provide benefits for acquiring a [“first mover advantage”](#) with their platform. As most of our communications have been surrounding Bundlr and their scaling solutions, Kyve offers an alternative option for establishing Permacheive’s method for bundling data, however, it would be up to the Permacheive team to decide whether their incentivised testnet is worth the time required.

With that in mind, Bundlr’s accessibility provides for a fast road-to-market, increased ease of use, and beneficial affordability for this initial MVP. Once solidifying that this protocol would be ideal for creating a working product with the infrastructure we currently had available, we realized that the main differences between uploading directly to Arweave vs through Bundlr were **expected costs** and **increased risk during data submission**. According to [Arweave News](#), the blockchain will cache all submitted data, yet will delete all data that remains unmined by the time the cache is cleared. This was an interesting admission from the Arweave team, but also emphasizes the benefits of bundling transactions prior to uploading to their network. The authors




also included that similar transactions were also at risk through Nodes hitting their rate limits or if block difficulty changes mid-transaction and the user must supply an additional reward. These were all important highlights to understand before interacting with Arweave directly, especially by confirming that bundling solutions were the correct way to proceed. Fortunately, expected costs as the next differentiation between the two protocols assisted this viewpoint as well.

Understanding the amount of capital that we would require to begin uploading was our next priority, and initially favored Bundlr with their promoted [free transactions](#) for any files under 100 kilobytes (kb). While this would greatly assist in uploading smaller content, we have not completed testing various social media and news sites to determine whether our image and content size will be larger or smaller than 100 kb.

#### Free Transactions

Free for data uploads under 100 kb

From here, we will need to focus on file size research as we enter into Phase 2 and 3 to fully calculate further expected costs as Permawave scales out. Our current guidelines for this revolve around recent comments in the [Bundlr Discord](#) from [joshb](#) (join Bundlr Discord to view initial [conversation](#)), a Bundlr developer, who explained that Bundlr is our cheapest option for anything

**joshb** 06/07/2022

Bundlr is cheaper than Arweave for txs <200kib. It's 20-30% more for >200kib

Bundlr acts as a caching layer allowing fast read and write

ArDrive is just a different thing. ArDrive is just OneDrive for Arweave vs Bundlr which is a scaling solution for Arweave

less than 200 kb, while Arweave is 20-30% more expensive for file sizes greater than 200 kb. Once we determine various file sizes for each social media site and the average size for news sites during Phase 2, we can test out rotating between both protocols depending on our results.

From here, we can optimize for each content site that we create archivers for, beginning with news & articles.

## A Framework for NewsAPI

With [NewsAPI](#) as our main goal for pulling news articles from thousands of different sites, we prioritized setting up this archiver using an API Key. This can be found in their documentation [here](#).

As explained in the Project Overview, the Bundlr ARchivers repository provided our next tool with the [ARTicle](#) program. This section uses the NewsAPI API Key and their various endpoints to achieve similar steps as the TwittAR tool. This process would contain similar JSON parsing, which is the process of converting our text formatted objects into JS objects that can be used



inside a program. This was a good starting point as the repository displayed what we could begin with and what else was needed, but the overall framework required additional improvements.

Once we finalized the general process for news sites, we found Twitter to be the most productive and accessible platform for pulling data. For our social media sites, we will display the benefits of Twitter's data uploads and our initial issues and concerns with Facebook's data uploads.

## Testing Social Media

To begin with social media sites, we continued to test out Bundlr's ARchivers with their [TwittAR](#) library to pull filtered streams of various tweets. We proceeded to then parse the data from these tweets into a JSON file, then upload the JSON file to Arweave using [Bundlr](#).

Our goal from here after being able to pull tweets from various searches was to create a visual image to display our results. Using the previously described JS library, Puppeteer, we realized that screenshot capability was possible. This library simulated a browser for our display and allowed for URLs to be used as our access point.

With the combination of these two tools and continued research on linking the screenshot and the plain text data together, we came to the conclusion that data and attributes could be uploaded in the tags of an Arweave upload.

## Twitter's Upload

After making a quick test program to pull tweets using keywords from the Tigrayan war, these tweets were screenshotted, saved within a PNG file and presented with **a file size of only 66 kb.**

Once we tested out a few more tweets, it became clear that Bundlr would be the best option to use for individual tweets, as this falls perfectly in line with their free transactions for file sizes under 100 kb.







## Facebook Posts

The primary issues we had with this initial Discovery phase were with Facebook and Mark Zuckerberg's disdain for open-source programs. It truly is astonishing to see how closed off certain Web2 entities currently are, as Facebook aka Meta put us in a tough position when trying to work around their new API. Facebook has closed down a large amount of their open endpoints to pull public posts, leaving very little to work off of using the official Facebook APIs. Our new path here was to then use a [web scraper](#), a tool that extracts data from a website, or a [post crawler](#), a tool that indexes the content of websites all over the internet, to pull public posts from the application.

When looking in the [Node Package Manager](#) libraries (npm), the best example we could find for this was the [Facebook Keyword Crawler](#) that we will test out in Phase 2, just in case we're able to bypass the official API. We initially thought this may change if it ends up not properly returning data, however, this became the least of our worries. Unfortunately, once we ended up working through more of the Facebook Graph API, we realized that they require a fully functioning app before we can use their APIs.

### Additional Details

- This permission or feature **requires successful completion of the App Review** process before your app can access live data. [Learn More](#)
- This permission or feature is **only available with business verification**. You may also need to sign additional **contracts before your app can access data**. [Learn More](#)
- While you are testing your app and before you submit it for review, your app can only access content on a Page for which the following is true: The person who holds the admin role for the Page also holds an admin, developer, or tester role on the app. If you want the app to be able to access public content on other Pages, you must submit this feature for review. Once you set your app to live mode, it will not be able to see any Page public content without this feature.

One interesting workaround we attempted was applying the same scrapers to search results in order to pull IDs and posts from Facebook, followed by using [GraphQL](#) to check if the post had been updated in a certain time period. This would have allowed us to determine when posts were published with a plan to pull said data and archive it once it fit a certain timeframe. We ran into similar problems here and opened up our options to see if Facebook would still be a viability but ran into too many issues once actually launching these programs..

## Conclusion

The NewsAPI and Twitter section on the archiver program is working great. Although Facebook turned out to not work, we believe we have enough evidence of this product being sustainable and easily accessible for Permacheive to use Bundlr and Arweave as their data upload services. For the archivers, we will be interacting with NewsAPI, Twitter API, and Facebook Keyword Crawler to pull data, Puppeteer to get a screenshot of the URL associated with said data, then upload the image with the plain text contents as a tag using the Bundlr network.