Silesian University of Technology in Gliwice

Interdisciplinary studies in automatic control and robotics, electronics and telecommunication, informatics

Engineering thesis

Containerization of data warehouse creation and management processes

Author: Karol Latos
Supervisor: dr inż. Anna Gorawska

Gliwice, November 2021

Contents

Chapter 1

Introduction

The subject of this engineering thesis is a standalone data warehouse built using multiple Docker containers. The thesis describes how the data is gathered, processed and analyzed, and what components make up each container, as well as their mutual coordination. Additionally, practical results of the project are presented in form of answers to real-world questions, together with steps taken to achieve performance optimization and code robustness.

The basic assumptions of the system are the following:

1. The application is portable, standalone and platform-independent.
2. The application requires the minimal amount of setup in order to run.
3. The code is robust in recovering from errors and able to run continuously.
4. Proper programming practices are followed whenever it is possible.

1.1 Scope of the work

The work done in relation to one of the containers in this project is partially based on a group project[1], concerned with the acquisition of data from a web service in order to answer simple questions with SQL queries. For that project, the author of this thesis implemented the data gathering part using Python language with Selenium module.

[1] Group members: Jakub Sieńko, Kacper Garcon and the author, collaborating during an university course *Data Warehouses and Data Mining Systems*

This individual work has been transfered and heavily modified in order to compose a container-based solution, with other containers written by the author from scratch for the purpose of this thesis.

In totality, the following parts were implemented:

- data gathering container,
- database manager container,
- data mining container,
- *nodejs* container with simple web application,
- logging service,
- flag management service,
- database management service,
- web scraping Selenium-based service,
- data transformation service,
- data mining service.

Additionally, the author researched topics concerned with container orchestration as a simplification of system architecture; Selenium framework for web scraping; optimization of “big data” SQL queries; full-stack web development in Node.js environment and other related fields.

1.2 Thesis contents

Chapter (2) is concerned with the changes in technology leading to various options of exploiting data availability on the Internet, while also deliniating the tools which are at the core of this project. It also explains the motivation for choosing such thesis topic.

Chapter (3) describes the domain of the problem, functional and non-functional requirements pertaining to it, as well as the structure of the data warehouse and the data inside it.

Chapter (4) contains technical information about the orchestration of the containers, composition of the services and an overview of the data pipeline.

Chapters (5), (6) and (7) are centered around the three containers, taking a deep dive inside each of them and following the data on its path from the website to market reports.

Chapter (8) reflects on the practical results generated as a result of the data analysis, while constrasting them with initially posed questions. Additionally, the chapter describes the performance of the code under normal and abnormal conditions, and how the project changed with time.

Chapter (9) summarizes the effects achieved during the implementation of the system and formulated conclusions about possibilities of using containers to create and manage data warehouses.

Chapter 2

Project basis

> *"Data is the sword of the 21st century,*
> *those who wield it well, the Samurai."*
>
> — Jonatan Rosenerg, former Senior Vice President of Products, Google

In the contemporary world, the Internet facilitates the backbone of world-wide services, including e-commerce, transportation, entertainment and businesses. This relatively recent change presented an open world of possibilities, with new creative ideas emerging every year. In comparison to the reality of the past, the amount of information available at hand is unprecedently greater than ever before, constituting an advancement so significant, it can be argued to be as important as the invention of the printing press or even written language. Tapping into this immense ocean of knowledge is possible by utilizing adequate tools, like programming languages capable of producing specifically targeted scripts. This potentially is a fertile ground for automatization, and to take advantage of it means to extend one's capability of processing information, possibly by orders of magnitude, to access previously unavailable knowledge.

2.1 Data on the internet

What is the history of internet? What is the state of the internet today? How good is the average bandwidth? How are machines communicating with each other and to what means? Instant availability of information is only practical for computers, need to automate the gathering and analysis of data. Enter web scraping.

