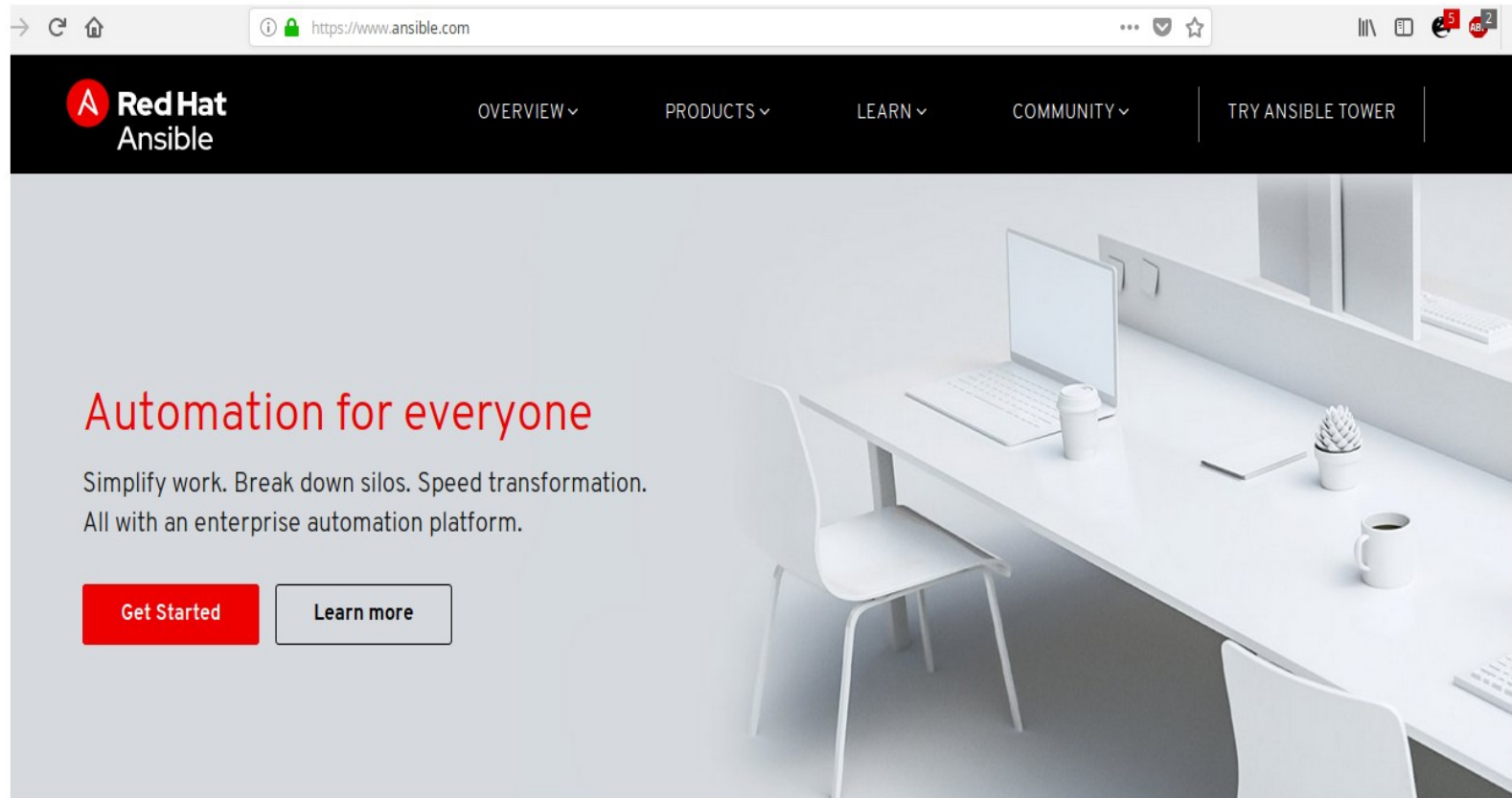


# Ansible

Marcin Wiktorowicz

# www.ansible.com



# ANSIBLE FOR EVERYONE

Ansible opiera się na zrozumiałym i prostym języku YAML

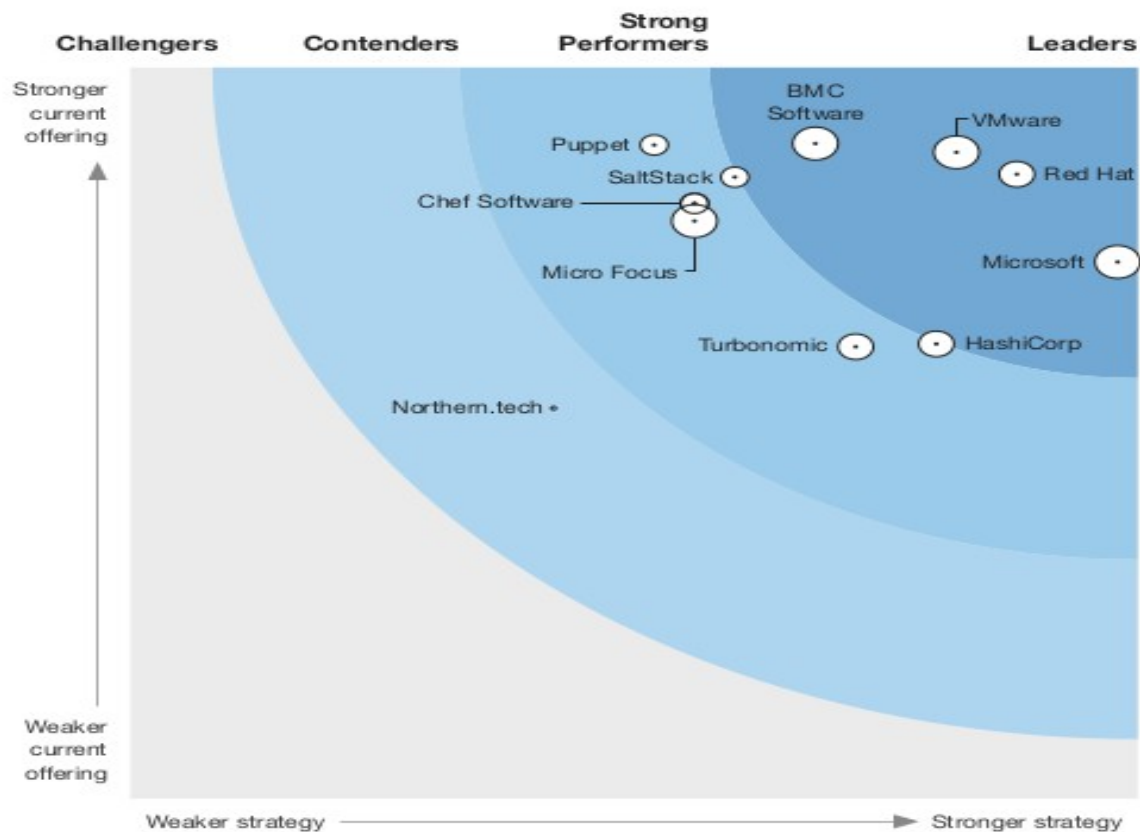
YAML Ain't Markup Language (tzw. Human-readable: prosty w odczycie, zrozumieniu i modyfikacji )

- Używanie ansible nie wymaga specjalnych umiejętności programowania
- Prosta instalacja – ansible dostępny jest w repozytoriach wszystkich dystrybucji lub na Githubie
- Od razu po instalacji ansible-engine mamy dostęp do bogatej biblioteki modułów – aktualnie ponad 3000
- Ansible Galaxy – rozwijane przez społeczność ogromne repozytorium gotowych rozwiązań do zastosowania przy niewielkiej zmianie konfiguracji dostępne online pod adresem <https://galaxy.ansible.com/>

# Raport Forrester Wave - 3 kwartał 2019r.

FIGURE 1 Forrester Wave™: Infrastructure Automation Platforms, Q3 2019

## THE FORRESTER WAVE™ Infrastructure Automation Platforms Q3 2019



# ANSIBLE IS... SIMPLE

- Prosty w odczycie, zrozumieniu i modyfikacji język YAML
- Łatwy w użyciu nawet dla osób nie mających wiele wspólnego z programowaniem
- Zadania wykonywane są jedno po drugim, zgodnie z kolejnością
- Szybkie wdrożenie rozwiązania w środowiskach produkcyjnych

# ANSIBLE IS... POWERFUL

- Szybki we wdrożeniach aplikacji
- Doskonale się sprawdza w automatyzacji konfiguracji serwerów, routerów itp.
- Wydajny w orkiestracji cyklu pracy aplikacji

# ANSIBLE IS... AGENTLESS

- Ansible nie potrzebuje instalacji dodatkowego oprogramowania do komunikacji z zarządzanymi systemami
- Ansible może być użyty do komunikacji z każdym systemem posiadającym ssh lub WinRM (rodzina systemów Windows)
- Przez to jest bezpieczniejszy – brak agentów do eksploatacji lub konieczności updatu agentów

# Ansible - instalacja

- Ansible dostępny jest w większości dystrybucji w podstawowych repozytoriach, w **CentOS v8** występuje w repozytorium epel-release
- Instalacja Ansible:
  - SUSE Linux:  
zypper install ansible
  - Debian:  
apt install ansible
- Instalacja **Ansible** w systemie **CentOS**:
  1. yum install -y epel-release
  2. yum search ansible
  3. yum info ansible
  4. yum -y install ansible



# Ansible - instalacja

- Ansible (w najnowszej wersji developerskiej – aktualnie 2.10.0.dev0) dostępny jest także poprzez serwis github pod adresem:

<https://github.com/ansible/ansible>

Instalacja:

1. yum install -y git-core
  2. git clone <https://github.com/ansible/ansible.git>
  3. cd ansible
  4. pip3 install packaging
  5. make
  6. make install
- Instalacja za pomocą menedżera instalacji pakietów python3:
    1. yum install -y python3-pip
    2. pip3 search ansible
    3. pip3 install ansible

## Ansible - moduły

- Ansible v. 2.0 – 500+ modułów  
m.in. Linux management, cloud provisioning, application deployment,  
Windows management
- Ansible v.2.8 – 3000+ modułów  
doszły m.in. Network, storage, security i wiele innych
- Lista dostępnych modułów możemy sprawdzić poleceniem:  
**ansible-doc -l**
- Moduły są wbudowane w ansible engine, dostępne  
wraz z instalacją samego ansible

# Ansible - konfiguracja zarządzanych hostów

- Domyślnym sposobem komunikacji hosta zarządzającego Ansible z hostami zarządzanymi odbywa się poprzez protokół ssh.
- Generowanie kluczy ssh niezbędnych do poprawnej komunikacji:
  1. ssh-keygen
  2. ssh-copy-id [altkom@192.168.10.151](#)
  3. weryfikacja (logowanie powinno przebiegać bez podawania hasła użytkownika):  
ssh [altkom@192.168.10.151](#)
  4. Weryfikacji można dokonać za pomocą modułu 'ping':  
ansible hosts -i inventory.cfg -m ping

# Ansible - wersja oprogramowania

- Aby sprawdzić wersję Ansible, na której pracujemy :

- **ansible-playbook --version**
- **ansible --version**

- Przykładowy wynik:

```
ansible --version
```

```
ansible 2.8.5
```

```
config file = /etc/ansible/ansible.cfg
```

```
configured module search path = ['/root/.ansible/plugins/modules', '/usr/share/ansible/plugins/modules']
```

```
ansible python module location = /usr/lib/python3.6/site-packages/ansible
```

```
executable location = /usr/bin/ansible
```

```
python version = 3.6.8 (default, Nov 21 2019, 19:31:34) [GCC 8.3.1 20190507 (Red Hat 8.3.1-4)]
```

```
[root@server1 ~]# ansible-playbook --version
```

```
ansible-playbook 2.8.5
```

```
config file = /etc/ansible/ansible.cfg
```

```
configured module search path = ['/root/.ansible/plugins/modules', '/usr/share/ansible/plugins/modules']
```

```
ansible python module location = /usr/lib/python3.6/site-packages/ansible
```

```
executable location = /usr/bin/ansible-playbook
```

```
python version = 3.6.8 (default, Nov 21 2019, 19:31:34) [GCC 8.3.1 20190507 (Red Hat 8.3.1-4)]
```

# Ansible - główne składowe

- **ansible**, **ansible-playbook** oraz **ansible-galaxy** to niezależne programy
  - **ansible-playbook** – służy do wykonywania playbookow (uschematyzowanych list zadań)
  - **ansible** – służy do wykonywania pojedynczych zadań 'ad-hoc'
  - **ansible-galaxy** – zarządza rolami (tworzy role, pobiera z repozytorium [galaxy.ansible.com](https://galaxy.ansible.com) itp.)

## Ansible - plik konfiguracyjny

- Plik konfiguracyjny ansible może być zlokalizowany w wielu miejscach:
  - **ANSIBLE\_CONFIG** – określa to zmienna środowiskowa
  - **ansible.cfg** – umiejscowiony w głównym katalogu
  - **~/ansible.cfg** – umiejscowiony w katalogu domowym użytkownika
  - **/etc/ansible/ansible.cfg** – domyślny, podstawowy plik konfiguracyjny

Plik konfiguracyjny (wraz z poszczególnymi parametrami) może być nadpisany w playbooku lub poprzez ustawienie odpowiedniej flagi w poleceniu 'ad-hoc'

# Ansible - plik konfiguracyjny

```
# config file for ansible -- https://ansible.com/
# =====

# nearly all parameters can be overridden in ansible-playbook
# or with command line flags. ansible will read ANSIBLE_CONFIG,
# ansible.cfg in the current working directory, .ansible.cfg in
# the home directory or /etc/ansible/ansible.cfg, whichever it
# finds first

[defaults]

# some basic default values...

#inventory      = /etc/ansible/hosts
#library        = /usr/share/my_modules/
#module_utils   = /usr/share/my_module_utils/
...
```

## Ansible - plik konfiguracyjny

- Najważniejsze opcje w pliku konfiguracyjnym Ansible:
  - sekcja **[defaults]** zawiera główne ustawienia
  - **inventory** – ścieżka do pliku Inventory
  - **remote\_user** – nazwa usera, na którego Ansible łączy się z zarządzanymi hostami
  - **ask\_pass** – opcja połączenia z nodami za pomocą hasła.  
Jeśli Ansible łączy się z nodami za pomocą klucza ssh, ustawiony jest na False. ask\_pass = True – pytanie o hasło (zwyczajowo root)



# Ansible - plik konfiguracyjny

Najważniejsze opcje w pliku konfiguracyjnym Ansible:

- sekcja **[privilege\_escalation]** zawiera konfiguracje uprawnień z jakimi działa Ansible na hostach (nodach):
  - **become** – przełączenie się na usera po zalogowaniu przez ssh (zwyczajowo root)
  - **become\_method** – metoda przełączenia na usera (zwyczajowo sudo)
  - **become\_user** – określa na jakiego usera Ansible ma się przełączyć po zalogowaniu (zwyczajowo root)
  - **become\_ask\_pass** – pytanie o hasło dla become\_method

## Ansible – plik konfiguracyjny

Dla każdego nowego projektu zalecane jest tworzenie nowego pliku konfiguracyjnego oraz nowego pliku inventory.

Pliki **ansible.cfg** oraz **inventory.cfg** powinny być przechowywane w folderze danego projektu.

Wszelkie opcje pominięte w lokalnym pliku konfiguracyjnym zostaną wczytane z głównego pliku konfiguracyjnego:

**/etc/ansible/ansible.cfg**

# INVENTORY FILE

- Zawiera listę hostów, którymi chcemy zarządzać
- Zapisany w formacie Initialization (INI)
- Jeśli nie określimy innego położenia pliku Inverntory, Ansible pobierze położenie pliku ze zmiennej **host\_file** z pliku konfiguracyjnego **ansible.cfg**. Standardowo jest to plik **/etc/ansible/hosts**
- Jeśli korzystamy z innego pliku, wywołujemy go z przełącznikiem **-i** lub **--inventory-file**, np:

**ansible-playbook -i** moje\_hosty.yml

# INVENTORY FILE

- Najprostszy plik **Inventory** zawiera **listę nazw domenowych** hostów, którymi chcemy zarządzać (każdy host powinien się znajdować w osobnym wierszu), np:

krakow.altkom.pl  
wroclaw.altkom.pl

lub po prostu **listę adresów IP**, np.:

192.168.0.10  
172.16.10.10

# INVENTORY FILE

- W pliku **Inventory** możemy pogrupować hosty wedle konfiguracji, położenia geograficznego, rodzaju dystrybucji itp., np:

```
[webservers]  
krakow.atkom.pl  
wroclaw.atkom.pl  
192.168.10.155
```

```
[databases]  
katowice.atkom.pl  
warszawa.atkom.pl
```

- Dzięki temu możemy wywołać danego playbooka dla określonej grupy hostów, np.:  
ansible-playbook webservers -i inventory.cfg bazy\_danych.yml
- Zmiany zostaną wykonane tylko na krakow.atkom.pl i wroclaw.atkom.pl

# INVENTORY FILE

W pliku INVENTORY możemy tworzyć także grupy potomne, np:

```
[webservers]
```

```
krakow.altkom.pl
```

```
wroclaw.altkom.pl
```

```
[databases]
```

```
katowice.altkom.pl
```

```
warszawa.altkom.pl
```

```
[centos:children]
```

```
webservers
```

```
databases
```

ansible centos -i inventory.cfg -m ping

wywoła moduł ping dla grupy centos zawierającej grupy potomne (webservers, databases). Moduł zostanie wykonany na wszystkich hostach.

# INVENTORY FILE

Poszczególne hosty mogą przynależeć do kilku grup, np.:

[webservers]

krakow.altkom.pl

wroclaw.altkom.pl

[databases]

katowice.altkom.pl

warszawa.altkom.pl

krakow.altkom.pl

Host krakow.altkom.pl przynależy do grupy webservers oraz databases

# INVENTORY FILE

- W pliku INVENTORY możemy także zastosować wyrażenia regularne, np:

[webservers]

krakow.altkom.pl

wroclaw.altkom.pl

192.168.10.[1:10]

[databases]

katowice[2-4].altkom.pl

warszawa[A-D].altkom.pl

Do grupy webservers przynależą m.in. hosty: 192.168.10.1, 192.168.10.2, 192.168.10.3 itp.

Do grupy databases przynależą m.in. hosty: katowice2.altkom.pl, katowice3.altkom.pl, katowice4.altkom.pl itp.



