Abstract

With the penetration of new technologies, there is a rapid growth of internet users and data generated by those users on the internet. As scraping is one of the major sources for extraction of unstructured data from the Internet, we have analysed the scraping process when introduced to a bulk of data extraction. We faced several challenges while scraping large amount of data, such as encountering captcha, storage issue for a large volume of data, need for intensive computation capacity and reliability of data extraction. In this paper, we investigate cloud-based web scraping architecture able to handle storage and computing resources with elasticity on demand using Amazon Web Services. Our solution tries to address both scraping and feasibility for big data applications in a single cloud-based architecture for data-based industries. We discuss Scrapy as one of our tool for web scraping because of web drivers it supports which simulates a real user working with a browser. We also analyse the scalability and performance of the proposed cloud-based scraper.

Contents

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

Introduction

1.1 Introduction Description

Web scraping is a technique for extracting information from websites. This can be done manually but it is usually faster, more efficient and less error-prone to automate the task.

Web scraping allows you to acquire non-tabular or poorly structured data from websites and convert it into a usable, structured format, such as a .csv file or spreadsheet.

Scraping is about more than just acquiring data: it can also help you archive data and track changes to data online.

There are several features that make the data on this page easier to work with. The search, reorder, refine features and display modes hint that the data is actually stored in a (structured) database before being displayed on this page. The data can be readily downloaded either as a comma separated values (.csv) file or as XML for re-use in their own database, spreadsheet or computer program. Thus, web scraping typically targets one web site at a time to extract unstructured information and put it in a structured form for reuse. Web scraping is used in a variety of digital businesses that rely on data harvesting. Legitimate use cases include:

- Search engine bots crawling a site, analysing its content and then ranking it.
- Price comparison sites deploying bots to auto-fetch prices and product descriptions for allied seller websites.
- Market research companies using scrapers to pull data from forums and social media (e.g., for sentiment analysis).

1.2 Organization of report

Describe every chapter (what every chapter contain)

- Ch.1 Introduction: What is web scraping and why this project helps to determine the necessity of web scraping.
- Ch.2 Literature Review: About web scraping more in detail with reference to the research conducted behind Our Proposed System.
- Ch.3 Proposed System: What we finalized on and the technical aspect of web scraping in detail.
- Ch.4 Results & Discussion: Final output of out project along with the conclusion.

CHAPTER 2

Literature Review

2.1 Survey existing system

The web scraper is a tool which integrates the functionality required for big data applications like analysing data and visualizing information from data.
This paper proposes the Scraper which scrapes the specific data and stores in the cloud storage. In this paper they try to implement Selenium as a base for scraper which is the python library called Beautiful Soup.

It did in some way solve the problem of the applications which were facing the storage crisis. But managing that much storage was constantly becoming the bigger problem.

In this system the users who need the constant updating of the data need to wait for considerate amount of time as they need to get the data manually.

2.2 Limitation existing system or Research gap

• Difficult to analyse – For anybody who is not an expert, the scraping processes are confusing to understand. Although this is not a major problem, but some errors could be fixed faster if it was easier to understand for more software developers.

• Data analysis – The data that has been extracted will first need to be treated so that they can be easily understood. In certain cases, this might take a long time and a lot of energy to complete.

• Time – It is common for new data extraction applications to take some time in the beginning as the software often has a learning curve. Sometimes web scraping services take time to become familiar with the core application and need to adjust to the scrapping language. This means that such services can take some days before they are up and running at full speed.

• Speed and protection policies – Most web scrapping services are slower than API calls and another problem is the websites that do not allow screen scrapping. In such cases web scrapping services are rendered useless. Also, if the developer of the website decides to introduce some changes in the code, the scrapping service might stop working.

2.3 Problem Statement and Objectives

Our project will describe the most common reasons for Automating Web Scraping and make easier for the users connected to this topic. Techniques of Web Scraping with appropriate explanation are presented in separate chapters. Currently available software tools are listed with brief summary of their functionalities. The Process of Web Scraping is entirely explained by an practical example at the end of the paper.

2.3.1 Objectives

The objective of this research project is to develop a webpage that uses a concept to collect a specific piece of information from a particular website and represent this information. Further, this application comprises of two major components. The first component is a utility that is responsible on connecting to the Wold Wide Web, collecting the information using the web scraping techniques, and store the collected information into a database. Where the other component is a website that is responsible on presenting on a web page the collected information which stored in a database.

