Open in app ↗

Search

Write

Web Scraping with Puppeteer Extra, Typescript, & AWS Lambda

An in-depth guide on how to build a web scraper using the best tools available.

Joe Osborne · Follow

9 min read · Jan 20, 2024

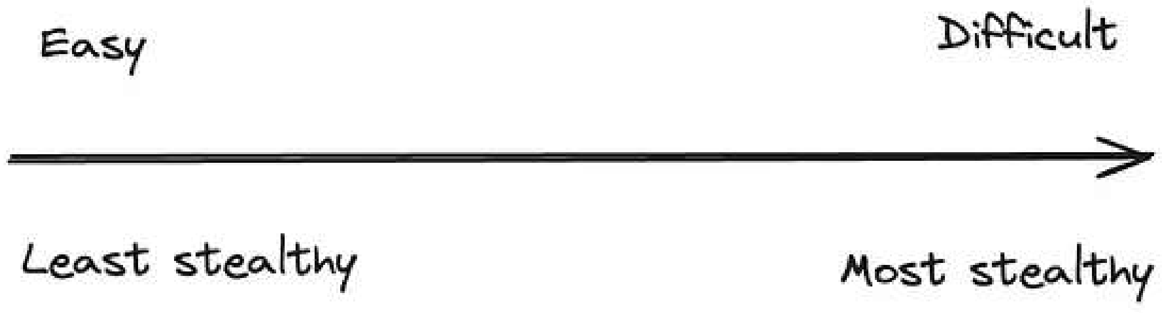
64 2

This guide will walk through step-by-step how to build and deploy robust, stealthy web scrapers utilizing these tools:

- **Typescript** (Typescript >>> Javascript)
- **Puppeteer Extra** (lots of cool plugins available with Extra!)
- **AWS Lambda** for compute (EC2/ECS is FAR cheaper at scale, but Lambda is great for one-offs and quick deployment)
- **Serverless** for deployment
- **AWS CodeBuild** (in place of running memory-hogging Docker locally)
- **AWS Elastic Container Registry** (ECR)
- **GitHub**

- **Proxy services from oxylabs.io**

These tools above have become my holy grail for web scraping difficult sites. When I approach a new project, I go through a list of methods from easiest to hardest to see what level of scraper I need to successfully get the data I'm looking for. If the website blocks a method, I move up a tier. The scraping continuum typically looks like this:



If the website we're scraping employs no blocking, no authentication, no anti-scraping measures, then we probably don't need Puppeteer. Here's my basic tier list:

1. Python `requests`+ `BeautifulSoup` or Javascript/Typescript `fetch` + `cheerio` to get HTML from the webpage.
2. Internal API calls found in the network tab that don't require strict auth — client side requests that populate the data you see on the webpage.
3. A combination of the above two using a cheap proxy service to rotate IP addresses if blocked.

4. Puppeteer to spawn a browser and the ability to interact with the site if needed.
5. Puppeteer Extra using the stealth plugin in combination with a proxy service.

#5 is what I'll be walking through in this guide. It is so far unbeaten in accessing tough websites for me. It's my last resort as it requires the most set up, but it's as close as you can get to mimicking human behavior with web automation.

Typescript + Initial Setup

I highly prefer Typescript over Javascript. In my opinion it makes code infinitely easier to work with and understand. However, this guide will still work with Javascript if that's what you prefer.

Set up a new directory and run `npm init` . Here's a sample `package.json` with the needed dependencies:

```
{
  "name": "Puppeteer Extra with AWS Lambda",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "deploy:dev": "serverless deploy --stage dev --verbose"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "aws-lambda": "^1.0.7",
    "puppeteer": "^21.6.1",
    "puppeteer-extra": "^3.3.6",
    "puppeteer-extra-plugin-proxy": "^1.0.2",
    "puppeteer-extra-plugin-stealth": "^2.11.2"
  },
  "devDependencies": {
    "@types/aws-lambda": "^8.10.130",
    "ts-node": "^10.9.2"
  }
}
```

Here's a sample `tsconfig.json` I would recommend:

```
{
  "compilerOptions": {
    "module": "commonjs",
    "noImplicitAny": true,
    "removeComments": false,
    "preserveConstEnums": true,
    "sourceMap": true,
    "outDir": "./dist",
    "rootDir": "./",
    "strict": true,
    "target": "es2022",
    "strictPropertyInitialization": false,
    "experimentalDecorators": true
  }
}
```

