Dydaktyka Informatyki 11(2016)
ISSN 2083-3156
DOI: 10.15584/di.2016.11.20
http://www.di.univ.rzeszow.pl

Wydział
Matematyczno-Przyrodniczy UR
Laboratorium Zagadnień
Społeczeństwa Informacyjnego

Jacek WOŁOSZYN

Dr inż., Uniwersytet Technologiczno-Humanistyczny w Radomiu, Wydział Informatyki i Matematyki, Katedra Informatyki, ul. Malczewskiego 29, 26-600 Radom; jacek@delta.pl

POTOKOWE PRZETWARZANIE DANYCH Z DZIENNIKÓW SYSTEMOWYCH Z WYKORZYSTANIEM FUNKCJI GENERATORÓW

PIPELINED PROCESSING OF DATA FROM SYSTEM LOGS USING THE FUNCTION GENERATORS

Słowa kluczowe: system operacyjny, dzienniki, przetwarzanie potokowe.
Keywords: operating system logs, pipelining.

Streszczenie

W artykule przedstawiono problem wyszukiwania informacji w dziennikach systemowych. Zapisy z dzienników pozwalają na rozwiązanie wielu problemów nieprawidłowego działania systemu, czy wybranej aplikacji. Zastosowana procedura przetwarzania potokowego składa się z kilku etapów i jest bardzo elastyczna. Można ją łatwo zmodyfikować i wykorzystać do innych zastosowań.

Summary

This article presents the problem of searching for information in the system log. Records from the logs allow you to solve many problems of a system malfunction. The procedure of processing pipeline consists of several steps and it is very flexible. It can be easily modified and used for other applications.

Wstęp

Przetwarzanie potokowe jest często stosowane w systemach, gdzie wymagane jest połączenie ze sobą różnych interfejsów. Wyniki w postaci strumienia danych produkowane przez pierwszy obiekt są pobierane jako dane wejściowe w drugim obiekcie, gdzie po odpowiednim przetworzeniu według zadanego algorytmu są prezentowane w postaci ostatecznego wyniku lub traktowane jako dane wejściowe dla kolejnego obiektu. Zastosowanie tego typu rozwiązania do

logów systemowych pozwala szybko wyszukać poszukiwany wpis wśród setek tysięcy wierszy zapisów.

Aplikacje działające pod kontrolą systemu operacyjnego odnotowują ślady swojej działalności w dziennikach systemowych. W miarę upływu czasu gromadzone dane w plikach mocno przybierają w swojej objętości. W przypadku nieprawidłowego działania systemu szukanie przyczyny poprzez zgromadzone zapisy w dziennikach staje się bardzo uciążliwe. Rozwiązanie przedstawione w dalszej części artykułu przedstawia jedno z rozwiązań tego problemu polegające na automatycznym przeszukiwaniu treści zawartej w plikach i wyświetlaniu tylko informacji pasującej do kryterium zadanego wzorca.

Opis dzienników systemowych

Dzienniki systemowe to nic innego jak pliki tekstowe zapisujące zdarzenia z przebiegu działania aplikacji w kolejnych wierszach. Większość tych plików w systemie operacyjnym Linux znajduje się w katalogu /var/log. Wydając polecenie ls -l można się przekonać, że jest to pokaźny zbiór informacji.

```
-rw-r--r-- 1 root root  1569 sty 10 11:48 alternatives.log
-rw-r--r-- 1 root root  9048 gru 29 10:35 alternatives.log.1
-rw-r—r--  1 root root 245 kwi  2  2015 alternatives.log.10.gz
-rw-r--r-- 1 root root 297 lut 28  2015 alternatives.log.11.gz
-rw-r--r-- 1 root root 508 sty 27  2015 alternatives.log.12.gz
-rw-r--r-- 1 root root 398 lis  7 11:19 alternatives.log.2.gz
…...
-rw-r----- 1 root adm        6993 sty 11 18:11 user.log
-rw-r----- 1 root adm        3762 sty  3 12:32 user.log.1
-rw-r----- 1 root adm         626 gru 26 19:49 user.log.2.gz
-rw-r----- 1 root adm        1021 gru 16 10:33 user.log.3.gz
-rw-r----- 1 root adm         986 gru  8 15:21 user.log.4.gz
-rw-rw-r-- 1 root utmp     465792 sty 11 19:24 wtmp
-rw-rw-r-- 1 root utmp    1083264 sty  2 14:07 wtmp.1
-rw-r--r-- 1 root root      22468 sty 11 19:23 Xorg.0.log
-rw-r--r-- 1 root root      23200 sty 11 18:11 Xorg.0.log.old
```

