LABORATORIUM PROGRAMOWANIA W CHMURACH OBLICZENIOWYCH

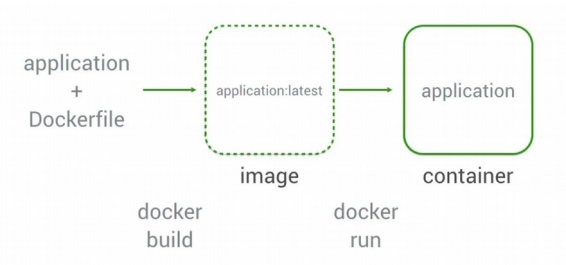
LABORATORIUM NR 6.

Do tej pory wykorzystanie kontenerów Docker ograniczało się do pojedynczych usług lub aplikacji uruchamianych w środowisku Docker. Jenak w ogromnej większości przypadków, na bazie kontenerów tworzone są złożone usługi bazujące na koncepcji mikrousług. Potrzebne jest zatem narzędzie które pozwoli na opis takich wielokontenerowych systemów. Sposób definiowania takich opisów powinien z jednej strony pozwalać na elastyczne odzwierciedlenie dowolnej logiki funkcjonowania projektowanego systemu usługowego za pomocą sformalizowanej składni języka opisu a z drugiej, na umożliwienie implementacji tej logiki w dowolnym (lub możliwie szerokiej gamie) środowisku chmurowym wyposażonym w narządzie automatycznej orkiestracji struktur wielokontenerowych.

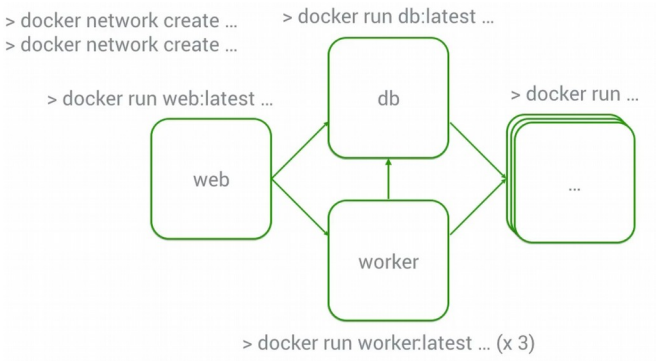
Obecnie, w kontekście środowiska Docker, istnieją trzy dominujące rozwiązania. Pierwsze i najstarsze opiera się na narzędziu Docker Compose https://docs.docker.com/compose/ . Pozwala na stworzenie pliku, który opisuje dowolną architekturę wielokontenerową a w oparciu o ten plik możliwe jest uruchamianie, modyfikowanie, testowanie i proste skalowanie określonej usługi. Kolejnym rozwiązaniem jest tworzenie opisów w postaci plików snap przeznaczone do uruchamiania w strukturach klastrowych Swarm (Swarm to natywny klaster dla środowiska Docker https://docs.docker.com/engine/swarm/) na jednej lub wielu maszynach fizycznych, wirtualnych lub serwerach bare metal. Opisy w postaci plików *docker-compose* jak i *snap* mogą (zazwyczaj po modyfikacjach) być podstawa do bezpośredniego wdrażania usług w chmurach publicznych jak i w chmurach prywatnych. Trzecia metoda oparta jest o wdrażanie usług wielokontenerowych w oparciu o orkiestrator Kubernetes https://kubernetes.io/ . W chwili obecnej wdreżenia takie są możliwe od pojedynczych maszyn (laptop, server, serwer wirtualny), przez dedykowane struktury klastrowe aż po specjalizowane serwisy największych dostawców usług chmurowych.

Niniejsze laboratorium poświęcone jest narzędziom Docker Compose oraz zasadom tworzenia i wykorzystania plików docker-compose.yml.

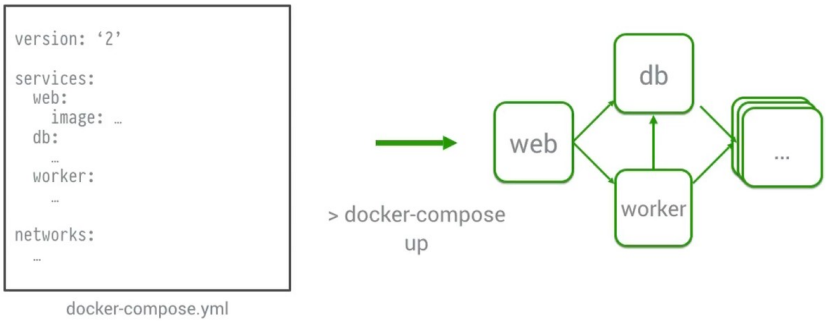
Na podstawie poprzednich laboratoriów powinno być jasne wykorzystanie środowiska Docker do budowy obrazu i na jego podstawie uruchomienia kontenera z określoną usługą/aplikacją, tak jak schematycznie ilustruje to rysunek poniżej.



Również na podstawie poprzednich laboratoriów, wykorzystując wiedzę o funkcjonowaniu komunikacji pomiędzy kontenerami i zasadach wykorzystania wolumenów, jesteśmy w stanie „ręcznie” uruchomić usługę, która składa się z więcej niż jednego kontenera, np. według schematu postępowania pokazanego na kolejnym rysunku.