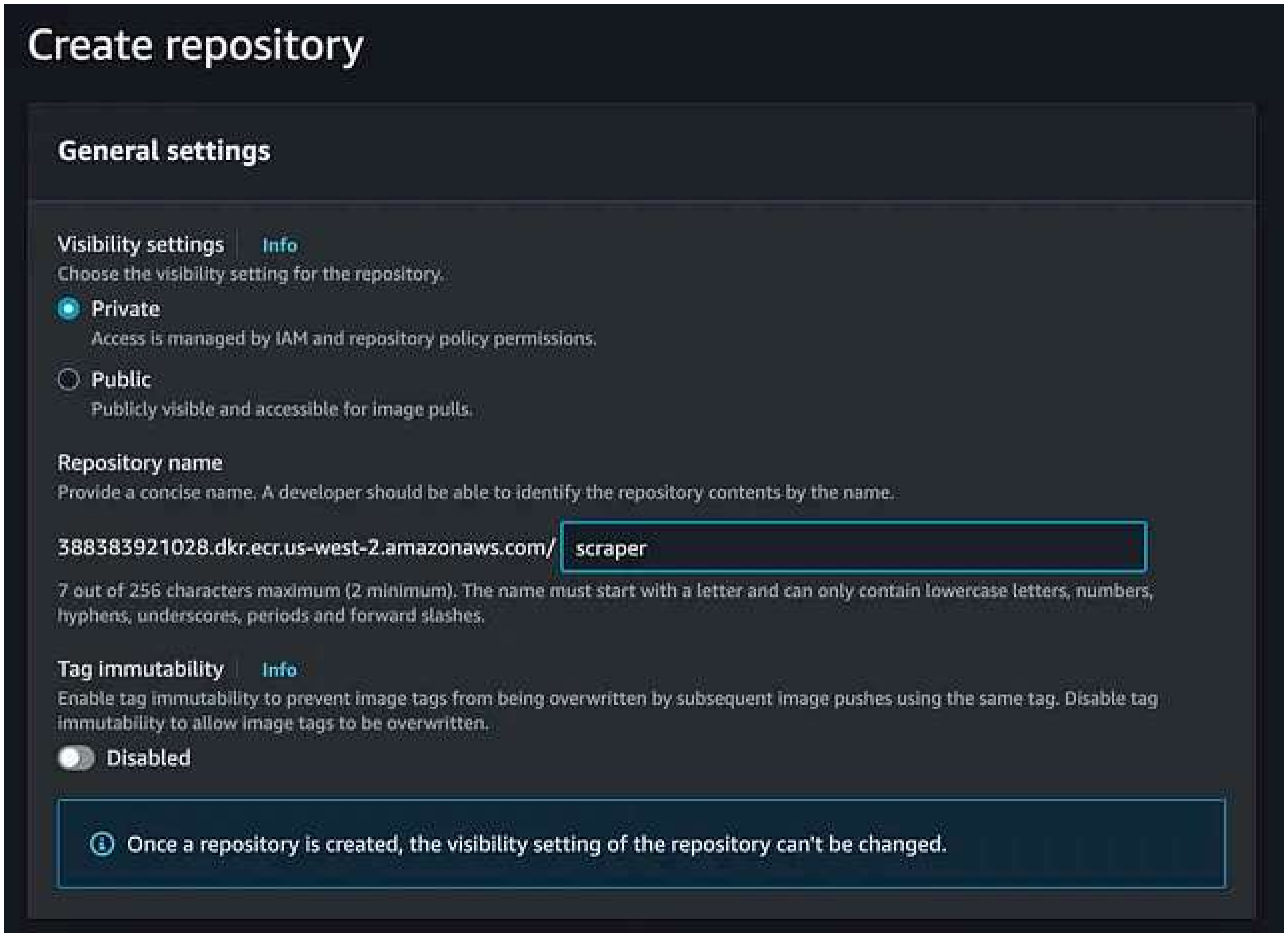
Important! — Before running `npm install` , we need to set up a `.puppeteerrc.cjs` file to set up some configuration that will help our scraper find the browser executable both locally and in Lambda. Without it, your Lambda function by default won't be able to find the correct executable path.

```
const { join } = require("path");

/**
 * @type {import("puppeteer").Configuration}
 */
module.exports = {
  // Changes the cache location for Puppeteer.
  cacheDirectory: join(__dirname, ".cache", "puppeteer"),
  downloadPath: join(__dirname, ".cache", "puppeteer", "downloads"),
};
```

Now, go ahead and run `npm install` . At this point, you should also set up a GitHub repository for this project that we will use later to connect to AWS CodeBuild.

AWS Elastic Container Registry

Set up a new repository in AWS. For the sake of this guide, we'll name ours `scraper` :



Create repository

General settings

Visibility settings | Info

Choose the visibility setting for the repository.

[✓ Private] Access is managed by IAM and repository policy permissions.

Public Publicly visible and accessible for image pulls.

Repository name

Provide a concise name. A developer should be able to identify the repository contents by the name.

388383921028.dkr.ecr.us-west-2.amazonaws.com/ scraper

7 out of 256 characters maximum (2 minimum). The name must start with a letter and can only contain lowercase letters, numbers, hyphens, underscores, periods and forward slashes.