Listing 1. Fragment listingu plików w katlogu /var/log

Ze względu na dużą ilość plików listing 1 zawiera tylko kilka przykładowych plików. Całkowita ilość plików wynosi w tym przypadku około 150. Ta ilość jest oczywiście zmienna i zależy od aktywności aplikacji i generowanych przez nią częstotliwości zapisów do dziennika. Należy nadmienić, że prezentowane dane nie są pobierane z systemu pracującego jako serwer. W takim przypadku ruch generowany przez klientów korzystających z usług znacznie zwiększyłby ilość zapisanych danych.

Do omówienia problemu w tej pracy zostanie użyty jeden z dzienników, zapisujący aktywność do pliku /var/log/auth.log. Do pliku dopisywane są w przypadku zgłoszenia przez aplikację kolejne wiersze z informacjami o tym co dzieje się w systemie. Przykładowy zapis takiej struktury znajduje się w listingu 2. Jak można zauważyć, zapis, który się znajduje w kolejnych wierszach powinien zwrócić uwagę administratora systemu, ponieważ z jego treści wynika, że nieuprawniony klient używający maszyny o numerze IP 198.74.100.10 próbuje się dostać do systemu za pomocą usługi wykorzystując usługę ssh. Co więcej, próbuje uwiarygodnić swój adres IP wymuszając próbę dopisania do /etc/hosts.allow[1], co ma uwiarygodnić jego adres IP. Mechanizm działania tej procedury nie jest istotny jednak z punktu naszego artykułu, a jedynie fakt, że takie działanie zostało odnotowane w dzienniku.

Kolejne linie zapisywane do pliku zwiększają jego objętość, aż do określonej wartości, przy której zostaje otwarty nowy plik auth.log, a zawartość poprzedniego zostaje zapisana jako auth.log.1. Po osiągnięciu odpowiedniej wielkości przez kolejny plik zostaje otwarty nowy plik auth.log, a zawartość poprzedniego przeniesiona do auth.log.1, a tego z kolei do auth.log.2.gz i tak dalej zgodnie z opisaną wcześniej procedurą. Istotne z naszego punktu widzenia jest to, że kolejne pliki są spakowane i posiadają rozszerzenia gz.

```
root@dlt:/var/log# ls -l auth*
-rw-r----- 1 root adm 77674 sty 11 22:17 auth.log
-rw-r----- 1 root adm 66467 sty  3 12:39 auth.log.1
-rw-r----- 1 root adm  2585 gru 27 10:39 auth.log.2.gz
-rw-r----- 1 root adm  3887 gru 16 10:39 auth.log.3.gz
-rw-r----- 1 root adm  4101 gru  8 15:39 auth.log.4.gz

Jan 11 20:39:01 dot CRON[31981]: pam_unix(cron:session): session closed for user root
Jan 11 21:01:28 dot sshd[32072]: warning: /etc/hosts.allow, line 13: host name/address mismatch:
    198.74.100.10 != server.harik.com
Jan 11 21:01:28 dot sshd[32072]: refused connect from 198.74.100.10 (198.74.100.10)
Jan 11 21:09:02 dot CRON[32091]: pam_unix(cron:session): session opened for user root by
    (uid=0)
Jan 11 21:09:02 dot CRON[32091]: pam_unix(cron:session): session closed for user root
Jan 11 21:17:01 dot CRON[32145]: pam_unix(cron:session): session opened for user root by
    (uid=0)
Jan 11 21:17:01 dot CRON[32145]: pam_unix(cron:session): session closed for user root
Jan 11 21:27:36 dot sshd[32174]: warning: /etc/hosts.allow, line 13: host name/address mismatch:
    198.74.100.10 != server.harik.com
Jan 11 21:27:36 dot sshd[32174]: refused connect from 198.74.100.10 (198.74.100.10)
Jan 11 21:39:01 dot CRON[32203]: pam_unix(cron:session): session opened for user root by
    (uid=0)
Jan 11 21:39:01 dot CRON[32203]: pam_unix(cron:session): session closed for user root
Jan 11 21:47:27 dot sshd[32258]: refused connect from 112.254.6.109.rev.sfr.net (109.6.254.112)
```