Z oczywistych powodów powyższe podejście jest pracochłonne, podatne na błędu ludzkie a co najważniejsze, nie może być automatycznie replikowane do różnych środowisk docelowych. Wobec tego zaproponowany został schemat opisu złożonych usług wielokontenerowych przy pomocy plików YAML. Tą ideę ilustruje rysunek poniżej:



Docker Compose jest zestawem narzędzi pozwalających na konfigurowanie i uruchamianie aplikacji opartych o wiele kontenerów Docker. Podstawą do zautomatyzowania procesu wdrażania aplikacji jest tzw. plik compose (*docker-compose.yml*). Wykorzystanie Docker Compose obejmuje następujące trzy zadania cząstkowe:

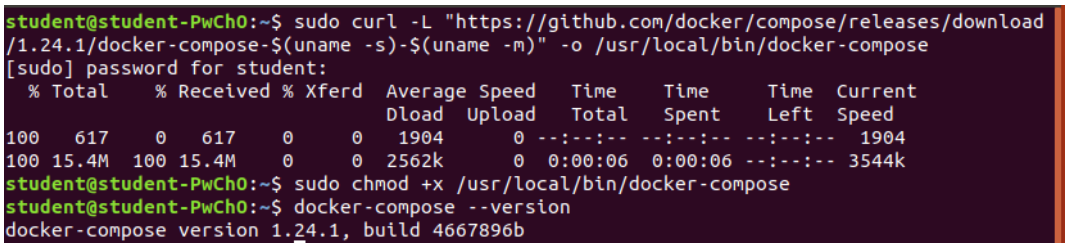
- utworzenie pliku (-ów) *Dockerfile*, dzięki którym możliwe jest określenie środowiska programistycznego i składowych aplikacji, jaka ma się znajdować w danym Dockerze,
- utworzenie pliku docker-compose.yml, dzięki któremu opisane są związki pomiędzy poszczególnymi kontenerami (aplikacjami w kontenerach Docker). Na tej podstawie tworzone jest izolowane środowisko programistyczne i sieciowe, niezbędne dla całego systemu wielokontenerowego,
- uruchomienie opracowanego (opisanego) środowiska i systemu/aplikacji za pomocą narzędzia *docker-compose up*.

Instalacja środowiska docker compose dla poszczególnych systemów operacyjnych jest opisana w dokumentacji, pod adresem: https://docs.docker.com/compose/install/#install-compose . W systemie Linux proces instalacji przebiega kilku etapowo:

1. Pobranie najnowszej wersji compose. Numer wersji należy sprawdzić na stronie projektu. W momencie opracowywania niniejszej instrukcji była to wersja 1.24.1.

```
sudo curl -L ”https://github.com/docker/compose/releases/download/1.24.1/docker-compose-(unames)−(uname -m)” -o /usr/local/bin/docker-compose
```

2. Nadaniu uprawnień -x dla popranych plików.
3.Weryfikacji poprawności instalacji.



Odinstalowanie środowiska compose polega na usunięciu katalogu:

```
sudo rm /usr/local/bin/docker-compose
```

A. Pliki docker-compose.yml

Każdy plik *docker-compose.yml* rozpoczyna się od instrukcji wskazującej na wersję składni pliku, np.:

```
version: '3.7'
```

UWAGA: Wybór wersji w pliku *docker-compose.yml* jest istotny ze względu na możliwość przyszłej implementacji opisanej usługi w różnych środowiskach. W przypadku implementacji w chmurach publicznych czy też w klastrze Swarm ZDECYDOWANIE powinno się wykorzystywać wersję 3 i większe.

Szczegółową informację na temat poszczególnych wersji plików docker-compose.yml w połączeniu z kolejnymi wrersjami środowiska Docker można uzyskać pod adresem: https://github.com/docker/compose/releases . Natomiast różnice w składni poszczególnych wersji wraz z przykładami można znaleźć w dokumentacji środowiska Docker, pod adresem: https://docs.docker.com/compose/compose-file/compose-versioning/

Kluczową zmianą w stosunku do wcześniej poznanego pliku dockerfile.yml jest wprowadzenia pojęcia usług (ang. services). Każdy plik docker-compose.yml musi opisywać elementy składowe każdej usługi cząstkowej (obrazy, ich konfigurację, sposób wykorzystania wolumenów, sieci itd.) oraz niezbędne składniki infrastruktury (typowo: sieci, wolumeny). W związku z tym szkielet struktury pliku docker-compose.yml można przedstawić następująco:

```
version: 'x'

services:
      service1:
      service2:
      service3:

networks:

volumes:
```

Oczywiście po dwukropkach należy utworzyć opisy poszczególnych składników pliku. W tym celu należy korzystać z instrukcji dostępnych dla wybranej wersji składni pliku. Pełna dokumentacja składni w wersji 3 jest dostępna pod adresem: https://docs.docker.com/compose/compose-file/

UWAGA 1: Zapoznając się z powyższą dokumentacją proszę zwracać uwagę na kompatybilność poszczególnych instrukcji i flag z klastrem Swarm.

