Permachive - Phase 3

Final Technical Code Document

Protocol Installation & Demo

08.02.2022

GG Capital

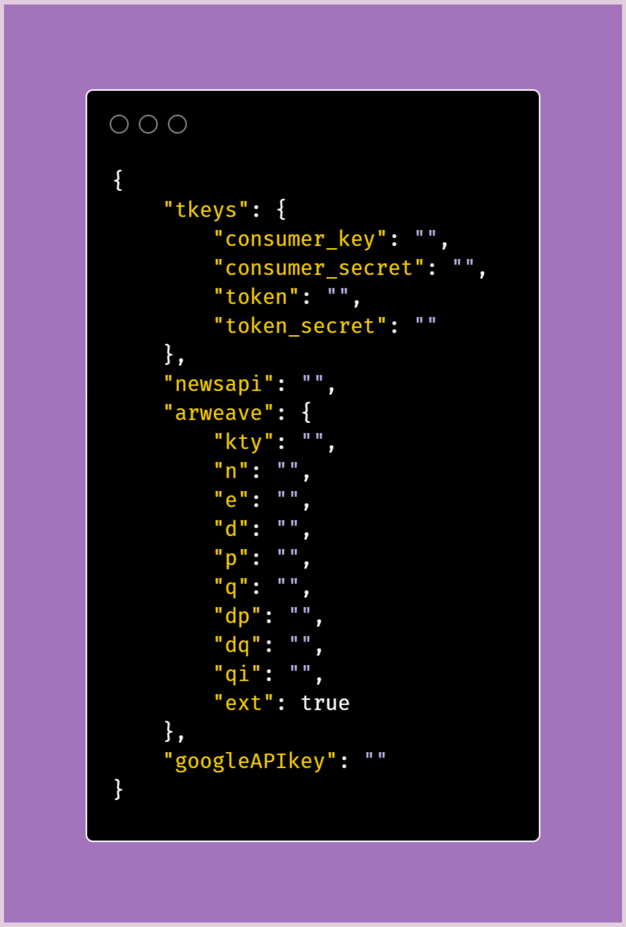
A Digital Asset Research Firm

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

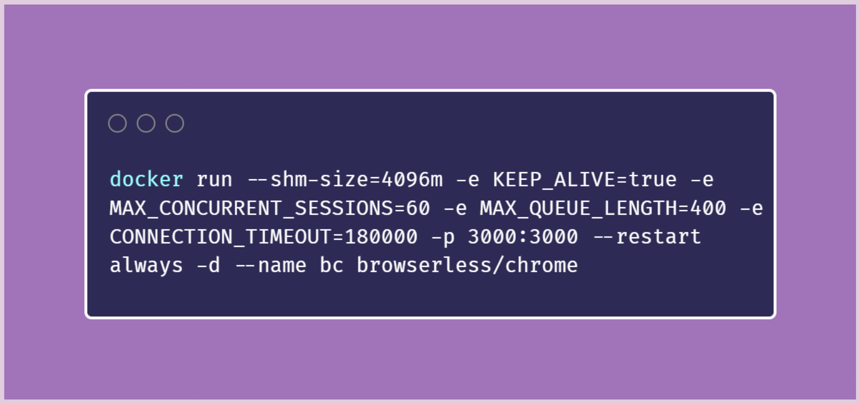
To run either the Twitter or Article Archiver, you must have access to an Arweave wallet file. All resources can be found in the **Summary of Specifications** section of the document. From here, you need to copy the contents of this wallet file into the "arweave section" of the wallet.json example here:



```
{
    "tkeys": {
        "consumer_key": "",
        "consumer_secret": "",
        "token": "",
        "token_secret": ""
    },
    "newsapi": "",
    "arweave": {
        "kty": "",
        "n": "",
        "e": "",
        "d": "",
        "p": "",
        "q": "",
        "dp": "",
        "dq": "",
        "qi": "",
        "ext": true
    },
    "googleAPIkey": ""
}
```

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 and then continue to execute the Docker commands.

