

PODSTAWY INŻYNIERII OPROGRAMOWANIA



Ogólne informacje na temat laboratorium

Prowadzący laboratoria
mgr inż. Konrad Miśkiewicz
konrad@iis.p.lodz.pl
konrad.miskiewicz@p.lodz.pl

Docker

Źródła www:

1. <https://www.docker.com/>
2. <https://www.uml-diagrams.org/component-diagrams.html>
3. <http://mmorejon.github.io/en/blog/deployment-diagram-docker-django/>

Czym jest Docker?

Docker jest platformą służącą tworzeniu, dostarczaniu oraz uruchamianiu aplikacji w zwirtualizowanym środowisku kontenerów.

Komponenty Docker:

1. Docker Engine
2. Docker Hub
3. Docker Machine
4. Docker Swarm
5. Docker Compose
6. niezależne narzędzia (otwarte API)

Problemy jakie rozwiązuje Docker:

- a) powolne wdrażanie
- b) olbrzymie koszty
- c) zmarnowane zasoby
- d) trudne w skalowaniu
- e) trudne w migracji

Zalety rozwiązania Docker:

1. Jedną fizyczną maszynę można podzielić na wiele wirtualnych
2. Lepsze zarządzanie zasobami
3. Łatwiejsze w skalowaniu (łatwo dodać nowe kontenery i hosty (Docker Engines) jeśli zajdzie taka potrzeba)
4. Nowy model opłat w cloud: pay-as-you-go
5. Podział odpowiedzialności (deweloperzy budują aplikację, operatorzy budują środowisko w ramach wspólnego narzędzia (Infrastructure as Code))
6. Szybki cykl wytwarzania: wszystko w jednym miejscu, brak problemów z kompatybilnością między środowiskami, brak problemów z zależnościami

Wady korzystania z wirtualizacji np. VirtualBox, VMWare Workstation:

1. Każda VM nadal wymaga alokacji zasobów - CPU - RAM - dysk - system operacyjny
2. Im więcej VM, tym więcej potrzebnych zasobów
3. Wiele systemów (Guest OS) to wiele zmarnowanych zasobów

W celu optymalnego wykorzystania zasobów twórca Dockera Solomon Hykes stworzył nowe rozwiązanie oparte na kontenerach. Z początku Docker był wewnętrznym projektem rozwijanym w firmie dotCloud. W marcu 2013 Docker został udostępniony publicznie. Od tego czasu zyskuje coraz większą popularność: profil na portalu [GitHub](#) został oznaczony

gwiazdką ponad 35 tysięcy razy, a członkowie społeczności dokonali ponad 10 tysięcy forków projektu.

Wirtualizacja oparta o kontenery używa kernela systemu operacyjnego hosta do uruchamiania wielu instancji aplikacji gości. Każda instancja gościa jest nazywana kontenerem. Każdy kontener posiada:

1. system plików (rootfs)
2. procesy
3. pamięć
4. urządzenia
5. porty sieciowe

Kontenery vs. VM

1. Kontenery są lżejsze
2. Brak konieczności instalacji systemu operacyjnego (Guest OS)
3. Mniejsze wymagania względem zasobów (CPU/RAM)
4. Większa przenośność

Terminologia

Serwer Dockera (Docker Machine)

uruchomiony w trybie daemona. Zamienia komputer lokalny lub zewnętrzny w serwer Dockera na którym można uruchamiać, budować i wyłączać kontenery za pomocą zdalnego klienta.

