Quick Start Guide
Scraping APIs

- `url`
- `domain`
- `query`
- `num_pages`
- `device_type`
- `locale`
- `headless`

Properties:
- `request`
- `target`
- `price_upper`: float
- `manufacturer`: string
- `customer_reviews`: string
- `price`: float
- `pos`: integer
- `images`: string
- `seller_id`: string
- `color`: string
- `brand`: string
- `customer_reviews`: string
# Table of Contents

1. About Smartproxy

2. Introduction to Scraping APIs

3. How web scraping APIs work

4. Scraping APIs products
   - Web Scraping API
   - SERP Scraping API
   - eCommerce Scraping API
   - Social Media Scraping API

5. Overview and integrations
   - Pricing
   - Authentication methods
   - API playground
   - Requests usage

6. Resources
   - GitHub
   - Postman collections

7. Conclusion
1. About Smartproxy

Smartproxy’s data collection infrastructure helps you effortlessly extract web data from even the most challenging targets. Our products come with award-winning 24/7 support, intuitive self-service dashboard, and flexible pricing plans.
2. Introduction to Scraping APIs

Our Scraping APIs are designed to simplify real-time data collection at scale. They lift the burden of managing proxies, running headless browsers, and overcoming bot detection systems. With a single API call, you can get structured data from the biggest search engines, social media platforms, and eCommerce stores, or raw HTML from any website anywhere in the world.

These APIs are highly scalable and charge only for successful requests, making your expenses predictable. If needed, you can even integrate them in place of a proxy server with very few adjustments.
# Proxies vs Scraping APIs

Smartproxy’s product family consists of proxies and web scraping APIs. You may be wondering how the two compare. The table below explains their main features:

<table>
<thead>
<tr>
<th></th>
<th>Proxies</th>
<th>Scraping API</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Integration methods</strong></td>
<td>Proxy</td>
<td>API, proxy-like</td>
</tr>
<tr>
<td><strong>Proxy servers</strong></td>
<td>✅</td>
<td>✅</td>
</tr>
<tr>
<td><strong>Locations</strong></td>
<td>Worldwide</td>
<td>Worldwide</td>
</tr>
<tr>
<td><strong>IP rotation</strong></td>
<td>Automatic or manual</td>
<td>Automatic</td>
</tr>
<tr>
<td><strong>Automated website unblocking</strong></td>
<td>—</td>
<td>✅</td>
</tr>
<tr>
<td><strong>CAPTCHA solving</strong></td>
<td>—</td>
<td>✅</td>
</tr>
<tr>
<td><strong>JavaScript rendering</strong></td>
<td>—</td>
<td>✅</td>
</tr>
<tr>
<td><strong>Data parsing</strong></td>
<td>—</td>
<td>✅</td>
</tr>
<tr>
<td><strong>Pricing model</strong></td>
<td>Traffic and/or IPs</td>
<td>Successful requests</td>
</tr>
</tbody>
</table>
3. How Scraping APIs work

Scraping APIs let you send API requests with your target and optional parameters like geolocation or JavaScript rendering. The API automatically applies appropriate proxies, headers, and retries the request if necessary until it scrapes the page. It then returns the result over an open HTTP connection.
4. Scraping APIs
products
Web Scraping API

Web Scraping API tries to scrape any URL you pass its way. It returns the page’s HTML.

How does Web Scraping API work?

Step 1
Select the target’s URL and set the JavaScript rendering parameter.

Step 2
Send a POST request to our Web Scraping API

Step 3
Receive full-page HTML data via API
Web Scraping API can return the HTML of any webpage. It makes a perfect choice for use cases where you need a scalable web scraper that can render JavaScript and overcome website protection mechanisms.

**Main features**

- Ability to scrape any website you throw its way
- JavaScript rendering for pages that require headless browsers
- Rich targeting options with over 100 supported countries
- Ability to load pages using mobile & desktop device headers

**Parameters**

Web Scraping API accepts the following parameters. Most of them are optional. The only obligatory parameter is `url`.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>target</td>
<td>string</td>
<td>Should be always set to <em>universal</em> for Web Scraping API</td>
</tr>
<tr>
<td>url</td>
<td>string</td>
<td>Direct URL (link)</td>
</tr>
<tr>
<td>locale</td>
<td>string</td>
<td>This will change the web interface language. Example: – en-US – en-GB</td>
</tr>
<tr>
<td>geo</td>
<td>string</td>
<td>The geographical location that the result depends on. City location names, state names, country names, coordinates and radius, Google’s Canonical</td>
</tr>
<tr>
<td>device_type</td>
<td>string</td>
<td>Device type and browser. Supported: desktop, desktop_chrome, desktop_firefox, mobile, mobile_android, mobile_ios.</td>
</tr>
<tr>
<td>headless</td>
<td>string</td>
<td>Enable JavaScript rendering. Supported: html, png</td>
</tr>
</tbody>
</table>
Authentication methods

