COMMON CRAWL

The Data Resources Community About Search Contact Us

Get Started

Accessing the Data

Crawl data is free to access by anyone from anywhere.

The data is hosted by **Amazon Web Services' Open Data Sets Sponsorships** program on the bucket `s3://commoncrawl/`, located in the `US-East-1` (Northern Virginia) AWS Region.

You may process the data in the AWS cloud or download it for free over HTTP(S) with a good Internet connection.

Choose a crawl...

You can process the data in the AWS cloud (or download directly) using the URL schemes `s3://commoncrawl/[...]`, `https://ds5q9oxwqwsfj.cloudfront.net/[...]` and `https://data.commoncrawl.org/[...]`.

To access data from outside the Amazon cloud, via HTTP(S), the new URL prefix `https://data.commoncrawl.org/` – must be

used.

For further detail on the data file formats listed below, please visit the **ISO Website**, which provides format standards, information and documentation. There are also helpful explanations and details regarding file formats in other GitHub projects.

The status of our infrastructure can be monitored on our Infra Status page.

Accessing the data in the AWS Cloud

It's mandatory to access the data from the region where it is located (`us-east-1`).

The connection to S3 should be faster and you avoid the minimal fees for inter-region data transfer (you have to send requests which are charged as outgoing traffic).

Be careful using an Elastic IP address or load balancer, because you may be charged for the routed traffic.

You may use the **AWS Command Line Interface** but many AWS services (e.g EMR) support the `s3://` protocol, and you may directly specify your input as `s3://commoncrawl/path_to_file`, sometimes even using wildcards.

On Hadoop (not EMR) it's recommended to use the **S3A Protocol**: just change the protocol to `s3a://`.

Accessing the data from outside the AWS Cloud

If you want to download the data to your local machine or local cluster, you can use any HTTP download agent, such as **cURL** or **wget**. The data is accessible via the `https://data.commoncrawl.org/[...]` URL scheme.

There is no need to create an AWS account in order to access the data using this method.

Using the AWS Command Line Interface

The **AWS Command Line Interface** can be used to access the data from anywhere (including EC2). It's easy to install on most operating systems (Windows, macOS, Linux). Please follow the **installation instructions**.

Please note, access to data from the Amazon cloud using the S3 API is only allowed for authenticated users. Please see our **blog announcement** for more information.

Once the AWS CLI is installed, the command to copy a file to your local machine is:

```
aws s3 cp s3://commoncrawl/path_to_file <local_path>
```

You may first look at the data e.g, to list all `WARC` files of a specific segment of the April 2018 crawl:

```
> aws s3 ls s3://commoncrawl/crawl-data/CC-MAIN-2018-
17/segments/1524125937193.1/warc/
2018-04-20 10:27:49 931210633 CC-MAIN-20180420081400-20180420101400-
00000.warc.gz
2018-04-20 10:28:32 935833042 CC-MAIN-20180420081400-20180420101400-
00001.warc.gz
2018-04-20 10:29:51 940140704 CC-MAIN-20180420081400-20180420101400-
00002.warc.gz
```

The command to download the first file in the listing is:

```
aws s3 cp s3://commoncrawl/crawl-data/CC-MAIN-2018-
17/segments/1524125937193.1/warc/CC-MAIN-20180420081400-
20180420101400-00000.warc.gz <local_path>
```

The AWS CLI supports recursive copying, and allows for pattern–based inclusion/exclusion of files.

For more information check the AWS CLI user guide or call the command-line help (here for the `cp` command):

```
aws s3 cp help
```

Using HTTP download agents

To download a file using an HTTP download agent add the full path to the prefix `https://data.commoncrawl.org/`, e.g:

```
wget https://data.commoncrawl.org/crawl-data/CC-MAIN-2018-
17/segments/1524125937193.1/warc/CC-MAIN-20180420081400-
20180420101400-00000.warc.gz
```

Example Code

If you’re more interested in diving into code, we’ve provided introductory **Examples** that use the Hadoop or Spark frameworks to process the data, and many more examples can be found in our **Tutorials Section** and on our **GitHub**.