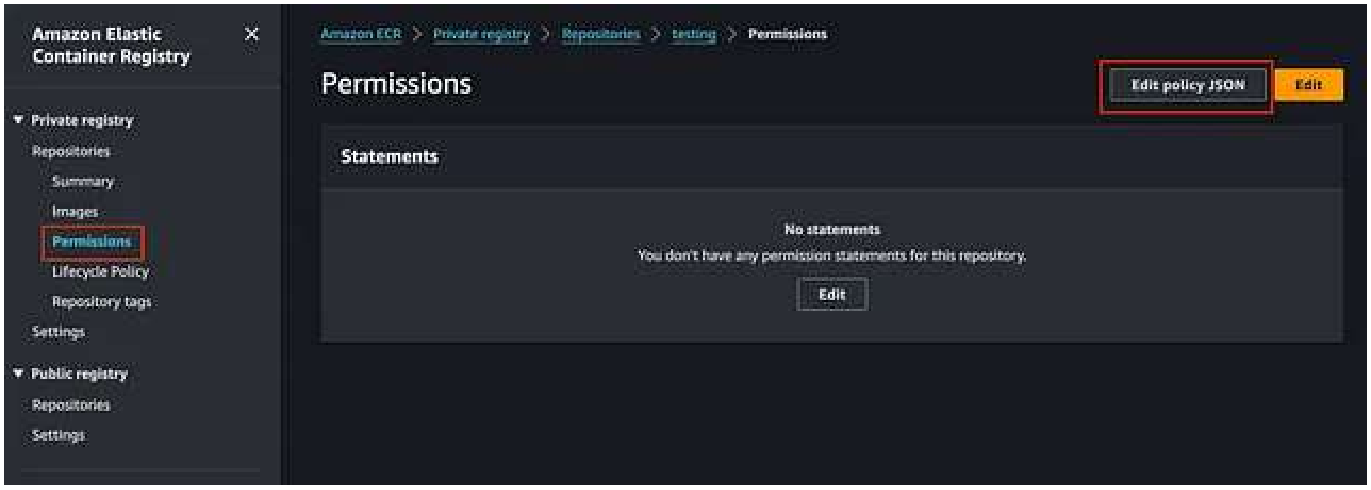
Tag immutability | Info

Enable tag immutability to prevent image tags from being overwritten by subsequent image pushes using the same tag. Disable tag immutability to allow image tags to be overwritten.

Disabled

Once a repository is created, the visibility setting of the repository can't be changed.

Once it's created, we'll need to give it the needed permissions. Click into your registery and select "Permissions" on the left sidebar. Click "Edit policy JSON" in the top right:



No statements
You don't have any permission statements for this repository.

Copy this JSON object into the editor and replace `<aws-account-id>` with your account ID:

```
{
  "Version": "2008-10-17",
  "Statement": [
    {
      "Sid": "LambdaECRImageRetrievalPolicy",
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Action": [
        "ecr:BatchGetImage",
        "ecr:DeleteRepositoryPolicy",
        "ecr:GetDownloadUrlForLayer",
        "ecr:GetRepositoryPolicy",
        "ecr:SetRepositoryPolicy"
      ],
      "Condition": {
        "StringLike": {
          "aws:sourceArn": "arn:aws:lambda:us-west-2:<aws-account-id>:function:*
        }
      }
      }
    ]
  }
```

AWS Lambda + Serverless

AWS Lambda is the service we will use for compute. It's extremely useful for one-off projects and quick deployment. It can be a double edged sword in that it can give instant access to thousands of GB of RAM, but can quickly get very expensive. If the goal is to build a long lasting scraping system, I recommend using ECS to keep costs down.

Prerequisite for this section — an AWS account with local credentials set up, and necessary permissions for Cloudformation, Cloudwatch, S3, ECR, and Lambda. I may be missing some, but it's not too difficult to add permissions as you go.

Set up a `serverless.yml` file and input your AWS account ID in the `uri` field.

```
service: puppeteer-scraper

provider:
  environment:
    ENV: ${self:provider.stage}
  name: aws
  region: us-west-2
  runtime: nodejs20.x
  stage: ${opt:stage, "dev"}

functions:
  scraper:
    memorySize: 512
    timeout: 300
    image:
      uri: <aws-account-id>.dkr.ecr.us-west-2.amazonaws.com/scraper:latest
      command:
        - dist/index.handler
```

AWS CodeBuild

We will use CodeBuild as a replacement for locally running Docker. I'm impatient with the amount of memory Docker Desktop takes up when running, and it has crashed my machine on more than one occasion. However if you prefer to use it instead of CodeBuild, the deploy script I use should still work fine.

We'll need to set up three files, a `Dockerfile` , `buildspec.yml` , and `deploy.sh`.

Feel free to copy these files directly:

`Dockerfile`:

```
# Define custom function directory
ARG FUNCTION_DIR="/function"

FROM node:bookworm as build-image

RUN ls -la

RUN cat /etc/os-release
RUN uname -a

ENV NPM_CONFIG_CACHE=/tmp/.npm3

# Include global arg in this stage of the build
ARG FUNCTION_DIR

# Install build dependencies
RUN apt-get update && \
    apt-get install -y \
    g++ \
    make \
    cmake \
    unzip \
    libcurl4-openssl-dev

# Copy function code
RUN mkdir -p ${FUNCTION_DIR}
COPY . ${FUNCTION_DIR}

WORKDIR ${FUNCTION_DIR}

RUN pwd
RUN ls -la

RUN npm install

RUN npm install aws-lambda-ric

RUN npx puppeteer browsers install chrome

# Install some extra dependencies
RUN apt-get install -y \
    libnss3 libnss3-dev \
    libnspr4 libnspr4-dev \
    libdbus-1-3 \
    libatk1.0-0 libatk-bridge2.0-0 \
    libcups2 \
    libdrm2 \
    libxkbcommon0 \
    libatspi2.0-0 \
    libxcomposite1 \
    libxdamage1 \
    libxfixes3 \
    libxrandr2 \
    libgbm1 \
    libasound2 \
    ca-certificates \
    fonts-liberation \
    libappindicator3-1 \
    libc6 \
    libcairo2 \
    libexpat1 \
    libfontconfig1 \
    libgcc1 \
    libglib2.0-0 \
    libgtk-3-0 \
    libpango-1.0-0 libpangocairo-1.0-0 \
    libstdc++6 \
    libx11-6 libx11-xcb1 libxcb1 \
    libxcursor1 \
    libxdamage1 \
    libxext6 \
    libxfixes3 \
    libxi6 \
    libxrandr2 \
    libxrender1 \
    libxss1 \
    libxtst6 \
    lsb-release \
    wget \
    xdg-utils

RUN pwd
RUN ls -la

# Set runtime interface client as default command for the container runtime
ENTRYPOINT ["/usr/local/bin/npx", "aws-lambda-ric"]
```

