Tydzień 3 do EX294: Ansible od instalacji do szablonów

Poziom: średnio

Dotyczy: ansible-core 2.16 (RHEL 10), RHEL 9 i 10 jako węzły zarządzane, cele EX294

Czas: 6 dni po 1,5 do 2 godzin

Dni 18 do 23 programu: instalacja i konfiguracja Ansible, inwentarz i polecenia ad hoc, pierwsze playbooki, zmienne i fakty, pętle, warunki i handlery oraz szablony Jinja2. Dalsze dni (błędy, sekrety, role, automatyzacja zadań RHCSA) opisuje następna strona.

ZAŁOŻENIA SCENARIUSZA

węzłem sterującym jest `rhlab01` (RHEL 10, ansible-core 2.16 z AppStream), a węzłami zarządzanymi `rhlab02` (RHEL 10) oraz `rhlab03` i `rhlab04` (RHEL 9), wszystkie w sieci 10.20.150.0/24 z strony Plan 30 dni i laboratorium RHEL w Zielonej. Strona EX294 Red Hat mówi o RHEL 9, więc jeśli chcemy dokładnie tych warunków, instalujemy `rhlab03` jako węzeł sterujący na RHEL 9.

Skróty użyte na tej stronie (11)

Skrót	Co znaczy
AAP	Ansible Automation Platform: produkt Red Hat z interfejsem webowym, kontrolą dostępu i środowiskami wykonawczymi dla Ansible
DNS	Domain Name System: książka telefoniczna internetu, zamienia nazwy na adresy IP
EPEL	Extra Packages for Enterprise Linux: dodatkowe repozytorium pakietów dla RHEL i klonów
EX294	numer egzaminu praktycznego RHCE w ścieżce Ansible: kandydat pisze playbooki, które Red Hat uruchamia na świeżych systemach
JSON	JavaScript Object Notation: tekstowy format danych z klamrami i cudzysłowami
RHCSA	Red Hat Certified System Administrator: certyfikat Red Hat za praktyczną obsługę systemu RHEL, zdaje się go egzaminem EX200
RHEL	Red Hat Enterprise Linux: komercyjna dystrybucja Linuksa
SELinux	Security-Enhanced Linux: dodatkowa warstwa uprawnień w jądrze RHEL

Skrót	Co znaczy
SSH	Secure Shell: szyfrowane zdalne logowanie do serwera
UID	User ID: numer użytkownika w Linuksie
YAML	YAML Ain't Markup Language: format plików konfiguracyjnych oparty na wcięciach

Fiszka

Komendy wykonujemy w bash jako `tomek` na węźle sterującym, w katalogu projektu `~/ansible`. Nazwy takie jak `NazwaPlaybooka` czy `NazwaGrupy` zastępujemy własnymi. Cele pochodzą ze strony Red Hat o egzaminie EX294, stan na 10 października 2026.

◫◫ ZMIENIA SYSTEM

Instalacja ansible-core z repozytorium systemu: dnf install

```
sudo dnf install ansible-core
```

Pakiet jest w AppStream. Zawiera `ansible-playbook`, `ansible-doc`, `ansible-galaxy` i moduły z przestrzeni `ansible.builtin`, a moduły z kolekcji, na przykład `ansible.posix`, są osobno.

◫◫ **dnf** = Dandified YUM: menedżer pakietów RHEL/Rocky

◫◫ TYLKO ODCZYT

Sprawdzenie wersji i ścieżek: ansible --version

```
ansible --version
```

Wynik pokazuje wersję `ansible-core`, używany plik `ansible.cfg` i ścieżki do modułów oraz kolekcji. Jeżeli plik konfiguracji ma wartość `None`, Ansible nie czyta naszego projektu, bo jesteśmy w złym katalogu.

◫◫ **ansible** = bazowe polecenie Ansible do wykonania pojedynczej komendy ad-hoc na grupie maszyn, bez playbooka

◫◫ TYLKO ODCZYT

Pokazanie ustawień zmienionych względem domyślnych: ansible-config

```
ansible-config dump --only-changed
```

Wypisuje tylko te ustawienia, które nasz plik `ansible.cfg` zmienił. To najszybsza kontrola, czy konfiguracja w ogóle została wczytana.

□□ **ansible-config** = od Ansible i configuration: wypisuje bieżącą konfigurację Ansible i pokazuje, które ustawienia zmienił plik ansible.cfg

□□ **ZMIENIA SYSTEM**

Zainstalowanie ansible-navigator na laboratorium: pip install

```
sudo dnf install python3-pip
python3 -m pip install --user ansible-navigator
```

W systemie z subskrypcją Ansible Automation Platform pakiet `ansible-navigator` instalujemy z jej repozytorium. W laboratorium bez tej subskrypcji wystarczy `pip`, a narzędzie uruchamiamy bez kontenera (`--ee false`).