Why Scrape the Web?

There are many specific reasons why businesses may want to scrape their website;
one of the vital reason being the unavailability of APIs. Some of the other major reasons which may lead a company into scraping their website are:

1. Expand Market Share

Due to the lack of availability of APIs the possibility of collaborating with business partners is limited. By exposing the data available in their website as APIs enterprises can open up new channels, possibilities to expand the market share and increase sales.

2. Enter New Markets with Early go-to Market Strategy

API being the long-time strategy, Web Scraping solution can potentially enable organizations to build an early go-to market strategy.

3. Access to Renewed and Structured Data

Scraping the website of the organization through a Web Scraping solution gives an organization the chance to access renewed, structured and up to date data through the scraped APIs.

Benefits of Scraping Solution:

In order to remain competitive, businesses must be able to act quickly and assuredly in the markets. Web Scraping plays a big role in the development of various business organizations that use the services. The benefits of these services are:

1. Low Cost

Web Scraping service saves hundreds of thousands of man-hours and money as the use of scraping service completely avoids manual work.

2. Less Time

Scraping solution not only helps to lower the cost, it also reduces the time involved in data extraction task. This tool ensures and gathers fast results required by people.

3. Accurate Results

Web Scraping solutions help to get the most accurate and fast results that cannot be collected by human beings. It generates correct product pricing data, sales leads, duplication of online database, captures real estate data, financial data, job postings, auction information and many more.

4. Time to Market Advantage

Fast and accurate results help businesses to save time, money and labour and get an obvious time-to market advantage over the competitors.

5. High Quality

A Web Scraping solution provides access to clean, structured and high-quality data through scraping.

2.4 Scope

The project will be useful for the users who deal with data engineering and need to collect data for respective purposes. The project target is to create a web application that analyses it automates the process and make it easy for the naïve as well as experienced user. It can be very useful for scraping the data for analyst who obviously needs data for analysis in a easier way as compared to other scrapers. The main feature of this project is to scrape data automatically and periodically and help big data application to store the data in cloud storage.

CHAPTER 3

Proposed System

3.1 Analysis/Framework/Algorithm

For the creation of the web scraper and the web application, several tools must be utilised. To keep all code and software produced in a safe environment with maximum compatibility, a virtual machine (VM) running the Kali distribution of Linux. This VM is used for portability so that when moving between machines, the VM box can be copied and transferred to the new machine. For the creation of the web scraper and the web application, Python 3.6.3 will be used, with Scrapy 1.4 for the web scraper, along with Scrapy-do daemon and Flask 0.12.2 and Bootstrap v4.0 for the web application and Amazon Web Services storage facility S3 for storing the scraped data. The latest stable releases of each language and software will be used to ensure code will be up to date and will not be deprecated in the foreseeable future.

3.1.1 Scrapy

Scrapy is a fast high-level web crawling and web scraping framework, used to crawl websites and extract structured data from their pages. It can be used for a wide range of purposes, from data mining to monitoring and automated testing.

As diverse the internet is, there is no “one size fits all” approach in extracting data from websites. Many a time ad hoc approaches are taken and if you start writing code for every little task you perform, you will eventually end up creating your own scraping framework. Scrapy is that framework.

3.1.2 Amazon Web Services

Amazon Web Services (AWS) is a subsidiary of Amazon providing on-demand cloud computing platforms and APIs to individuals, companies, and governments, on a metered pay-as-you-go basis. These cloud computing web services provide a variety of basic abstract technical infrastructure and distributed computing building blocks and tools. One of these services is Amazon Elastic Compute Cloud (EC2), which allows users to have at their disposal a virtual cluster of computers, available all the time, through the Internet. AWS's version of virtual computers emulates most of the attributes of a real computer, including hardware central processing units (CPUs) and graphics processing units (GPUs) for processing; local/RAM memory; hard-disk/SSD storage; a choice of operating systems; networking; and pre-loaded application software such as web servers, databases, and customer relationship management (CRM).