The Docker command to create headless chrome host is the following:



```
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.

If the image size is greater than 100kb, the photo is compressed using the compress-images npm package. Even after compressing, there may be some files over 100kb, but the majority we have ran into while testing were compressed to under 100kb. This diminishes cost of upkeep for this archiver thanks to Bundlr's policy of free transactions under 100kb.

Error logging to a file has been added. Errors will log to a "Twitter-errorlog.txt" file.

New changes were added to remove some elements from the page before taking a screenshot to improve the readability of the screenshot. Some screenshots still contain the loading wheel, but due to how Twitter's code handles its visibility, there is no current way of hiding this.

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.

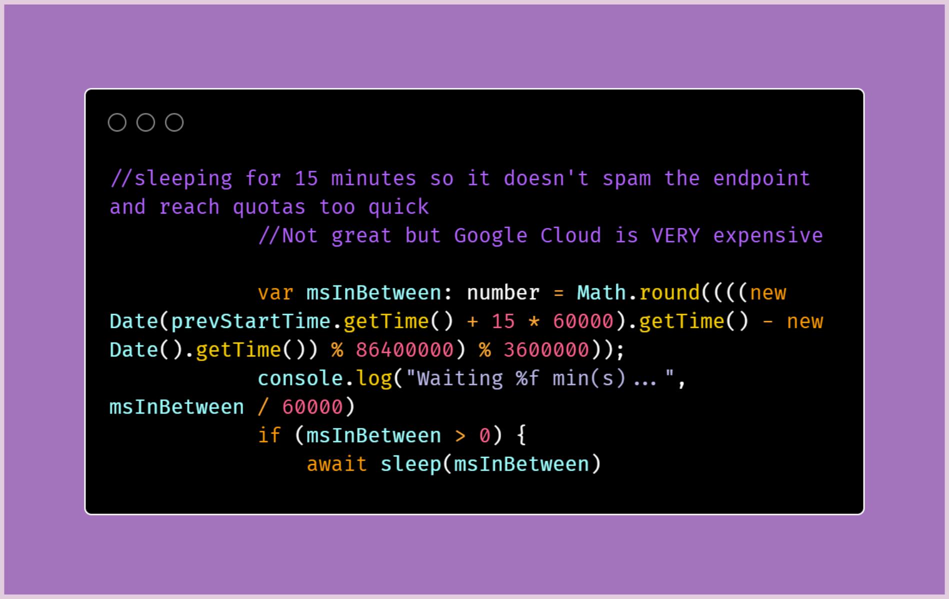
If you are noticing too many re-uploads of unchanged data, or that the system is not responding to changes, adjust the difference value in the config (lower = more sensitive to changes). Remember to change the queryID value in the configuration to distinguish your collection from others - you can always 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 a 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.

YouTube Archiver

For this archiver, a Google API Youtube Data V3 API key is the main requirement. You will need to create a project in Google Cloud and enable the YouTube Data API v3 API. AFter this, create an API key using the Google Cloud console.

Due to the quotas on this API, only 100 requests a day are allowed without paying. Quotas are tracked through "points" with a daily limit of 10,000. Each call to the YouTube API costs 100 points. Google Cloud is not the cheapest platform, thus why we went in this direction. If you would like to pay for a higher quota, you can remove lines 84-90 in the YouTube.ts file.



```
//sleeping for 15 minutes so it doesn't spam the endpoint
and reach quotas too quick
            //Not great but Google Cloud is VERY expensive

            var msInBetween: number = Math.round((((new
Date(prevStartTime.getTime() + 15 * 60000).getTime() - new
Date().getTime()) % 86400000) % 3600000));
            console.log("Waiting %f min(s)... ",
msInBetween / 60000)
            if (msInBetween > 0) {
                await sleep(msInBetween)
```

This extra code limits the archiver from calling the YouTube API once every 15 minutes. The time it waits is the difference in 15 minutes after it last called it and the time it finishes archiving.

So if it calls the API and takes 5 minutes to parse and upload the videos, the program will then wait 10 minutes after. This program concatenates the keywords from config.json and uses that as the query.

Due to the nature of videos being a larger size, this section using Arweave directly to upload instead of Bundlr like previous pieces. This is due to Arweave being more cost efficient than Bundlr for files larger than 200kb.

Error logging to a file has been added. Errors will log to a YouTube-errorlog.txt file.

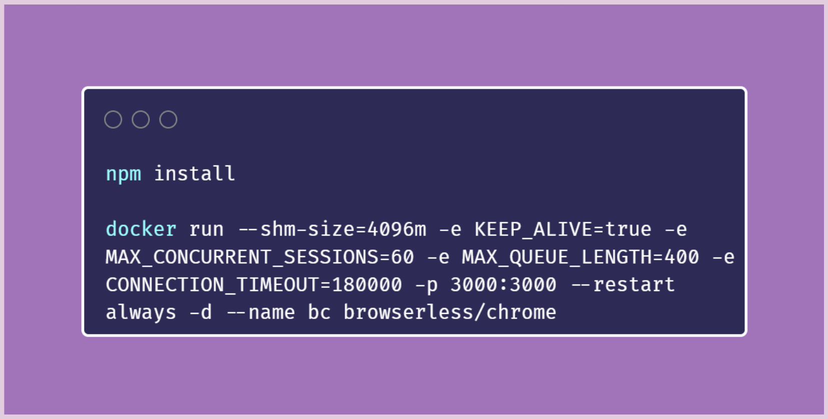
Running the program

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.