Klient Dockera (Docker Client)

używany do kontrolowania oraz komunikacji ze środowiskami dockera. Po skonfigurowaniu Serwera możemy go kontrolować Klientem

Kontener Dockera (Docker container)

jest izolowaną platformą uruchomieniową dla aplikacji. Zawiera system plików, biblioteki, zależności, które są potrzebne do uruchomienia aplikacji

Obraz Dockera (Docker Image)

Obrazy możemy interpretować jako szablony do tworzenia kontenera. Można utworzyć własny obraz lub wykorzystać jeden z dostępnych na Docker Hub

Wolumen (Volumen)

Wolumen to określony katalog w kontenerze, który zostaje przeznaczony na przechowywanie danych trwałych np. kodu aplikacji, niezależnie od cyklu życia kontenera. Wolumeny nie są częścią obrazu. Są nadal utrzymywane, jeśli obrazy zostaną skasowane. Mogą być zmapowane w wielu kontenerach (współdzielone).

Dockerfile

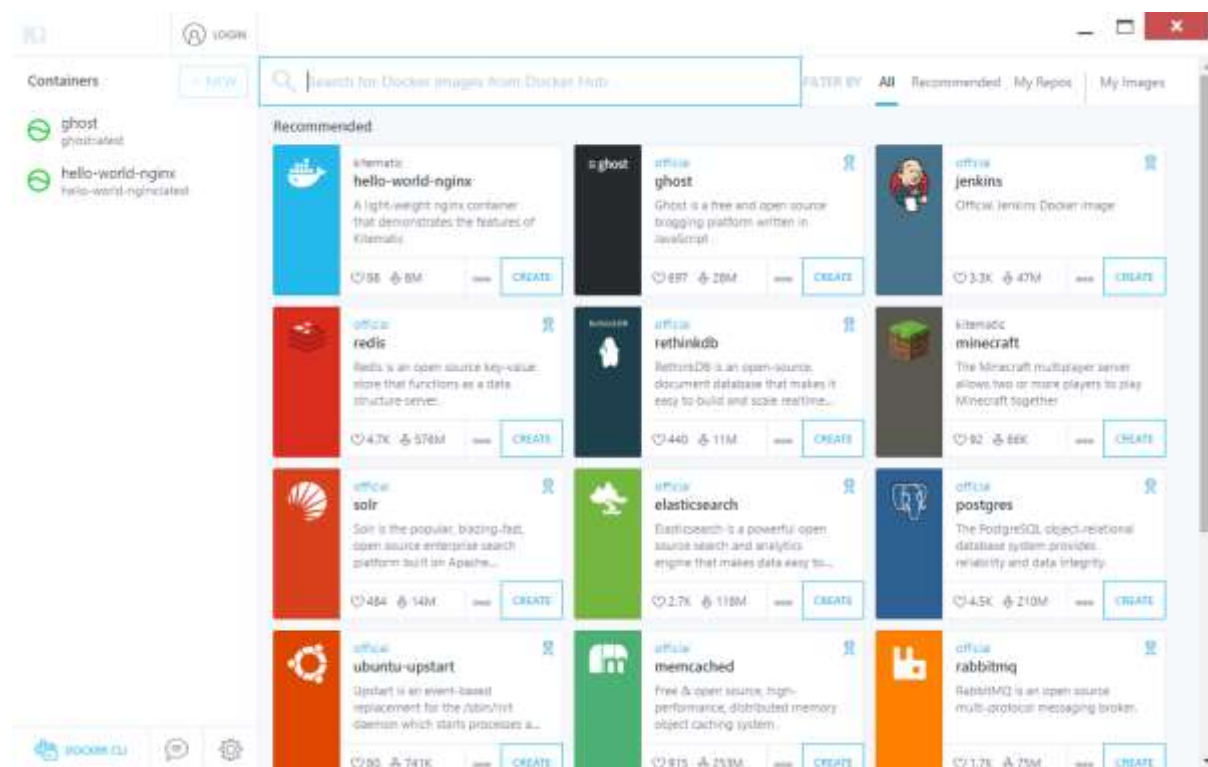
to plik konfiguracyjny zawierający instrukcje potrzebne do budowy obrazu. W sposób programowy i powtarzalny tworzy własny obraz. Instrukcje Dockerfile to kroki wykonywane w celu zbudowania obrazu.