Amazon S3 has a simple web services interface that you can use to store and retrieve any amount of data, at any time, from anywhere on the web. It gives any developer access to the same highly scalable, reliable, fast, inexpensive data storage infrastructure that Amazon uses to run its own global network of web sites. The service aims to maximize benefits of scale and to pass those benefits on to developers.

3.1.3 Flask

Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself. Extensions exist for object-relational mappers, form validation, upload handling, various open authentication technologies and several common framework related tools.

53 3.2 Details of Hardware and Software

3.2.1 Hardware requirements

Minimum Hardware Requirements:

- Processor: Intel core i3 or higher
- RAM: 2GB or above

3.2.2 Software requirements

Minimum Software Requirements

- Operating System: Windows 8.1/10
- Programming Language: Python
- Web Applications: AWS, Scrapy daemon, Scrapy, Flask.
- User Interface: HTML, CSS

53 **3.3 Design Details**

3.3.1 System Flow/ System Architecture

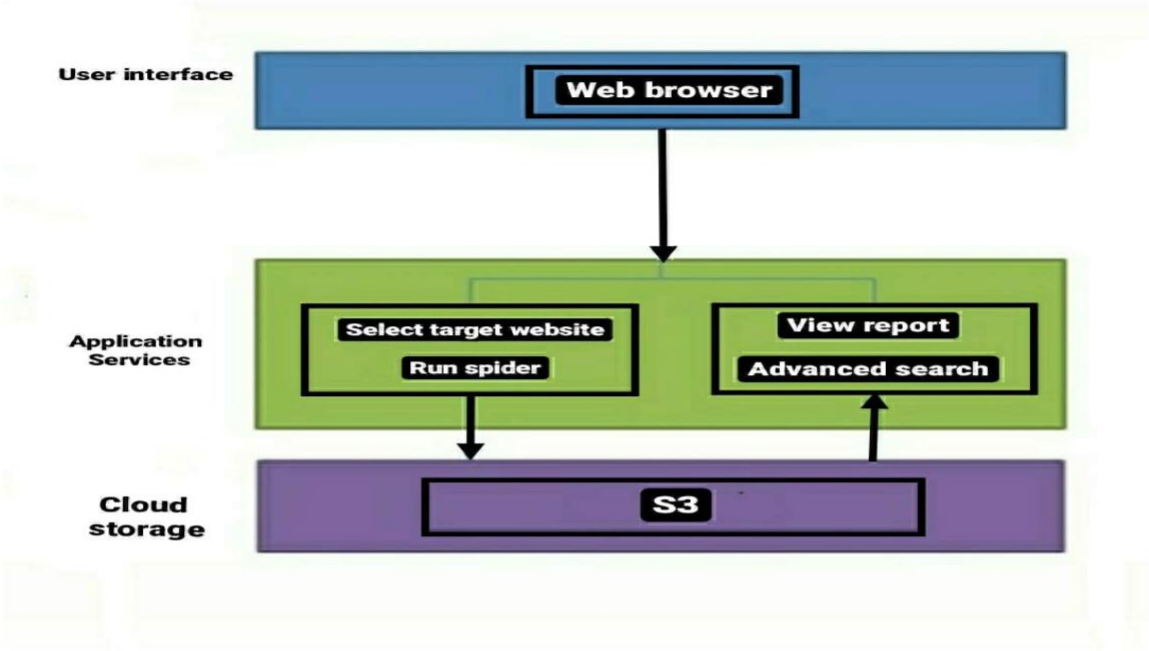

Fig. No.3.3.1:Architecture diagram of the proposed system

This is the architecture diagram for the proposed web scraping system. Here a layered architecture was chosen, as each layer can only communicate with the layer one above or below. This means that the core functionality of the web application can take place close to the database after authentication, as the application makes use of information stored in the database for all aspects of the application services layer.

The services writing to the database are those that interact with the spider, as it scrapes a user specified website and stores any findings in the database. The services reading from the database are those that generate reports from the stored findings of the spider. This includes a generic report with all information displayed, and custom reports generated by a search query.

The spider itself has a documented architecture, as scraping websites has a set of required steps for getting and parsing the data found online. The architecture of a Scrapy spider along with a description of the data flow is shown in the figure.

