Algolia docsearch crawler

View on GitHub

This tutorial shows you how to use Modal to run the Algolia docsearch crawler to index your website and make it searchable. This is not just example code - we run the same code in production to power search on this page (`Ctrl+K` to try it out!).

Basic setup

Let's get the imports out of the way.

```
import json
import os
import subprocess

from modal import App, Image, Secret, web_endpoint
```

Modal lets you use and extend existing Docker images, as long as they have `python` and `pip` available. We'll use the official crawler image built by Algolia, with a small adjustment: since this image has `python` symlinked to `python3.6` and Modal is not compatible with Python 3.6, we install Python 3.11 and symlink that as the `python` executable instead.

```
algolia_image = Image.from_registry(
    "algolia/docsearch-scraper:v1.16.0",
    add_python="3.11",
    setup_dockerfile_commands=["ENTRYPOINT []"],
)
```

```
app = App(
    "example-algolia-indexer"
)  # Note: prior to April 2024, "app" was called "stub"
```

Configure the crawler

Now, let’s configure the crawler with the website we want to index, and which CSS selectors we want to scrape. Complete documentation for crawler configuration is available here.

```
CONFIG = {
    "index_name": "modal_docs",
    "start_urls": [
        {"url": "https://modal.com/docs/guide", "page_rank": 2},
        {"url": "https://modal.com/docs/examples", "page_rank": 1},
        {"url": "https://modal.com/docs/reference", "page_rank": 1},
    ],
    "selectors": {
        "lvl0": {
            "selector": ".sidebar .active",
            "default_value": "Documentation",
            "global": True,
        },
        "lvl1": "article h1",
        "lvl2": "article h2",
        "lvl3": "article h3",
        "lvl4": "article h4",
        "text": "article p,article ol,article ul,article pre",
    },
}
```

Create an API key

If you don’t already have one, sign up for an account on Algolia. Set up a project and create an API key with `write` access to your index, and with the ACL permissions `addObject`, `editSettings` and `deleteIndex`. Now, create a secret on the Modal Secrets page with the `API_KEY` and `APPLICATION_ID` you just created. You can name this anything you want, we named it `algolia-secret`.

The actual function

We want to trigger our crawler from our CI/CD pipeline, so we’re serving it as a web endpoint that can be triggered by a `GET` request during deploy. You could also consider running the crawler on a schedule.

The Algolia crawler is written for Python 3.6 and needs to run in the `pipenv` created for it, so we're invoking it using a subprocess.

```python
@app.function(
    image=algolia_image,
    secrets=[Secret.from_name("algolia-secret")],
)
def crawl():
    # Installed with a 3.6 venv; Python 3.6 is unsupported by Modal, so use a subprocess i
    subprocess.run(
        ["pipenv", "run", "python", "-m", "src.index"],
        env={**os.environ, "CONFIG": json.dumps(CONFIG)},
    )
```

We want to be able to trigger this function through a webhook.

```python
@app.function()
@web_endpoint()
def crawl_webhook():
    crawl.remote()
    return "Finished indexing docs"
```

Deploy the indexer

That's all the code we need! To deploy your application, run

```
modal deploy algolia_indexer.py
```

If successful, this will print a URL for your new webhook, that you can hit using `curl` or a browser. Logs from webhook invocations can be found from the apps page.

The indexed contents can be found at https://www.algolia.com/apps/APP_ID/explorer/browse/, for your APP_ID. Once you're happy with the results, you can set up the `docsearch` package with your website, and create a search component that uses this index.

Entrypoint for development

To make it easier to test this, we also have an entrypoint for when you run `modal run algolia_indexer.py`

```python
@app.local_entrypoint()
def run():
    crawl.remote()
```