Narzędzia Dockera

Kitematic

Jest graficznym interfejsem Dockera dostarczającym maszyny wirtualne. Pozwala zarządzać Obrazami i Kontenerami.

<https://kitematic.com/>



Docker Hub

Repozytorium obrazów, które można uruchomić z Dockera.

Docker Client (Obsługa obrazów i kontenerów na serwerze)

Podstawowe komendy wykorzystywane w Docker Client:

Ściągnięcie obrazu:

```
docker pull [nazwa obrazu]
```

Uruchomienie obrazu:

```
docker run [nazwa obrazu]
```

(w przypadku gdy obrazu nie ma na komputerze, Docker postara się go ściągnąć z Docker Hub.)

np uruchomienie serwera Tomcat: `docker run -d -P tomcat:7`

Wyświetla listę dostępnych obrazów:

```
docker images
```

Uruchomienie systemu ubuntu:

Składnia:

```
docker run [opcje] [obraz] [komenda] [argumenty]
```

Przykład uruchomienie konsoli Ubuntu:

```
docker run -i -t ubuntu:latest /bin/bash
exit
```

Startowanie i zatrzymywanie:

Wyświetla uruchomione Kontenery:

```
docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
961d1bc30aa9	ghost:latest	"docker-entrypoint.s..."	About an hour ago	Up About an hour	0.0.0.0:32769->2368/tcp	ghost

a. Znajdź wszystkie (uruchomione i zatrzymane) Kontenery po ID lub nazwie:

```
docker ps -a
```

b. `docker start [nazwa kontenera]`

c. `docker stop [nazwa kontenera]`

Usuwanie zatrzymanych kontenerów:

```
docker rm [nazwa kontenera]
```

Ćwiczenia

Ćwiczenie 1. Wykonaj następujące komendy w celu uruchomienia aplikacji testowej:

Ściągnij aplikację z Dock Hub:

```
docker pull hello-world
```

Sprawdź czy obraz został ściągnięty poprawnie, w tym celu wyświetl wszystkie obrazy dostępne na Serwerze. W celu utworzenia Kontenera z obrazu hello-world wykonaj:

```
docker run hello-world
```

przeczytaj informację, która została wyświetlona

Wyświetl wszystkie kontenery:

```
docker ps -a
```

Usuń niepotrzebne kontenery:

```
docker rm [container id]
```

Wyświetl dostępne obrazy:

```
docker images
```

Usuń niepotrzebne obrazy:

```
docker rm [image id]
```

Ćwiczenie 2. Uruchamianie prostego serwera ze statyczną stroną internetową

Strona którą zamierzamy uruchomić będzie pobrana z przygotowanego wcześniej obrazu. Dzięki temu w szybki sposób postawimy nasz serwer webowy, który w dalszej części będziemy mogli wykorzystać, a jednocześnie zmodyfikować. Strona która będzie wyświetlana jest statyczną pojedynczą stroną.

Nasz obraz dotyczący pojedynczej strony: **dockersamples/static-site**

Każdy z obrazów ma różne pola które podczas uruchamiania kontenera możemy ustawić. I tak uruchommy nasz kontener, gdzie przypiszemy mu nazwę: ioweb, w zmiennej AUTHOR wpisujemy swój numer indeksu, oraz uruchamiamy go w trybie „ukrytym” – detach mode.

Przykład:

```
docker run --name [nazwa kontenera] -e AUTHOR="[numer indeksu]" --detach -P [nazwa obrazu]
```

W celu skontrolowania na jakim porcie jest włączona nasza strona używamy polecenia:

```
docker port [nazwa kontenera]
```

Dzięki temu w przeglądarce możemy wpisać adres tak by można było uruchomić naszą stronę:

localhost:[port odczytany z polecenia wyżej]

Jeśli jednak chcielibyśmy ustawić naszą stronę na innym porcie to możemy zrobić tzw. mapowanie portów. W tym celu najpierw zatrzymaj uruchomiony kontener ze statyczną stroną oraz usuń go. Następnie tym samym poleceniem którym uruchamialiśmy nasz serwer webowy, uruchamiamy kontener raz jeszcze z tą różnicą że po podaniu autora wpisujemy **--p 8080:80**. Opcja ta pozwoli nam na to że w przeglądarce będziemy wpisywali localhost:[port] i nie będzie to domyślny port który został losowo przypisany, tylko ten który ustawiliśmy czyli 8080.

Zweryfikuj czy strona działa na ustawionym porcie.

Po zakończeniu tego ćwiczenia usuwamy nasz kontener i obraz.

Ćwiczenie 3. Montowanie własnego kodu źródłowego do kontenera.

Wolumen (Volumen)

Wolumen to określony katalog w kontenerze, który zostaje przeznaczony na przechowywanie danych trwałych np. kodu aplikacji, niezależnie od cyklu życia kontenera. Wolumeny nie są częścią obrazu. Są nadal utrzymywane, jeśli obrazy zostaną skasowane. Mogą być zmapowane w wielu kontenerach (współdzielone).

Montowanie na przykładzie Frameworka Express opartego na Node.js

(<https://expressjs.com/>).

W celu wykonania ćwiczenia wymagane jest zainstalowanie Node.JS oraz NPM z linku:

<https://nodejs.org/en/>

Uruchom Express lokalnie, w tym celu wykonaj kolejno poniższe komendy:

```
npm install express-generator -g
express ExpressSite --hbs
cd ExpressSite
npm install
```

w celu sprawdzenia czy aplikacja działa bez wirtualizacji*:

```
npm start
```

wejdź na adres w przeglądarce internetowej: localhost:3000

Uruchomienie kontenera

```
docker run -p 8080:3000 -v /var/www node
```

Wyświetl wszystkie kontenery:

```
docker ps -a
```

Wyświetl informacje na temat kontenera z zainstalowanym Express:

```
docker inspect [container id]
```

Usuń kontener razem z woluminem:

```
docker rm -v [container id]
```

przejdź do lokalizacji gdzie został ściągnięty framework Express i wylistuj pliki

```
cd ExpressSite
```

```
ls
```

Uruchom wolumen z framework Express bezpośrednio z podanej lokalizacji, przypnij go do kontenera node i uruchom

```
docker run -p 8080:3000 -v C:\Users\mada\ExpressSite:/var/www node npm start
```

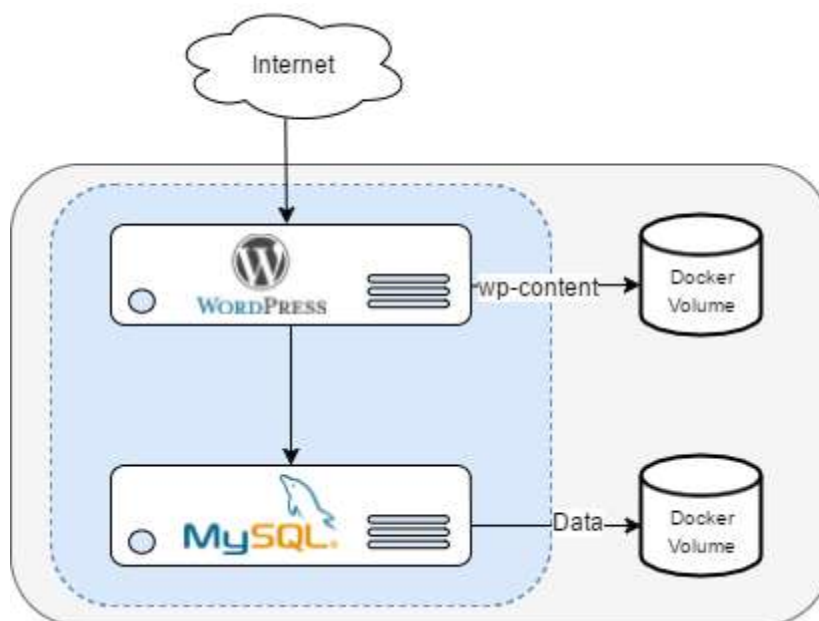
dodanie możliwości natychmiastowej edycji uruchomionego kodu:

```
run -p 8080:3000 -v C:\Users\mada\ExpressSite:/var/www -w"/var/www" node npm start
```

Wejdź do lokalizacji np. C:\Users\mada\ExpressSite i edytuj stronę views/index.hbs.

Następnie wejdź przez przeglądarkę internetową na adres localhost:8080 i odśwież stronę.

Ćwiczenie 4. Wykonanie prostej usługi webowej



Tak jak na załączonym przykładzie skonfiguruj dockera i jego kontenery tak by mieć osobno kontener dla wordpresa oraz dla bazy danej. Wszelkie informacje są dostępne poniżej.

Obraz kontenera wordpresa dostępny jest tutaj: <https://hub.docker.com/r/bitnami/wordpress-nginx/>

Obraz kontenera bazy danych: https://hub.docker.com/_/mysql/

Ansible + Vagrant

Źródła www

1. <https://docs.ansible.com/>
2. <https://www.vagrantup.com/docs/index.html>

Czym jest ansible?

Ansible to silnik za pomocą którego możemy w łatwy sposób zautomatyzować proces tworzenia kontenerów i maszyn wirtualnych, a zwłaszcza elementów na nich instalowanych. Platforma ta jest bez agentowa, czyli oznacza to że do jej działania nie potrzebna jest żaden dodatkowy program zainstalowany na maszynie zarządzanej. Pozwala to na używanie Ansible ad-hoc. Domyślnie platforma ta ma tryb push czyli wypychanie zmian z zarządcy (administratora) do urządzeń/maszyn zarządzanych. Za pomocą playbooków łączymy się z odpowiednimi maszynami po ssh i możemy wgrać elementy, które klient lub użytkownik potrzebuje na już a całość jest wykonywana zdalnie.

Czym jest vagrant?

Jest narzędziem do automatycznej konfiguracji środowiska programistycznego w postaci maszyn wirtualnych. Od VirtualBoxa różni się szybkością w ustawianiu nowych instancji wirtualnych oraz sprawia że pewne rzeczy można konfigurować od razu. Samo narzędzie jest proste, a dokumentacja techniczna jest dobrze napisana.

Przygotowanie środowiska pracy

Tworzymy sobie katalog w którym będziemy przechowywać nasze maszyny, może to być folder na pulpicie. Z Menu Start wybieramy cmd oraz przechodzimy do wcześniej utworzonego katalogu. Następnym krokiem jest inicjalizacja Vagranta, w tym celu wpisujemy:

```
vagrant init
```

Po wykonaniu tego polecenia utworzony zostanie przykładowy plik z konfiguracją dla vagranta. Otwieramy go za pomocą notatnika lub innego programu do edycji tekstowej (preferowany Visual Studio Code). Po otwarciu pliku dla naszych potrzeb zostawiamy tylko poniższe linie kodu, lub je po prostu dodajemy:

```
vagrant.configure(2) do |config|  
end
```

teraz kolejnym korkiem będzie skonfigurowanie maszyn wirtualnych. W tym celu należy w pliku konfiguracyjnym wewnątrz pozostawionych wcześniej linii wpisać ustawienia dla maszyny wirtualnej:

```
config.vm.define "nazwa_maszyny" do |nazwa_maszyny|  
  nazwa_maszyny.vm.box = "ubuntu/trusty64"  
  nazwa_maszyny.vm.hostname = "nazwa_maszyny"
```

