Google Maps Scraper Kit: The Full Setup Guide

Run a free, open-source Google Maps scraper on your own computer and let Claude Code pull local leads for you, no subscription, no data cap.

Why I built this into my workflow

Most lead databases charge a monthly fee just to look up business contact info you could technically find yourself. And when you ask a general AI chatbot to just go grab it from Google Maps, it usually comes back empty because Google Maps sits behind a consent wall and renders everything with JavaScript. A plain fetch request sees none of it.

Google Maps Scraper Kit fixes this by running a real scraping engine on your own computer through Docker, and letting Claude Code drive it through a local API instead of trying to read the page directly. You tell Claude a city and a business type, and it hands back a clean list: name, phone, email, website, address, rating, and review count.

The tool itself is a wrapper around an open-source scraper called gosom/google-maps-scraper, built by developer Georgios Komninos under the MIT license. This kit adds the Docker setup, the scripts, and a Claude skill on top so you don't have to read the underlying project's API docs yourself.

- What it pulls by default: name, phone, email, website, category, address, rating, review count. The scraper actually captures around 34 raw fields per business, but the kit strips out geo coordinates, internal IDs, hours, and image blobs so you're left with a usable lead list.
- What the --socials flag adds: each business's Instagram, Facebook, and LinkedIn, found by scanning its own website. This runs as plain HTTP requests plus regex matching, so it costs zero AI tokens, though coverage only covers businesses that actually link their socials on their site.
- What it does not do: touch Instagram, TikTok, or YouTube directly. It's built for Google Maps business listings only, on purpose.

01

How it actually works

1. You start the scraper locally with Docker. It runs at http://localhost:8080 and only accepts connections from your own machine, nothing is exposed to the internet.
2. You tell Claude Code the city and business type, either in plain English or with a slash command. Claude builds a scrape job with the required fields (keywords, lat, lon, max_time) and submits it to the local API.
3. Claude polls the job until it finishes, downloads the results, and strips the raw output down to a usable lead list. If you turned on the --socials option, it also scans each business's website for Instagram, Facebook, and LinkedIn links.

Speed depends on depth and concurrency. The upstream scraper measures roughly 120 places per minute at a concurrency of 8 and a depth of 1, so a modest job (a couple hundred businesses at depth 5) is realistically a few minutes of waiting, not an overnight job.

02

The one thing almost everyone gets wrong

People treat this like a chatbot that can scrape anything, and run it too hard. This is a real scraper hitting Google's servers, not a magic trick, so it behaves like one.

> **WATCH OUT**
>
> Running big jobs back to back, setting depth very high, queuing many keywords at once, or scheduling repeated runs without proxies can get your IP temporarily rate-limited by Google. It does not ban your Google account, and it usually clears in minutes to a few hours, but while it's happening jobs come back failed or with empty or unusually short results. Start at depth 5, run one job at a time, and only add proxies (socks5, socks5h, http, or https) once you're doing larger or repeated scrapes.

03

The files

> **SOURCE CODE**
>
> The full repo, including the Docker setup, the scripts, and the Claude skill:
>
> **https://github.com/Mahanaicoach/google-maps-scraper-kit**

EXAMPLES/QUERIES.EXAMPLE.JSON

```
{
  "job": {
    "name": "my scrape",
    "keywords": ["coffee shops in Austin TX"],
    "lang": "en",
    "zoom": 15,
    "lat": "30.2672",
    "lon": "-97.7431",
    "fast_mode": false,
    "radius": 10000,
    "depth": 5,
    "email": false,
    "max_time": 300,
    "proxies": null
  }
}
```

SCRIPTS/SCRAPE.SH USAGE

```
./scripts/scrape.sh "coffee shops in Austin TX" 30.2672 -97.7431 5
```

SCRIPTS/SCRAPE.PY USAGE (WITH AUTO-GEOCODING)

```
python3 scripts/scrape.py "cafes in Austin TX" --city "Austin, TX" --depth 5
```

