Permachive - Phase 2

Summary of Specifications

Technical Code Document

07.25.2022

GG Capital

A Digital Asset Research Firm

Overview

Following the results from our Summary of Discovery report, we conclude that the scope of this program is strictly the archiver and its use cases within the Permachive ecosystem. The set of archivers implemented here would interact with various external resources to acquire information to store on Arweave that can then be retrieved by a front end through Arweave's Graph API.

From an end-user perspective, the archivers used to filter uncensored data from Arweave and display content onto the Permachive website is a relatively straightforward program. With our current model, we can run specific social media archivers individually or we can have each program running simultaneously using a JavaScript process manager. In summary, a process manager will allow us to observe current services and facilitate administrative tasks that would otherwise be extensive and/or mundane. Our goal here is to apply the following requirements that we compiled from our Summary of Discovery to the initial product for running an archiver.

The requirements for this archiver include:

- A funded Arweave wallet
- A NewsAPI API Key
- Access tokens to the Twitter API with elevated developer access
- Running Docker, an open software platform that eases the process of building and managing applications
- Node.js: a compiler & runtime for Javascript, basically where our code is executed when running the program
- And npm: a package and library manager for Node.js used for managing development

When the above requirements are met, the program can be left running. However, the key points here are that the Docker command for starting the Docker instance of Chrome must be run, this way the rest of the program is executed from the script itself. The most important part here is that this script must have funds available within the Arweave wallet to ensure that the program can continue running.

Archiver Background

Using the Twitter Archiver

The Twitter archiver operates by using the Twitter API, or Application Programming Interface, to constantly send new tweets related to a specific keyword using what they call a filtered stream. This endpoint group from Twitter effectively allows us to filter the real-time stream of public Tweets. Whenever a new tweet is sent to the archiver through this stream, the tweet is processed not only using the data received from the Twitter API, but also through the Docker instance of Chrome and a library called Puppeteer. This library provides a high-level API to

control our connections to Chrome. In doing so, Puppeteer then connects to the docker instance of Chrome, which acts as our connection to a web browser. This instance, which is a realized form of any object in programming (such as a document type, element, or a web browser in our case), then navigates to the tweet URL. Once this occurs, the program analyzes the HyperText Markup Language or HTML code (the standard language for displaying data via web browser) that compiles even more information about the acquired tweet. This tweet is put into a JavaScript Object Notation or JSON format to then be attached as a tag to the Bundlr upload.

Tags are similar to metadata on files or images where this piece of information describes the data it is assigned. They can be set to any desired datapoint and accessed when the data is pulled. After the data is formatted correctly, Puppeteer is then used again to capture a screenshot of the tweet. This screenshot is saved as a PNG image file.

Bundlr is then called to create the transaction with the PNG file as the upload data and the tags are passed as a parameter. The transaction is then signed and uploaded. This process continues with each new tweet until it is stopped by a user. If an error occurs, it will print out the error and abort the upload the error occurred on, then continue with the next upload.

NewsAPI Archiver

For our NewsAPI Archiver, data is structured and pulled in a slightly different manner compared to our Twitter Archiver. Since NewsAPI has no variation of a filtered stream, finding new articles will occur manually. In essence, this API is called using a keyword and all articles returned that relate to this keyword.

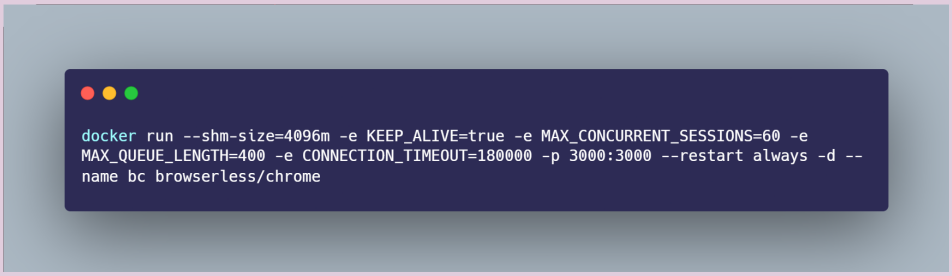
The key part of this Archiver is that it creates a local database stored as a file that keeps records of all the articles that our previous searches have returned. When searching previous keywords or topics, this Archiver checks against the database to see if a new article was returned or if an existing article that was already archived has been updated. If these conditions are met, Puppeteer applies the HTML code and converts this into a standalone HTML file, which benefits this database by holding no dependencies on JavaScript or any external resources.

This is crucial to the process of acquiring and uploading data, even more so since one of our main objectives for Permachive is to preserve unobstructed data and content from the past. When these measures are not completed, the newest version of data will be retrieved during new searches, since the program is calling the website's resources, which defeats the purpose of saving an exact copy of the article as it was published.

