PP > WE > Stacjonarne II st. Informatyka > Sem. 9	Integracja i eksploracja danych. Laboratorium.	Rok ak. 2017/18 Sem. Zimowy
TI Konrad Dysput	ELK + Docker	18.12. 2017r

Spis treści

Podstawowe pojęcia

Docker
Platforma umożliwiająca budowanie, zarządzanie oraz bezpieczeństwo wielu aplikacji uruchamianych w skonfigurowanym wydzielonym środowisku. Platforma w odróżnieniu od rozwiązań opartych o maszyny wirtualne nie potrzebuje operować na emulowanej warstwie sprzętowej oraz systemu operacyjnego. Docker w kontenerze uruchamia procesy aplikacji, które działają w oparciu o system operacyjny, a nie emulowane środowisko. Nic nie stoi na przeszkodzie, aby uruchamiać więcej niż jeden kontener dockera, ponieważ działają one niezależnie od siebie.

Kibana
Aplikacja www umożliwiająca wizualizacje oraz analizę danych przechowywanych w Elastic Search.

Logstash
Aplikacja, umożliwiająca zbieranie, a następnie filtrowanie przetwarzanych danych. Dla dowolnego strumieniu danych Logstash pozwala przetworzyć wejście, wyfiltrować je, a następnie zapisać/ przetworzyć wyjście.

Elastic Search
Aplikacja umożliwiająca indeksowanie danych poddawanych przetwarzaniu i analizie. Zaindeksowane dane możemy następnie poddać pełnotekstowemu przeszukiwaniu w celu otrzymania wyników. Możliwości dostarczane przez Elastic Search widoczne są szczególnie w wyszukiwaniu pełnotekstowym, w którym użytkownik wpisując tekst liczy na znalezienie wyników oraz informacji z nimi powiązanymi.

Przygotowanie środowiska

Docker vs Docker Toolbox

Przygotowane zadania zostały zrealizowane przy pomocy platformy *Docker* przygotowanej na system operacyjny *Windows 10*. W celu uruchomienia platformy Docker na systemie operacyjnym Windows można było skorzystać z dwóch opcji:

- Docker for Windows
- Docker Toolbox

Docker Toolbox jest narzędziem przygotowanym dla użytkowników z systemem operacyjnym Windows dla wersji Home. Swoje działanie opiera o maszyny wirtualne działające przy użyciu narzędzia VirtualBox. Przydatnym narzędziem dla pakietu Docker Toolbox jest program Kitematic, który w sposób graficzny prezentuje aktualny stan kontenerów oraz ich podstawowe parametry konfiguracyjne – zmienne środowiskowe, konfiguracje sieci, czy stan kontenera. Narzędzie Docker Toolbox często nie oferowało tyle możliwości co oprogramowanie Docker for Windows. Jednakże jest to bardzo dobra alternatywa dla osób nie posiadających systemu Windows w wersji wyższej niż Home. W odróżnieniu od oprogramowania Docker na inne systemy operacyjne oraz oprogramowania Docker for Windows, aby dostać się do aplikacji działających w Docker Toolboxie, należy odnosić się do adresu IP wirtualnej maszyny. Zachowanie te nowych użytkowników oprogramowania może wprowadzać w błąd, ale również umożliwia eksportowanie portów kontenerów bez obawy, że dany port jest zajęty przez aplikację działającą w systemie operacyjnym.

Docker for Windows jest alternatywą dla narzędzia Docker Toolbox. Microsoft oparł działanie oprogramowania o maszyny wirtualne działające w oparciu o Hyper-V oraz umożliwił korzystanie z kontenerów zarówno działających na Windowsie jak i na Linuxie. Niestety możemy wybrać jeden docelowy system operacyjny. Jest to narzędzie, z którym bardzo dobrze zintegrowane są środowiska programistyczne takie jak np. Visual Studio, które min. umożliwi debuggowanie aplikacji znajdujących się w kontenerach albo przygotowanie paczek kontenerów. Znaczącym minusem rozwiązania jest wykorzystanie Hyper-V, które nie cieszy się dużą popularnością wśród osób zajmujących się maszynami wirtualnymi. Aby Windows mógł korzystać z Hyper-V należy zmienić konfigurację w Panelu Sterowania. Edycja konfiguracji jest licznie krytykowana przez użytkowników Docker for Windows, ze względu na wprowadzanie inwazyjnych zmian do systemu operacyjnego.