□□ **dnf** = Dandified YUM: menedżer pakietów RHEL/Rocky · **python3** = uruchom interpreter Pythona w wersji 3 albo wskazany skrypt

□□ **TYLKO ODCZYT**

Rozesłanie klucza SSH na węzły zarządzane: ssh-copy-id

```
for h in rhlab02 rhlab03 rhlab04; do ssh-copy-id -i ~/.ssh/id_ed25519.pub tomek@"$h";
done
```

Klucz mamy już z dnia 1, ale kroku nie pomijamy w ćwiczeniu, bo egzamin ma osobny cel o rozsyłaniu kluczy.

□□ **SSH** = Secure Shell: szyfrowane zdalne logowanie do serwera

□□ **TYLKO ODCZYT**

Wyświetlenie inwentarza jako drzewa: ansible-inventory --graph

```
ansible-inventory --graph --vars
```

Pokazuje grupy, ich członków i zmienne. Wynik `--list` w formacie JSON zawiera wszystko, łącznie ze zmiennymi hostów.

JSON = JavaScript Object Notation: tekstowy format danych z klamrami i cudzysłowami · **ansible-inventory** = od Ansible i inventory: pokazuje inwentarz jako drzewo grup lub jako JSON razem ze zmiennymi hostów

TYLKO ODCZYT

Sprawdzenie łączności ze wszystkimi węzłami: ansible -m ansible.builtin.ping

```
ansible all -m ansible.builtin.ping
```

Moduł `ping` sprawdza SSH, Pythona i uprawnienia, a nie sieć. Wynik `pong` oznacza, że można zlecać zadania.

SSH = Secure Shell: szyfrowane zdalne logowanie do serwera · **ansible** = bazowe polecenie Ansible do wykonania pojedynczej komendy ad-hoc na grupie maszyn, bez playbooka

TYLKO ODCZYT

Wykonanie polecenia ad hoc z podniesieniem uprawnień: ansible -b

```
ansible NazwaGrupy -b -m ansible.builtin.dnf -a "name=NazwaPakietu state=present"
```

Opcja `-b` to `become`, czyli wykonanie jako root. Polecenia ad hoc służą do jednorazowych zmian i szybkiego sprawdzania stanu.

ansible = bazowe polecenie Ansible do wykonania pojedynczej komendy ad-hoc na grupie maszyn, bez playbooka

TYLKO ODCZYT

Odczytanie dokumentacji modułu: ansible-doc

```
ansible-doc ansible.posix.firewalld
```

To jedyna dokumentacja, którą mamy na egzaminie. Sam przykład zadania wypisuje opcja `-s`, a listę modułów z częścią nazwy `ansible-doc -l | grep NazwaFragmentu`.

ansible-doc = od Ansible i documentation: pokazuje dokumentację modułu wraz z przykładami, bez dostępu do internetu

ZMIENIA SYSTEM

Sprawdzenie składni playbooka: --syntax-check

```
ansible-playbook NazwaPlaybooka.yml --syntax-check
```

Wykrywa błędy wcięć i nazw modułów bez łączenia się z węzłami. Wykonujemy go przed każdym pierwszym uruchomieniem.

ansible-playbook = uruchom scenariusz Ansible

TYLKO ODCZYT

Próba na sucho z podglądem zmian: --check --diff

```
ansible-playbook NazwaPlaybooka.yml --check --diff
```

Pokazuje, co playbook by zmienił, razem z różnicą w plikach, ale niczego nie zmienia. Nie każdy moduł wspiera ten tryb, na przykład `ansible.builtin.command` jest pomijany.

ansible-playbook = uruchom scenariusz Ansible

ZMIENIA SYSTEM

Wypisanie zadań playbooka: --list-tasks

```
ansible-playbook NazwaPlaybooka.yml --list-tasks
```

Przydaje się do ustalenia nazwy zadania dla `--start-at-task "Nazwa zadania"`, które pomija wszystko przed wskazanym zadaniem.

ansible-playbook = uruchom scenariusz Ansible

ZMIENIA SYSTEM

Przekazanie zmiennej z linii poleceń: -e

```
ansible-playbook NazwaPlaybooka.yml -e "NazwaZmiennej=Wartosc"
```

Zmienna z `-e` ma najwyższy priorytet i przesłania każdą inną definicję.