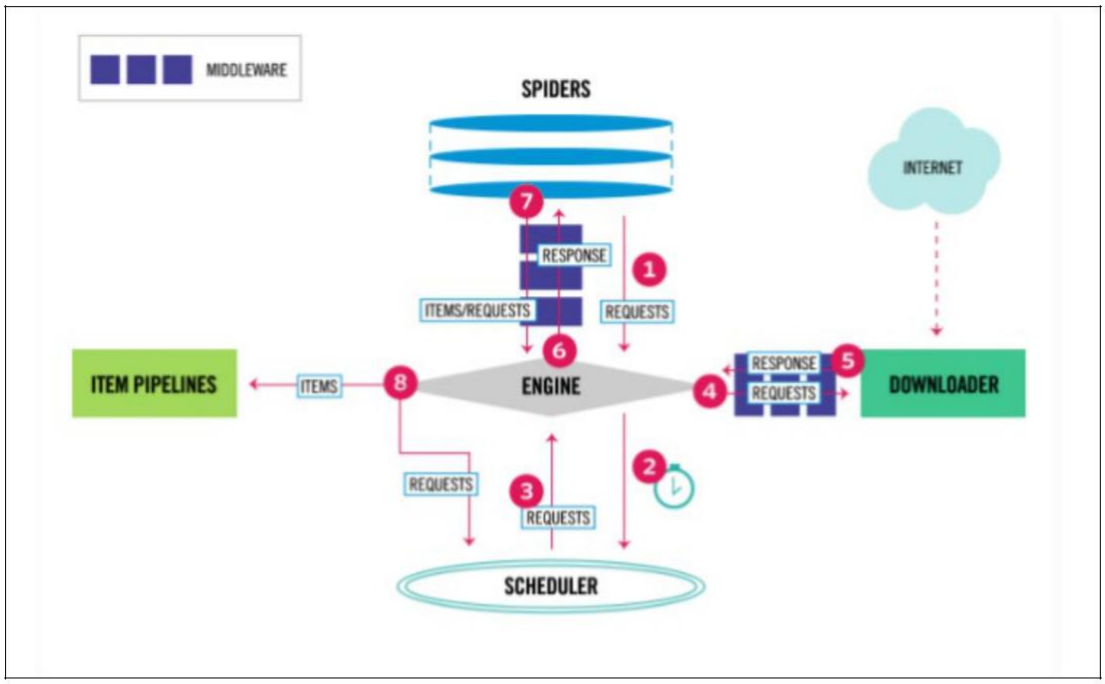

Fig. No.3.3.2:Architecture of Scrapy Spider

The data flow in scrapy is controlled by the execution engine and goes like:

1.The **Engine** gets the initial Requests to crawl from the **Spider**.

2.The **Engine** schedules the requests in the **Scheduler** and asks for the next Request to crawl.

3.The **Scheduler** returns the next Request to the **Engine.**

4.The **Engine** sends the Requests to the **Downloader**, passing through the **downloader middleware.**

5.Once the page finishes downloading the **Downloader** generates a response (with that page) and sends it to the **Engine**, passing through the **downloader middleware**.

6.The **Engine** receives the response from the **Downloader** and sends it to the **Spider** for processing, passing through the **spider middleware**.

7.The **Spider** processes the response and returns scrapped items and new requests (to follow) to the **Engine**, passing through the **spider middleware**.

8.The **Engine** sends processed items to **Item Pipelines**, then send processed requests to the **Scheduler** and asks for possible next requests to crawl.

3.3.2 Detailed Design

Once the user visits our website , they are taken to the index, or main, page, where there is a field to enter the website the user wishes to scrape and a navigation bar at the top of the page, which allows users to access the other pages in the application. If the user chooses to scrape a website, the spider is initialised and the application prompts the user that the target website is being scraped. Once this is finished, a report is generated which can be easily downloaded over S3 bucket.