```
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
```

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.

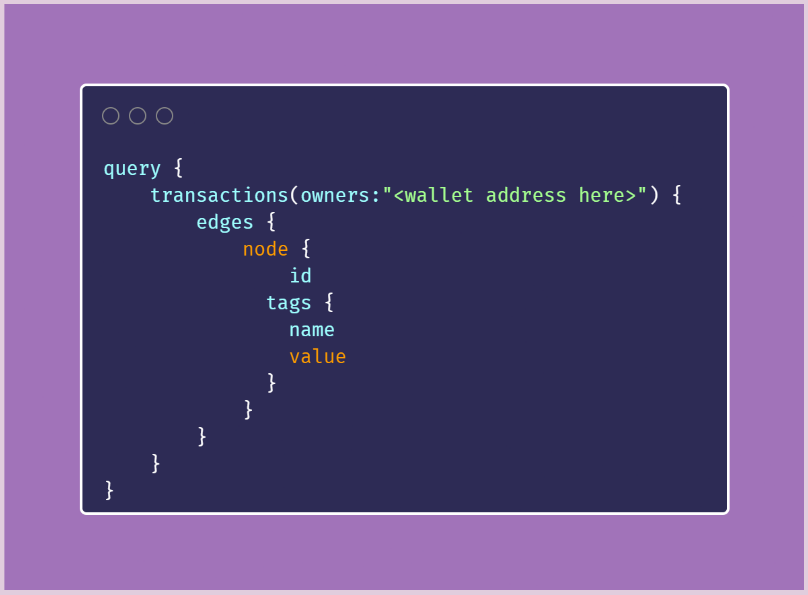
The Puppeteer library is then used to scrape data as well as take screenshots prior to uploading.

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.

Puppeteer library is used to scrape data as well as take screenshots.

On-chain info and communication to frontend

Using Arweave's GraphQL, you can run a query to pull all transactions from a specific account.

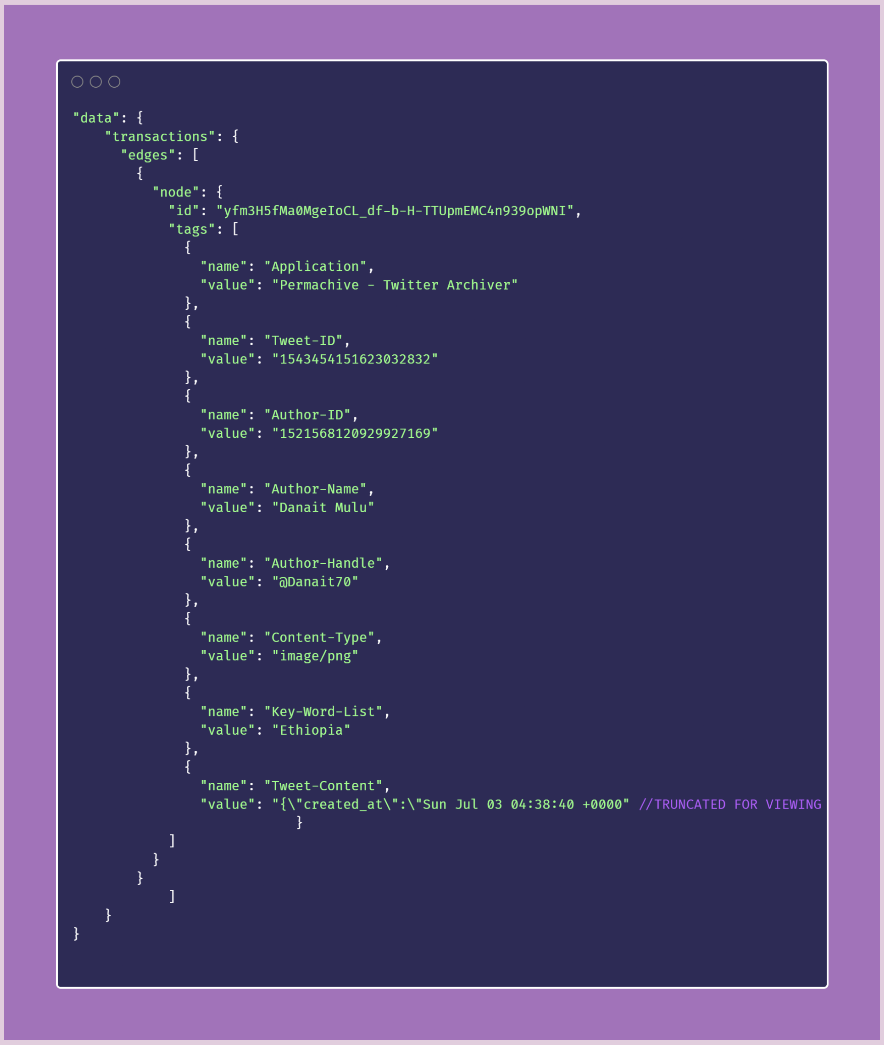