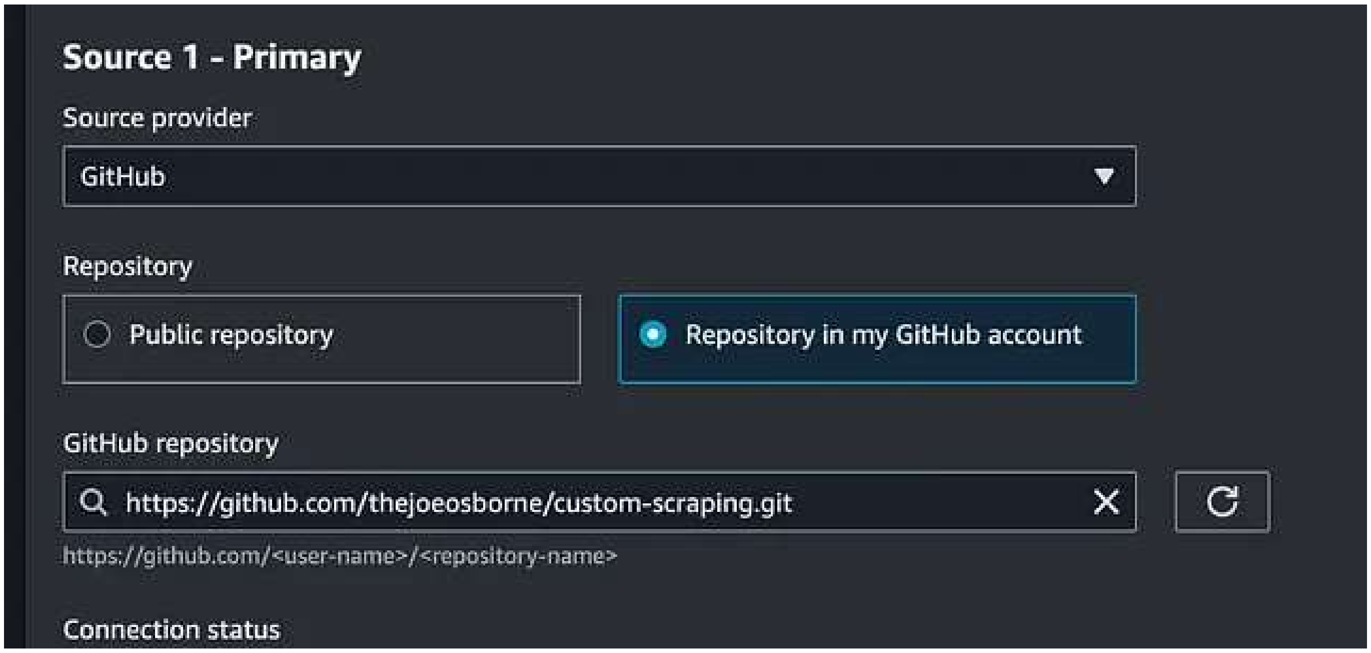
`deploy.sh` — replace <aws-account-id> with your account ID. Also, make sure to run `chmod +x deploy.sh` to make it executable.

```
#!/bin/bash

echo "command - npx tsc"
npx tsc

echo "command - aws ecr get-login-password --region us-west-2 | docker login --u
aws ecr get-login-password --region us-west-2 | docker login --username AWS --pa

echo "command - docker build --platform linux/amd64 --progress=plain -t scraper
docker build --platform linux/amd64 --progress=plain -t scraper .

echo "command - docker tag scraper:latest <aws-account-id>.dkr.ecr.us-west-2.ama
docker tag scraper:latest <aws-account-id>.dkr.ecr.us-west-2.amazonaws.com/scrap

echo "command - docker push <aws-account-id>.dkr.ecr.us-west-2.amazonaws.com/scr
docker push <aws-account-id>.dkr.ecr.us-west-2.amazonaws.com/scraper:latest
```

`buildspec.yml`

```
version: 0.2
env:
  variables:
    REGION: us-west-2
    VERSION: "1"

phases:
  install:
    on-failure: ABORT
    runtime-versions:
      nodejs: 20
  pre_build:
    on-failure: ABORT
    commands:
      - echo "Installing dependencies..."
      - npm install
  build:
    on-failure: ABORT
    commands:
      - echo "Building..."
      - $CODEBUILD_SRC_DIR/deploy.sh
```

Now go ahead and set up a new build project in CodeBuild. You'll need to connect your GitHub account and choose the repo you set up for this project. Here are the main configurations needed to set, the defaults will work for the rest:


Source 1 - Primary

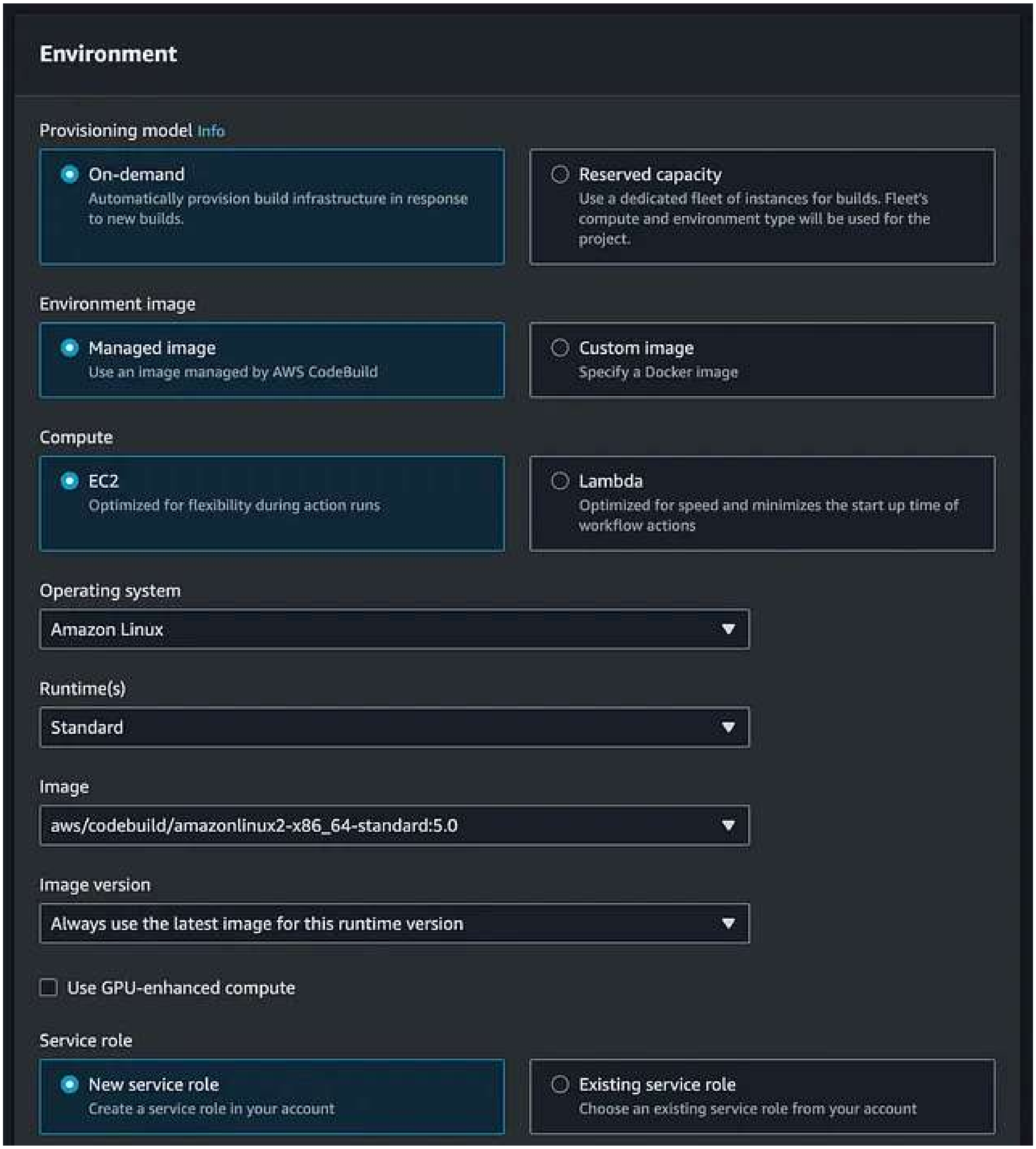
Source provider: GitHub

Repository: ( ) Public repository [✓ Repository in my GitHub account]

GitHub repository: https://github.com/thejoeosborne/custom-scraping.git

https://github.com/<user-name>/<repository-name>

Connection status


Environment

Provisioning model Info

[✓ On-demand] Automatically provision build infrastructure in response to new builds.

Reserved capacity: Use a dedicated fleet of instances for builds. Fleet's compute and environment type will be used for the project.