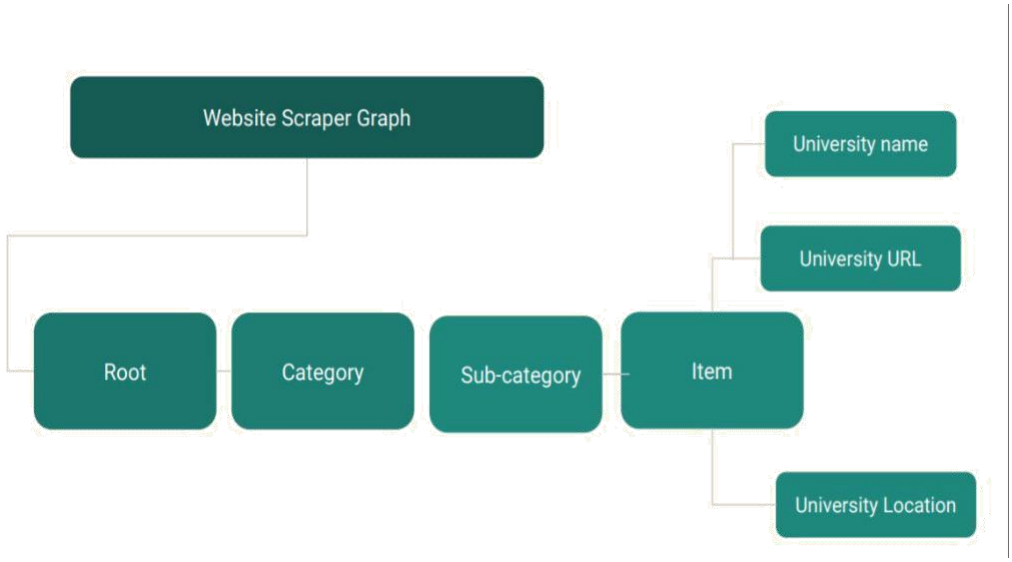

Fig. No.3.3.3:Sitemap of Proposed System

3.4 Methodology/Procedure (Approach to solve the problem)

The approach taken during the development of the web application was an incremental one. In this approach each segment was completed before moving on to another. The first segment was building the first spider, focusing on a local page taken from Wikipedia.org, where an understanding of how Scrapy works was gained. After this, the database was developed, and the spider was adapted to write its findings to the database. Once this was completed, work began on the web application, where Flask was set up and the first webpages were made; index.html and layout.html. The layout page is a page that never loads by itself, as it provides the layout of the parts of the webpages that do not change between pages. An example of this is the menu, or navbar, at the top of each page, which remains constant no matter which page the application loads. After this, the spider(url) method was created to allow the spider to be activated from within the web application. The report.html page was then created to display the findings of the spider in an easy-to-read fashion.

The tools and languages used to develop this system were as follows:

• Backend: Python and its libraries; Scrapy, Scrapy-do, Flask and AW'S

• Frontend: HTML, CSS, jQuery

These technologies were chosen for two reasons. The first is due to already having experience in using these languages and libraries for developing web applications, with the exception of the Scrapy library, and the second is that Scrapy is a Python library, so the application also had to be built using Python.

As Scrapy was the only unfamiliar tool, time was spent learning how to use the framework at the beginning of development before any work began on the other, more familiar components.

In order to automate the process we use the Scrapy-Do Daemon which helps us to schedule the spiders periodically. It can either do immediately, or run them periodically, at specified time intervals. It has convenient CLI to manage the spiders and also user friendly web interface.

CHAPTER 4
Results and Discussion

4.1 Results

As mentioned earlier, we proposed a automated approach to web scraping to make the process easier.
Here the scrapy-do scheduled tasks are showed in the web interface which shows the status of the tasks scheduled.

