Smart Proxy Service

The Charity Engine Smart Proxy allows the running of JavaScript applications within fully-featured headless web browsers on a vast network of volunteer computing devices. Smart Proxy v2 currently uses Puppeteer to run a Chrome-based headless browser, which can be controlled via the Puppeteer JavaScript API. Additional browsers and APIs may be supported in the future.

This is a Beta Service

- Features of Smart Proxy may be added, changed, or removed at any time and without advance notice.
- This document is intended to support testing on the path to public release of the service and should not be considered final documentation.

Unless otherwise noted, the mechanisms established by the Charity Engine Distributed Proxy for authentication and configuration still apply, as Smart Proxy is an extension of the Distributed Proxy. Refer to Charity Engine Distributed Proxy Service documentation for details.

Initiating Smart Proxy crawls

Smart Proxy crawls are initiated by connecting to the Distributed Proxy service and supplying additional HTTP headers.

x-proxy-puppeteer-script-url

This header indicates the URL of the script that Smart Proxy should run. Nodes on the Charity Engine network will download and cache the script from this URL when processing a Smart Proxy request that requires it. The URL of the target page to crawl will then be passed as an argument to this script.

While any URL is accepted currently, the service may only allow whitelisted URLs or hostnames in the future.

x-proxy-puppeteer-script-md5

To ensure the integrity of Smart Proxy results, an MD5 hash of the script file defined by the `x-proxy-puppeteer-script-url` HTTP header is required with each request. A script that does not match the supplied MD5 hash will not be run.

Puppeteer script

A script that executes Smart Proxy crawls must define an async function `smartproxy()` that is the entry point to the script. Two parameters - a Puppeteer Page object and the starting URL - will be passed to the function, and a return value of the function will be returned via the proxy.

```
async function smartproxy(page, url) {
  await page.goto('https://example.com');
  const text = await page.evaluate(() => {
    const el = document.querySelector('h1');
    return el.textContent;
  });
  return text;
}
```

Testing locally

To test your script locally, a wrapper function that approximates the Smart Proxy execution environment can be used. The following example code expects the `async function smartproxy(){}` to be placed in a variable `str`, and the starting url to be placed in a variable `url`. The example code will return the main heading from the website `https://www.example.com/`.

```
// Starting URL
const url = 'https://www.example.com/';

// The function that gets executed by the Smart Proxy
const str = 'async function smartproxy(page, url) {' +
    '  await page.goto(url);\n' +
    '  const text = await page.evaluate(() => {\n' +
    '    const el = document.querySelector(\'h1\');\n' +
    '    return el.textContent;\n' +
    '  });' +
    '  return text;' +
    '}';

const puppeteer = require('puppeteer-extra');

// Enable stealth plugin
puppeteer.use(require('puppeteer-extra-plugin-stealth')());

const chromePath = process.env['PROGRAMFILES(X86)'] + '\\Google\\Chrome\\Application\\chrome.exe';

(async () => {
    // Launch the browser in headless mode and set up a page
    const browser = await puppeteer.launch({
        executablePath: chromePath,
        args: ['--no-sandbox'],
        headless: true
    })
    const page = await browser.newPage();

    eval('async function execSP(page, url) {' + str + '; return smartproxy(page, url)}');
    let res = await execSP(page, url);
    console.log('res: ' + res);
    browser.close();
})();
```

Response structure

Any data that is returned from the `smartproxy()` function will be returned back to the server as a string. In case objects are returned, they will be encoded as JSON.

HTTP headers are not returned, and the status code is set to 200 after a successful return from the script. In case HTTP headers or status code from the page are needed, they should be included within the function return value. For example, if you crawl multiple pages and need to return the status codes of them, function return value could look like this:

```
{
  responses: [
    {
      "url": "https://www.example.com/",
      "statusCode": 200
    },
    {
      "url": "https://www.example.com/non-existant-page",
      "statusCode": 404
    }
  ]
}
```

Complex Crawls

Running complex crawls consisting of multiple requests or doing compute-intensive post-processing can provide notable performance benefits, and greatly extend capabilities for knowledge creation and discovery.

Batching requests

Proxy service performance benefits from batching multiple requests together into a single script run. For example, it would be significantly more efficient to download 10 URLs through a single Smart Proxy script in comparison to doing 10 runs with a single URL each, when considering both time and data transfer.

We do not impose limits on the data being returned, but some nodes may be on slow connections, so they may be able to do fewer requests in the allotted time. Script execution duration could be monitored to stop sending additional requests in case the time limit is being approached.

Post processing at the edge

A collection of post-processing functions are available to enhance the capabilities of Smart Proxy scripts. These functions include running text inference via LLMs, integration with container images on Docker Hub, and also processing data using proprietary applications.

For more information on this functionality, including example usage, see documentation on Smart Proxy Post Processing.

Timeout considerations

By default, Smart Proxy requests are terminated after 20 seconds just like Distributed Proxy requests. It is possible to extend the Smart Proxy request timeout by sending `X-Proxy-Timeout-Hard` and `X-Proxy-Timeout-Soft` headers (see Distributed Proxy documentation for details). Both "soft" and "hard" timeouts can be set to a maximum of 120 seconds (2 minutes) for Smart Proxy requests.

Connection will be kept open until Smart Proxy request resolves. This, combined with a long timeout, may cause problems if running a chain of software such as Haproxy. Each step in the chain must be configured to allow long-duration timeouts.

> ⚠ Extending the timeout interval may slightly decrease the success rate of your crawls as nodes can disappear at any time without advance warning. If you see a high rate of timed out crawls try decreasing the size of the crawl executed in a single node.

Cookie persistence

Cookies are not currently persisted between Smart Proxy crawls, but this may change in the future. Please let us know if you need this functionality.