2.2 Python language

Python is high-level general-purpose programming language. It provides levels of abstraction from the machine architecture, so that user doesn't have to worry about e.g. managing memory allocation, and can be used to develop applications in many domains. Its history begins in December 1989, when a Dutch programmer, Guido von Rossum, became working on a successor to the ABC language [3]. Released in 1991, with major consecutive versions in 2000 and 2008 (Python 3, current version), it today became a robust language for just about anything, from statistics and modelling, to webservers and RestAPIs. Python is taught in schools as an entry-level language, while at the same time being used by NASA[1].

Python offers a variety of modules and packages (also called libraries), i.e. collections of functionalities with provided interface for the user to utilize. Among the most popular libraries we have numpy — for managing matrices and multi-dimensional tables similar to MATLAB; openCV — for processing and analyzing images and videos; requests — a simple HTTP library for communicating with the server; and many more.

In my code I'm using several crucial libraries, which I will briefly describe below, as well as packages with less significance, which I list here:

- checksumdir: utility for calculating a checksum of a given directory, similar to a file checksum, which I use to determine whether new data has arrived and whether that data is complete and can be passed further;
- time: popular package for measuring time between two points in code, sleeping (i.e. stopping the execution) and getting information on the current date, mostly used for scheduling the run until the day changes and measuring performance of various code parts;
- shutil: module for creating and removing non-empty directories, used by me to additionally isolate the shared data inside each container, so that it's invulnerable to changes in the original folder once running;

[1] `https://github.com/nasa/podaacpy` is used for crucial communications with Jet Propulsion Laboratory

- os: another popular library, utilized in my code for managing local files and ensuring proper directory structure on the first run, i.e. creating shared folders for storing data, logs and flags.

2.2.1 Selenium and Beautiful Soup

These libraries make it possible to connect to any website and emulate user behaviour in an automatic fashion. Selenium is a framework responsible for creating a webdriver (virtual, in-code browser object) and visiting requested URLs, while looking for specific elements (like buttons for expanding the page) or deciding whether the response from the page is complete or it needs more time to load. Beautiful Soup on the other hand is a simple but powerful system for navigating HTML code. It provides support for finding particular webpage elements like *div* or *span*, by requesting their *class*, *id*, or any other attribute. Every HTML page is converted into a soup object, similar to a nested dictionary, which then can be searched using provided methods.

2.2.2 Numpy, Pandas and Scikit-learn

Numpy, briefly mentioned before, is a mathematical library which is a basis for many other libraries, such as Pandas or Scikit-learn. Pandas revolves around Dataframes and Series — two- and one-dimensional data structures similar to Excel or SQL tables. One of the advantages is the speed of processing; selecting tens of thousands rows out of millions, based on some condition, is almost instantaneous. Dataframes come with set of tools for their manipulation, i.e. aggregation functions, joins, concatenation, etc. When this data is passed, scikit-learn is responsible for creating statistical models trying to find correlations, predict outcomes, as well as provide decision support based on the data. It is arguably one of the most important libraries in machine learning domain.

2.2.3 Matplotlib and Seaborn

Matplotlib and Seaborn are libraries for data visualization, where the latter is an extension of the former. Matplotlib gives basic plotting functionalities and proves to be a robust module, which can be used on its own. However, in order to reduce the amount of code written and standardize the outcomes, Seaborn is used to provide more complex visualizations without an extensive use of Matplotlib.