Z kolei wykorzystanie środowiska docker-compose jest opisane w dokumentacji, pod adresem: https://docs.docker.com/compose/

Ze względu na obszerność przytoczonych wyżej dokumentacji, najlepszą metodą zapoznawania się z Docker Compose jest samodzielne uruchamianie przykładowych usług wielokontenerowych. Proszę pamiętać, że poza licznymi blogami tematycznymi i tutorialami dostępnymi w Internecie, twórcy narzędzi Compose prezentują zestaw przykładów, które można traktować jako przykłady referencyjne i warto się z nimi zapoznać przed realizacją własnych projektów. Są one dostępne pod adresem: https://docs.docker.com/compose/samples-for-compose/

W dalszej części tego punktu instrukcji przedstawiona będzie przykładowa usługa monitorowania środowiska Docker. Poza rolą ilustracyjną, może stać się ona podstawą do tworzenia własnych systemów monitorowania i nadzoru tak środowiska Docker, pojedynczych kontenerów, klastrów jak i całych usług wielokontenerowych. Usługa monitorowania złożona jest z trzech usług/aplikacji składowych:

- Cadvisor - https://github.com/google/cadvisor , https://hub.docker.com/r/google/cadvisor/
- Prometheus – https://prometheus.io/ , https://hub.docker.com/r/prom/prometheus/
- Grafana - https://grafana.com/ , https://hub.docker.com/r/grafana/grafana/

Struktura usługi monitorowania zawiera „backend” zawierający kontenery z aplikacjami Cadvisor i Prometheus oraz „frontend” w postaci kontenera Grafana, który dostępny ma być dla klientów zewnętrznych. Wobec tego ogólna struktura pliku docker-compose.yml powinna być następująca:

```
version: '3'

services:

      cadvisor:
            image: google/cadvisor:latest
            container_name: cadvisor
            networks:
                  - back

      prometheus:
            image: prom/prometheus
            container_name: prometheus
            depends_on:
                  - cadvisor
            networks:
                  - back

      grafana:
```

```
            image: grafana/grafana
            container_name: grafana
            depends_on:
                  - prometheus
            networks:
                  - front
                  - back

volumes:

networks:
      front:
      back:
```

UWAGA: Ponieważ nazwy kontenerów muszą być unikalne to nie ma możliwości skalowania (tworzenia replik) kontenerów z nazwą zdefiniowaną przez użytkownika. Wobec tego instrukcja *container_name* nie może być używana w przypadku implementacji w środowiskach klastrowych, np. Swarm.

W przedstawionym powyżej szkielecie pliku *docker-compose.yml* użyta została również instrukcja *depends on*. Pozwala ona na definiowanie kolejności uruchamiania kontenerów (polecenie *docker-compose up*) oraz ich zatrzymywania (kolejność odwrotna, polecenie *docker-compose stop*)

Wykorzystanie instrukcji *depends on X* nie gwarantuje poprawności (i gotowości do działania) kontenera X a jedynie, że kontener ten został uruchomiony i można uruchomić kontener powiązany z X. Bardziej zaawansowane metody nadzoru nad procesem uruchamiania są przedstawione w dokumentacji pod adresem: https://docs.docker.com/compose/startup-order/

UWAGA: Podobnie jak poprzednio, instrukcji *depends on* nie powinno się używać w połączeniu z implementacjami usług w klastrach ponieważ w tych przypadkach kontrolę nad metodą i kolejnością startu i zatrzymywania kontenerów przejmuje zazwyczaj orkiestrator.

Mając szkielet pliku docker-compose,yml można przystąpić do opisu ustawień poszczególnych usług skladowych (kontenerów).

1. Cadvisor

Cadvisor to narzędzie opracowane przez firmę Google w celu szybkiego dostępu do informacji diagnostycznych dla środowiska Docker wraz z budowanym serwerem http pozwalającym na zdalną prezentację graficzną zebranych wyników. Narzędzie to było już wykorzystywane w instrukcji nr 4. W tym przypadku, będzie ono użyte jako źródło danych dla bardziej zaawansowanego narzadzia analitycznego jakim jest pakiet Prometheus. W związku z tym, w usłudze cadvisor należy zdefiniować (zgodnie z dokumentacją obrazu) podstawowe vulumeny typu bind-mount, które posłużą do zbierania niezbędnych danych. Ponieważ cadvisor wykorzystuje API wewnętrznego serwera do dostępu do danych (domyślnie skonfigurowane na port 8080), należy użyć instrukcji *expose* z wartością 8080. Będzie to „end-point” dla aplikacji prometheus.

UWAGA: Użycie instrukcji *expose* w pliku *docker-compose* oznacza, wskazany port będzie „wystawiony” dla wszystkich powiązanych usług (w ramach danej sieci) i nie powoduje publikowania tego portu na zewnątrz tej sieci.

Opis usługi cadvisor jest zatem następująca:

```
cadvisor:
      image: google/cadvisor:latest
      container_name: cadvisor
      volumes:
            - /:/rootfs:ro
            - /var/run:/var/run:rw
            - /sys:/sys:ro
            - /var/lib/docker/:/var/lib/docker:ro
      restart: unless-stopped
      expose:
            - 8080
      networks:
            - back
```

W powyższym opisie użyta została również instrukcja *restart* z wartością unless-stopped. Restart definiuję politykę restartu usługi. Domyślna wartość to „no”. Inne możliwe wartości to: „always” oraz „on-failure”.

UWAGA: Ponownie, instrukcja ta jest ignorowana w przypadku wykorzystania docker-compose w strukturach klastrowych. W zamian należy wykorzystywać instrukcję deploy, np. w postaci:

```
      deploy:
            restart_policy:
                  condition: unless-stopped
```

Instrukcja *deploy* jest dostępna wyłącznie w wersji 3+ i jest wykorzystywana w implementacjach klastrowych ale jest ignorowana przez polecenia *docker-compose up* , *docker-compose run*.