# INVENTORY FILE

- Zawsze istnieją 2 grupy hostów:
  - **all** – wszystkie hosty, niezależnie jak są pogrupowane
  - **ungrouped** – wszystkie hosty nieprzynależące do żadnej grupy
- W pliku **Inventory** możemy nadpisać pewne zmienne z pliku konfiguracyjnego ansible, np.:
  - **ansible\_user** – user, z uprawnieniami którego zostanie wykonane zadanie, playbook
  - **ansible\_port** – port ssh, na który loguje się master
  - **ansible\_host** – adres IP hosta
  - **ansible\_connection** – rodzaj połączenia, które zostanie wykorzystane
  - **ansible\_private\_key\_file** – ścieżka do pliku z kluczem prywatnym

# INVENTORY FILE

- Jeśli w pliku **Inventory** znajdują się 2 hosty lub grupy hostów o tej samej nazwie, Ansible zgłosi błąd
- Weryfikacji hostów w pliku **Inventory** można dokonać za pomocą polecenia:
  - ansible webservers -i inventory.cfg --list-hosts
  - ansible all -i inventory.cfg --list-hosts
  - ansible ungrouped -i inventory.cfg --list-hosts

# DYNAMIC INVENTORIES

- W środowiskach, gdzie zmiany w konfiguracji środowiska występują często, możemy się posłużyć DYNAMIC INVENTORIES – plikami inventory generowanymi dynamicznie tuż przed użyciem komendy 'ad-hoc' bądź playbooka
- Jest to szczególnie przydatne w środowiskach chmurowych, np. AWS, Azure lub OpenStack
- Zamiast posługiwać się statycznym plikiem z listą hostów, ansible za pomocą odpowiednich skryptów generuje listę hostów i zwraca wynik w formacie JSON.
- Wiele takich gotowych skryptów możemy pobrać z repozytorium ansible pod adresem:  
<https://github.com/ansible/ansible/tree/devel/contrib/inventory>

## Ansible - moduły

- Moduły są głównym komponentem playbooków
- Można z nich też korzystać przy jednorazowym wykonaniu polecenia 'ad-hoc', np. moduł 'ping' do sprawdzania połączenia między masterem a nodami :

```
ansible -i inventory.cfg all -m ping
```

- Każdy moduł jest zaprojektowany do wykonania konkretnego zadania, dzięki nim możemy zarządzać wieloma:
  - serwerami
  - serwisami
  - aplikacjami
  - bazami danych
  - infrastrukturą zwirtualizowaną
  - środowiskami w chmurze

## Ansible - moduly

- Moduły są instalowane automatycznie wraz z Ansiblem
- Aby wyświetlić listę modułów wydajemy polecenie:  
**ansible-doc -l**
- Aby przejrzeć zmienne w konkretnym module oraz przejrzeć jego opis oraz możliwości:

**ansible-doc service** //dokumentacja modułu 'service'

- Moduły znajdują się w miejscu wskazanym w ansible.cfg i pogrupowane są na katalogi z ogólnym przeznaczeniem, np.: files, database, monitoring, system itp.

## Ansible - moduly

- główne moduły używane do zarządzania systemami linuxowymi:
- Moduły do zarządzania plikami:
  - **copy** oraz **fetch**
  - **file**
  - **lineinfile**
  - **synchronize**
- Moduły do zarządzania oprogramowaniem:
  - **package** – samodzielnie wykrywa menedżera oprogramowania
  - **yum** – dla dystrybucji z rodziny RedHat
  - **apt** – dla dystrybucji z rodziny Debian
  - **zypper** – dla dystrybucji z rodziny SUSE
  - **pip** – dla Pythona
  - **gem** – dla Ruby

## Ansible - moduły

- główne moduły używane do zarządzania systemami linuxowymi:
- Moduły systemowe:
  - **reboot**
  - **firewalld** – zarządzanie zaporą sieciową
  - **service/systemd** – zarządzanie serwisami w systemie
  - **user** oraz **group** – zarządzanie kontami użytkowników
- Moduły do zarządzania sieciami:
  - **nmcli** – zarządzanie połączeniami sieciowymi
  - **uri** – obsługa web service
  - **get\_url** – pobieranie plików przez protokoły HTTP, FTP...

## Ansible - moduly

- Większość modułów zawiera listę argumentów (możliwe do wyświetlenia poprzez 'ansible-doc moduł').
- Argumenty modułów podajemy poprzez przełącznik -a w zadaniach ad-hoc:

```
ansible all -i inventory.cfg -m user -a name=student uid=4001
```

lub jako listę w playbookach:

```
- name: "tworzenie usera"  
  user:  
    name: altkom  
    uid: 4001  
    state: present
```



## Ansible - moduly

- Wywołanie modułu w poleceniu ad-hoc lub w playbooku zwraca wynik, np.:  
[marcin@server1 ~]\$ ansible centos -i inventory.cfg -m shell -a \"cat /etc/passwd | grep altkom\"  
192.168.10.153 | CHANGED | rc=0 >>  
altkom:x:4000:4000::/home/altkom:/bin/bash

Pierwsza linia zwraca status wykonanego zadania

- 192.168.10.153 to adres hosta, na którym zostało wykonane polecenie
- **CHANGED** oznacza, że na hoście zarządzanym dokonano odpowiednich zmian
- **rc=0** oznacza, że zadanie zostało wykonane poprawnie

Druga linia zwraca wynik wykonanego polecenia

## Ansible - moduly

- To samo polecenie 'ad-hoc' wywołane z przełącznikiem -o zwraca wynik w postaci jednego wiersza:

```
[marcin@server1 ~]$ ansible centos -i inventory.cfg -m shell -a "cat /etc/passwd | grep altkom" -o  
192.168.10.152 | CHANGED | rc=0 | (stdout) altkom:x:4000:4000::/home/alkom:/bin/bash
```

## Ansible - moduly

- Przykład użycia modułu **ping** w poleceniu **'ad-hoc'**:

```
[marcin@server1 ~]$ ansible all -i inventory.cfg -m ping
192.168.10.152 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/libexec/platform-python"
  },
  "changed": false,
  "ping": "pong"
}
192.168.10.153 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/libexec/platform-python"
  },
  "changed": false,
  "ping": "pong"
}
```

# Ansible - moduly

- Przykład użycia modułu ping w playbooku:

```
[marcin@server1 ~]$ ansible-playbook -i inventory.cfg playbook_ping.yml
```

```
PLAY [all]
```

```
*****  
*****
```

```
TASK [Gathering Facts]
```

```
*****  
*****
```

```
ok: [192.168.10.152]
```

```
ok: [192.168.10.153]
```

```
TASK [sprawdzenie komunikacji]
```

```
*****  
*****
```

```
ok: [192.168.10.152]
```

```
ok: [192.168.10.153]
```

```
PLAY RECAP
```

```
*****  
*****
```

```
192.168.10.152      : ok=2   changed=0   unreachable=0   failed=0   skipped=0   rescued=0  
ignored=0
```

```
192.168.10.153      : ok=2   changed=0   unreachable=0   failed=0   skipped=0   rescued=0  
ignored=0
```

```
[marcin@server1 ~]$
```

## Ansible - moduly

- Implementacja modułu ping w playbooku:

---

- hosts: all  
  become: true

tasks:

- name: sprawdzenie komunikacji  
  ping:

# ANSIBLE - PLAYBOOKS

- główne cechy playbookow:
- Dostępne lokalnie
- Do wielokrotnego wykorzystania
- Napisane w języku YAML
- Uruchamiane za pomocą polecenia **ansible-playbook:**

```
ansible-playbook -i inventory.yml playbook.yml
```

- Mogą zawierać zmienne
- Mogą odwoływać się do innych playbooków
- Mogą odwoływać się do plików zewnętrznych (np. plik z hasłem)

# ANSIBLE - PLAYBOOKS

- Playbook to uporządkowany zbiór zadań,, plik tekstowy w formacie YAML. Zadania zawarte w playbooku wykonywane są na wybranych hostach z pliku inventory.
- Zadania z playbooka uruchamiane są w odpowiedniej kolejności, zgodnie z umiejscowieniem w pliku.
- Poszczególne elementy struktury pliku tego samego poziomu (np. lista paczek do zainstalowania) powinny być wyróżnione równym wcięciem z lewej strony, np.:
  - httpd
  - mariadb-server
- Elementy potomne zależne od rodzica powinny mieć wcięcie z lewej strony, np.:  
packages:
  - httpd
  - mariadb-server

# ANSIBLE - PLAYBOOKS

- Jako znaków wcięcia należy używać tylko i wyłącznie spacji, znaki tabulacji nie są dozwolone.

Dla przejrzystości i łatwości odczytu można stosować linie puste.

- W pliku ~/.vimrc można ustawić domyślne wcięcia tabulacji równe 2 spacje:

**autocmd FileType yaml setlocal ai ts=2 sw=2 et**



# ANSIBLE - PLAYBOOKS

- Każdy plik playbooka powinien się rozpoczynać znakami trzech myślników (---)
- Zakończenie pliku można oznaczyć trzema znakami kropki (...). Jednakże jest to rzadko stosowane, można to pominąć.
- Każde poszczególne zadanie w playbooku powinno się zaczynać znakiem myślnika i spacji, np.:  
tasks:
  - zadanie1
  - zadanie2

# ANSIBLE - PLAYBOOKS

- Każdy playbook zawiera zestaw **klucz: wartość**. Dla danego klucza jako wartość może być podana lista argumentów, np.:

name: Instalacja oprogramowania serwera

hosts: webservers

**tasks:**

- **name:** Instalacja httpd

- yum:

- name: httpd

- ...

- **name:** Instalacja bazy danych

- yum:

- ...