Docker compose YML

W celu przygotowania pliku docker-compose należy przygotować plik deklarujący użyte kontenery, wersje oraz ich właściwości – np. zmienne środowiskowe, czy wyeksportowane porty. Plik ten należy przygotować w formacie *YAML – YAML Ain't Markup Language*. Rozwiązanie te może nieść ze sobą dużo problemów w przypadku przygotowywania złożonych plików docker-compose. Bardzo często problemy z formatowaniem są przyczyną dużej ilości błędów wynikających z błędów takich jak dodatkowa spacja. Format pliku YAML nie jest zbyt jednoznaczny (poprzez dodatkowe białe znaki) do walidacji, przez co utrudnia on w znacznym sposób pracę z plikami docker-compose.

Parametry środowiskowe

Do uruchomienia platformy niezbędna była konfiguracja komputera uwzględniająca włączenie wirtualizacji oraz programu *Hyper-V*. Uruchomienie kontenerów dockera ze stosu technologii *ELK* jest możliwe tylko przy zmianie konfiguracji działania Dockera na systemy *Linuxowe*. Start aplikacji był możliwy dzięki *docker-compose*, czyli aplikacji do uruchamiania wielu kontenerach jednocześnie.

Aplikacje ze stosu ELK uruchamiane są na systemie Linux, na dystrybucji Centos. W celu łatwiejszej analizy na dystrybucjach zostały zainstalowane dodatkowe paczki umożliwiające przetwarzanie danych oraz typowe czynności wykonywane na serwerach jak np. wypakowanie archiwum zip.

W celu zbierania danych z wielu aplikacji uruchamianych na wielu kontenerach, stworzono aplikacje umożliwiające uruchomienie na systemach Linuxowych. W związku z tym przygotowano programu w oparciu o języki:

- *JavaScript,*
- *.NET Core,*
- *Python.*

Celem przygotowanych programów jest generowanie danych, które następnie mogą być przetwarzane oraz analizowane przez stos ELK. Kod źródłowy aplikacji jest dostępny pod adresem: https://github.com/konraddysput/ELKSample na repozytorium *Github*.

Aplikacje zostały przygotowane na platformie Windows, a docelowo działają na kontenerze dockera opartym o dystrybucje Linux – *Alphine*. Oprogramowanie użyte do stworzenia aplikacji to: *Visual Studio, Visual Studio Code* oraz *PyCharm.*

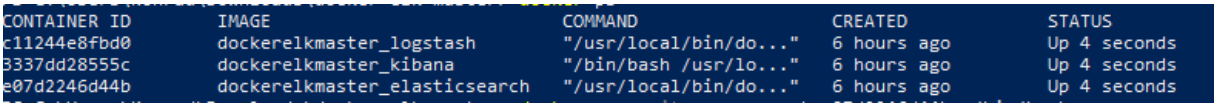
Uruchomienie i przygotowanie środowiska

Uruchomienie dockera oraz stosu ELK

Stos ELK dostępny jest pod adresem: https://github.com/deviantony/docker-elk W celu rozpoczęcia prac sklonowano repozytorium, a w folderze projektu wykonano komendę:

Docker-compose up

Pomyślne wykonanie polecenia możemy sprawdzić poprzez użycie komendy docker ps, które umożliwi podejrzenie aktualnie uruchomionych kontenerów. Konfiguracja docker-compose w repozytorium uwzględnia eksport portów używanych przez aplikację w kontenerze na zewnątrz kontenera. Do każdej z aplikacji można dostać się z przeglądarki poprzez adres lokalny oraz zdefiniowany port w pliku *docker-compose.yml*.



Aplikacje w pliku docker-compose mają wskazane porty, z których można się połączyć do odpowiednich portów w kontenerze. Domyślna konfiguracja pozwala między innymi na dostęp do kibany przy pomocy portu 9300. Wszystkie kontenery komunikują się wzajemnie przy pomocy sieci określonej w pliki docker-compose. Alternatywą do tworzenia

sieci w konfiguracji wszystkich kontenerów jest użycie słowa kluczowego link w definicji kontenera. Wskazanie jako argument klucza tablicy kontenerów pozwoli na stworzenie połączenia.

Oprogramowanie

W celu stworzenia własnych kontenerów logujących zdarzenia zachodzące w aplikacji zostały stworzone aplikacje przy użyciu: nodejs, .NET Core oraz pythona. Wykorzystanym systemem operacyjnym był Linux. Kontenery użyte do uruchomienia kodu zostały znalezione na stronie *hub.docker.com*. Kod niezbędny do uruchomienia kontenerów znajduje się również na repozytorium github. Celem napisanych aplikacji jest generowanie losowych informacji z użytkowania docelowo aplikacji produkcyjnych.