```
query {
    transactions(owners:"<wallet address here>") {
        edges {
            node {
                id
              tags {
                name
                value
              }
            }
        }
    }
}
```

Make sure to use the account that is linked to the archiver.

For examples of transactions created from our first set of tests, the testing account used was "`BGAuyDbdXuSJXfTEM90DbtwGg6sTvEOZNOWnTCHjl-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 the ease of search capabilities.

On-chain Proof & Costs

YouTube: https://arweave.app/tx/TnRu2xgfzx7ikfQoIGTtXCq9i42buW3njDBEAuWyS30
Twitter: https://arweave.app/tx/lptCjqSkRvOxqeUuiWjOQZMSeuC7-1CH7x4Mo6wk3oE
Article: https://arweave.app/tx/dMyr9BF7k3wRRpYFHTroXlSUi421ZJfcfs8jVNTxUGA

Cost Estimation

With the addition of the new compression feature on the Twitter archiver, cost of upkeep has been exponentially decreased. The largest file size for compressed images was ~120 KB, which would only cost a few cents. This was a rare occurrence in our testing with most files remaining under 100 KB. Our current estimate would suggest allocating for at minimum 1 AR token/month, and/or no more than $10/month.

The News Article archiver upload cost varies per upload. If there is a page with few pictures, the cost could be no more than 75 KB. However, if there are a lot of high quality pictures, this could result in multiple MBs costing up to fifty cents/transaction. Luckily, article uploads that match each keyword are not as frequent, sometimes only occurring a few times a day. At most, this would be more expensive than Twitter uploads, with an estimate of $30 a month, or around 2-2.5 AR. Keep in mind, this is a very conservative estimate and would be better adjusted with further testing over a longer period of time.

With the YouTube archiver, costs are extremely variable due to the unexpected upload frequency and length of videos. When applying similar keywords as the other archivers, the last video uploaded that matched was nearly two weeks ago. In the past few months, roughly one video was uploaded per week. In order to determine further testing, we applied a very broad keyword "cats" to find videos. The videos found were generally shorter in length, no longer than a minute, ranging from 120 KB to 1.4 MB. Similar to the News Article Archiver, this would require further testing over a longer period of time to receive a more accurate estimate. However, with these file sizes, we can estimate that the user most hold at least 3-4 AR per month in the wallet file.

Storage Size

Any files that are created by the archiver will be deleted from the machine once the upload is complete. To be safe, I would recommend 1 GB of extra storage space on the machine you are running it on. The biggest files will probably be the YouTube videos, which we observed ranging anywhere from 100 KB to 100 MB. Videos are received and uploaded asynchronously, so at most only one video will be saved locally at any given time.