This standalone HTML file is then passed to a Bundlr transaction with tags containing basic data. The transaction is then signed and uploaded, which means that when users retrieve an article from Arweave, it will be HTML instead of an image. However, the format in which the data will be saved on Arweave can be changed per request from business users.

Configuration

After installing the above requirements, there are a few configurations that must occur before the program can be executed. First, the docker command must be running to create a background instance of Chrome for Puppeteer to utilize. These commands are specified more in the Technical Code Document under the first section, with the docker command seen here.



The next step in confirming that the program is ready will be setting up the config.json and wallet.json files with their appropriate keys. These keys are what provide access to each API and are used to identify an application or user.

The config.json file clarifies each part of the process, which contains the query keywords, userIDs, wallet path, and other settings used to define the program and the functions that the search will be calling. This example provides a very precise similarity for what our configurations would look like when conducting testing for the program.

```
{
    "bundlrNode": "http://node2.bundlr.network",
    "keywords": [
        "#Ethiopia",
        "#Tigray",
        "#TigrayGenocide",
        "ethiopia war"
    ],
    "userIDs": [],
    "instances": 40,
    "query": "ethiopia",
    "walletPath": "./wallet.json",
    "difference": 0.02,
    "refreshPeriod": "-4 hours",
    "queryID": "Ethiopia1"
}
```

The next file is the wallet.json. This file defines the Arweave wallet that the program is connected to. As this will display private information, we highly advise that Permachive refrains from showing this file when launching archivers.

There are many occurrences where launching programs with viewable wallet.json files has led to the downfall of projects and where not having the JSON file has led to the erasure of user wallets. With that in mind, this becomes increasingly important for developers to prioritize the protection of this file, especially when the company scales out to user-hosted archivers.

In this screenshot, we can view the Twitter keys, noted as “tkeys”, that can be obtained from the Twitter developer site. The most important element here is that you must have elevated developer access for using this Twitter account.

The “newsapi” is the previously described NewsAPI Key, which is provided when first creating an account to access the API. From here, we would then input our necessary information into the “arweave” section, which is where you paste your wallet.json key that was given from Arweave. You can download your wallet’s wallet.json through Arweave’s wallet website.

Once you have confirmed that all of these conditions have been met and that each command is executed, you can start the program and allow it to continue running.

Risks, Issues, and Errors

Thankfully, a wide range of precedents has taken place to ensure that this project has a very low risk for operation. In line with the previously mentioned wallet.json example, leaking this file would be the main risk when launching this program as this puts you at risk of attackers draining available funds.

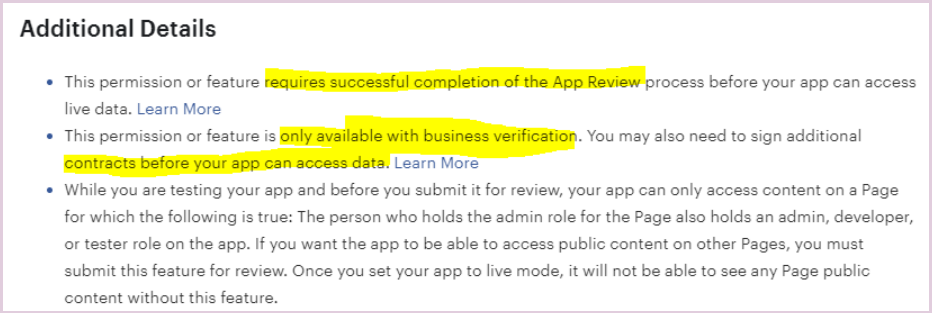
An additional risk would be the program uploading a large number of files at the same time, which could potentially extract necessary AR funds before the data could be uploaded. The main component here is that parameters would have to be set in advance to ensure that funds are available for all data uploads.

Currently, error reporting is only done locally by outputting any error to our console. Following this, the program then moves on to the next item to archive. One of the primary changes we plan on implementing will be improving error reporting so that we can maintain an updated database of past errors.

Non-functional requirements

Facebook archiving is the only requirement that was not met due to their extensive requirements for gaining access to the API. Our future goal will be to revisit this platform when Permachive’s project and website are fully established. This way we can provide the required information to Facebook for acquiring API access.

This feature is described in greater detail via Facebook's Public Content Access. The main requirement is having an established site with a Privacy Policy that complies with Facebook's policy.



Use cases

For the Proof of Concept we are creating, this program is currently two archivers in one: the Twitter API Archiver and the NewsAPI Archiver. These can either be executed together or separately depending on the desired purpose. The main use case for these Archivers is to record data based on a specific keyword and then display this content on the Permachive website for users to filter and search at their discretion.