[1] N. Gift, J. Jones, *Python for Unix and Linux system Administration*, O'Reilly 2008.

```
Jan 11 21:53:44 dot sshd[32276]: warning: /etc/hosts.allow, line 13: host name/address mismatch:
    198.74.100.10 != server.harik.com
Jan 11 21:53:44 dot sshd[32276]: refused connect from 198.74.100.10 (198.74.100.10)
Jan 11 21:55:05 dot sshd[32280]: refused connect from 112.254.6.109.rev.sfr.net (109.6.254.112)
Jan 11 22:09:01 dot CRON[32315]: pam_unix(cron:session): session opened for user root by
    (uid=0)
Jan 11 22:09:01 dot CRON[32315]: pam_unix(cron:session): session closed for user root
Jan 11 22:17:01 dot CRON[32369]: pam_unix(cron:session): session opened for user root by
    (uid=0)
Jan 11 22:17:01 dot CRON[32369]: pam_unix(cron:session): session closed for user root
```

Listing 2. Przykładowy fragment zawartości pliku auth.log

Opis problemu

Po lekturze poprzedniego rozdziału można dojść do wniosku, że odnalezienie potrzebnych informacji z tak obszernych zbiorów danych jest bardzo czasochłonne i żmudne.

Dlatego też naszym zadaniem będzie napisanie programu, którego zadaniem jest wyszukiwanie interesującego zdarzenia za nas. Zadanie na pewno utrudnia fakt, że większość plików jest w postaci zarchiwizowanej, czyli nie jest czystym plikiem tekstowym.

Program zostanie napisany w języku Python[2], który jest często wykorzystywany we współczesnych aplikacjach pracujących w środowiskach sieciowych. Do przetworzenia przez program będzie bardzo duża ilość danych, bez konieczności umieszczania ich w całości w pamięci.

Zadanie będzie składało się z kilku etapów.

Pierwszym z nich będzie wyszukanie w strukturze drzewa katalogów wszystkich nazw pasujących do wzorca wieloznacznego powłoki. Wykorzystane tu zostaną funkcje generatorów.

Kolejnym będzie otwieranie plików sekwencji po kolei i tworzenie obiektów plikowych. Pliki są kolejno otwierane i zamykane.

Następny etap będzie łączył poszczególne iteratory w całościową sekwencję.

Ostatnim etapem będzie wyszukiwanie w sekwencji wierszy pasujących do przyjętego regularnego.

Połączenie opisanych wyżej procedur pozwoli na przeszukanie dowolnie dużego zbioru danych w postaci plików w wybranym katalogu pasujący do zadanego wzorca i analizę zawartości treści pod kątem poszukiwanej informacji.

---

[2] A. Downey, *Python for Software Design*, Cambridge University Press 2009; A. Downey, *Think Python*, O'Reilly 2012; M. Goodrich, R. Tamassia, M. Goldwasser, *Data Structures and Algorithms in Python*, Wiley 2013; D. Hellman, *The Python Standard Library by Example*, Addison-Wesley 2011; Y. Hilpisch, *Derivatives Analytics with Python*, Wiley 2015; J. Payne, *Beginning Python*, Wrox 2010.