Environment image

[✓ Managed image] Use an image managed by AWS CodeBuild

Custom image: Specify a Docker image

Compute

[✓ EC2] Optimized for flexibility during action runs

Lambda: Optimized for speed and minimizes the start up time of workflow actions

Operating system: Amazon Linux

Runtime(s): Standard

Image: aws/codebuild/amazonlinux2-x86_64-standard:5.0

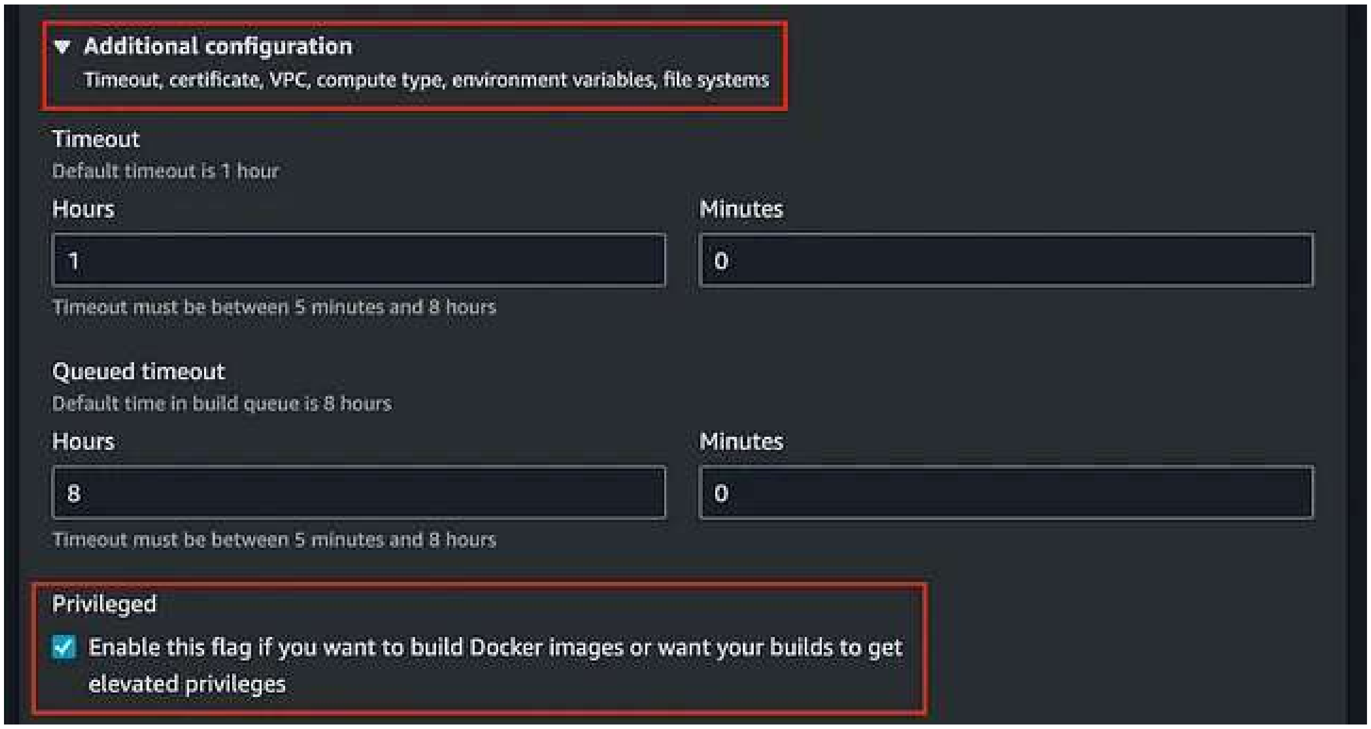
Image version: Always use the latest image for this runtime version

☐ Use GPU-enhanced compute

Service role

[✓ New service role] Create a service role in your account

Existing service role: Choose an existing service role from your account


▼ Additional configuration
Timeout, certificate, VPC, compute type, environment variables, file systems

Timeout
Default timeout is 1 hour

Hours	Minutes
1	0

Timeout must be between 5 minutes and 8 hours

Queued timeout
Default time in build queue is 8 hours

Hours	Minutes
8	0

Timeout must be between 5 minutes and 8 hours

Privileged
✓ Enable this flag if you want to build Docker images or want your builds to get elevated privileges

Scraper

Now we can set up our scraper. Go ahead and make an `index.ts` file at the root of your project. Here's a sample you can use. It includes launch configurations that check if you're running on Lambda or locally. It also includes a commented out section to use proxies, we'll get to that soon!

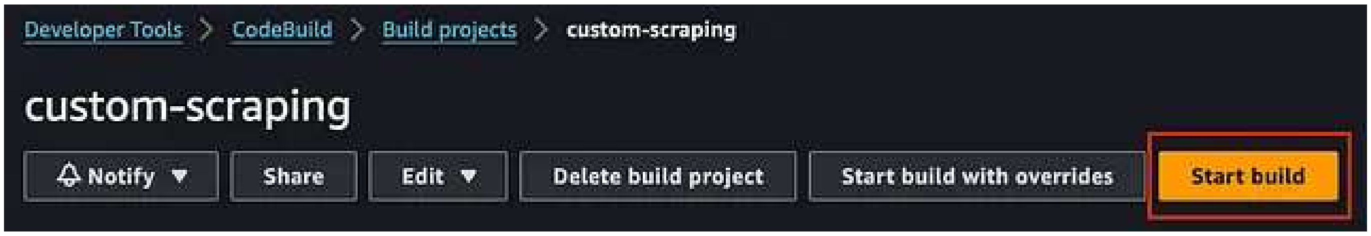
```
import { Callback, Context, Handler } from "aws-lambda";
import { Browser, Page, PuppeteerLaunchOptions } from "puppeteer";
import { PuppeteerExtra } from "puppeteer-extra";

interface ExampleEvent {
  //type your event here if desired
}

export const handler: Handler = async (
  event: ExampleEvent,
  context: Context,
  callback: Callback
): Promise<any> => {
  try {
    console.log("event:", event);
    const puppeteer: PuppeteerExtra = require("puppeteer-extra");
    const stealthPlugin = require("puppeteer-extra-plugin-stealth");
    puppeteer.use(stealthPlugin());

    // const proxyPlugin = require("puppeteer-extra-plugin-proxy");
    // puppeteer.use(
    //   proxyPlugin({
    //     address: "pr.oxylabs.io",
    //     port: 7777,
    //     credentials: {
    //       username: "customer-someUsername-cc-US",
    //       password: "somePassword",
    //     },
    //   })
    // );

    const launchOptions: PuppeteerLaunchOptions = context.functionName
      ? {
          headless: true,
          executablePath: puppeteer.executablePath(),
          args: [
            "--no-sandbox",
            "--disable-setuid-sandbox",
            "--disable-dev-shm-usage",
            "--disable-gpu",
            "--single-process",
            "--incognito",
            "--disable-client-side-phishing-detection",
            "--disable-software-rasterizer",
          ],
        }
      : {
          headless: false,
          executablePath: puppeteer.executablePath(),
        };

    const browser: Browser = await puppeteer.launch(launchOptions);
    const page: Page = await browser.newPage();
    await page.goto("https://www.example.com");
    await new Promise((resolve) => setTimeout(resolve, 5000));
    console.log(await page.content());
    await browser.close();
  } catch (e) {
    console.log("Error in Lambda Handler:", e);
    return e;
  }
};
```

Deploy + Run

Now we can test! To test locally, add this code snippet to the bottom of the file and run `npx ts-node index.ts` .