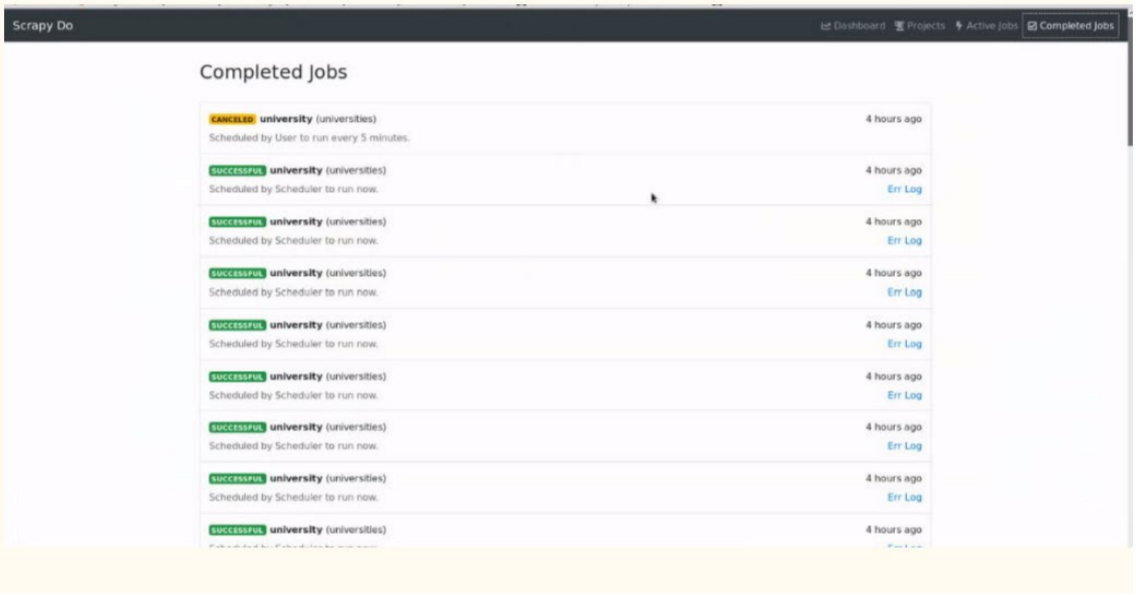

Fig. No. 4.1 Result Log

Later we can download the scraped data from the AWS S3 and get the desired data.

4.2 Discussions

Here you can discuss about output with some comparative study (tabular form), output analysis.

Table 4.2 Comparative Study of all designs

Criterion	Coefficient	Design #1	Design #2	Design #3
Accuracy	4	N/A	N/A	N/A
Precision	4	N/A	N/A	N/A
General Density	4	6 (1.82)	6.13 (1.84)	7.5 (2.25)
Homogeneity	5	1.4 (0.14)	1.1 (0.11)	1.9 (0.19)
Completeness (buildings)	4	3 (30)	5 (50)	4.5 (45)
Completeness (bridge)	3	2 (20)	4.5 (45)	7 (70)
Completeness (road)	3	9.5 (95)	9.5 (95)	9.5 (95)
Total score	**-**	**77.5**	**92.02**	**107**

CHAPTER 5

REFERENCES

Conclusion and Future Work

It can be concluded that a sophisticated technology like S3 provides a better platform for the web scrapper. Proposed web scrapper provides reliable data extraction from the web. S3 provides elastic resources for computing and storage in a distributed environment on demand. We tested the scalability and performance of the architecture with the experiment above on top of Amazon's cloud services. We analyzed the proposed cloud-based web scraping architecture implemented using Amazon's S3 service for scraping websites using multiple instances of virtual computing machines. This architecture can also be implemented using other cloud service provider. We suggest using the alternative of selenium web renderer if better performance is the only requirement without web automation. As it is based on completely scalable resources, big data applications can be implemented over this architecture. The proposed architecture includes almost all resources required for big data solutions. We discussed advantages of our architecture and briefly compared it with traditional/other web scraping architecture.

Note: References should be written in IEEE format

[1] Ram Sharan Chaulagain, Santosh Pandey, Sadhu Ram Basnet, Subarna Shakya, "Cloud Based Web Scraping for Big Data Applications". 2017

[2] Vidhi Singrodia;Anirban Mitra; Subrata Paul, "A Review on Web Scraping and its Applications" 2019

[3] A Dive Into Web scraper World Publisher: IEEE, Authors: Deepak Mahto, Lisha Singh. 2016

[4] Web Scraping: State-of-the-Art and Areas of Application. Authors: Rabiatou Diouf, Edouard Ngor Sarr,Ousmane Sall 2019

You can add url of website from which you take guidance.

1. Web scraping and data analysis using selenium webdriver and python. http://www.datasciencecentral.com/profiles/blogs/web-scraping-and-data-analysisusing-selenium-webdriver- and, July 2,2016. Accessed: 2017-08-1.
2. Osmar Castillo-Fernandez. Web scraping: Applications and tools. European Public Sector Information Platform, pages 13–15, 2007.
3. EliteDataScience. 5 tasty python web scraping libraries. https://elitedatascience.com/python-web-scraping-libraries, Oct 28, 2016. Accessed: 2017-08-7