2. Prometheus

Dane zbierane przez cadvisor i dostępne na zdefiniowanym w nim end-poicie mają być wykorzystywane przez aplikację/usługę prometheus. Prometheus, opracowany przez firmę SoundCloud, jest jednym z najbardziej popularnych narzędzi do monitorowania, szczególnie w obszarze środowisk rozproszonych.

Konfiguracja usługi prometheus jest przechowywana w pliku YAML. Wobec tego należy stworzyć plik ./prometheus/prometheus.yml i podłączyć go do systemu plików kontenera. Minimalny plik konfiguracyjny ma następującą zawartość:

```
global:
      scrape_interval: 15s
      evaluation_interval: 15s
      external_labels:
            monitor: 'monitoring'

rule_files:

scrape_configs:
```

```
- job_name: 'prometheus'
static_configs:
      - targets: ['localhost:9090']
- job_name: 'cadvisor'
static_configs:
      - targets: ['cadvisor:8080']
```

Plik informuje, że dane będą zbierane i przetwarzane w interwałach 15 sekundowych. Dodatkowo określa dwa end-pointy, odpowiednio dla cadvisor oeaz dla prometheus. Ten ostatni będzie wykorzystywany dla dostarczania danych do usługi grafana.

Dodatkowo, zbierane dane powinny być umieszczene na zewnętrznym wolumenie (np. ze względów bezpieczeństwa czy też dalszego wykorzystania za pomocą innych narzędzi). Dla tego zadania utworzymy wolumen o nazwie: *prometheus_data* .

Końcowa postać sekcji pliku docker-compose dla usługi prometheus wygląda zatem następująco:

```
prometheus:
      image: prom/prometheus
      container_name: prometheus
      volumes:
            - ./prometheus/prometheus.yml:/etc/prometheus/prometheus.yml
            - prometheus_data:/prometheus
      restart: unless-stopped
      expose:
            - 9090
      depends_on:
            - cadvisor
      networks:
            - back
```

3. Grafana

Grafana jest narzędziem open source, które służy do prezentacji grafik z wartościami zdefiniowanych metryk pochodzących z przyłączonych źródeł. Wybór tych źródeł (backend-ów) jest dokonywany poprzez konfigurację pluginów. Natomiast wizualizacje są zorganizowane w postaci pulpitów graficznych (ang. dashboards). Konfiguracja obu tych elementów (plugins oraz dashboard) nie jest omawiana w tej instrukcji ale szczegółowe informacje i przykłady można znaleźć w dokumentacji narzędzia Grafana.

Analogicznie jak dla usługi Prometheus, należy dane wykorzystywane przez usługę umieścić na zewnętrznym wolumenie, tym razem nazwanym *grafana_data*. Jednocześnie konfiguracja pluginów/end-point-ów oraz zawartości dashboarda musi zostać „podmountowana” do systemu plików kontenera, w miejsce wynikające z dokumentacji wykorzystanego obraz grafana. W niniejszym przykładzie ustawienia te znajdują się w katalogu *./grafana/provisioning/*

Grafa wymaga zdefiniowania zmiennych środowiskowych, w minimalnym zakresie odnoszących się do polityki dostępu do ustawień aplikacji. W Docker-compose istnieją dwie podstawowe metody definiowania zmiennych środowiskowych:
- poprzez deklarację w instrukcji *environment:* ,
- poprzez wykorzystanie z pliku zewnętrznego zdeklarowanego w instrukcji *env_file:* .

UWAGA1: Wpisy w instrukcji *environment:* pokrywają deklaracje wartości zmiennych w pliku z instrukcji *env_file:* .

UWAGA2: W przypadku budowania obrazów na podstawie deklaracji w *docker-compose.yml* (mowa będzie o tym w dalszej części instrukcji), zmienne środowiskowe, niezależnie od sposobu deklaracji, nie są „widziane" przez proces *build*. W takim przypadku należy rozważyć użycie instrukcji ARG poznanej na poprzednim laboratorium.

W omawianym przykładzie stworzony został plik o nazwie grafana.config, który pełni rolę pliku definiującego niezbędne zmienne środowiskowe. Jego zawartość jest następująca:

```
GF_SECURITY_ADMIN_USER=admin
GF_SECURITY_ADMIN_PASSWORD=password
GF_USERS_ALLOW_SIGN_UP=false
```

Jak widać, zawiera on ustawienia użytkownika i hasła. Zawarcie tych danych na zewnętrznej lokalizacji jest dodatkowo wygodne, bo pozwala zmieniać te ustawienia (i dodawać inne) bez konieczności edytowania pliku *docker-compose.yml*.

Ostateczna postać opisy usługi grafana dla Docker-compose jest następująca:

```
grafana:
      image: grafana/grafana
      container_name: grafana
      volumes:
            - grafana_data:/var/lib/grafana
            - ./grafana/provisioning/:/etc/grafana/provisioning/
      env_file:
            - ./grafana/grafana.config
      restart: unless-stopped
      ports:
            - 3000:3000
      depends_on:
            - prometheus
      networks:
            - front
            - back
```

W powyższej konfiguracji użyta została instrukcja *ports*. Może być ona wykorzystywana według dwóch schematów. Format krótki polega na podaniu dwóch wartości:

```
HOST:CONTAINER
```

W przypadku braku wartości dla części HOST, compose zdefiniuje tymczasowy port. Możliwe jest też definiowanie adresów IP (v4 oraz v6 przy czym obecnie v6 nie jest wspierana w klastrach Swarm), zakresów portów oraz protokołów transportowych. Ilustrują to przykłady poniżej:

```
 - "127.0.0.1:8001:8001"
 - "127.0.0.1:5000-5010:5000-5010"
 - "6060:6060/udp"
```

UWAGA1: W przykładach powyżej, wszystkie definiowane wartości objęte są cudzysłowami. Ich użycie jest konieczne w przypadku korzystania z portów nizszych niż 60. Zaleca się jednak by zawsze, niezależnie od zawartości, stosować cudzysłowy.