SCRIPTS/SCRAPE.PY BATCH USAGE

```
python3 scripts/scrape.py --keywords-file examples/queries.example.txt --city "Denver,
CO"
```

City coordinates cheatsheet from examples/queries.example.json, so you don't have to look them up:

- Austin, TX: 30.2672, -97.7431
- Miami, FL: 25.7617, -80.1918
- Denver, CO: 39.7392, -104.9903
- New York, NY: 40.7128, -74.0060
- Los Angeles, CA: 34.0522, -118.2437

Set it up in Claude Code

1. Install Docker Desktop if you don't have it, then confirm it's running with docker --version and docker ps (the second command should not error).
2. Clone the repo: git clone https://github.com/Mahanaicoach/google-maps-scraper-kit and cd into the folder it creates.
3. The skill folder is already inside the repo at .claude/skills/google-maps-scraper/SKILL.md, and CLAUDE.md at the project root auto-loads when Claude Code opens this folder, so there's nothing extra to wire up.
4. Copy the config template if you want to customize anything: cp .env.example .env. Defaults work as-is for a first run.
5. Start the scraper: docker compose up -d. First run pulls the scraper's Docker image, which takes a few minutes depending on your connection.
6. Confirm it's alive: curl http://localhost:8080/api/v1/jobs should print an empty list [] or existing jobs, not a connection error.
7. Open the folder in Claude Code by running claude from inside it, then just talk to it: "scrape coffee shops in Austin" or "find gyms in Miami with phone numbers."
8. Watch for the first successful run: Claude reports the job as done, downloads a CSV, and gives you a table with real business names, phone numbers, and websites, not an empty result.

> 💡 **QUICK TIPS**
>
> - On Apple Silicon Macs, the scraper image is linux/amd64 and runs under emulation. If the container won't start, uncomment the platform: linux/amd64 line in docker-compose.yml.
> - On Windows, use scripts/scrape.py with py or python3. The bash script scrape.sh needs WSL2 or Git Bash.
> - This project doesn't need an API key to run locally, so there's nothing to add to your shell profile for the core scraper itself.

04

The commands I actually run

FIRST HEALTH CHECK AFTER STARTING DOCKER

```
Check that the Google Maps scraper container is running, and confirm the API at
localhost:8080 is responding.
```

A BASIC LOCAL LEAD PULL

```
Scrape [BUSINESS TYPE] in [CITY, STATE], depth 5, and give me a table with name, phone,
and website.
```

PULLING SOCIALS ALONG WITH THE LEADS

```
Scrape [BUSINESS TYPE] in [CITY, STATE] and also pull their Instagram, Facebook, and
LinkedIn from their websites.
```

A BATCH RUN ACROSS SEVERAL KEYWORDS

```
Run a batch scrape using examples/queries.example.txt for [CITY, STATE], one job, and summarize the total leads found when it's done.
```

CHECKING ON A RUNNING JOB

```
List the current scrape jobs and tell me if any are still in progress or came back failed.
```

SCALING UP A LARGER JOB SAFELY

```
I need to scrape [BUSINESS TYPE] across [NUMBER] cities. Set up proxies so we don't get rate-limited, and run the jobs one at a time.
```

When it breaks

ERROR	FIX
curl: connection refused on :8080	Container isn't up. Run docker compose up -d, then check docker ps.
422 missing max time	The job body needs max_time in seconds, e.g. 300.
422 missing geo coordinates	The job body needs lat and lon as strings, e.g. "30.2672".
Job stuck on working forever	Lower the depth setting, or your IP is being throttled by Google. Wait it out or add proxies.
Empty CSV back	Keyword is too narrow or the geo is wrong. Widen the radius or double check lat/lon.
Docker pull is very slow	Normal on the first run only. It caches the image after that.
Jobs suddenly returning failed after working fine	Classic rate-limit signal. Slow down, drop to one job at a time, lower depth, or add proxies.