Kod aplikacji – nodejs

```
class Main{
    static doWork(){
        const names = ["Jenifer", "John", "Abby", "Bailey", "Rian"];
        const actions =["set new visualisation", "prepare new template", "upl
        setInterval(function(){
            let name = names[Math.floor(Math.random() * names.length)];
            let action =actions[Math.floor(Math.random() * actions.length)];
            console.log(`${name} ${action}`);
        }, 2500);
    }
}

Main.doWork();
```

Kod aplikacji – python

```
import random
from datetime import datetime, date, time, timedelta
from time import sleep

user = ['konrad', 'tiger', 'nno44', 'blue_sky']
arr = [' join a chanel', 'create new event', 'cancel event', 'logout from server', 'receive new mess

def period_log():
    while True:
        print(user[random.randint(0, len(user) - 1)] + " " + arr[random.randint(0, len(arr) - 1)])
        sleep(1)

if __name__ == "__main__":
    period_log()
```

Kod aplikacji – dotnet core

```
namespace dotnet
{
    0 references
    class Program
    {
        2 references
        private static string[] _users = {"Administrator", "SuperUser", "Technical User", "John", "Bo
        2 references
        private static string[] _actions = {"create new user", "change application configuration", "s
        0 references
        static void Main(string[] args)
        {
            PeriodLog();
        }
        1 reference
        private static void PeriodLog()
        {
            Random urandom = new Random();
            while(true){
                Console.WriteLine($"{_users[urandom.Next(0,_users.Length)]} {_actions[urandom.Next(0,
                Thread.Sleep(500);
            }
        }
    }
}
```

Praca ze środowiskiem

Wdrożenie aplikacji do ekosystemu ELK

Plik docker-compose został wyposażony w dodatkowe wpisy, które mają na celu uruchomienie pozostałych kontenerów jeden po drugim. W pliku YAML w atrybucie build została wskazana ścieżka służąca do zbudowania obrazu. Uruchomienie zbioru kontenerów przy użyciu komendy *docker-compose up* spowoduje pobranie oraz zbudowanie brakujących kontenerów. W przypadku gdy chcielibyśmy zbudować ręcznie kontener (aby np. udostępnić go innej osobie) należałoby wejść do odpowiedniego folderu oraz wykonać komendę *docker build*. W komendzie warto uwzględnić dodatkowy atrybut -t , który spowoduje nazwanie

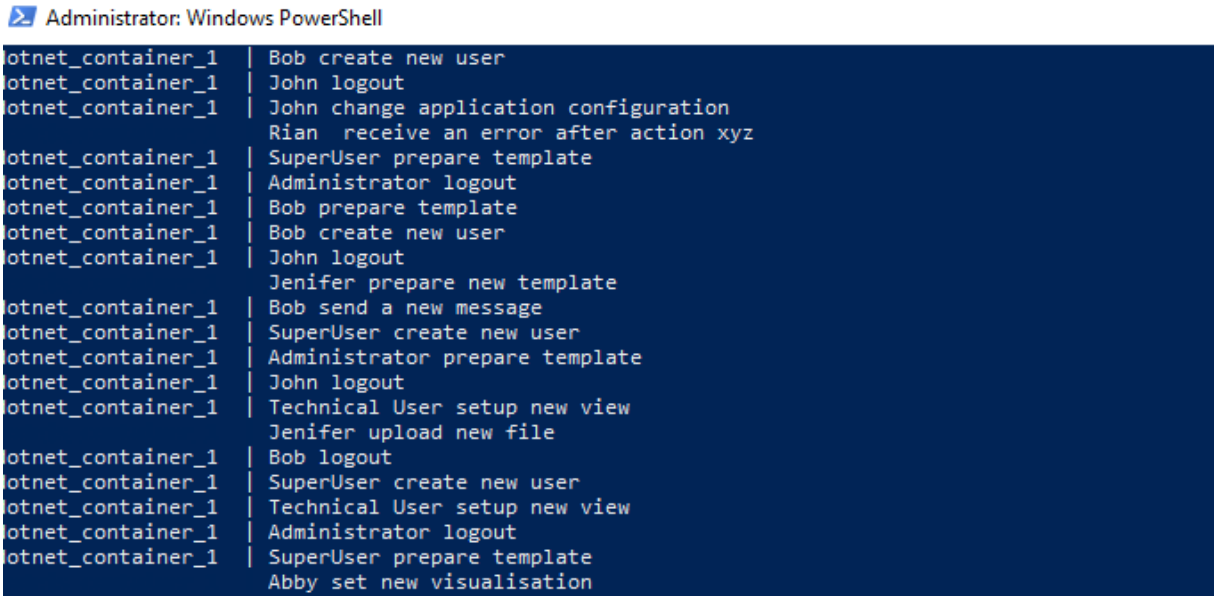
Sposób logowania danych z kontenerów do logstash