**Klucz** 'hosts' posiada **wartość** 'webservers'

**Klucz** 'tasks' jako **wartość** posiada **listę** zadań do wykonania zaczynająca się od 'name'

# ANSIBLE - PLAYBOOKS

W powyższym playbooku klucz **'name:'** (**nazwa zadania**) nie jest konieczny, ale zalecany ze względu na jasność odczytu. Jeśli playbook zawiera dużą ilość zadań, łatwiej zrozumieć, za jaką funkcjonalność poszczególne zadanie jest odpowiedzialne

- Klucz **'hosts:'** zawiera informacje na jakich hostach zostaną wykonane zadania playbooka. Jest to odpowiednik: ansible **webservers** -i inventory playbook.yml
- Klucz **'yum:'** odpowiada za przywołanie modułu 'yum' w celu instalacji oprogramowania. Poniżej modułu 'yum' występuje lista atrybutów dla danego modułu.

# ANSIBLE - PLAYBOOKS

- Kolejność zadań zamieszczonych w playbooku jest bardzo ważna, ponieważ dokładnie w takiej kolejności zostaną wykonane zmiany na hostach.
- Aby zobaczyć szczegóły wykonania poszczególnych zadań z playbooka można się posłużyć przełącznikiem **-v** lub jego wielokrotnością, np.:

```
ansible-playbook -v -i inventory.cfg webservers-playbook.yml
```

- **-v** zwraca rezultat zadania
- **-vv** zwraca również konfiguracje zadań
- **-vvv** dodaje informacje dotyczące połączenia z zarządzanymi hostami
- **-vvvv** zwraca dodatkowe opcje, np. informacje o użytkowniku

# ANSIBLE - PLAYBOOKS

- Ansible posiada wbudowaną opcję sprawdzania składni playbooka. Przed każdym uruchomieniem playbooka warto sprawdzić, czy nie posiada on błędów syntaktycznych:

```
ansible-playbook playbook.yml -i inventory.cfg --syntax-check
```

- Kolejnym sposobem na sprawdzenie poprawności działania naszego playbooka jest wywołanie go z opcją **-C**. Dzięki temu playbook wykona się tylko w formie symulacji. Na hostach zarządzanych nie zostaną dokonane żadne zmiany:

```
ansible-playbook -C playbook.yml -i inventory
```

# ANSIBLE PLAYBOOKS - ZMIENNE

- Playbooki mogą wykorzystywać zmienne. Najprostszym sposobem użycia zmiennych w playbookach jest umieszczenie sekcji **vars** w samym playbooku, np.:

**vars:**

**folder: /var/www/html**

**folder2: /backup**

**tasks:**

- name: kopiowanie folderu

copy:

path: {{ folder }}

dest: {{ folder2 }}

...

Zmienne przyjmują format **nazwa: wartość**

# ANSIBLE PLAYBOOKS - ZMIENNE

- Innym sposobem na wykorzystanie zmiennych w playbooku jest dołączenie osobnego pliku z zapisanymi zmiennymi, np.:

**vars\_files:**

- **zmienne.yml**

tasks:

- name: kopiowanie folderu

copy:

path: {{ folder }}

dest: {{ folder2 }}

...

W playbooku możemy wykorzystać zmienne zapisane w kilku plikach.

# ANSIBLE PLAYBOOKS - ZMIENNE

- Do wyświetlania wartości zmiennych możemy posłużyć się modulem **debug**, np.:
  - debug: var=folder
- Innym sposobem na podgląd zmiennych jest ich rejestracja za pomocą instrukcji **register**, np.:
  - name: wynik polecenia id  
shell: id  
**register: login**
  - debug: var=**login.stdout**



# ANSIBLE PLAYBOOKS - ZMIENNE

- Jeśli zwracaną wartością zmiennej jest słownik, możemy odwołać się do niej za pomocą kropki (.) lub nawiasów kwadratowych ([]).
- Oba poniższe wyrażenia zwracają ten sam wynik:
  - `{{ login.stdout }}`
  - `{{ login['stdout'] }}`
- Taką formę zapisu można również łączyć:
  - `{{ ansible_eth0.ipv4['address'] }}`
- **stdout** - zawiera zwracany komunikat
- **stdout\_lines** - zawiera kilka linii komunikatu
- **stderr** - zwraca komunikat o błędzie

## ANSIBLE PLAYBOOKS - ZMIENNE (FACTS)

- Podczas wykonywania playbookow Ansible zbiera we wstępnej fazie dane dotyczące zarządzanego hosta zwane **facts**.
- Są to informacje o architekturze procesora, dyskach, pamięci, konfiguracji sieci itp.
- Wszystkie zmienne gromadzone przez Ansible ( **facts** ) możemy wyświetlić za pomocą modułu **setup**:
  - 'ad-hoc':

```
ansible centos -i inventory -m setup
```

- playbook:

tasks:

- name: wyswietl gather\_facts

debug:

var: **ansible\_facts**

# ANSIBLE PLAYBOOKS - ZMIENNE (FACTS)

- Aby zrezygnować z automatycznego gromadzenia **facts** (np. przy pracy z bardzo złożonym środowiskiem) należy użyć w playbooku:

**gather\_facts: False**

albo w pliku konfiguracyjnym ansible.cfg formuły:

**gather\_facts = False**

# ANSIBLE PLAYBOOKS - ZMIENNE (FACTS)

- Najczęściej używane zmienne **ansible\_facts**:
  - **ansible\_all\_ipv4\_addresses** – adresy IP
  - **ansible\_interfaces** – nazwy interfejsów sieciowych
  - **ansible\_distribution** – nazwa dystrybucji Linuxa
  - **ansible\_os\_family** – np. RedHat, Debian itp.
  - **ansible\_architecture** – architektura procesora
  - **ansible\_env** – zmienne systemowe
  - **ansible\_nodename** – pełna nazwa FQDN
  - **ansible\_hostname** – nazwa maszyny  
(systemowy hostname)
  - **ansible\_devices** – lista urządzeń blokowych

# ANSIBLE PLAYBOOKS - ZMIENNE (FACTS)

- Przykład zastosowania **ansible\_facts**:

---

hosts: centos

tasks:

- name: wyswietl IP

- debug:

- msg: >

- Host {{ ansible\_facts['nodename'] }}

- posiada adres IP {{ ansible\_facts['all']['ipv4']['addresses'] }}

## ANSIBLE PLAYBOOKS - ZMIENNE (FACTS)

- Informacje zwracane przez **ansible\_facts** przekazywane są w formie słownika. Zagłębiając się w głąb słownika **ansible\_facts** możemy dotrzeć do konkretnej informacji (np. informacji o wielkości partycji systemowej):
- Poniższy playbook zwraca wszelkie informacje o partycji sda1:

---

```
- name: testy testy testy
  hosts: server2
  become: true
  tasks:
    - name: distro watch
      debug:
        msg: "{{ ansible_devices['sda']['partitions']['sda1'] }}"
```

## ANSIBLE PLAYBOOKS - ZMIENNE (FACTS)

```
ok: [192.168.10.152] => {
  "msg": {
    "holders": [],
    "links": {
      "ids": [
        "ata-VBOX_HARDDISK_VB0c091213-3b4009f5-part1"
      ],
      "labels": [],
      "masters": [],
      "uuids": [
        "024ae99e-8baf-4678-ac0b-e76f6e3291e3"
      ]
    },
    "sectors": "2097152",
    "sectorsize": 512,
    "size": "1.00 GB",
    "start": "2048",
    "uuid": "024ae99e-8baf-4678-ac0b-e76f6e3291e3"
  }
}
```

## ANSIBLE PLAYBOOKS - ZMIENNE (FACTS)

Jeśli z wyniku naszego playbooka potrzebujemy wydobyć tylko informacje o wielkości partycji ( klucz '**size:**') możemy zmodyfikować naszego playbooka :

---

```
- name: okreslanie wielkosci partycji
  hosts: server2
  become: true
  tasks:
    - name: partycja sda1
      debug:
        msg: >
          "Wielkosc partycji sda1:
           {{ ansible_devices.sda.partitions.sda1.size }}"
```



## ANSIBLE PLAYBOOKS - ZMIENNE (FACTS)

- Moduł **setup** w playbookach wywołuje się automatycznie wraz z uruchomieniem playbooka
- W module **setup** możemy stosować filtry w celu wyświetlenia tylko zawężonych, potrzebnych nam informacji, np.:

```
ansible centos -i inventory -m setup -a 'filter=ansible_os_family'
```

Ansible pozwala na tworzenie własnych faktów **set\_fact**, do których możemy się później odwoływać za pomocą określonego modułu , np.:

- set\_fact: ip="{{ ansible\_eth0.ipv4.address }}"
  - name: wyswietlenie adresu IP
- debug:
- msg: "Moj adres IP: {{ ip }}"

## ANSIBLE PLAYBOOKS - ZMIENNE ( FACTS )

- Zmienną zapisaną w playbooku można nadpisać w komendzie uruchamiającej playbooka za pomocą przełącznika **-e** lub **--extra-vars**, np.:

playbook.yml:

**vars:**

**folder: /var/www/html**

**tasks:**

- name: kopia folderu

...

ansible-playbook -i inventory playbook.yml **-e folder=/www**

- Zmienna zdefiniowana poprzez przełącznik **-e** ma zawsze najwyższy priorytet

# ANSIBLE PLAYBOOKS

- Każdy prosty playbook musi zawierać co najmniej:
  - **grupę hostów**, na której wykonywane będą zadania (**hosts**)
  - **listę zadań**, które mają być wykonane (**tasks**)
- Ponadto w playbooku można umieścić:
  - **name** – opis akcji wyświetlany wraz ze startem zadania (zalecane)
  - **become** – możliwość wykonania zadania z konta roota (musi być ustawiona wartość **True**)
  - **vars** – lista zmiennych i przypisanych im wartości
- Nazwa zadania (**name**) może być wykorzystywana w argumencie **--start-at-task** – rozpoczęcie playbooka od dowolnego miejsca:

ansible-playbook --start-at-task 'Instalacja httpd' playbook.yml

# ANSIBLE PLAYBOOKS

- W każdym zadaniu playbooka może występować tylko jeden moduł
- Gdy chcemy korzystać z kilku modułów w celu dokonania zmian na serwerach, musimy je umieścić w osobnych zadaniach:

tasks:

- **name:** zadanie nr 1

- yum:**

- name: httpd

- **name:** uruchomienie serwisu httpd

- service:**

- name: httpd

- state: started

# ANSIBLE PLAYBOOKS

- Aby ograniczyć wykonanie playbooka tylko do części hostów zawartych w pliku **inventory**, należy posłużyć się przełącznikiem **-- limit**
- Plik inventory.cfg:  
[web]  
192.168.0.104  
[databases]  
192.168.0.105
- `ansible-playbook -i inventory --limit web playbook.yml`
- Zadania z playbooka zostaną wykonane tylko na hostach z sekcji [web]

# ANSIBLE PLAYBOOKS - PETLE

- W Ansible mamy możliwość skorzystania z **pętli (loop)**. Zapobiega to nadmiernemu tworzeniu zadań.
- Najprostsza i najczęściej wykorzystywana jest pętla **loop**:

tasks:

- name: tworzenie kont użytkowników

user:

name: **{{ item }}**

state: present

**loop:**

- user1

- user2

- Pętle należy zacząć od tego samego wcięcia od jakiego zaczyna się moduł

# ANSIBLE PLAYBOOKS - PETLE

- Pętla loop zastąpiła pętle **with\_items**, aczkolwiek **with\_items** ciągle jeszcze można umieszczać w playbookach.
- Pętla **with\_items**:

tasks:

- name: tworzenie kont użytkowników

user:

name: **{{ item }}**

state: present

**with\_items:**

- user1

- user2

- Zaleca się jednak używanie petli **loop**.

# ANSIBLE PLAYBOOKS - PETLE

- Wynik pętli **loop** możemy przechwycić do dalszego wykorzystania lub debugowania:

- name: nowi uzytkownicy

shell:

echo {{ item }}

**loop:**

- user1

- user2

**register: echo**



# ANSIBLE PLAYBOOKS - PETLE

- Wartości przekazywane do pętli **loop** mogą także zostać określone jako wartości zmiennych w sekcji **vars**:

**vars:**

**uzytkownik:**

- user1
- user2

**tasks:**

- name: tworzenie kont uzytkownikow

**user:**

name: “{{ item }}”

state: present

**loop:** “{{ uzytkownik }}”

## ANSIBLE PLAYBOOKS - PETLE

- Do pętli **loop** możemy też przekazać wartości w postaci słownika. W takim przypadku każda zmienna jako parametr przyjmuje klucz słownika:

tasks:

- name: zarządzanie użytkownikami

user:

name: {{ **item.name** }}

state: present

groups: {{ **item.groups** }}

**loop:**

- { **name**: 'marcin', **groups**: 'student' }

- { **name**: 'piotr', **groups**: 'profesor' }

# ANSIBLE PLAYBOOKS - PETLE

- Do każdego zadania możemy użyć osobnej pętli **loop** :

tasks:

- **name**: zarządzanie użytkownikami

**user**:

name: {{ **item.name** }}

state: present

groups: {{ **item.groups** }}

**loop**:

- { **name**: 'marcin', **groups**: 'student' }

- **name**: zarządzanie serwisami

**service**:

name: {{ **item** }}

state: started

**loop**:

- firewallld

# ANSIBLE PLAYBOOKS - WARUNKI

- W playbookach możemy także wykorzystywać warunki.  
Dane zadanie zostanie zrealizowane pod warunkiem spełnienia pewnych zależności, np.:

tasks:

- name: instalacja httpd na maszynie wirtualnej

yum:

name: httpd

state: latest

**when:** ansible\_os\_family == "RedHat"

- Instalacja pakietu httpd zakończy się pomyślnie  
jeśli dystrybucja naszego Linuxa to RedHat, CentOS, Fedora itp.

# ANSIBLE PLAYBOOKS - WARUNKI

Warunków do spełnienia może być kilka, np.:

- `ansible_distribution == "Centos" and ansible_processor_cores == 1`
- `ansible_distribution == "Centos" or ansible_distribution == "Debian"`
- `(ansible_distribution == "Centos" and ansible_processor_cores == 1)  
or  
(ansible_distribution == "Debian" and ansible_fqdn == "s1.altkom.pl")`

# ANSIBLE PLAYBOOKS - WARUNKI

## Sprawdzanie warunków:

- równa się (string):
  - `ansible_machine == "x86_64"`
- równa się (wartość numeryczna):
  - `max_memory == 1024`
- mniejsze niż
  - `max_memory < 1024`
- większe niż
  - `max_memory > 1024`
- mniejsze lub równe
  - `max_memory <= 1024`
- większe lub równe
  - `max_memory >= 1024`

# ANSIBLE PLAYBOOKS - WARUNKI

## Sprawdzanie warunków:

- różne od
  - min\_memory **!=** 256
- zmienna jest zdefiniowana ( istnieje )
  - min\_memory **is defined**
- zmienna nie jest zdefiniowana
  - min\_memory **is not defined**
- wartość boolowska (**True, yes** lub **1**)
  - memory available
- wartość boolowska (**False, not** lub **0**)
  - not memory\_available
- wartość pierwszej zmiennej jest zawarta także w wartości drugiej zmiennej
  - ansible\_distribution **in** supported\_distros

# ANSIBLE - HANDLERS

- **Handlers** to zadania uwarunkowane, uruchamiane tylko jeśli mają odwołanie do **notify** poprzedniego zadania:

tasks:

- name: kopia pliku index.html  
copy:  
src: index.html  
dest: /var/www/html/index.html  
**notify: kopia\_pliku**

handlers:

- name: kopia\_pliku  
service:  
name: httpd  
state: restarted



# ANSIBLE - HANDLERS

- **Handlers** mają unikalne nazwy (globalnie) i są wykonywane dopiero po wykonaniu wszelkich pozostałych zadań
- **Handlers** nie mogą być dołączane do playbooka, są one zawsze integralną częścią playbooka
- Zadanie zdefiniowane w sekcji **Handlers** wykona się tylko i wyłącznie wtedy, gdy zadanie z **notify** ma status **changed**
- **Handlers** wykonywane są zawsze w kolejności w jakiej występują w sekcji **handlers** playbooka

# ANSIBLE - HANDLERS

- Istnieje możliwość wywołania **handlers** nawet jeśli zadanie z notify zakończy się błędem. Decyduje o tym parametr **force\_handlers**:

hosts: centos

**force\_handlers: yes**

tasks:

- name: Instalacja httpd

**notify:** restart httpd

**handlers:**

- **name: restart httpd**

service:

name: httpd

state: restarted

# ANSIBLE - HANDLERS

- W przypadku zadań playbooka sami możemy określić, kiedy zadanie zakończy się błędem za pomocą parametru **failed\_when**:

tasks:

- name: sprawdzenie selinuxa

debug:

msg: "{{ ansible\_selinux.status }}"

register: selinux\_status

**failed\_when**: "'disabled' in selinux\_status.stdout"

- Podobnie działa parametr **changed\_when**, tylko z pozytywnym skutkiem

# ANSIBLE - BLOCK

- Parametr **block** zastosowany w playbooku pozwala na połączenie kilku zadań w logiczną całość. Często występuje z parametrem **when**

- tasks:

- **block**:

- **name**: instalacja httpd

yum:

...

- **name**: uruchomienie serwisu httpd

service:

...

**when**: ansible\_distribution == "Centos"

## ANSIBLE - BLOCK

- Dzięki zastosowaniu parametru **block** możemy zarządzać błędami poprzez dodatkowe parametry **rescue** oraz **always**:
  - tasks:
    - **block**:
      - name: zadanie1
    - rescue**:
      - name: zadanie 2
    - always**:
      - name: zadanie 3
- W powyższym playbooku zadania 2 i 3 wykonają się nawet wtedy, gdy zadanie 1 zakończy się błędem. Zadanie 3 wykona się zawsze.

# ANSIBLE - YAML

- W języku **YAML** najczęściej występują podstawowe formy składni:

- **deklaracje**, np.:

name: "Instalacja serwera www"

- **listy**, np.:

users:

- Marcin
- Piotr
- Dominik

- **słowniki** (tablice asocjacyjne), np.:

user:

name: Marcin

comment: dzial IT

# ANSIBLE - YAML

- Formy te można stosować razem, np.:

tasks:

- name: "Instalacja serwera Apache"

yum:

name: httpd

state: present

- name: "Uruchomienie serwisu httpd"

service:

name: httpd

state: started

enabled: True

# ANSIBLE - YAML

- Komentarze w języku **YAML** dodaje się poprzedzając je '#':

tasks:

```
# pierwsze zadanie playbooka  
- name: instalacja httpd
```

...