The web scraping APIs use username and password authentication. We provide the username, and you can create a password in the dashboard. The credentials are encoded in Base64 and passed as an authorization header.

curl --request POST \
    -- url http://scrape.smartproxy.com/v1/tasks \
    -- header 'Content-Type: application/json' \
    -- header 'accept: application/json' \
    -- header 'authorization: Basic U1B1c2VybmFtZToTUHBhc3N3b3Jk' \
    {
        "target" "google search",
        "query" "world",
        "parse" "true",
        "locale" "en-GB",
        "geo" "London, England, United Kingdom",
    }
Integration methods

All our APIs support two integration methods: real-time and proxy-like. Both return data over an open connection, meaning that you send a request and wait for the response.

- **Real-time**
  This is the main integration method. It lets you send POST requests to the API endpoint with parameters in a JSON payload. This way, you can specify data sources (such as Google Search) instead of providing the full URL.

```bash
curl -u username:password 'https://scrape.smartproxy.com/v1/tasks' -H "Content-Type: application/json" -d '{"target": "google_search", "domain": "com", "query": "world"}"
```

- **Proxy-like**
  This method lets you integrate the APIs as a proxy server. It’s useful when your infrastructure is based on the proxy format, or you’re transitioning from proxies. The method requires passing a full URL with parameters in the request headers.

```bash
```
**Response codes**

The two tables below show the possible response codes you may encounter while using the APIs.

<table>
<thead>
<tr>
<th>Response</th>
<th>Description</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>200 - Success</td>
<td>Server has replied and given the requested response.</td>
<td>Celebrate!</td>
</tr>
<tr>
<td>204 - No content</td>
<td>Job not completed yet.</td>
<td>Wait a few seconds before trying again.</td>
</tr>
<tr>
<td>400 - Multiple error messages</td>
<td>Bad structure of the request.</td>
<td>Re-check your request to make sure it is in the correct format.</td>
</tr>
<tr>
<td>401 - Invalid / not provided authorization header</td>
<td>Incorrect login credentials or missing authorization.</td>
<td>Re-check your provided credentials for authorization.</td>
</tr>
<tr>
<td>403 - Forbidden</td>
<td>Your account does not have access to this resource.</td>
<td>Make sure your Google target is supported by us</td>
</tr>
<tr>
<td>404 - Not found</td>
<td>Your target was not found.</td>
<td>Re-check your targeted URL.</td>
</tr>
<tr>
<td>429 - Too many requests</td>
<td>Exceeded rate limit for your subscription.</td>
<td>Make sure you still have at least one request left. Wait a couple minutes and try again. If you are encountering the error often – chat with us to see if your rate limit can be increased.</td>
</tr>
<tr>
<td>500 - Internal error</td>
<td>Service unavailable, possibly due to some issues we are encountering.</td>
<td>Wait a couple minutes and send another request. Contact us for more information.</td>
</tr>
<tr>
<td>524 - Timeout</td>
<td>Service unavailable, possibly due to some issues we are encountering.</td>
<td>Wait a couple minutes and send another request. Contact us for more information.</td>
</tr>
</tbody>
</table>
### Parsed result response codes:

<table>
<thead>
<tr>
<th>Response</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>12000 - Success</td>
<td>Server has replied and given the requested response.</td>
</tr>
<tr>
<td>12002 - Error</td>
<td>Parsing has failed completely.</td>
</tr>
<tr>
<td>12003 - Not supported</td>
<td>Targeted website parsing is not supported.</td>
</tr>
<tr>
<td>12004 - Response not full</td>
<td>Some fields were not parsed and are missing.</td>
</tr>
<tr>
<td>12005 - Response not fully parsed</td>
<td>Some fields might not have been parsed and are returned unparsed.</td>
</tr>
<tr>
<td>12006 - Error</td>
<td>Unexpected error. Let us know the task ID and we will check what went wrong.</td>
</tr>
<tr>
<td>12007 - Unknown</td>
<td>We could not determine whether the data was parsed correctly.</td>
</tr>
<tr>
<td>12008 - Error</td>
<td>Failed to parse all the data.</td>
</tr>
<tr>
<td>12009 - Error</td>
<td>Target not found. Make sure the parameters you passed are correct and supported.</td>
</tr>
</tbody>
</table>
SERP Scraping API

SERP Scraping API lets you scrape Google, Baidu, Bing, and Yandex by entering a URL or sending the search query as a parameter. It returns data in HTML or, in the case of Google, parsed JSON.