ansible-playbook = uruchom scenariusz Ansible

TYLKO ODCZYT

Odczytanie faktów węzła: setup z filtrem

```
ansible NazwaHosta -m ansible.builtin.setup -a "filter=ansible_distribution*"
```

Fakty to dane o węźle zebrane przed zadaniami: system, adresy, pamięć, dyski. Wynik bez filtra jest bardzo długi, więc zawężamy go.

ansible = bazowe polecenie Ansible do wykonania pojedynczej komendy ad-hoc na grupie maszyn, bez playbooka

TYLKO ODCZYT

Uruchomienie playbooka narzędziem ansible-navigator: run

```
ansible-navigator run NazwaPlaybooka.yml --ee false --mode stdout
```

Opcja `--ee false` wyłącza kontener, a `--mode stdout` wypisuje wynik jak zwykły `ansible-playbook`. Bez nich narzędzie otwiera interaktywny ekran tekstowy i uruchamia playbook w środowisku wykonawczym.

ansible-navigator = od Ansible i navigator: nowsze polecenie do uruchamiania playbooków, domyślnie w kontenerze zwanym środowiskiem wykonawczym

TYLKO ODCZYT

Wypisanie zainstalowanych kolekcji: ansible-galaxy collection list

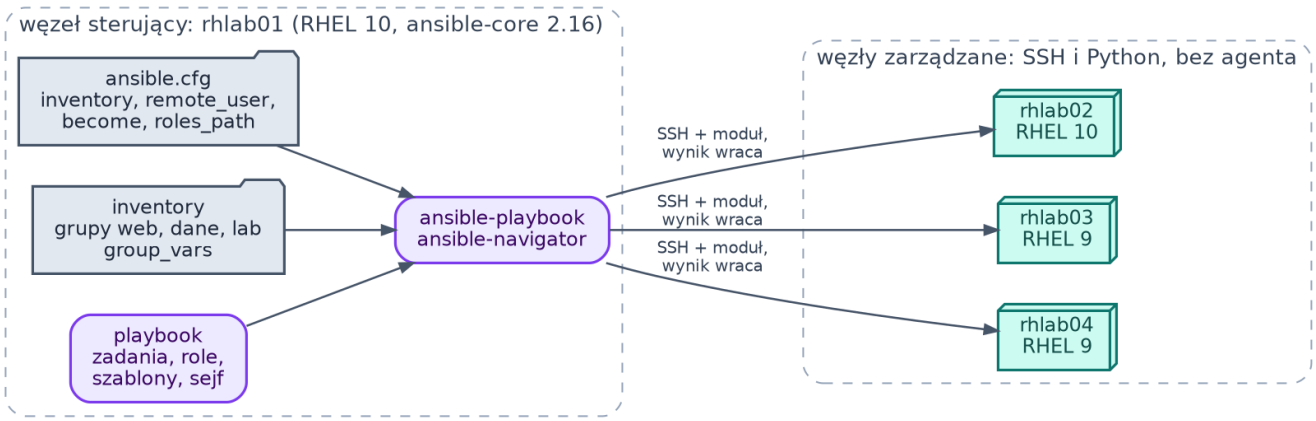
```
ansible-galaxy collection list
```

Kolekcja to paczka modułów, wtyczek i ról z nazwą w postaci `przestrzen.nazwa`, na przykład `ansible.posix`. Bez kolekcji moduł z jej przestrzeni nie zadziała.

ansible-galaxy = od Ansible i galaxy: polecenie do tworzenia szkieletów ról i kolekcji oraz ich instalowania ze źródeł i z archiwów

Jak to działa: węzły, narzędzia i pliki

Ansible działa po SSH: z jednej maszyny, węzła sterującego, wysyła do pozostałych, węzłów zarządzanych, małe programy w Pythonie zwane modułami, uruchamia je i odbiera wynik. Na węzłach zarządzanych nie instalujemy żadnego agenta, wystarczy SSH i Python.



Rysunek 1. Węzeł sterujący rhlab01 czyta ansible.cfg i inwentarz, łączy się po SSH z rhlab02, rhlab03 i rhlab04 i uruchamia tam moduły (kliknięcie otwiera pełny rozmiar)

Pojęcie: węzeł sterujący i węzeł zarządzany. Węzeł sterujący to maszyna, na której zainstalowano Ansible i z której uruchamiamy playbooki. Węzły zarządzane to maszyny, które konfigurujemy; według dokumentacji RHEL 10 węzeł sterujący z ansible-core 2.16 obsługuje tylko węzły zarządzane z RHEL 9 i RHEL 10, nie starsze.