Tworząc aplikacje programista dąży do stworzenia kodu, który nie jest zależny od warunków środowiska developerskiego/produkcyjnego. Bardzo często może zajść potrzeba konfiguracji środowiska produkcyjnego według pewnych wytycznych, które mogą np. uniemożliwić nam logowania danych do stosu ELK. Docker wyposażony jest w mechanizm przekazywania logów ze wskazanych kontenerów pod odpowiedni adres (np. adres logstasha). Poza tym istnieje wiele innych sposobów przekazywania informacji z kontenerów do innych kontenerów np. wykorzystanie oprogramowanie logspout.

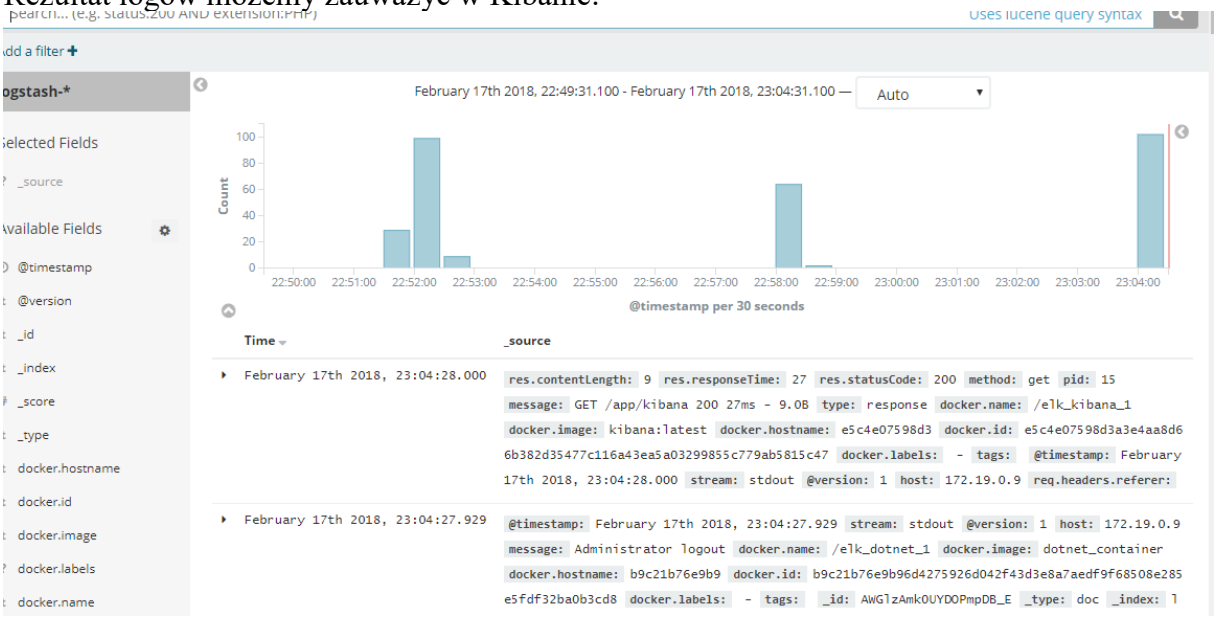
Sposób uruchomienia

W katalogu głównym repozytorium umieszczony został plik *docker-compose.yml*. Wpisanie przy użyciu dowolnego terminala komendy *docker-compose up* spowoduje pobranie wszystkich używanych kontenerów oraz uruchomienie oprogramowania na nim. Jeżeli zauważymy że aplikacji nie uruchomiła wraz z wpisaniem komendy, nie powinniśmy się przejmować błędami. W pliku *docker-compose.yml* zostały zawarte odpowiednie komendy, które gwarantują między innymi restart kontenera w przypadku gdyby aplikacji przestała by działać. Konfiguracja zawiera cztery aplikacje generujące dane oraz cztery aplikacje zbierające i przetwarzające je. Poza aplikacjami w trzech różnych językach programowania (python, nodejs oraz .NET core) został dodany również serwer nginx, który wygeneruje log w przypadku wejścia pod adres: localhost:8000.