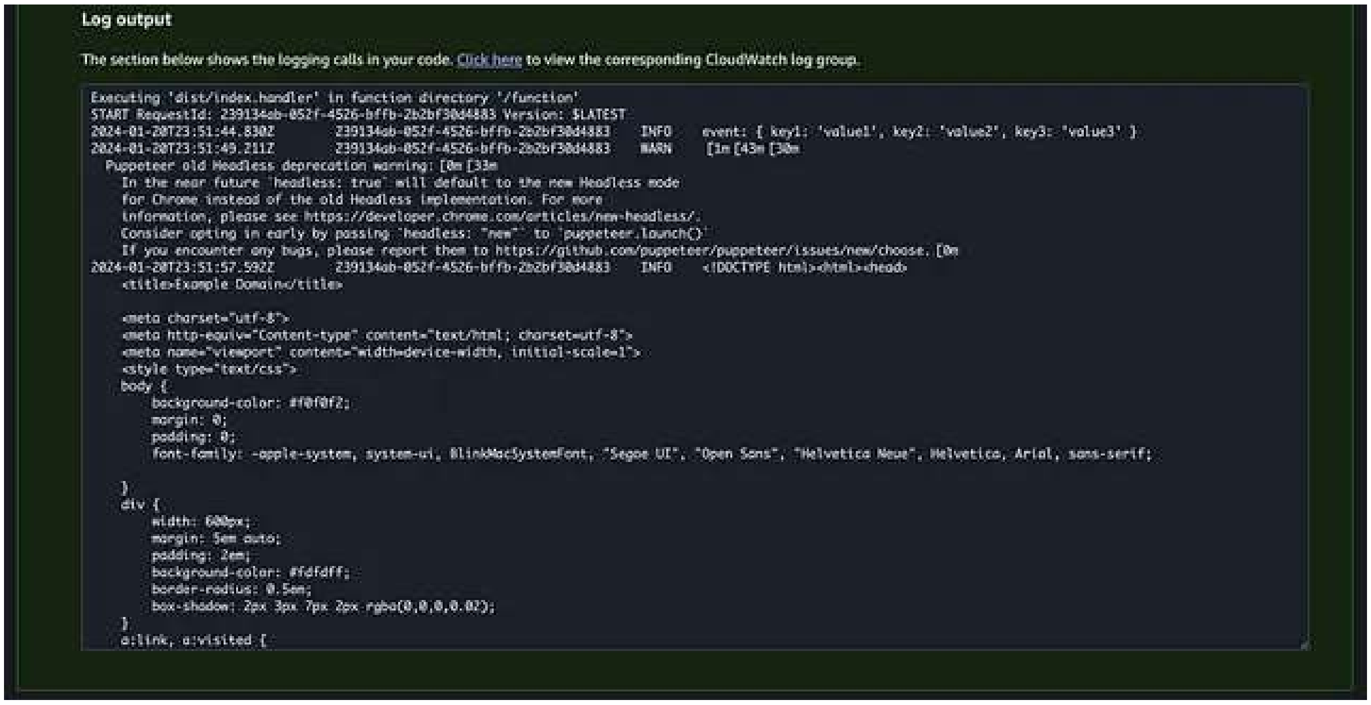
```
// Test - npx ts-node index.ts
(async () => {
  try {
    const event: ExampleEvent = {};
    //@ts-ignore
    await handler(event, {}, () => {});
  } catch (e) {
    console.log("Error in Lambda Handler:", e);
  }
})();
```

To deploy and test on Lambda, make sure everything is pushed up to GitHub before this section with a `git push` . Head to your CodeBuild project and hit "Start Build"


Developer Tools > CodeBuild > Build projects > custom-scraping
custom-scraping
Notify
Share
Edit
Delete build project
Start build with overrides
Start build

Wait for the build to complete, it should take ~5 minutes. Once it's complete head back to your project and open a terminal at the project root. Install the serverless CLI if needed: `npm install -g serverless` . To deploy to AWS, run `serverless deploy --stage dev --verbose` . This will create our Lambda function and connect it to our Docker image in ECR.

Head to your Lambda function and go to the "Test" tab. Run a test, and if it's successful, you should see the logs print out the HTML from example.com:


Log output

The section below shows the logging calls in your code. Click here to view the corresponding CloudWatch log group.

```
Executing 'dist/index.handler' in function directory '/function'
START RequestId: 239134ab-052f-4526-bffb-2b2bf30d4883 Version: $LATEST
2024-01-20T23:51:44.830Z	239134ab-052f-4526-bffb-2b2bf30d4883	INFO	event: { key1: 'value1', key2: 'value2', key3: 'value3' }
2024-01-20T23:51:49.211Z	239134ab-052f-4526-bffb-2b2bf30d4883	WARN	[1m [43m [30m
  Puppeteer old Headless deprecation warning: [0m [33m
    In the near future `headless: true` will default to the new Headless mode
    for Chrome instead of the old Headless implementation. For more
    information, please see https://developer.chrome.com/articles/new-headless/.
    Consider opting in early by passing `headless: "new"` to `puppeteer.launch()`
    If you encounter any bugs, please report them to https://github.com/puppeteer/puppeteer/issues/new/choose. [0m
2024-01-20T23:51:57.592Z	239134ab-052f-4526-bffb-2b2bf30d4883	INFO	<!DOCTYPE html><html><head>
    <title>Example Domain</title>

    <meta charset="utf-8">
    <meta http-equiv="Content-type" content="text/html; charset=utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <style type="text/css">
    body {
        background-color: #f0f0f2;
        margin: 0;
        padding: 0;
        font-family: -apple-system, system-ui, BlinkMacSystemFont, "Segoe UI", "Open Sans", "Helvetica Neue", Helvetica, Arial, sans-serif;

    }
    div {
        width: 600px;
        margin: 5em auto;
        padding: 2em;
        background-color: #fdfdff;
        border-radius: 0.5em;
        box-shadow: 2px 3px 7px 2px rgba(0,0,0,0.02);
    }
    a:link, a:visited {
```