- Jeśli musimy zapisać dłuższy ciąg znaków, możemy posłużyć się znakiem '>' dzielącym wiersze:

adres: >

```
Altkom Akademia S.A.  
Warszawa 00-867  
ul. Chłodna 51
```



## ANSIBLE PLAYBOOKS - Jinja2 templates

- W Ansible dostępnych jest kilka modułów do edycji plików tekstowych, m.in. **lineinfile**, **blockinfile**.
- Aby wprowadzić elastyczność i dynamiczne zmiany w plikach możemy wykorzystać moduł **template** oraz wzorce **Jinja2**.
- Składnia playbooka przedstawia się następująco:

tasks:

- name: plik index.html

template:

src: /tmp/index.html.j2

dest: /var/www/html/index.html

- Podajemy źródło pliku, w którym umieszczony jest wzorzec **Jinja2** oraz docelową ścieżkę na zarządzanym hoście, gdzie ten plik ma zostać umieszczony

# ANSIBLE PLAYBOOKS - Jinja2 templates

- Pliki wzorców **Jinja2** standardowo posiadają rozszerzenie **.j2**
- Aby wykorzystać potencjał wzorców **Jinja2** w pliku możemy posłużyć się zmiennymi.

Przykład pliku z zastosowaniem wzorca **Jinja2**:

Witaj na stronie www **{{ ansible\_nodename }}**

Strona została utworzona w: **{{ ansible\_date\_time.date }}**

- W miejsca “**{{ }}**” zostają podstawione dynamicznie wartości zmiennych:
  - zdefiniowane w playbooku
  - ansible facts
  - zmienne z dołączanych do playbooka plików

# ANSIBLE PLAYBOOKS - Jinja2 templates

- Aby w pełni wykorzystać potencjał plików z wzorcami **Jinja2** możemy użyć:

- **prostych zmiennych**, np.:

`{{ ansible_date_time.date }}`

- **zmiennych przechowujących element tablicy**, np.:

`{{ TABLICA['klucz'] }}`

- **zmiennych przechowujących atrybuty obiektu**, np.:

`{{ OBIEKT.atrybut }}`

# ANSIBLE PLAYBOOKS - Jinja2 templates

- Aby w pełni wykorzystać potencjał plików z wzorcami **Jinja2** możemy użyć:

- **filtrów**, np.:

```
{{ ZMIENNA | capitalize }}
```

- **sprawdzania warunków**, np.:

```
{% if ansible_eth0.active == True %}
```

```
    IP = {{ ansible_eth0.ipv4.address }}
```

```
{% endif %}
```

# ANSIBLE PLAYBOOKS - Jinja2 templates

- Aby w pełni wykorzystać potencjał plików z wzorcami **Jinja2** możemy użyć:

- **pętle**, np.:

```
{% for maszyna in groups['hosts'] %}
```

```
  {{ hosts }}
```

```
{% endfor %}
```

# ANSIBLE VAULT

- Domyślnie Ansible przechowuje wrażliwe dane, klucze API oraz hasła w formie plików tekstowych, bez szyfrowania.
- Aby zmniejszyć ryzyko wycieku danych i zminimalizować incydenty bezpieczeństwa Ansible posiada mechanizm **ANSIBLE VAULT** służący do szyfrowania plików (domyślnie dane szyfrowane są przez zewnętrzną bibliotekę Pythona python3-cryptography - klucz symetryczny AES256).
- Za pomocą **ansible-vault** możemy zaszyfrować:
  - pliki haseł
  - pliki zawierające zmienne wykorzystywane w playbookach
  - pliki zewnętrzne wykorzystywane w playbookach
  - pliki inventory itp.

# ANSIBLE VAULT

- Tworzenie nowego pliku szyfrowanego. Po wydaniu polecenia:

**ansible-vault create** szyfrowany.yml

Ansible zapyta o hasło, po czym otworzy plik do edycji w domyślnym edytorze systemowym:

```
[marcin@server1 ~]$ ansible-vault create szyfrowany.yml
```

New Vault password:

Confirm New Vault password:

# ANSIBLE VAULT

- Podgląd zaszyfrowanego pliku:

```
[marcin@server1 ~]$ cat szyfrowany.yml
$ANSIBLE_VAULT;1.1;AES256
35376364616332623961643839316565663566646161333636363137
396162303034323937626433
6162623731316563363965306638666336613432363234390a633964
323231653739643433316662
63653637353138613664393537303566336561356238373232656235
373731633337313134616138
3437633430633138300a616563666338383665373663316262373764
643366396432383163653433
37643539323930366530333638373635356335313362666230663833
306636313162
```



# ANSIBLE VAULT

- Podgląd zaszyfrowanego pliku w ansible-vault:

**ansible-vault view** szyfrowany.yml

```
[marcin@server1 ~]$ ansible-vault view szyfrowany.yml
```

Vault password:

dane zaszyfrowane

```
[marcin@server1 ~]$
```

# ANSIBLE VAULT

- Edycja zaszyfrowanego pliku w ansible-vault:

**ansible-vault edit** szyfrowany.yml

```
[marcin@server1 ~]$ ls -l szyfrowany.yml
```

```
-rw-----. 1 marcin marcin 419 Feb  6 18:12 szyfrowany.yml
```

```
[marcin@server1 ~]$ ansible-vault edit szyfrowany.yml
```

Vault password:

Domyślnie ze względów bezpieczeństwa pliki tworzone za pomocą **ansible-vault** mają prawo odczytu i zapisu tylko i wyłącznie dla właściciela pliku.

## ANSIBLE VAULT

- Za pomocą mechanizmu **ansible-vault** możemy również szyfrować już istniejące pliki:

**ansible-vault encrypt** szyfr2.yml

```
[marcin@server1 ~]$ cat szyfr2.yml
```

dane szyfrowane - plik 2

```
[marcin@server1 ~]$ ansible-vault encrypt szyfr2.yml
```

New Vault password:

Confirm New Vault password:

Encryption successful

```
[marcin@server1 ~]$ cat szyfr2.yml
```

```
$ANSIBLE_VAULT;1.1;AES256
```

```
30343539623264383836613162376136336131643237356163656261  
666437313062316334663331 ...
```

# ANSIBLE VAULT

- Ten sam mechanizm **ansible-vault** działa również w drugą stronę:

**ansible-vault decrypt** tajny.yml

```
[marcin@server1 ~]$ cat tajny.yml
```

```
$ANSIBLE_VAULT;1.1;AES256
```

```
64383633366561343963323530386636656532396237383939313065  
383233323430376263623739
```

```
...
```

```
[marcin@server1 ~]$ ansible-vault decrypt tajny.yml
```

```
Vault password:
```

```
Decryption successful
```

```
[marcin@server1 ~]$ cat tajny.yml
```

```
plik scisle tajny
```

# ANSIBLE VAULT

- Zarówno przy szyfrowaniu, jak i przy deszyfrowaniu plików możemy użyć przełącznika:

**--output=**

- **ansible-vault encrypt plik1.yml --output=plik\_szyfrowany.yml**  
stworzy nowy plik szyfrowany z danymi z pliku plik1.yml  
pod nazwą szyfrowany.yml
- **ansible-vault decrypt szyfrowany.yml --output=deszyfrowany.yml**  
odszyfrowane dane z pliku szyfrowany.yml zapisze w plaintext  
do pliku deszyfrowany.yml

# ANSIBLE VAULT

- Zmiany hasła dokonuje się za pomocą komendy:

**ansible-vault rekey** szyfrowany.yml

- Hasła chroniące nasze pliki z wrażliwymi danymi możemy również przechowywać w osobnych plikach (warto je zabezpieczyć przed nieuprawnionym dostępem):

**ansible-vault create --vault-password-file=passwd** szyfrowany.yml

- W podobny sposób możemy podejrzeć zaszyfrowany plik:

**ansible-vault view --vault-password-file=passwd** szyfrowany.yml

# ANSIBLE VAULT

```
[marcin@server1 ~]$ echo "altkom" > passwd
[marcin@server1 ~]$ ansible-vault create --vault-password-
file=passwd tajne.yml
[marcin@server1 ~]$ cat tajne.yml
$ANSIBLE_VAULT;1.1;AES256
38393636323230616162366237613032623532666435306466353061
323934313463373331643565 ...
[marcin@server1 ~]$ chmod 0600 passwd
[marcin@server1 ~]$ ls -l passwd
-rw-----. 1 marcin marcin 7 Feb  6 18:57 passwd
[marcin@server1 ~]$ ansible-vault view --vault-password-
file=passwd tajne.yml
bardzo tajne dane
```

# ANSIBLE VAULT

- Wywołanie playbooka zawierającego część zaszyfrowaną (np. zewnętrzny plik z hasłami userów, których konta tworzymy) zakończy się niepowodzeniem, dopóki nie użyjemy przełącznika **--vault-id @prompt** dla podania hasła w trybie interaktywnym:

```
ansible-playbook -i inventory.cfg --vault-id @prompt  
playbook.yml
```

- lub powołując się na plik z zapisanymi hasłami:  
**--vault-password-file= :**

```
ansible-playbook -i inventory.cfg --vault-password-file=passwd  
playbook.yml
```



# ANSIBLE VAULT

- Oprócz **ansible-vault** istnieją też inne rozwiązania, mogące służyć do szyfrowania wrażliwych plików Ansible:
  - HashiCorp Vault
  - Microsoft Azure Key Vault
  - AWS Key Management Service

# ANSIBLE - PARALLELISM

- Standardowo Ansible wykonuje zadania playbooka zgodnie z kolejnością, jedno po drugim. Każde z tych zadań wykonywane jest równocześnie na wszystkich zarządzanych hostach.
- Dopiero po wykonaniu zadania nr 1 na wszystkich hostach Ansible przechodzi do zadania nr 2 itd.
- Parametr określający na ilu jednocześnie hostach pracuje Ansible to **forks** = zdefiniowany w pliku konfiguracyjnym **ansible.cfg**.
- Domyślnie Ansible pracuje na 5 hostach jednocześnie.