Rozwiązanie problemu

Opierając się na opisach procedur zawartych w poprzednim rozdziale wiemy, że przetwarzanie danych będzie odbywało się w czterech kolejnych krokach. Każdy etap przekazuje wynik swojej pracy jako parametr do kolejnego etapu, aż do momentu uzyskania żądanej informacji.

Etap 1

Pierwszym elementem jest wskazanie wzorca plików, które zamierzamy przeszukać oraz miejsca, w którym się znajdują.

Utworzono do tego celu funkcję przyjmującą dwa parametry: pierwszy parametr to poszukiwany wzorzec, a drugi katalog, w którym mogą się znajdować pliki. Po zaimportowaniu modułu os, zawierającego między innymi funkcję do manipulowania plikami i katalogami, wykorzystujemy os.walk[3] do przejrzenia zawartości katalogu podawanego jako jeden z parametrów. Drugim parametrem podawanym na tym etapie pracy jest miejsce w drzewie katalogów, gdzie należy szukać plików pasujących do wzorca ('auth.log*', '/var/log/').

```
for sciezka, katlist, pliklist in os.walk(top):
    for wzc in fnmatch.filter(pliklist, plikpat):
        yield os.path.join(sciezka,wzc)
```

Listing 3. Fragment funkcji wyszukującej pliki wg zadanych wzorców

Wynikiem działania tej funkcji jest lista plików pasujących do szukanego wzorca:

```
/var/log/auth.log.3.gz
/var/log/auth.log
/var/log/auth.log.2.gz
/var/log/auth.log.1
/var/log/auth.log.4.gz
```

Listing 4. Wynik działania I etapu pracy

Etap 2

Po udanej selekcji listy plików, która może zawierać interesujące nas dane należy je otworzyć jeden po drugim, a następnie zamknąć. W tym przypadku należy utworzyć funkcję otwierającą pliki, przyjmującą jako parametr po kolei pliki uzyskane w poprzedniej procedurze. Dla tego przypadku należy skorzystać z biblioteki gzip. Kolejno są sprawdzane pliki dostarczone w etapie pierwszym. Jeśli są spakowane, czyli koniec nazwy pliku jest zapisany jako .gz wówczas otwierane są za pomocą wspomnianej wcześniej biblioteki. Jeśli jest to typowy plik tekstowy otwarcie następuje za pomocą standardowej funkcji open. Wynik

[3] M. Summerfield, *Programming in Python 3*, Addison-Wesley 2010; T. Ziade, *Packt*, *Expert Python Programming*, Publishing 2008.

każdej operacji jest zapisywany do zmiennej f i dołączany w kolejnym przejściu pętli za pomocą generatora.

```
for plikname in pliknames:
    if plikname.endswith('.gz'):
        f = gzip.open(plikname, 'rt')
    else:
        f = open(plikname, 'rt')
    yield f
    f.close()
```

Listing 5. Otwieranie plików

Etap 3

Uzyskane w ten sposób zasoby informacji należy połączyć w jedną sekwencję. Uzyskujemy to tworząc kolejną funkcję w potokowym przetwarzaniu danych. Także i w tym przypadku parametrem funkcji będzie produkt działania funkcji z etapu 2.

```
        for c in iterators:
            yield from c
```