```
    nazwa_maszyny.vm.network "private_network", ip: "192.168.33.10"
end
```

Ćwiczenie 1: Nawiązując do powyższego przykładu stwórz 3 maszyny (jeden o nazwie serwer a dwa pozostałe jako web i database) z adresami ip z sieci 192.168.10.0/24.

Po stworzeniu pliku konfiguracyjnego czas na uruchomienie naszych maszyn wirtualnych. W tym celu należy wywołać komendę:

```
vagrant up
```

W celu zweryfikowania czy wszystkie maszyny działają, używamy do tego celu polecenia

```
vboxmanage list runningvms
```

W celu podłączenia się do maszyny i jej konfiguracji wywołujemy polecenie

```
vagrant ssh nazwa_maszyny
```

Natomiast w celu zainstalowania platformy ansible po zalogowaniu się do odpowiedniej maszyny serwerowej wywołujemy komendę

```
sudo apt install ansible sshpass.
```

Ćwiczenie 2: Zaloguj się na maszynę serwer i zainstaluj platformę ansible.

Ćwiczenie 3: Testowanie działania ansible.

W pierwszej kolejności na zalogowanym serwerze tworzymy sobie katalog w którym będziemy umieszczali nasze konfiguracje, oraz przechodzimy do niego. Wewnątrz tworzymy plik **inventory**, do którego wpisujemy nasze adresy IP dla maszyny **web i database**, czyli pojawią się dwa adresy IP jeden pod drugim.

Teraz zapisujemy całość i wychodzimy z edycji pliku.

W celu zweryfikowania połączeń, czy ansible działa i komunikuje się z pozostałymi maszynami uruchamiamy polecenie:

```
Ansible [adres IP maszyny którą chcemy sprawdzić] -i inventory -u vagrant
-m ping -k
-i -> mówimy gdzie znajduje się plik inventory
-u -> wymusza logowanie użytkownika po SSH
-m -> wybierz moduł który masz zastosować
-k -> czeka na hasło które jest linkowane z SSH
```

(Standardowy login i hasło to vagrant)

Niestety po uruchomieniu powyższego polecenia wyrzuca błąd że nie znaleziono takiego klucza pasującego do podanego SSH. W tym celu aby uaktywnić całość musimy wpierw połączyć się na maszynie serwer po SSH z maszyną web:

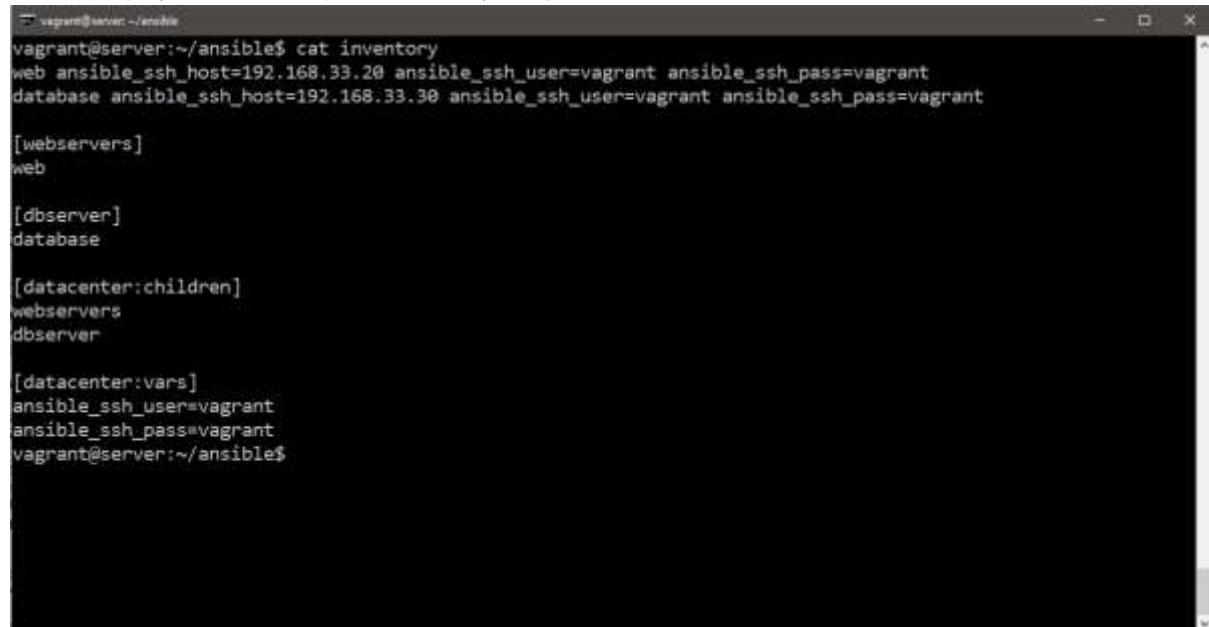
Ssh vagrant@192.168.33.20,

zatwierdzamy pytanie czy zapisać dany klucz i wpisujemy hasło vagrant. Po tej operacji wychodzimy z połączenia poleceniem exit i wywołujemy raz jeszcze polecenie testujące czy ansible zadziałało.

Po zweryfikowaniu zrób to samo dla maszyny database.

Ćwiczenie 4: Zdalne instalowanie oprogramowania

Rozbudujmy teraz nasz plik inventory do postaci:



```
vagrant@server: ~/ansible
vagrant@server:~/ansible$ cat inventory
web ansible_ssh_host=192.168.33.20 ansible_ssh_user=vagrant ansible_ssh_pass=vagrant
database ansible_ssh_host=192.168.33.30 ansible_ssh_user=vagrant ansible_ssh_pass=vagrant

[webservers]
web

[dbserver]
database

[datacenter:children]
webservers
dbserver

[datacenter:vars]
ansible_ssh_user=vagrant
ansible_ssh_pass=vagrant
vagrant@server:~/ansible$
```

Oraz utwórzmy plik ansible.cfg w którego wnętrzu wpisujemy następujące linie:

```
[defaults]
Hostfile=inventory
```

Dzięki tym plikom mamy trochę lepiej opisane zależności dla ansible. Plik konfiguracyjny ansible.cfg mówi programowi z jakiego pliku ma korzystać jeśli chodzi o dostęp do maszyn na których ma wykonywać tzw. Taski (instalować, weryfikować itd.). W pliku inventory wcześniej mieliśmy wpisane tylko adresy IP naszych maszyn. Aktualnie jest on już bardziej rozbudowany bo oprócz tego że przypisaliśmy każdemu nazwę to jednocześnie podaliśmy login i hasło, oraz pogrupowaliśmy je tematycznie, lub na to jaką pełnią rolę. I tak jak widać wyróżniamy trzy grupy, webserver, dbserver i datacenter. Jest to bardzo pomocne, gdyż w przypadku kiedy tych serwerów lub maszyn mamy więcej, a każda pełni inną rolę w łatwy sposób możemy powiedzieć że maszyny posiadające bazy danych mają zaktualizować bazę z wersji jednej do drugiej lub zainstalować dodatkowe pakiety czy programy. Jak dokonać szybkiej instalacji potrzebnych rzeczy na danej maszynie? W prosty sposób. Stwórzmy nowy plik o nazwie instalacja.yaml a w jego środku wpisujemy poniższy kod:

```
---
- hosts: webservers
  Sudo: yes
  Tasks:
    - Name: Ensure that Apache is installed
```

```
Apt: name=apache2 state=present
-name: Start Apache Service
Service: name=apache2 enabled=yes state=started
```

Po zapisaniu pliku, należy uruchomić go poleceniem

```
ansible-playbook instalacja.yaml,
```

a w wyniku otrzymamy kolejne informacje o tym czy każdy z tasków został wykonany poprawnie, na danej maszynie.

W celu przeciwiczenia, wykonaj powyższy przykład dla maszyny database tak by zainstalować na nim serwer mysql