Przykład działających aplikacji możemy zauważyć na konsoli:



Rezultat logów możemy zauważyć w Kibanie:



Napotkane problemy

Docker-compose oraz niepoprawne zamykanie maszyn

Problem ten jest charakterystyczny dla oprogramowania Docker na systemy operacyjne Windows. Kończąc pracę z konfiguracją kontenerów, osobie konfigurującej zależy na jak najszybszym zamknięciu uruchomionych kontenerów. W wielu przypadkach mimo poprawnego wysłania sygnału zamknięcia systemów operacyjnych, tworzone były locki, które miały zapobiegać uruchamianiu wielu instancji tej samej aplikacji na serwerze. Powodowało to sytuacje, w których ponowne uruchomienie np. docker-compose powodowało błędy. Utworzone locki można było usunąć poprzez zabicie odpowiednich procesów w kontenerze, albo poprzez reset oprogramowania Docker. Pierwszy z wymienionych sposób był bardzo często trudny do uzyskania ze względu na krótki czas życia kontenera. System uruchamiał się, napotykał problem, po czym następnie kończył pracę ze statusem innym niż 0. W tym przypadku pomocne może okazać się użycie w pliku docker-compose tymczasowo komendy command, która przed uruchomieniem aplikacji w kontenerze wykona komendę zabicia – np. procesu ElasticSearch. W przypadku ponownego uruchomienia dockera problem nie koniecznie musi być rozwiązany, ze względu na zachowanie oprogramowania na systemach operacyjnych Windows.

Połączenie z bazą danych oraz oczekiwanie na kontenery

Aplikacje internetowe w większości przypadków otrzymane dane zapisują w bazie danych. Przy uruchamianiu nowej instancji aplikacji z nowym kontenerem bazy danych, tworzone są struktury bazodanowe przy pomocy narzędzia ORM. Brak działającej bazy danej na adresie i porcie zdefiniowanym w plikach konfiguracyjnych może powodować otrzymanie wyjątku, a sama aplikacja może się nie uruchomić. Docker w pliku docker-compose zapewnia metodę synchronizacji uruchomień kontenerów, która jest zależna od min. kolejności wpisów kontenerów oraz od informacji zawartych w węźle depends_on. Nie oznacza to jednak, że kolejna aplikacja zostanie uruchomiona po skończeniu inicjalizacji poprzedniej. Wszystkie z kontenerów uruchamiane są w tle, a kolejność metody synchronizacji umożliwiają kolejność ich uruchamiania, W przypadku synchronizacji aplikacji www z bazą danych często napotykanym problemem był czas uruchomienia kontenerów. W celu rozwiązania tego problemu można skorzystać dodatkowo ze skryptów *wait-for.sh, dockerize*. Aby wykorzystać te skrypty, należałoby modyfikować pliki docekrfile oraz w niektórych przypadkach instalować dodatkowe paczki na serwerze, które mogłyby służyć do sprawdzenia połączenia z bazą danych. Innym rozwiązaniem jest zabezpieczenie kodu przed próbą połączenia do nie istniejącej bazy danych oraz próba ponownego wznowienia połączenia po określonym czasie działania.

Podsumowanie

Zyski, które przyniosło mi zadanie laboratoryjne

1. Migracja projektów webowych na kontenery w Dockerze – aktualnie przeniosłem jeden duży projekt, który działa w oparciu o różne bazy danych, kolejki i różne języki programowania na dockera
2. Migracja oprogramowania na komputerze – bardzo często bywało, że oprogramowanie niezbędne do sprawozdania laboratoryjnego używałem tylko raz na komputerze. Oprogramowanie mimo tego że zajmowało miejsce na dysku oraz nanosiło zmiany w systemie, to bardzo często działało w tle, co spodowowało zużywanie zasobów i dyskomfort z pracy na komputerze. Przykład oprogramowania które przeniosłem do dockera ze swojego prywatnego komputera: RabbitMQ, MSSQL, MongoDb, RServer, CentOS
3. Rozwój nowo wytwarzanego oprogramowania w oparciu o konteneryzacje (mikroserwisy) oraz inne przydatne kontenery (redis)

Dalsze kroki

1. Przejście na zarządzanie kontenerami z wykorzystaniem Kubernetesa,
2. Migracja kontenerów do chmury,

Koszty, które niosą ze sobą kontenery w dockerze

1. Ogromne koszty utrzymania kontenerów w chmurze,
2. Trudności z pracą z oprogramowaniem Docker for Windows lub Docker Toolbox,