UWAGA: Użycie instrukcji *network_mode: host* wyklucza możliwość wykorzystania instrukcji *ports* .

Druga metoda, tzw. długa, jest dostępna od wersji 3.2. Pozwala ona na zdefiniowanie wartości dla następujących kluczy:

- *target:* port wewnętrzny kontenera,
- *published:* port zewnętrzny (publiczny),
- *protocol:* tcp lub udp,
- *mode:* wartość host oznacza opublikowanie portu hostu na wszystkich nodach. Inna możliwa wartość to ingress która oznacza port w trybie swarm przeznaczony do wykorzystania przez mechanizmy równoważenia obciążeń (ang. load balancing).

Przykład użycia trybu długiego jest pokazany poniżej:

```
ports:
  - target: 80
    published: 8080
    protocol: tcp
    mode: host
```

4. Definiowanie infrastruktury

Na końcu pliku docker-compose.yml należy opisać wykorzystane wolumeny i sieci. W omawianym przykładzie wystarczy podać:

```
volumes:
    prometheus_data:
    grafana_data:

networks:
  front:
  back:
```

Szczegółowe informacje o bardziej złożonych opisach elementów infrastruktury usługi w postaci sieci można odnaleźć w dokumentacji plików *docker-compose*. Proszę poza sekcjami *network_mode* , *networks* oraz *volumes*, KONIECZNIE zapoznać się z sekcją *network configuration reference* https://docs.docker.com/compose/compose-file/#network-configuration-reference oraz *volume configuration reference* .

UWAGA: Wszystkie pliki powiązane z opracowanym projektem są dostępne na moodle, plik: **monitoring_v2019.tar.gz**

5. Uruchomienie usługi

Całość projektu opisanego w poprzednich podpunktach przedstawia znajduje się w dedykowanym katalogu o nazwie monitoring, będącym podkatalogiem katalogu domowego użytkownika. Drzewo katalogowe projektu jest następujące:

```
student@student-PwChO:~$ tree monitoring
monitoring
├── docker-compose.yml
├── grafana
│   ├── grafana.config
│   └── provisioning
│       ├── dashboards
│       │   ├── dashboard.yml
│       │   └── Docker Monitoring.json
│       └── datasources
│           └── datasource.yml
└── prometheus
    └── prometheus.yml

5 directories, 6 files
```

Przed uruchomieniem usługi zawsze warto sprawdzić poprawność przygotowanego pliku *docker-compose.yml*. Do tego celu służy polecenie

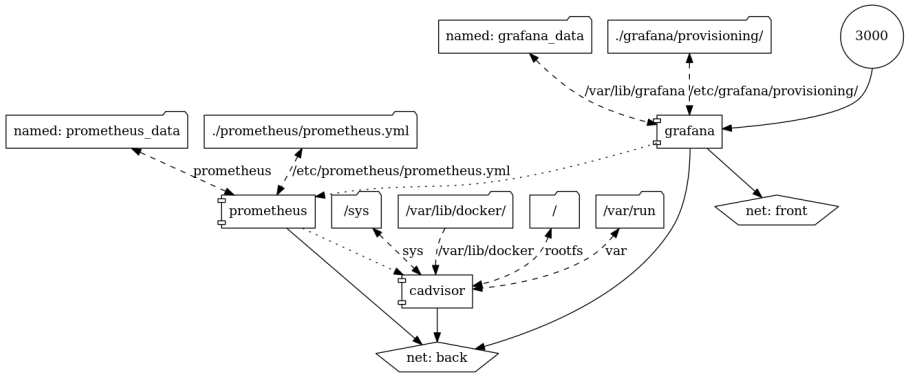
```
$ docker-compose config -q
```

Polecenie to należy wydać z poziomu katalogu, w którym umieszczony jest plik *docker-compose.yml*. Flaga *-q* oznacza, że wyświetlane mają być tylko ewentualne błędy.

Innym, przydatnym narzędziem jest *docker-compose-viz* opracowanym przez PMSIpilot. Narzędzie jest dostępne w postaci obrazu Docker i służy do graficznej reprezentacji zależności pomiędzy składowymi usługi zdefiniowanymi w danym pliku *docker-compose.yml* . Narzędzie to należy uruchomić z poziomu katalogu, w którym znajduje się wybrany plik *docker-compose.yml* poprzez wydanie polecenia:

```
$ docker container run --rm -it --name dcv -v $(pwd):/input
pmsipilot/docker-compose-viz render -m image docker-compose.yml
```

W rezultacie, w katalogu pojawi się plik docker-compose.png. Dla usługi monitoringu, opracowanej w tej części instrukcji, wygląda on następująco:

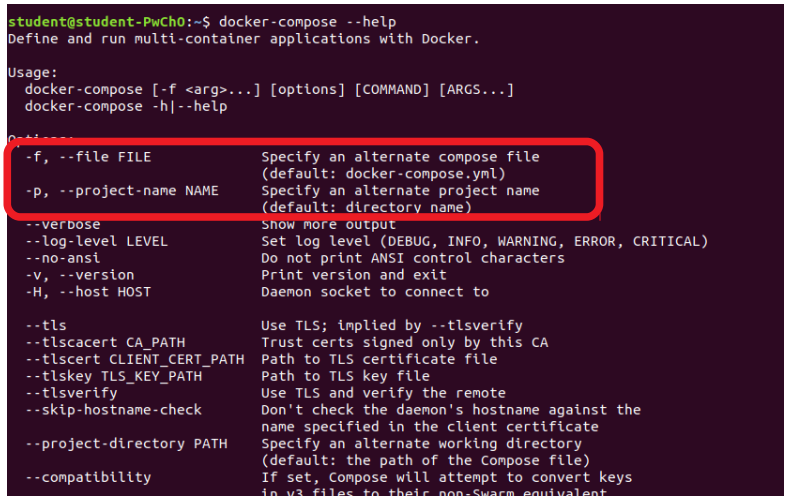


B. Środowisko docker-compose

Środowisko docker-compose posiada kilkanaście predefiniowanych poleceń, których działanie jest analogiczne do tych, jakie były już przedstawione na poprzednich laboratoriach. Pełną listę można uzyskać przez wydanie polecenia:

$ docker-compose - - help

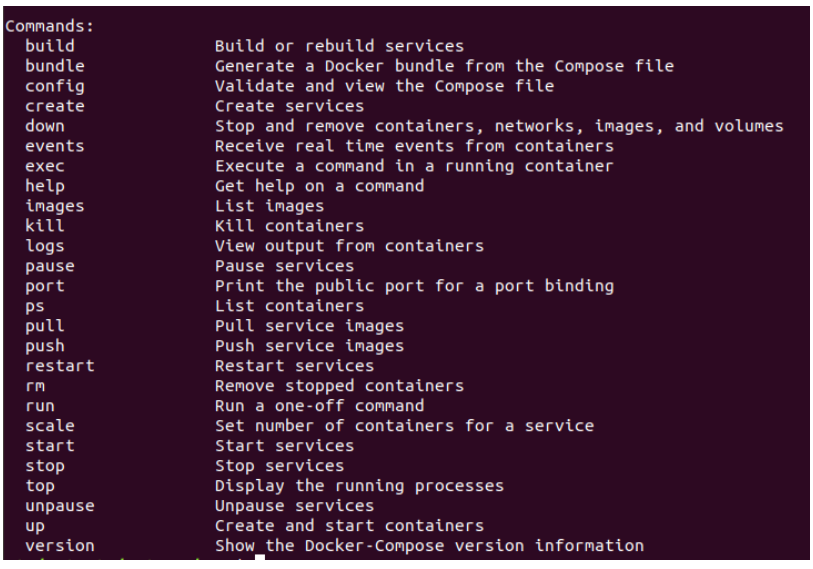
W pierwszej części wyniku działania przedstawionej komendy należy zwrócić uwagę na dwa wpisy:



Docker-compose dopuszcza użycie alternatywnych plików konfiguracyjnych (alternatywnych *docker-compose.yml*). Wykorzystuje się je w przypadku konieczności zmiany/dodania ustawień w podstawowym pliku *docker-compose.yml*. Kolejność podawania tych plików w poleceniu docker-compose decyduje o kojejności wprowadzanych zmian. Szczegóły wraz z prostymi przykładami wykorzystania plików alternatywnych można znaleźć pod adresem: https://docs.docker.com/compose/reference/overview/ .

Nazwa projektu jest przyjmowana domyślnie jak nazwa katalogu, w którym umieszczony jest plik *docker-compose.yml*. Aby tą nazwę zmienić, to przy uruchamianiu usługi należy podać flagę *-p* .

Polecenia dostępne w środowisku docker-compose są przedstawione poniżej:



UWAGA: Zdecydowaną większość z wypisanych powyżej poleceń należy wykonywać z miejsca, w którym znajduje się plik *docker-compose.yml*.

Działanie części tych poleceń można zilustrować poprzez uruchomienie opracowanej wcześniej usługi monitorowania środowiska docker.

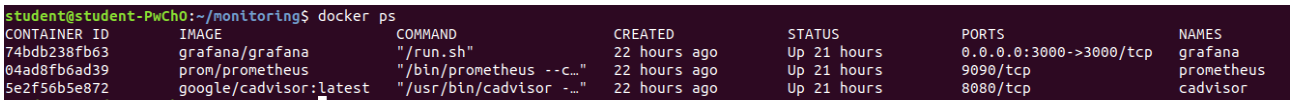
Do tej pory wykorzystaliśmy polecenie config, by sprawdzić poprawność konfiguracji usługi. Czas bu ją uruchomić. Dobrym zwyczajem jest przed uruchomieniem, poprać niezbędne obrazy za pomocą polecenia:

```
$ docker-compose pull
```

Można ten etap pominąć i uruchomić usługę w trybie „w tle”:

```
$ docker-compose up -d
```

Po pomyślnym zakończeniu uruchamiania i konfiguracji usługi monitoringu, można sprawdzić usługi składowe (zazwyczaj z kolejnej konsoli):