2.3 Docker containers

Containerization (or OS-level virtualization) is a way of isolating resources inside an operating system without using virtual machines (VMs), to create self-enclosed, lightweight executables. The key difference from VMs is that virtualization uses a hypervisor — software that hosts guest operating systems and distributes hardware resources among them. Any process running inside a virtual machine only sees the guest operating system. Meanwhile, containerization uses only the host operating system and a container engine (e.g. *Docker*). From the point of view of a process running inside a container, the directory structure may largely differ from what user sees on the disk. Container should have only the minimal number of libraries and dependencies required to run it. Generally speaking, containerization is a paradigm for the operating system kernel to allow many isolated *user spaces* to exist in a shared environment, which translates to container sharing the filesystem with the host operating system without conflicts. However, recreating the container may mean losing all of our data and starting fresh. To avoid that, methods for data persistence are available, two of which are most popular and used in this project:

- Docker volumes — internal Docker storage, which persists removing the container image if stated explicitly. These volumes are not seen by the host OS.
- Bind mounts — specified directories inside the container, that will correspond to actual directories on the host operating system. This technique is similar to mounting an USB device in a filesystem.

The goal of containerization is to deploy applications securely and fast, without worrying about OS compatibility. The act of abstracting our software from the host operating system allows containers to be portable and stand-alone. Additionally, one can orchestrate many containers to run together and share the OS resources in order to perform a common task. This creates a perfect opportunity to use containerization for an application managing data warehouse, since the data has to go through many different stages in a pipeline, which correspond to separate processes, each inside its own container.

2.4 Data warehouses

Data warehouse is a data organization concept that originated in late 1980s in IBM [1]. Barry Devlin and Paul Murphy were trying to find a way to optimize the processing of data from common source to different destinations, called decision support systems. These systems were composed of software taking various data as input, and producing a metric for finding solutions to a given problem. One example is using a decision support system highlighting unusual areas of a brain scan from a MRI, for faster recognition of potentially malicious changes. Before the practice of data warehousing, multiple systems had to acquire data independently from a business source, process it and then perform needed analysis. However, this approach yeilds several problems, most obvious of which is computational redundancy and consequently wasting of resources.

Devlin and Murphy's idea was to find commonalities between different decision support systems, gather all the needed data at once, process it and then store it in a ready-to-use format. This way, any application could request a specific set of data, tailored for its needs, without the need to perform heavy computation every time. These datasets are called *data marts*, and the isolation of decision support systems from their individual data sources proved to be a robust solution, that has quickly been implemented in businesses around the world.

In my project I gather the data from various pages on the card market website, then I process and save it in a *.csv* file format. These files are then read and converted to dataframes, which are converted to tables in a database. In order for another process to use the data efficiently, it is transformed into various data marts inside the database, creating a data pipeline from the source to the final program, which performs analysis. This way I'm utilizing data warehousing concepts, although I am only collecting data from a single source (one process responsible for data gathering) and delivering it to a single application.

2.5 Justification of the thesis topic

How do I use existing technology to solve an existing problem? What questions do I aim to answer?

Chapter 3

Specification

What is the actual example that I'm using? Describe the specification from the outside (helicopter view).

Implementation of the thesis topic will by nature involve a containerization software (*here Docker*), which will host several subapplications written as semi-standalone scripts, each responsible for a part of the data pipeline. Using Python and its vast collection of libraries I'm handling the data scraping, intial cleaning and maintenance of the pre-stage database in compressed CSV files; as well as managing the MySQL database, creating helper tables, extracting new information and visualizing the data to answer user's questions. With JavaScript, SQL and HTML, I'm able to present the results in form of a simple web application, querying the database connected to the *Node.js* server.

3.1 Functional requirements

1. The user should be able to collect data about chosen cards expansion

2. The data gathering system should run continuously and gather data once a day

3. The gathering system should visit all card sites from specified expansion and save (a) card information, dynamic and static, and (b) full information about sale offers of this card to CSV files

4. The gathering system should visit profile pages of all newly-encountered sellers and save their public data to a CSV file

5. The gathering system should keep track of the date and save the date data into a CSV file
6. The database manager system should run continuously and update the database whenever new complete batch of data has been gathered
7. The data miner system should run continuously and create data marts every hour, given the database has the newest data
8. The web application should run continuously, recovering from fatal errors and providing the user with at least four ways of getting useful information from the data