Once Permachive in its entirety is established, the working product will transition from being self-hosted to the eventual user-interactive website. A key goal here is that the program should allow users to download our code and run their own archiver. This expands Permachive's current use cases by applying the same globally expansive network that other open-sourced programs incentivize. From here, use cases can include news sources and media sites integrating our archivers into their websites and non-profit organizations that want to host uncensored content.

Within the realm of crypto projects and firms, each of these archivers eases the path to uploading data for teams that require uncensored and accessible information at a moment's notice. One interesting aspect of this could be for certain situations where users have posted fraudulent tweets or incriminating information, where Permachive could be used to clarify the validity of these statements by hosting the original content on a filtered page.

An additional use case here regards the difference in available news between domestic and foreign countries across the world. As Rahwa has explained, the purpose of Permachive originated from news about the Ethiopian War not being accessible for citizens within the country. When searching specific keywords on Google via a United States IP Address, the results are expansive and provide a large number of links. In comparison to searching the same keyword in China, the result is only *ONE* link. This opens up a massive opportunity for displaying uncensored data to anyone in the world, yet could also be pitched to governments and organizations that need to search previously censored content. We hope to expand on potential use cases as we build out our final Proof of Concept in Phase 3.

Future Social Media Archivers

Reddit

We have found that it is possible to search for posts on Reddit using keywords. Using the search endpoint here and setting the “restrict_sr” parameter to false, it will return posts from all subreddits and their accompanying threads.

Alternatively, we could use an NPM library called snoowrap, an effective Reddit API caller that provides an interface for accessing every Reddit API. This library parses the response for us, cutting out an extra step we would have to do on our end and allowing the flow of data to occur with the library’s built-in ratelimit protection.

Quora

Quora does not offer an official API. There are additional NPM libraries to perform specific searches for questions and posts, however, this only occurs by username. Most of these libraries, such as this unofficial Quora API, have not been updated in years and have notes saying that the Quora closed API changes often.

As of now, creating an archiver for Quora is not feasible. There are multiple posts and questions that inquire whether or not there is an open and accessible API, yet all answers have led to Quora either sending cease & desist letters or blocking out any competition. The main reason seems to be that they close off their API to the public and make constant changes to avoid any external resources or businesses from acquiring data outside of using the Quora site itself.

TikTok

Fortunately, there are available access points for searching keywords and videos through TikTok’s API. Using the Video Query endpoint, you can search for videos given specified filters. This returns a list of objects containing all details about the video, including the URL to the video.

We can then use an NPM package called tiktok-downloader to pass in the URL and download the TikTok as a video file. There is a demo built out from this NPM package that allows for downloading videos using a given URL. This would then be uploaded to Arweave similar to how the Twitter Archiver uploads screenshots.

Permachive - Phase 2

Technical Code Document

Summary of Application

The following is an overview of the application beginning with the instructions and requirements for launching the protocol. The Permachive repository displays a quick guide to the Twitter and News Article Archivers. These tools have been implemented to record tweets, posts & news articles, then permanently store each dataset on Arweave via the Bundlr platform.

The program is heavily influenced by Bundlr's open source program ARchivers

To run either the Twitter or Article Archiver, you must have access to an Arweave wallet file. All resources can be found in the **Summary of Specifications** section of the document. From here, you need to copy the contents of this wallet file into the "arweave section" of the "wallet.json" example given above.

Once setup has been completed, make sure you have Docker and a set of build tools installed (this is primarily for those not using Long Term Support or LTS versions of Node.js). Run yarn to install dependencies.

Docker command to create headless chrome host:

```
docker run --shm-size=4096m -e KEEP_ALIVE=true -e
MAX_CONCURRENT_SESSIONS=60 -e MAX_QUEUE_LENGTH=400 -e
CONNECTION_TIMEOUT=180000 -p 3000:3000 --restart
always -d --name bc browserless/chrome
```

Tweak the “MAX_CONCURRENT_SESSIONS” value as required. Using a higher setting leads to more load, however, this provides a higher chance of content being archived (download requests are dropped if the queue gets too full).

Twitter Archiver

To run the Twitter Archiver you need Twitter API keys, which you can get via a Twitter developer account. **You will also need elevated API access.** Follow this answer here, and fill in the relevant fields in the wallet.json image: https://stackoverflow.com/a/6875024/18012461. Name this wallet.json.

Then in the developer portal, request elevated access - this should be approved almost immediately.

This section gets a filtered stream of tweets using “node-tweet-stream” JavaScript library, which leverages the official Twitter API. This is filtered based on keywords, which are pulled from the “config.json” file. When a tweet comes in, it goes to the processTweet function, scraping all the data out of it. This data is compiled into an object.