**forks = 5**

- Parametr ten można zmienić w pliku **ansible.cfg** lub bezpośrednio w poleceniu **ad-hoc** lub **ansible-playbook**:
  - `ansible centos -i inventory.cfg -f 10 -m ping`
  - `ansible-playbook -i inventory.cfg --force 10 playbook.yml`

## ANSIBLE - SERIAL

- Jeśli chcemy, aby Ansible wykonał wszystkie zawarte w playbooku zadania (od zadania nr 1 do zadania ostatniego) na określonej liczbie hostów, a następnie na pozostałych, możemy użyć parametru **serial** zdefiniowanego bezpośrednio w playbooku.

- name: update klastra www

hosts: centos

**serial: 3**

tasks:

- name: update pliku index.html

...

Po wykonaniu całego playbooka na pierwszych 3 hostach, Ansible będzie kontynuował na pozostałych.

## ANSIBLE - IMPORT i INCLUDE

- Do głównego playbooka możemy dołączyć zewnętrzne playbooksi (importując w ten sposób wszystkie zadania w nich skonfigurowane) za pomocą parametrów:
  - **import\_playbook**  
zadania zaimportowane zostaną wstępnie przetworzone zanim wykona się playbook główny
  - **include\_playbook**  
działa dynamicznie, zadania zaimportowane będą dołączane w trakcie działania playbooka głównego

# ANSIBLE - IMPORT i INCLUDE

- Na podobnej zasadzie działa także importowanie plików zawierających jedynie **zadania** do wykonania:
  - **import\_tasks**
  - **include\_tasks**
- Dyrektywy te muszą być umiejscowione w odpowiednim miejscu playbooka:

hosts: centos

tasks:

- **import\_tasks**: webserwer.yml
- **include\_tasks**: database.yml

## PRZYSPIESZANIE ANSIBLE - PIPELINING

- Aby uzyskać informacje o szybkości wykonywania zadań można w `ansible.cfg` w sekcji `[default]` ustawić parametr:  
**`callback_whitelist = profile_tasks`**
- W celu przyśpieszenia działania playbooków możemy zastosować **pipelining**. Podczas wykonywania zadań playbooksa plik jest przesyłany na hosty zarządzane i dopiero wtedy wykonywany.
- Poprzez zastosowanie **pipelining** komendy zadań playbooksa są wykonywane bezpośrednio (bez konieczności transferu plików) jako opcje połączenia ssh.
- Aby włączyć **pipelining** w sekcji `[ssh_connection]` `ansible.cfg` należy ustawić parametr:

```
[ssh_connection]  
pipelining = True
```

## PRZYSPIESZANIE ANSIBLE - moduł MITOGEN

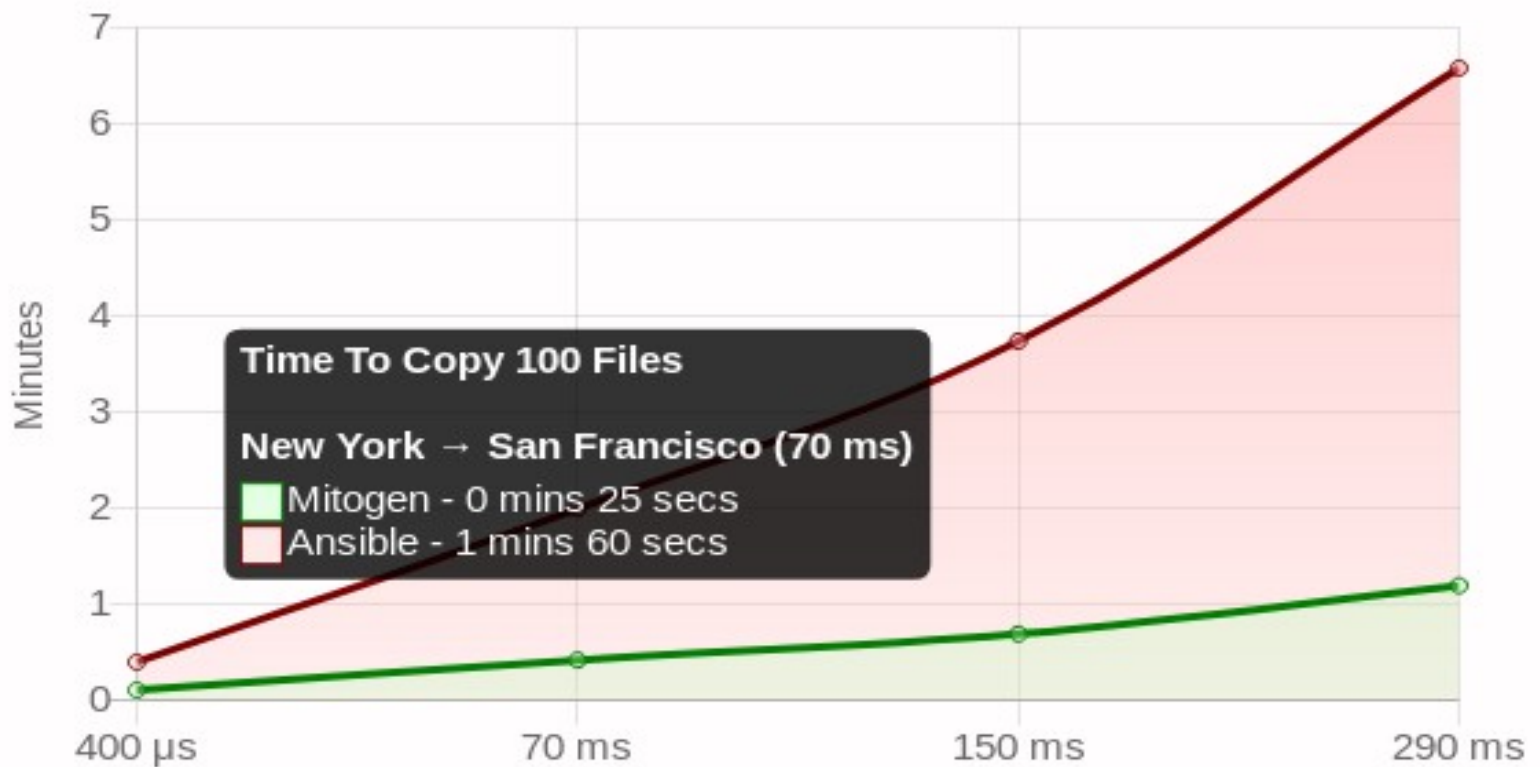
- Innym sposobem na przyspieszenie działania Ansible jest użycie dodatkowego modułu **Mitogen**.
- **Mitogen** przejmuje kontrolę nad warstwą połączenia z zarządzanymi hostami oraz sposobem działania modułów Ansible.
- Potrafi przyśpieszyć połączenia nawet siedmiokrotnie.
- Aby użyć **Mitogen** należy pobrać moduł ze strony producenta:  
<https://networkgenomics.com/ansible/>  
oraz rozpakować go (najlepiej w katalogu z projektami Ansible)
- Następnie w sekcji **[default]** ansible.cfg dodajemy 2 wiersze:

**[default]**

```
strategy_plugins=./plugins/mitogen-0.2/ansible_mitogen/plugins/strategy  
strategy = mitogen_linear
```

# PRZYSPIESZANIE ANSIBLE - moduł MITOGEN

- Wykres obrazujący wydajność modułu Mitogen:





## ANSIBLE - ROLES

- Ogromnym zbiorem gotowych do wykorzystania ról jest portal Ansible Galaxy, znajdujący się pod adresem:  
**[www.galaxy.ansible.com](http://www.galaxy.ansible.com)**
- Role w Ansible Galaxy tworzone są przez społeczność i są one dostępne bezpłatnie dla każdego
- Poszukując roli dla zadania, które mamy wykonać, możemy przeszukiwać Ansible Galaxy pod kątem:
  - ilości pobrań,
  - ilości punktów (każda rola może być oceniana)
  - daty ostatniej modyfikacji itp.
- Role możemy przechowywać także lokalnie oraz tworzyć własne

## ANSIBLE - ROLES

- Przy automatyzacji większych projektów playbooki znacznie się rozrastają i pojawia się konieczność podzielenia playbooka na kilka plików zewnętrznych, dołączanych do głównego playbooka (np. plik ze zmiennymi, poświadczeniami, inwentaryzacyjne itp.).
- Taki podział jednego wielkiego playbooka na kilka plików sprzyja czytelności oraz zachowaniu odpowiedniego porządku. Każda rola zachowuje odpowiednią strukturę katalogów.
- Przydaje się to szczególnie w pracy zespołowej.
- Aby usprawnić ten proces w Ansible pojawiły się **ROLES**

# ANSIBLE - ROLES


Browser address bar: [https://galaxy.ansible.com/search?deprecated=false&keywords=apache&order\\_by=-download\\_c](https://galaxy.ansible.com/search?deprecated=false&keywords=apache&order_by=-download_c)

Navigation: Home, Search, Community

Galaxy Header: About, Help, Documentation, Login

Search Results for 'apache':

- phar**
- Roles** 2463



geerlingguy

### php

PHP for RedHat/CentOS/Fedora/Debian /Ubuntu.


build passing

✓ 4.8 / 5 Score

3442710 Downloads

Last Imported: 4 days ago

development drupal fpm joomla language magento php web wordpress



geerlingguy

### apache

Apache 2.x for Linux.


build passing

✓ 4.8 / 5 Score

3244829 Downloads

Last Imported: a day ago

apache html httpd web webserver



yatesr

### timezone


Role for managing timezone.