3.2 Non-functional requirements

1. The total time of data processing should be less than 6 hours
2. The system should be standalone (bootstrapping itself from *docker-compose.yml* and code), and cross-platform compatible
3. The gathering system should adapt the requests frequency to the server condition
4. No user input is required after running the system

3.3 Problem domain

3.3.1 Trading card games

Trading card games (*TCG*), also known as collectible card games (*CCG*), are types of card games combining the elements of strategic gameplay and features of trading cards. The first TCG was released in 1993 under the name Magic: The Gathering (*MTG*). The game, released by an eight-person company, was an overnight success, with over 10 million cards sold in just 6 weeks. Two years later, this basement business became a gaming corporation. Today, *MTG* is among the most popular TCGs with roughly 35 million players as of December 2018 [2].

During the game, the goal of each player is to reduce the opponent's life points by strategically playing cards from the hand, before the other one succeeds. However, each player can compile their own deck of 60[1] cards out of the thousands available. This makes every gameplay unique on a level which is fundamentally different from the classic cards. One doens't have to be an expert in the field to understand that the more cards one has, especially good cards, the higher the chances of winning. Thus, trading the cards becomes an aspect as crucial as the strategy itself.

One of the places where *MTG* cards can be bought is `www.cardsmarket.com/` — an online market with a myriad of cards from the game listed for sale, by users from around the world, majority of which lives in Europe. Users are divided into three categories: *Amateur*, *Professional* and *Powerseller*, depending on their setup and *modus operandi* (individuals, zealous hobbyists, card stores). To spend the least amount of money while collecting the most wanted cards, i.e. to optimize the shopping, one would have to analyze thousands of bits of data, from the average prices of all cards, to the velocity of sales and of restocking the virtual shelves.

3.3.2 Decision optimization

Describe the cards (with their statistics — static entity, dynamic entity). Also sellers and sale offers. Describe what we want to do with the gathered data and to what means.

When the data from the card market is collected, it is stored in five CSV files and one text file. The text file contains card names in the order of first visit and it's used to maintain consistent card-to-card progression between runs. The card entity has only static attributes: the card's **id**, **name**, **rarity** and what **expansion** is this card in.

3.4 Data warehouse modelling

Describe how is the data structured, based on the entities mentioned before. Show how does the data warehouse look like including all steps, explain why is it like this. Mention DWDMS and the outcome of that project, as well as differences between it and this thesis.

[1] Some variations require a deck of 40 cards, which are selected from a random pool of cards

Chapter 4

Project architecture

What are the components of the project? What containers are used, and what are their tasks? How are they orchestrated, how do they communicate or share data?

4.1 Components internal organization

What services do my subprojects use? What are the functions of these modules?

4.1.1 Services

What are the assumptions of the services? What do they do? Which services are shared? Which functions are shared?

4.2 Containers orchestration

How are the containers setup and connected? How are they synchronized or managed?

4.3 Data pipeline

High level overview of the data pipeline (schema). Where does the data come from? How do I get it? Where do I validate it? What happens in the directories?

What happens in Docker volumes? What happens in the database? What is the end result of the pipeline? What is the speed of consecutive steps?

4.4 Compatilibity

Is my project compatible with main operating systems? How does the installation differ on various systems?

Chapter 5

Data gathering implementation

What is implemented in this subproject, so that it fulfills its purpose? What does the program do? Step by step.

5.1 Program configuration

What is in config.py file? Which values are shared among containers? Which are modifiable and which are not?

5.2 Used services

What is my service? (What is a service?) Why do I use modules as services? Why don't I use objective programming? What is the purpose of each of the services in this subproject?

5.2.1 Web service

What is this service all about?

5.3 Local directories

How are they setup, what will they contain?

5.4 Run scheduling

How is the run scheduled? Does it keep trying when faced with exceptions?