Here's an example of how to fetch a page using the Common Crawl Index using Python:

```
import requests
import json

# For parsing URLs:
from urllib.parse import quote_plus

# For parsing WARC records:
from warcio.archiveiterator import ArchiveIterator

# The URL of the Common Crawl Index server
SERVER = 'http://index.commoncrawl.org/'

# The Common Crawl index you want to query
INDEX_NAME = 'CC-MAIN-2024-33'      # Replace with the latest index name

# The URL you want to look up in the Common Crawl index
target_url = 'commoncrawl.org/faq'  # Replace with your target URL

# It’s advisable to use a descriptive User-Agent string when developing your own applicat
# This practice aligns with the conventions outlined in RFC 7231. Let's use this simple o
myagent = 'cc-get-started/1.0 (Example data retrieval script; yourname@example.com)'
```

```python

# Function to search the Common Crawl Index
def search_cc_index(url):
    encoded_url = quote_plus(url)
    index_url = f'{SERVER}{INDEX_NAME}-index?url={encoded_url}&output=json'
    response = requests.get(index_url, headers={'user-agent': myagent})
    print("Response from server:\r\n", response.text)
    if response.status_code == 200:
        records = response.text.strip().split('\n')
        return [json.loads(record) for record in records]
    else:
        return None

# Function to fetch content from Common Crawl
def fetch_page_from_cc(records):
    for record in records:
        offset, length = int(record['offset']), int(record['length'])
        s3_url = f'https://data.commoncrawl.org/{record["filename"]}'

        # Define the byte range for the request
        byte_range = f'bytes={offset}-{offset+length-1}'

        # Send the HTTP GET request to the S3 URL with the specified byte range
        response = requests.get(
            s3_url,
            headers={'user-agent': myagent, 'Range': byte_range},
            stream=True
        )

        if response.status_code == 206:
            # Use `stream=True` in the call to `requests.get()` to get a raw
            # byte stream, because it's gzip compressed data

            # Create an `ArchiveIterator` object directly from `response.raw`
            # which handles the gzipped WARC content

            stream = ArchiveIterator(response.raw)
            for warc_record in stream:
                if warc_record.rec_type == 'response':
                    return warc_record.content_stream().read()
        else:
            print(f"Failed to fetch data: {response.status_code}")
            return None

    print("No valid WARC record found in the given records")
    return None

# Search the index for the target URL
records = search_cc_index(target_url)
if records:
    print(f"Found {len(records)} records for {target_url}")

    # Fetch the page content from the first record
    content = fetch_page_from_cc(records)
    if content:
        print(f"Successfully fetched content for {target_url}")
        # You can now process the 'content' variable as needed
        # using something like Beautiful Soup, etc
else:
```

cc_fetch_page.py hosted with ❤️ by **GitHub** view raw

Data Types

Common Crawl currently stores the crawl data using the **Web ARChive (WARC) Format**. Previously (prior to Summer 2013) the data was stored in the **ARC Format**.

The `WARC` format allows for more efficient storage and processing of Common Crawl's free multi-billion page web archives, which can be hundreds of terabytes in size.

If you want all the nitty–gritty details, the best source is the **IIPC document on the WARC Standard**.

Click the panels below for an overview of the differences between:

`WARC` files which store the raw crawl data

`WAT` files which store computed metadata for the data stored in the `WARC`

`WET` files which store extracted plaintext from the data stored in the `WARC`

WARC WAT WET

The WARC Format

The `WARC` format is the raw data from the crawl, providing a direct mapping to the crawl process.

Not only does the format store the HTTP response from the websites it contacts (WARC-Type: response), it also stores information about how that information was requested (WARC-Type: request) and metadata on the crawl process itself (WARC-Type: metadata).

For the HTTP responses themselves, the raw response is stored. This not only includes the response itself, (what you would get if you downloaded the file) but also the HTTP header information, which can be used to glean a number of interesting insights.

In the example below, we can see the crawler contacted `https://en.wikipedia.org/wiki/Saturn` and received HTML in response.