Listing 6. Łączenie w całość

```
….
Dec 14 23:33:11 dlt dbus[2760]: [system] Rejected send message, 10 matched rules; type="error",
    sender=":1.25" (uid=0 pid=3378 comm="/usr/bin/pulseaudio --start ") interface="(unset)"
    member="(unset)" error name="org.bluez.MediaEndpoint1.Error.NotImplemented" request-
    ed_reply="0" destination=":1.0" (uid=0 pid=2815 comm="/usr/sbin/bluetoothd ")
Dec 14 23:33:11 dlt dbus[2760]: [system] Rejected send message, 10 matched rules; type="error",
    sender=":1.25" (uid=0 pid=3378 comm="/usr/bin/pulseaudio --start ") interface="(unset)"
    member="(unset)" error name="org.bluez.MediaEndpoint1.Error.NotImplemented" request-
    ed_reply="0" destination=":1.0" (uid=0 pid=2815 comm="/usr/sbin/bluetoothd ")
Dec 15 10:13:42 dlt kdm: :0[2987]: pam_unix(kdm:session): session opened for user root by
    (uid=0)
Dec 15 10:13:42 dlt kdm: :0[2987]: pam_ck_connector(kdm:session): nox11 mode, ignoring
    PAM_TTY :0
….
```

Listing 7. Wynik działania etapu 3

Etap 4

Skoro poprzednia funkcja zapewniła nam dostęp do całej zawartości wyszukanych wcześniej plików, to pozostało tylko zastosować wzorzec do każdej przetwarzanej linii, aby wyselekcjonować tylko te wpisy z dziennika, które niosą ze sobą interesującą informację selekcjonowaną wg zadanego wzorca. Także w tej funkcji parametrem będzie zmienna zawierająca wynik działania procedury z poprzedniego etapu, ale i dodatkowo zmienna ze zdefiniowanym wzorcem do wyszukiwania.

```
pat = re.compile(pattern)
for line in lines:
```

```
    if sciezka.findall(str(line)):
        yield line
```

Listing 8. Przeszukiwanie wierszy wg zadanego wzorca

```
…
Dec 14 23:17:01 dlt CRON[6397]: pam_unix(cron:session): session opened for user root by (uid=0)
Dec 14 23:17:01 dlt CRON[6397]: pam_unix(cron:session): session closed for user root
Dec 15 10:17:01 dlt CRON[3790]: pam_unix(cron:session): session opened for user root by (uid=0)
Dec 15 10:17:01 dlt CRON[3790]: pam_unix(cron:session): session closed for user root
…
```

Listing 9. Ostateczny wynik działania programu

Wnioski

Opisane rozwiązanie opisuje ogólne podejście do problemu. Można łatwo zmodyfikować program pod kątem przeszukiwanych plików, jak i szukanej zawartości. Zastosowane w artykule podejście z potokowym przetwarzaniem danych stanowi, że każda funkcja na poszczególnym etapie produkuje dane za pomocą generatora yield, a kolejna przyjmuje wynik tej produkcji i przetwarza w pętli jej zawartość wg kolejnych wzorców generując dane dla kolejnej funkcji, aż do uzyskania oczekiwanego wyniku. Należy zauważyć, że zastosowane rozwiązanie jest krótkie, czytelne i poszczególne etapy są niezależne od siebie.

Rozbudowa interfejsu komunikacji z klientem oraz użycie słowników z wpisanymi standardowymi słowami kluczowymi, może znacznie uprościć i uprzyjemnić pracę z aplikacją. W przypadku często powtarzających się niepożądanych wpisów w logach można dopisać moduł samoczynnie przeszukujący określone treści i informujący nas o tym.

Bibliografia

Downey A., *Python for Software Design*, Cambridge University Press 2009.
Downey A., *Think Python*, O'Reilly 2012.
Gift N., Jones J., *Python for Unix and Linux sysytem Administration*, O'Reilly 2008.
Goodrich M., Tamassia R, Goldwasser M, *Data Structures and Algorithms in Python*, Wiley 2013.
Hellman D., *The Python Standard Library by Example*, Addison-Wesley 2011.
Hilpisch Y., *Derivatives Analytics with Python*, Wiley 2015.
Payne J., *Beginning Python*, Wrox 2010.
Summerfield M., *Programming in Python 3*, Addison-Wesley 2010.
Ziade T., *Packt, Expert Python Programming*, Publishing 2008.