5.5 Data pickling and validation

What are the formats of the data? Why is csv compressed? What are pickles for? What were the obstacles (dictionary fiasco)?

5.6 Loop over cards

Page url crafting, getting the page, explicit wait. Trying to exhaust the Load more button, explicit wait. Getting the soup object from HTML page source. Decomposition of the soup. Increament START_FROM. Clean and unpickle the data, revisit scheduling.

5.7 Soup decomposition

Getting card name, and adding this card if not present. Card id. Getting card statistics, adding them.

5.7.1 Sale offers table

Understanding the table of sale offers. Getting all seller names from the table. Filtering the seller names as a set difference with saved. Iterating over pages of sellers profiles, scraping the data. Sellers added one by one, with append(seller: dict) — pickles fiasco. Updating sale offers.

5.8 Data pipeline output

What happens when the program is finished? How does the data look, where is it? How do other containers know when to act?

Chapter 6

Data warehouse creation

How is the data stored so far? How it can be stored? Why use database? What kinds of engines we can use? What does the data warehouse contain? What is its purpose?

6.1 Program configuration

What is in config.py file? Which values are shared among containers? Which are modifiable and which are not?

6.2 Used services

What modules are used in the subproject? What are the differences in the services?

6.2.1 Database service

What does it do, and how does it do it?

6.3 Run scheduling

How is the run scheduled? What indicators are present?

6.4 Tables from staging area

How do the tables look like when the data is gathered by the first container? What are the data types? What data can be missing or faulty? Do we have any metadata? How can it be used?

6.5 Data transformation

How will the data be transformed? What new tables will be created? How will these tables be utilized?

Chapter 7

Data mining

What information is ready to be read right away? What questions can be answered immediately? What questions are more complex? What data can be predicted from the current data?

7.1 Program configuration

What is in config.py file? Which values are shared among containers? Which are modifiable and which are not?

7.2 Used services

What modules are used in the subproject? What are the differences in the services?

7.2.1 Mining service

What does it do, and how does it do it?

7.3 Run scheduling

How is the run scheduled? What indicators are present?

7.4 Feature extraction

What new features can be discovered? `https://www.analyticsvidhya.com/blog/2021/04/guide-for-feature-extraction-techniques/`

Chapter 8

Results

What is generated by the project? What questions are answered? How does my program behave under different cases? Is it resiliant?

8.1 Testing

Testing of the proper functioning of each module. What is the performance of the code under normal and abnormal conditions?

8.2 Version control

How was the project developed? How were different versions maintained?

8.3 Project variation in time

How did the project evolve with time? What challenges did I encounter? How did I solve problems along the way? Which problems remained?

Chapter 9

Summary

What was the subject of this thesis? How did the assumptions or constraints influence the project? What did I implement and how did I do it?

9.1 Result assessment

What is the end result of the program? Did I manage to answer all of my questions? Are the effect satisfactory?

9.2 Scaling and customizability

How can this project be used outside of the case described in this thesis?

9.3 Conclusions

Any other closing remarks.

Bibliography

[1] Barry A. Devlin and Paul T. Murphy. "An Architecture for a Business and Information System". In: *IBM Syst. J.* 27 (1988), pp. 60–80.

[2] Suresh Kotha. "Wizards of the Coast". Archived from the original (PDF) on September 1, 2006, retrieved August 11, 2013. Oct. 1998. URL: https://web.archive.org/web/20060901100217/http://faculty.bschool.washington.edu/skotha/website/cases%20pdf/Wizards%20of%20the%20coast%201.4.pdf.

[3] Guido Van Rossum. "The History of Python: A Brief Timeline of Python". Archived from the original on 5 June 2020. Retrieved 5 March 2021. Jan. 2009. URL: https://python-history.blogspot.com/2009/01/brief-timeline-of-python.html.

Appendix A

Used symbols

$M(i, j)$ — measure between points i and j.

Appendix B

Another appendix