Once the data scraping is complete, the URL is pulled from the data and passed to the Puppeteer functions to take screenshots. The screenshot is saved in a .png file. The tags are created from the Twitter data, and the original data object is converted into a JSON string and assigned to a tag. Bundlr is then called to create a transaction, passing the .png screenshot and the tags. After signing and uploading the transaction, the .png file is deleted locally, as it has already been pushed to Arweave.

Our following plans include adding image compression to cut down the size of images, which would also decrease transaction costs. With a current goal of achieving <100kb without diminishing the images, this could eliminate upload transaction fees as an upkeep cost due to Bundlr’s policy of free transactions under 100kb.

Article Archiver

For the Article Archiver, the first requirement is a NewsAPI API key. Add this to your wallet.json (or example.wallet.json - rename to wallet.json) (it can be run without as an external import - just don't invoke updateNewsApi).

Tweak config.json as required, adding in keyterms - tweak instances to about 80% of your MAX_CONCURRENT_SESSIONS value.

If you are noticing too many re-uploads of unchanged data, or that the system is not responding to changes, adjust the difference value in the config - lower = more sensitive to changes. Remember to change the queryID value in the configuration to distinguish your collection from others - you can always filter by owner address, but this allows for more fine grained control.

This section pulls from the NewsAPI endpoint using a keyword. Once it has its results, it checks a local SQLite database file to see if it has uploaded that before or if it is a new version of an article it has already uploaded. If it is new or an updated version of an existing one, the HTML code is pulled and converted to be fully standalone. Making this process standalone strips out any external resources, which is essential for keeping it stored forever. The tags are created using the data from the page, and Bundlr is called when passing in the HTML code and tags.

Running

Install PM2 globally (use elevated terminal): yarn global add pm2

Build the project: yarn build

Start the project (All archivers): pm2 start Archiver.ecosystem.config.js

Docker is required, some machines may not have this.

After cloning the repo, run these commands to get the docker started and get all packages required. This uses a process manager for Node.js called PM2 that allows multiple programs to run at the same time using. To monitor the applications after starting, run pm2 monit to view the programs and their outputs.

```
npm install

docker run --shm-size=4096m -e KEEP_ALIVE=true -e
MAX_CONCURRENT_SESSIONS=60 -e MAX_QUEUE_LENGTH=400 -e
CONNECTION_TIMEOUT=180000 -p 3000:3000 --restart
always -d --name bc browserless/chrome
```

The Puppeteer library is then used to scrape data as well as take screenshots prior to uploading.

On-chain info and communication to frontend

Using Arweave's GraphQL, you can run a query to pull all transactions from a specific account. Make sure to use the account that is linked to the archiver.

```
query {
    transactions(owners:"<wallet address here>") {
        edges {
            node {
                id
              tags {
                name
                value
              }
            }
        }
    }
}
```

For examples of transactions created from our first set of tests, the testing account used was "BGAuyDbdXuSJXfTEM90Dbtw Gg6sTvEOZNOWnTCHjl-s". Each test transaction can be displayed and provide a summary of what these uploads appear like.

For the next step, pulling tags and their accompanying metadata will provide the necessary content for each upload. Using the Twitter archiver, this includes information about the tweet shown below.


```
"data": {
    "transactions": {
      "edges": [
        {
          "node": {
            "id": "yfm3H5fMa0MgeIoCL_df-b-H-TTUpmEMC4n939opWNI",
            "tags": [
              {
                "name": "Application",
                "value": "Permachive - Twitter Archiver"
              },
              {
                "name": "Tweet-ID",
                "value": "1543454151623032832"
              },
              {
                "name": "Author-ID",
                "value": "1521568120929927169"
              },
              {
                "name": "Author-Name",
                "value": "Danait Mulu"
              },
              {
                "name": "Author-Handle",
                "value": "@Danait70"
              },
              {
                "name": "Content-Type",
                "value": "image/png"
              },
              {
                "name": "Key-Word-List",
                "value": "Ethiopia"
              },
              {
                "name": "Tweet-Content",
                "value": "{\"created_at\":\"Sun Jul 03 04:38:40 +0000" //TRUNCATED FOR VIEWING
                          }
            ]
          }
        }
          ]
    }
}
```


From this, you can pull the image using the URL, which is https://arweave.net/ + the id.

The “Tweet-Content” tag is a JSON string containing all the tweet information for increased searching functionality.

An example of this can be found in the repo EXAMPLE_TWEET .json.

Example: https://arweave.net/yfm3H5fMa0MgeIoCL_df-b-H-TTUpmEMC4n939opWNI.

For NewsAPI, the entire HTML is saved, so after getting the tags and URL similar to the Twitter section, pull the URL and render the information on the page. Extra information will be scraped in the frontend, however, we have additional plans to implement updated parameters that could increase ease of search capabilities.

For any questions or additional concerns, our developer Tab is available or reach out to us at yohangg@ggcapital.io.