How does SERP Scraping API work?

Step 1
Choose the integration method: either real-time or proxy-like

Step 2
Select a target domain and specify additional parameters (query, device type, parsing, etc.)

Step 3
Send a POST request to one of the available SERP Scraping API sources

Step 4
Receive raw HTML or formatted JSON in seconds
SERP Scraping API is able to extract structured real-time results from major search engines. It makes a perfect choice for search engine optimization, brand protection, and other use cases that involve search engine data.

**Main features**
- Ability to scrape Baidu, Bing, Google, and Yandex
- Localized results with country, state, city, and zip code targeting
- Option to enter a search query as a parameter for easier use
- Parsing capabilities for various Google data types like search results, ads, and Shopping

**Main targets**
- Google Search results with all page elements*
- Google Ads*
- Google Shopping (extract search, product, and pricing data)*
- Google Hotels
- Google Images
- Google Suggest
- Google Trends
- Baidu Search results
- Bing Search results
- Yandex Search results

* parsable
**Parameters**

SERP Scraping API accepts the following parameters. Most of them are optional. The only obligatory parameters are target and url if you're entering a link directly, or target and query.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>target</td>
<td>string</td>
<td>Data source. Available targets are listed <a href="#">here</a>.</td>
</tr>
<tr>
<td>url</td>
<td>string</td>
<td>Direct URL (link)</td>
</tr>
<tr>
<td>domain</td>
<td>string</td>
<td>Top-level domain of your target.</td>
</tr>
<tr>
<td>query</td>
<td>string</td>
<td>...</td>
</tr>
<tr>
<td>page_from</td>
<td>integer</td>
<td>Starting page number.</td>
</tr>
<tr>
<td>num_pages</td>
<td>integer</td>
<td>Number of results to retrieve in each page.</td>
</tr>
<tr>
<td>locale</td>
<td>string</td>
<td>This will change the Google search page web interface language (not the results). Example: – en-US – en-GB</td>
</tr>
<tr>
<td>geo</td>
<td>string</td>
<td>The geographical location that the result depends on. City location names, state names, country names, coordinates and radius, Google’s Canonical</td>
</tr>
<tr>
<td>device_type</td>
<td>string</td>
<td>Device type and browser. Supported: desktop, desktop_chrome, desktop_firefox, mobile, mobile_android, mobile_ios.</td>
</tr>
<tr>
<td><strong>Parameter</strong></td>
<td><strong>Type</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>----------------------------</td>
<td>-------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>parse</td>
<td>boolean</td>
<td>true' will return parsed output in JSON format. Leave blank for HTML – not all data sources can be parsed.</td>
</tr>
<tr>
<td>google_results_language</td>
<td>string</td>
<td>Shows results in a particular language. All of the supported languages are listed <a href="#">here</a>.</td>
</tr>
<tr>
<td>google_tbm</td>
<td>string</td>
<td>This parameter lets you filter Google Search results for specific types of content (news, apps, videos...).</td>
</tr>
<tr>
<td>google_tbs</td>
<td>string</td>
<td>This parameter contains parameters, like limiting/sorting results by date.</td>
</tr>
<tr>
<td>google_safe_search</td>
<td>string</td>
<td>Used to hide explicit content from the results.</td>
</tr>
<tr>
<td>stars</td>
<td>Integer array</td>
<td>2-5 stars, used with google_travel_hotels target</td>
</tr>
<tr>
<td>guests</td>
<td>Integer</td>
<td>Used with google_travel &amp; google_travel_hotels targets</td>
</tr>
<tr>
<td>date_range</td>
<td>string</td>
<td>Y-m-d,Y-m-d used with google_hotels &amp; google_travel_hotels targets</td>
</tr>
<tr>
<td>headless</td>
<td>string</td>
<td>Enable JavaScript rendering. Supported: html, png</td>
</tr>
</tbody>
</table>
Output example for