✓ 5 / 5 Score

874888 Downloads

Last Imported: 4 months ago

system



bootstrap

### bootstrap

Prepare your system to be managed by Ansible.

build passing

#### Popular Clouds

GenericLinux	849
opensuse	3,405
GenericUNIX	543
ArchLinux	514
MacOSX	3,058
Amazon	3,801
Alpine	495
google	10
azure	8
vmware	7
amazon	4
openstack	3
ovirt	3

# ANSIBLE - ROLES

- W zamierzeniu każda rola powinna odpowiadać konkretnemu obszarowi w systemie (np. obsługa serwisów: web, database itp.)
- Taka standaryzacja powoduje, że każda rola ma odpowiednią strukturę plików. Katalogi:
  - **defaults** – zawiera plik main.yml z domyślnymi zmiennymi i ich wartościami
  - **handlers** – zawiera main.yml z definicjami handlers
  - **tasks** - zawiera main.yml z zadaniami do wykonania
  - **files** – zawiera pliki statyczne
  - **meta** – zawiera main.yml z informacjami o roli, autorach, licencji, dystrybucjach na jakich działa oraz zależnościach
  - **vars** – zawiera main.yml z definicjami zmiennych

## ANSIBLE - ROLES

- Zmienne zawarte w danej roli mogą być nadpisywane w playbooku głównym
- Niekiedy playbook służący do automatyzacji grupy hostów może odwoływać się do kilku **ROLES**
- Ogromna ilość gotowych **ROLES** zawiera serwis Ansible Galaxy <https://galaxy.ansible.com>, utrzymywany przez społeczność.  
Zamieszczone tam role są oceniane i odpowiednio punktowane.  
Są one całkowicie darmowe do powszechnego wykorzystania.  
Każda z ról umieszczonych na Ansible Galaxy zawiera także opis instalacji i wdrożenia. Na Ansible Galaxy możemy także umieszczać swoje rozwiązania w postaci ról.
- W rolach bardzo przydatne są wzorce Jinja2 dzięki swojej elastyczności.

## ANSIBLE - ROLES

- Zmienne użyte w plikach **ROLES** mają nacechowanie:
- Najwyższy priorytet mają zmienne zdefiniowane w pliku **vars/main.yml**. Zmienne te nie mogą zostać nadpisane
- Domyślne zmienne zapisane w pliku **defaults/main.yml** mają niższy priorytet i mogą być nadpisane przez inne zmienne umiejscowione np. w **vars/main.yml**.
- Domyślnych zmiennych używamy, jeśli chcemy, by zostały nadpisane przez główny playbook. Wtedy tworzą swego rodzaju wzorzec, który najlepiej sprawdza się np. w pętli.

# ANSIBLE - ROLES

- Użycie **ROLES** w playbooku:

---

- name: role w playbooku

become: true

hosts: centos

**roles:**

- rola\_apache

- rola\_mariadb

# ANSIBLE - ROLES

- Poszczególne **ROLES** mogą się odwoływać do innych **ROLES**.
- Zależności takie są umieszczane w pliku **meta/main.yml**:

---

dependencies:

- { roles: serwer\_www, port: 80 }
- { roles: baza\_danych, dbname: altkom, admin\_user: marcin }



# ANSIBLE - ROLES

- Kolejność wykonywania zadań:
  1. zadania umieszczone w **ROLES**
  2. zadania umieszczone w playbooku
- Aby zarządzać kolejnością wykonywania zadań mamy możliwość skorzystania z:
  - **pre\_tasks** – zadania playbooka wyknają się przed zadaniami roli
  - **post\_tasks** – zadania playbooka wykonają się po wszystkich zadaniach z danej roli

# ANSIBLE - ROLES

- Dla zobrazowania kolejności wykonywania zadań:
- ---
- - hosts: centos
- **pre\_tasks:**
  - - debug:
  - msg: wystartowałem
- **roles:**
  - - role\_1
- **tasks:**
  - - debug:
  - msg: 'w trakcie pracy'
- **post\_tasks:**
  - - debug:
  - msg: 'koniec pracy';

# ANSIBLE - TROUBLESHOOTING

- Domyślnie żadne działania ansible nie są nigdzie logowane.
- Wyniki działań są przekierowywane na standardowe wyjście **stdout**
- Aby skonfigurować logowanie zdarzeń Ansible, należy ustawić opcję **log\_path** w pliku **ansible.cfg** w sekcji **[default]**:

**[default]**

**log\_path = error.log**

lub zmienną **\$ANSIBLE\_LOG\_PATH**

- Zaleca się ustawianie pliku logowania Ansible w bieżącym katalogu roboczym projektu
- Logowanie w katalogu **/var/log** wymaga dodatkowych praw dostępu dla użytkowników, którzy nie posiadają prawa roota

# ANSIBLE - TROUBLESHOOTING

- Najczęstsze problemy Ansible:
- Problem w połączeniu z hostami zarządzanymi:
  - hosty niedostępne
  - niewłaściwe uprawnienia
- Błędy syntaktyczne – można je sprawdzić za pomocą:
  - parametru **--syntax-check**  
ansible-playbook --syntax-check playbook.yml
  - parametru **-C** lub **--check**  
wywołanie playbooka bez wprowadzania żadnych zmian na hostach zarządzanych

# ANSIBLE - TROUBLESHOOTING

- W celu znalezienia błędu w wykonywanym playbooku lub komendzie ad-hoc należy przeprowadzić analizę wyniku:
- Nagłówek **PLAY** pokazuje, który play jest uruchomiony
- Nagłówek **TASKS** pokazuje aktualne zadanie playbooka
- Nagłówek **PLAY RECAP** pokazuje sumaryczny wynik naszego playbooka
- Aby uzyskać szczegółowe informacje odnośnie każdego elementu playbooka można się posłużyć przełącznikiem **-v**

# ANSIBLE - TROUBLESHOOTING

- Stopnie szczegółowości:
- **-v** – zwraca dane ( output )
- **-vv** – zwraca dane ( output i input )
- **-vvv** – zwraca dodatkowo informacje o połączeniu z zarządzanymi hostami
- **-vvvv** – najbardziej szczegółowe informacje pozwalające debugować playbooka, zwraca dodatkowe informacje o użytych pluginach, użytkownikach itp.

# ANSIBLE - TROUBLESHOOTING

- Wywołanie playbooka z parametrem **--step** spowoduje, że Ansible zapyta przed każdym zadaniem, czy ma je wykonać:
- `ansible-playbook --step playbook.yml`

```
[marcin@server1 ~]$ ansible-playbook -i inventory.cfg --step serwerwww.yml
PLAY [uruchomienie uslugi www]
*****

Perform task: TASK: Gathering Facts (N)o/(y)es/(c)ontinue: y
Perform task: TASK: Gathering Facts (N)o/(y)es/(c)ontinue:
*****

TASK [Gathering Facts]
*****

ok: [192.168.10.153]
ok: [192.168.10.152]
Perform task: TASK: instalka httpd (N)o/(y)es/(c)ontinue: y
```

# ANSIBLE - TROUBLESHOOTING

- Playbooki możemy także wykonywać od dowolnego zadania za pomocą przełącznika **--start-at-task**:

`ansible-playbook --start-at-task="Tworzenie użytkowników"`

- Wcześniejsze zadania zostaną pominięte



# ANSIBLE - TROUBLESHOOTING

- **Best practices:**

- używanie klucza **name** przy każdym zadaniu  
(nie jest konieczne do skutecznego wykonania zadania)
- stosowanie komentarzy
- używanie pustych wierszy w celu poprawienia czytelności playbooków
- tworzenie małych playbooków, podział na konkretne zadania
- tworzenie osobnych plików dla odrębnych zadań, zmiennych itp.  
Dołączanie za pomocą **include** do playbooka głównego
- używanie modułu **debug** do analizy wyników zadań, wartości zmiennych itp.

# ANSIBLE - TROUBLESHOOTING

- **Best practices:**

- sprawdzanie wartości zmiennej za pomocą modułu **debug**:

- debug:

- var: output

- verbosity: 2**

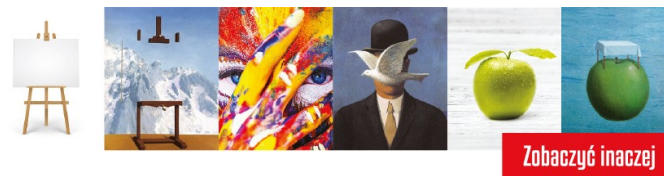
- **verbosity: 2** jest równoznaczne **-vv**

# ANSIBLE - TROUBLESHOOTING

- Podczas debugowania playbooków pomocne mogą być także moduły:
  - **uri** – łączy się z określonym URL i może sprawdzić zawartość np. serwisu httpd
  - **script** – pozwala na transfer i wykonanie na zarządzanym hoście określonego skryptu
  - **stat** – pozwala zbadać obecność plików i katalogów na zarządzanym hoście, a także zwrócić sumę kontrolną
- Ponadto w troubleshootingu pomocne będą pojedyncze komendy **ad-hoc**, np.:  
ansible centos -i inventory.cfg -m shell -a 'df -h'

## AWX - Instalacja

- `yum -y install epel-release`
- `yum install git ansible python-pip`
- `git clone https://github.com/ansible/awx.git`
- `docker --version`
- `pip install docker-compose==1.9.0`
- `cd awx/installer`
- `ansible-playbook -i inventory installer.sh`



# Zapraszamy do współpracy

**ALTKOM AKADEMIA**

ul. Chłodna 51,

00-867 Warszawa

Telefon: (+48 22) 460 99 99,

warszawa@altkom.pl