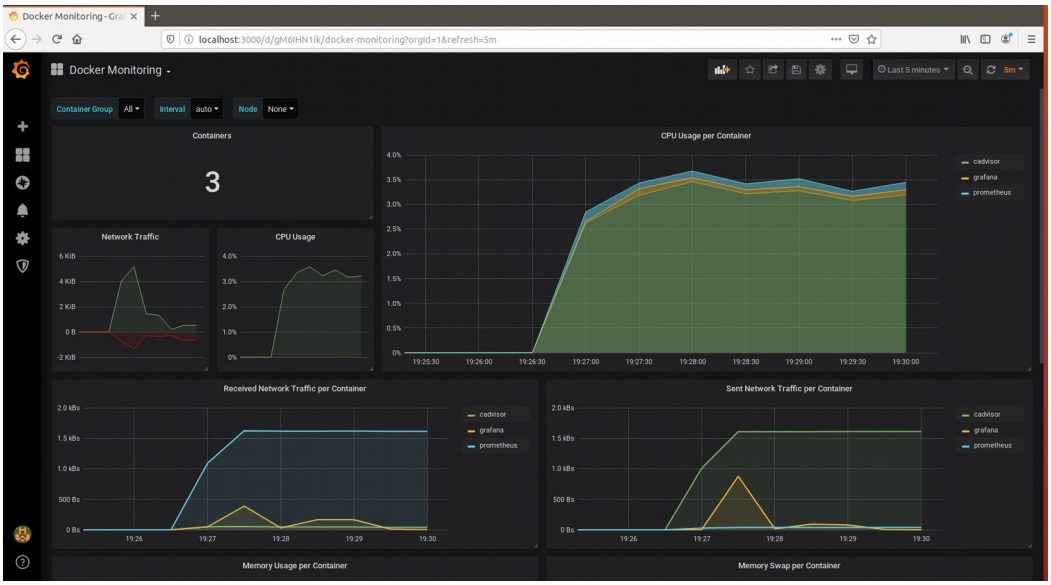
Zanim można będzie uruchomić front-end usługi, należy sprawdzić czy serwer http nasłuchuje” na porcie 3000 (zazwyczaj po uruchomieniu trzeba na to odczekać kilka, kilkanaście sekund – zależnie od wykorzystywanego sprzętu). Można tego dokonać obserwując informacje podczas uruchamiania usługi lub poprzez wydanie polecenia:

```
$ docker-compose logs -f grafana
```

Na koniec tej „szybkiej” diagnostyki, można wykorzystać polecenie:

UWAGA: Wszystkie poznane dotąd polecenia środowiska docker również mogą zostać użyte w odniesieniu do poszczególnych kontenerów, reprezentujących usługi cząstkowe.

Na koniec, należy otworzyć przeglądarkę i połączyć się z hostem lokalnym na porcie 3000. Powinno ukazać się okno logowania narzędzia grafana (user i password zostały zdefiniowane w pliku konfiguracyjnym ze zmiennymi środowiskowymi dla usługi grafana). Po zalogowaniu się powinniśmy przejść do panelu prometheus-a. W górnym, prawym rogu jest ikona do otwarcia okna dialogowego wyboru dashboardów (z etykietą Home). Wchodzimy tam i wybieramy skonfigurowany dashboard o nazwie „Docker monitoring”. Efekt tego wyboru powinien być taki jak poniżej:



Zatrzymanie usługi monitoringu następuje przez wydanie polecenia:

```
$ docker-compose stop
```

UWAGA: Alternatywnie można uzyć polecenia

```
$ docker compose kill
```

Należy jednak pamiętać, ż w tym przypadku wszystkie kontenery zostaną zatrzymane natychmiast i z pominięcie zdefiniowanych zależności. Może to spowodować utratę danych.

Po zatrzymaniu, możliwe jest usunięcie kontenerów związanych z projektem (opcja *rm*). Należy jednak pamiętać, że:
- usuwane są tylko kontenery w stanie „exited”.
- w ten sposób nie zostaną usunięte wolumeny, sieci i obrazy utworzone dla potrzeb poszczególnych usług cząstkowych.

Polecenie, które usuwa tak kontenery jak i sieci utworzone w projekcie to:

```
$ docker-compose down
```

Jeśli chcemy również usunąć kontenery to należy oddzielnie je usunąć (również dotyczy to sieci). Można też wydać polecenie:

```
$ docker-compose down --volumes --rmi all
```

W ten sposób usunięte zostaną wszystkie kontenery, wolumeny, obrazy i sieci. Pomoc dla polecenia down opisuje możliwe konfiguracje usuwania komponentów infrastruktury stworzonej w danym projekcie:

```
$ docker-compose down –help
```

C. Budowanie obrazów w połączeniu z docker-compose

Obecnie istnieje możliwość wykorzystania w środowisku doker predefiniowanego trybu *docker swarm mode*. Trym ten będzie jednym z tematów kolejnego laboratorium. W przypadku tego trybu nie trzeba instalować narzędzia *docker-compose* a można korzystać z jego plików *doker-compose.yml*. Podstawową cechą, która obecnie skłania do wykorzystania docker-compose, to możliwość równoległego budowania obrazów i uruchamiania bazujących na nich usług.

Budowa obrazów w oparciu o *docker-compose.yml* polega na zdefiniowaniu w nim kontekstu oraz opcjonalnie pliku Dockerfile oraz wartości przekazywanych przez ARG.

W najprostszym podejściu zawartość *docker-compose.yml* dla zbudowania hipotetycznej usługi cząstkowej może mieć następującą postać:

```
version: "3.7"
services:
  myidea:
    build: ./dir
```

UWAGA: Powyższy przykład zakłada, że *Dockerfile* znajduje się w podkatalogu dir katalogu głównego danego projektu. Składnia pliku *docker-compose.yml* w wrsjach 3+ pozwala również na podawanie zamiast katalogu, położenia URL repozytorium Github.