Google Shopping Pricing

```json
{
  "results": [
    {
      "content": {
        "url": "string",
        "title": "string",
        "rating": float,
        "pricing": [
          {
            "price": float,
            "seller": "string",
            "details": "string",
            "currency": "string",
            "condition": "string",
            "price_tax": float,
            "price_total": float,
            "seller_link": "string",
            "price_shipping": float
          },
          {
            "price": float,
            "seller": "string",
            "details": "string",
            "currency": "string",
            "condition": "string",
            "price_tax": float,
            "price_total": float,
            "seller_link": "string",
            "price_shipping": float
          }
        ],
        "review_count": integer,
        "parse_status_code": 12000
      }
    }
  ]
}
```

Bing, Yandex, Baidu

```json
{
  "results": [
    {
      "content": "<html> page content here</html>",
      "status_code": 200,
      "url": "string",
      "task_id": "string",
      "created_at": "string",
      "updated_at": "string"
    }
  ]
}
```
eCommerce Scraping API

eCommerce Scraping API lets you scrape Amazon and Wayfair by entering a URL or sending the query as a parameter. It returns data in HTML or, in the case of Amazon, parsed JSON.

How does eCommerce Scraping API work?

Step 1
Choose the integration method: either real-time or proxy-like

Step 2
Select a target domain and specify additional parameters (query, device type, parsing, etc.)

Step 3
Send a POST request to one of the available SERP Scraping API sources

Step 4
Receive raw HTML or formatted JSON in seconds
eCommerce Scraping API is able to extract structured real-time results from major eCommerce websites. It makes a perfect choice for price monitoring, market research, and other use cases that involve eCommerce data.

**Main features**

- Ability to scrape Amazon and Wayfair
- JavaScript rendering for pages that require headless browsers
- Option to enter a search query or item code as a parameter for easier use
- Parsing capabilities for various Amazon data types like search results, product pages, and reviews

**Main targets**

- Amazon search pages*
- Amazon product pages*
- Amazon product pricing*
- Amazon product reviews*
- Amazon product questions*
- Amazon sellers*
- Wayfair product pages

* parsable
### Parameters

eCommerce Scraping API accepts the following parameters. Most of them are optional. The only obligatory parameters are url if you’re entering a link directly, or target and query.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>target</td>
<td>string</td>
<td>Data source. Available targets are listed <a href="#">here</a>.</td>
</tr>
<tr>
<td>url</td>
<td>string</td>
<td>Direct URL (link)</td>
</tr>
<tr>
<td>parse</td>
<td>boolean</td>
<td>True’ will return parsed output in JSON format. Leave blank for HTML – not all data sources can be parsed.</td>
</tr>
<tr>
<td>domain</td>
<td>string</td>
<td>Top-level domain of your target.</td>
</tr>
<tr>
<td>query</td>
<td>string</td>
<td>...</td>
</tr>
<tr>
<td>page_from</td>
<td>integer</td>
<td>Starting page number.</td>
</tr>
<tr>
<td>num_pages</td>
<td>integer</td>
<td>Number of results to retrieve in each page.</td>
</tr>
<tr>
<td>locale</td>
<td>string</td>
<td>This will change the web interface language. Example: – en-US – en-GB</td>
</tr>
<tr>
<td>geo</td>
<td>string</td>
<td>The geographical location that the result depends on. City location names, state names, country names, coordinates and radius, Google’s Canonical</td>
</tr>
<tr>
<td>Parameter</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>device_type</td>
<td>string</td>
<td>Device type and browser. Supported: desktop, desktop_chrome, desktop_firefox, mobile, mobile_android, mobile_ios.</td>
</tr>
<tr>
<td>headless</td>
<td>string</td>
<td>Enable JavaScript rendering. Supported: html, png</td>
</tr>
</tbody>
</table>