Nice! We have deployed and ran a Puppeteer Extra scraper in Lambda 🐝

Bonus — Proxy Service

Sometimes the stealth plugin we used is not enough to get past blockers, captchas, etc. For tough sites that require a bit more artistry, you can use a proxy service. This will route our scraper through clean IP addresses and help us get the data we need. I recommend Oxylabs residential proxies because they've worked great for me and support HTTPS. If you use a proxy that only supports HTTP, you'll likely need to set `ignoreHTTPSErrors: true` in your launch options when you spawn your browser.

For a more simple proxy solution, I'd recommend Scraper API.

Refer back to our sample `index.ts` file, there is a commented out section that can be used to set up the proxy service:

```
const proxyPlugin = require("puppeteer-extra-plugin-proxy");
  puppeteer.use(
    proxyPlugin({
      address: "pr.oxylabs.io",
      port: 7777,
      credentials: {
        username: "customer-someUsername-cc-US",
        password: "somePassword",
      },
    })
  );
```

Thanks to Puppeteer Extra, there's a nice proxy plugin that makes it easy to use.

Conclusion

Thank you for reading this guide if you made it this far! I'm passionate about web scraping and enjoy building robust systems. This is my go-to toolkit for custom scraping solutions and I hope it works as well for you as it has for me.

If you'd like to support my work, you can buy me a coffee! I also do freelance web scraping projects, so feel free to DM me if there's any hard-to-get data that you'd like to collect.

Web Scraping AWS Lambda Typescript Data Collection Proxy Server

Written by Joe Osborne

Follow

10 Followers

Hi! I'm a software engineer with early stage startup experience. Check out some of my work at https://joeosborne.me :)

More from Joe Osborne

Joe Osborne

Intro to Web Scraping: Build Your First Scraper in 5 Minutes

A quick guide to help you build a simple web scraper.

5 min read · Apr 7, 2024

3 1

Joe Osborne

Web scraping is an art, not a science

Refining web scraping skills and processes requires a degree of artistry, not just raw...

5 min read · Mar 4, 2023

15 3

Joe Osborne

3 Barriers to Entry in Real Estate and How I Overcame Them

and how I got insanely lucky at age 20.

10 min read · Aug 30, 2023

2

See all from Joe Osborne

Recommended from Medium

Geoffrey Rekier

RUNNING PUPPETEER ON VERCEL

Or a possible solution on how to run Chromium in a serverless environment.

· 3 min read · Dec 6, 2023

6 2

Yash Bansal

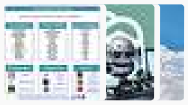
Running Selenium on Steroids : The AWS Lambda's Way

This article offers a concise argument on why Selenium works really nice with AWS Lambd...

· 5 min read · Feb 7, 2024

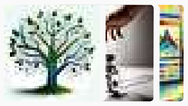
26 1

Lists



ChatGPT

21 stories · 574 saves

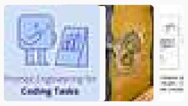


ChatGPT prompts

47 stories · 1429 saves

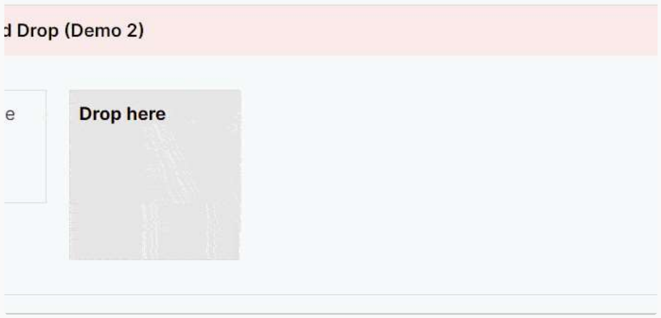
data science and AI

40 stories · 128 saves



Natural Language Processing

1377 stories · 870 saves



Akash

Selenium WebDriver Drag and Drop using Java

Introduction

3 min read · Mar 24, 2024

88 1

Ashar Malik

Undetected ChromeDriver

Do websites detect your selenium bot? Here is what you can do about it

3 min read · Dec 4, 2023

5 Web Scraping Tools to Collect Data from E-Commerce Websites in 2024
Bright Data, Oxylabs, Smartproxy, ScraperAPI, and Shifter—the best e-commerce scrapers for 2024

Sohom Das in JavaScript in Plain English

5 Web Scraping Tools to Collect Data from E-Commerce Websites ...

Bright Data, Oxylabs, Smartproxy, ScraperAPI, and Shifter—the best e-commerce scrapers...

10 min read · 5 days ago

67

Somnath Singh in Level Up Coding

The Era of High-Paying Tech Jobs is Over

The Death of Tech Jobs.

· 14 min read · Mar 31, 2024

6.8K 202

See more recommendations