We can also see the page sets caching details, and attempts to set a cookie (shortened for display here).

See the full WARC extract

```
WARC/1.0
WARC-Type: response
WARC-Date: 2024-11-30T14:52:51Z
WARC-Record-ID: <urn:uuid:6fad2bf3-f2b8-4755-ba48-2cef80f2a10b>
Content-Length: 636034
```

```
Content-Type: application/http; msgtype=response
WARC-Warcinfo-ID: <urn:uuid:37faa4c1-518b-47c1-8d06-0b368e5fb495>
WARC-Concurrent-To: <urn:uuid:90f1a666-d5ba-4e8d-806d-4d848e77a0f8>
WARC-IP-Address: 208.80.154.224
WARC-Target-URI: https://en.wikipedia.org/wiki/Saturn
WARC-Protocol: h2
WARC-Protocol: tls/1.3
WARC-Cipher-Suite: TLS_AES_128_GCM_SHA256
WARC-Payload-Digest: sha1:RNGUUH2LZ5GZAN4V6FJOEENFF56JZOJ3
WARC-Block-Digest: sha1:LRBPXRFQYN3VITSOMX3I4DOBNRBQ7CQV
WARC-Identified-Payload-Type: text/html

HTTP/1.1 200
date: Sat, 30 Nov 2024 11:13:30 GMT
server: mw-web.eqiad.main-864bbfd546-nnh82
x-content-type-options: nosniff
content-language: en
accept-ch:
vary: Accept-Encoding,Cookie,Authorization
last-modified: Sat, 30 Nov 2024 10:57:28 GMT
content-type: text/html; charset=UTF-8
X-Crawler-content-encoding: gzip
age: 13160
x-cache: cp1104 miss, cp1104 hit/3
x-cache-status: hit-front
server-timing: cache;desc="hit-front", host;desc="cp1104"
strict-transport-security: max-age=106384710; includeSubDomains; preload
report-to: { "group": "wm_nel", "max_age": 604800, "endpoints": [{ "url":
"https://intake-logging.wikimedia.org/v1/events?
stream=w3c.reportingapi.network_error&schema_uri=/w3c/reportingapi/network_error/1.0.0"
}] }
nel: { "report_to": "wm_nel", "max_age": 604800, "failure_fraction": 0.05,
"success_fraction": 0.0}
set-cookie: WMF-Last-Access=30-Nov-2024;Path=/;HttpOnly;secure;Expires=Wed, 01 Jan
2025 12:00:00 GMT
set-cookie: WMF-Last-Access-Global=30-Nov-
2024;Path=/;Domain=.wikipedia.org;HttpOnly;secure;Expires=Wed, 01 Jan 2025
12:00:00 GMT
set-cookie: WMF-DP=5b0;Path=/;HttpOnly;secure;Expires=Sun, 01 Dec 2024 00:00:00
GMT
x-client-ip: 44.207.1.179
cache-control: private, s-maxage=0, max-age=0, must-revalidate, no-transformset-
cookie: GeoIP=US:VA:Ashburn:39.05:-77.49:v4; Path=/; secure; Domain=.wikipedia.org
set-cookie: NetworkProbeLimit=0.001;Path=/;Secure;SameSite=Lax;Max-Age=3600
```

```
accept-ranges: bytes
X-Crawler-content-length: 113487
Content-Length: 634511

<!DOCTYPE html>
<html class="client-nojs vector-feature-language-in-header-enabled vector-feature-
language-in-main-page-header-disabled vector-feature-sticky-header-disabled
vector-feature-page-tools-pinned-disabled vector-feature-toc-pinned-clientpref-1
vector-feature-main-menu-pinned-disabled vector-feature-limited-width-clientpref-1
vector-feature-limited-width-content-enabled vector-feature-custom-font-size-
clientpref-1 vector-feature-appearance-pinned-clientpref-1 vector-feature-night-
mode-enabled skin-theme-clientpref-day vector-toc-available" lang="en" dir="ltr">
<head>
<meta charset="UTF-8"><title>Saturn - Wikipedia</title>
...
```