Output example for

Amazon Pricing

```json
{
   "results": [
     {
       "content": {
         "url": "string",
         "asin": "string",
         "page": integer,
         "title": "string",
         "pricing": [
           {
             "price": float,
             "seller": "string",
             "currency": "string",
             "delivery": "string",
             "condition": "string",
             "seller_id": "string",
             "seller_link": "string",
             "rating_count": integer,
             "price_shipping": float,
             "delivery_options": []
           },
           ]
       }
     },
     "asin_in_url": "string",
     "review_count": integer,
     "parse_status_code": 12000
   ]
}
}
Social Media Scraping API

Social Media Scraping API lets you scrape Instagram and TikTok by entering a URL or sending the query as a parameter. It returns data in HTML or parsed JSON.

How does Social Media Scraping API work?

Step 1
Choose the integration method: either real-time or proxy-like

Step 2
Select a target domain and specify additional parameters (query, device type, parsing, etc.)

Step 3
Send a POST request to one of the available SERP Scraping API sources

Step 4
Receive raw HTML or formatted JSON in seconds
Social Media Scraping API is able to extract structured results from major social media platforms in real time or on demand. It makes a perfect choice for sentiment analysis, influencer marketing, and other use cases that involve social media data.

**Main features**
- Ability to scrape Instagram and TikTok
- Option to target a GraphQL endpoint or fully render the page
- Real-time or on-demand data delivery
- Parsing capabilities for extracting structured results

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>url</td>
<td>Social Media URL</td>
</tr>
<tr>
<td>target</td>
<td>string</td>
<td>Desired target</td>
</tr>
<tr>
<td>locale</td>
<td>string</td>
<td>Language Locale</td>
</tr>
<tr>
<td>geo</td>
<td>string</td>
<td>Geolocation</td>
</tr>
</tbody>
</table>
Output example for TikTok

```json
{
    "data": {
        "content": {
            "nickname": "string",
            "verified": boolean,
            "avatarThumb": "string",
            "openFavorite": boolean,
            "ttSeller": boolean,
            "postInfo": {
                "id": "string",
                "description": "string",
                "postedAtTimestamp": integer,
                "postedAt": "string",
                "author": "string",
                "music": {
                    "id": "string",
                    "title": "string",
                    "playUrl": "string",
                    "coverLarge": "string",
                    "coverMedium": "string",
                    "coverThumb": "string",
                    "authorName": "string",
                    "original": boolean,
                    "duration": integer,
                    "scheduleSearchTime": integer
                },
                "shareCount": integer,
                "commentCount": integer,
                "playCount": integer,
                "accountLikes": integer
            }
        },
        "errors": [],
        "status_code": integer
    },
    "task_id": "string",
    "url": "string"
}
```
Output example for Instagram

```json
{
  "data": {
    "content": {
      "user": {
        "biography": "string",
        "bio_links": [
          {
            "title": "string",
            "lynx_url": "string",
            "url": "string",
            "link_type": "string"
          }
        ],
        "biography_with_entities": {
          "raw_text": "string",
          "entities": []
        },
        "blocked_by_viewer": boolean,
        "restricted_by_viewer": boolean,
        "country_block": boolean,
        "external_url": "string",
        "external_url_linkshimmed": "string",
        "edge_followed_by": {
          "count": integer
        },
        "fbid": "string",
        "followed_by_viewer": boolean,
        "edge_follow": {
          "count": 1111
        },
        "follows_viewer": boolean,
        "full_name": "string",
        "group_metadata": "string",
        "has_ar_effects": boolean,
        "has_clips": boolean,
        "has_guides": boolean,
        "has_channel": boolean,
        "has_blocked_viewer": boolean,
        "highlight_reel_count": integer,
        "has_requested_viewer": boolean,
        "hide_like_and_view_counts": boolean,
        "id": "string",
        "is_business_account": boolean,
        "is_professional_account": boolean
      }
    }
  }
}```
5. Overview and integrations

Each scraping API has its own section in the dashboard. There, you can manage your subscription, set up the API, and track usage statistics.
Pricing

The Pricing tab lets you buy, upgrade, or renew a subscription.

Authentication method

The Authentication method tab lets you create a password for accessing the APIs.
The API playground tab includes an interactive widget for configuring the API. You can use it to test requests even without a subscription, and it generates dynamic code samples for easier integration.
Statistics tab shows your request expenditure over time.

You can select a preset time period (such as week or month) or enter custom dates.
6. Resources

The following resources can help you learn more about the implementation and functionality of the scraping APIs.

GitHub

Our GitHub includes detailed code samples for the most popular programming languages like Python, PHP, and Node.js.

Postman collections

You can also take a look at our Postman recipes which explain each API line by line.
Smartproxy’s web scraping APIs were designed to help you effortlessly gather web data at any scale. With the ability to target a wide range of locations, render JavaScript, and parse major search, social media, and e-commerce websites, they can make your data collection operations more efficient and predictable.

We hope that you’ve found this guide helpful. We’d love to talk to you about how Smartproxy’s web scraping APIs can support your organization. You can book a call with us.