Bardziej złożona składnia dla zdefiniowania procesu build zawiera:

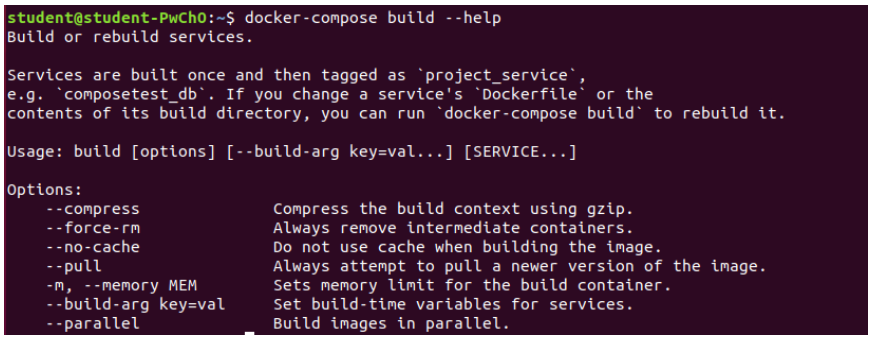
```
version: "3.7"
services:
  myidea:
    build:
      context: ./dir
      dockerfile: Dockerfile-alternate
      image: myidea:tag
      args:
        myargs: value
```

W powyższym przykładzie wykorzystano możliwość zdefiniowania opcjonalnego, alternatywnego pliku Dockerfile. Dodatkowo zdefiniowania nazwę obrazu (wraz z tag-iem) jaki ma powstać w wyniku procesu *build*. Zdefiniowano również wartość zmiennych dla instrukcji ARG. Należy pamiętać (z poprzedniej instrukcji), że wartości te są „widziane” wyłącznie przez proces *build*. Pełna dokumentacja jest dostępna pod adresem: https://docs.docker.com/compose/compose-file/

Budowanie obrazów odbywa się poprzez polecenie:

```
$ docker-compose up
```

W środowisku docker-compose istnieje również dedykowane polecenie *docker-compose build*. Jego składnia jest opisana w podręczniku systemowym



lub pod adresem: https://docs.docker.com/compose/reference/build/

Polecenie *build* jest głównie wtedy, gdy dokonano jakichkolwiek zmian w plikach Dockerfile (i ich otoczeniu). W wyniku jego działania powstają nowe wersje obrazów usług składowych.

UWAGA: Należy pamiętać, że wydanie poleceń *build* oraz *pull* w środowisku docker-compose nie prowadzi automatycznie do skonfigurowania na nowo usługi głównej (projektu). W tym celu należy wykorzystać polecenie *docker-compose create* lub opcje polecenia *docker-compose up –no-start* z dalszymi opcjami. Należy zapoznać się z podręcznikiem systemowym dla polecenia *docker-compose create* i *docker-compose up* lub z ich dokumentacją, odpowiednio strony: https://docs.docker.com/compose/reference/create/ oraz https://docs.docker.com/compose/reference/up/

Dla samodzielnego zapoznania się z prosem budowania obrazów w środowisku docker-compose ZDECYDOWANIE dobrym przykładem jest aplikacja do głosowania, opracowana na potrzeby dokumentacji środowiska Docker. Jej opis i wszystkie elementy składowe są dostępne pod adresem: https://github.com/dockersamples/example-voting-app

UWAGA: Poznanie zasad budowy i uruchamiania wskazanej wyżej aplikacji jest dodatkowo przydatne dla tych, którzy używają środowisk uruchomieniowych opartych o systemy Windows lub OSX. Jednocześnie aplikacja ta będzie wykorzystywana na kolejnym laboratorium.

Z61. Na poprzednich laboratoriach budowaliśmy i uruchamialiśmy kontenery z serwerem Apache, PHP. W tym zadaniu należy zbudować prosty plik docker-compose.yml, który pozwoli na uruchomienie znanej z innych zajęć, uslugi LAMP. Usługa ta składa się z trzech usług składowych:
- serwera Apacje (można wykorzystać np. obraz serwera httpd na bazie alpine)
- serwera PHP (można wykorzystać np. oficjalny obraz latest)
- bazy danych MySQL (można wykorzystać oficjalny obraz latest)
Założenia dla uslugi:
- serwery są budowane zgodnie z dokumentacją obrazów bazowych dostępnych na DockerHub i umieszczonych tam plików Dockerfile (nie korzysta się z gotowych obrazów),
- serwery PHP i MySQL są przyłączone do sieci backend a Apache do backend oraz frontend. Apache ma wystawiony na świat zewnętrzy port 6666.

W sprawozdaniu należy:
- podać zawartość plików Dockerfile dla każdej usługi cząstkowej,

- podać zawartość innych, niezbędnych plików,
- podać zawartość pliku docker-compose.yml,
- utworzyć i podać zawartość pliku README.md opisujący stworzony projekt

Opcjonalnie: sprawdzić czy jest możliwe wygenerowanie reprezentacji graficznej dla utworzonego pliku docker-compose.yml i jeśli tak, to umieścić go w sprawozdaniu.

Cały projekt może być też umieszczony w repozytorium na GitHub-ie. W takim przypadku proszę w sprawozdaniu podać link do tego repozytorium.