Pojęcie: ansible-core i Ansible Automation Platform. ansible-core to sam silnik: polecenia `ansible`, `ansible-playbook` i moduły podstawowe. Ansible Automation Platform (AAP) to produkt Red Hat, który dokłada interfejs webowy, kontrolę dostępu, kolekcje i środowiska wykonawcze. Według strony Red Hat domyślny ansible-core w AAP 2.5, 2.6 i 2.7 ma wersję 2.16, a środowiska wykonawcze z wersją 2.18 są też wspierane.

> **Pojęcie: ansible-navigator i środowisko wykonawcze.** ansible-navigator to nowsze polecenie do uruchamiania playbooków, które domyślnie robi to w kontenerze zwanym środowiskiem wykonawczym (execution environment): obrazie z ansible-core i kolekcjami. Cele EX294 wymieniają zarówno `ansible-navigator`, jak i `ansible-playbook`, więc umiemy oba. Konfiguracja navigatora leży w pliku `ansible-navigator.yml`.

Dzień 18: instalacja i konfiguracja Ansible

> **Z ŻYCIA WZIĘTE**
>
> Wtorek, 10:00. Ola otwiera notatnik z dwoma słowami: SSH i Python. „Ansible nie jest magią”, mówi. „To jest `ssh` z pętlą i dobrą pamięcią”. Tomek instaluje `ansible-core` na `rhlab01`, tworzy katalog projektu i po raz pierwszy widzi odpowiedź `pong` od trzech maszyn naraz.

Cele egzaminu

- tworzenie i modyfikacja plików `ansible.cfg` i `ansible-navigator.yml`
- konfiguracja węzłów zarządzanych: rozsyłanie kluczy SSH, podniesienie uprawnień
- instalacja pakietów potrzebnych do używania Ansible

Czytamy w bazie

- Instalacja Ansible i pierwszy inwentarz Zielonej
- Ansible: podstawy operacyjne
- SSH: klucze, ssh-agent, config, hartowanie

Krótko o tym, czego baza nie ma

Strona o instalacji Ansible w Zielonej opisuje instalację na stacji administratora i `app01` przez `pipx` i EPEL. Na egzaminie liczy się inna droga: `ansible-core` z repozytorium systemu, projekt w katalogu domowym z własnym `ansible.cfg` i nawiązanie połączenia z węzłami przez klucz SSH i `sudo` bez hasła. Ten sam `ansible.cfg` konfiguruje połączenie, uprawnienia i ścieżki ról.

W naszych ćwiczeniach węzły zarządzane są w pliku `/etc/hosts` z dnia 1, więc nazwy `rhlab02` do `rhlab04` rozwiązują się bez DNS.

Ćwiczenia

1. Instalujemy `ansible-core` na `rhlab01`, sprawdzamy wersję i pokazujemy, że w katalogu domowym Ansible nie czyta jeszcze żadnego pliku konfiguracyjnego.

Odpowiedź

```
sudo dnf -y install ansible-core
ansible --version
ansible-config dump --only-changed
```

Przy świeżej instalacji wiersz `config file` wynosi `/etc/ansible/ansible.cfg` albo `None`, a `--only-changed` nic nie wypisuje. Zmieni się to w następnym ćwiczeniu.

1. Tworzymy katalog projektu `~/ansible` z plikiem `ansible.cfg`, który ustawia inwentarz `./inventory`, użytkownika `tomek`, ścieżkę ról `./roles` i domyślne podnoszenie uprawnień przez `sudo`. Tworzymy też inwentarz z grupami `web` (`rhlab04`) i `dane` (`rhlab02`, `rhlab03`).

Odpowiedź

```
mkdir -p ~/ansible && cd ~/ansible
tee ansible.cfg <<'EOF'
[defaults]
inventory = ./inventory
remote_user = tomek
roles_path = ./roles

[privilege_escalation]
become = True
become_method = sudo
become_user = root
become_ask_pass = False
EOF
tee inventory <<'EOF'
[web]
rhlab04
```

```
[dane]
rhlab02
rhlab03

[lab:children]
web
dane
EOF
ansible-config dump --only-changed
ansible-inventory --graph
```

Plik `ansible.cfg` w katalogu bieżącym ma pierwszeństwo przed `~/.ansible.cfg` i `/etc/ansible/ansible.cfg`, więc Ansible uruchamiamy zawsze z katalogu projektu. Przykład wyniku `--graph`:

```
@all:
  |--@ungrouped:
  |--@lab:
  |  |--@web:
  |  |  |--rhlab04
  |  |--@dane:
  |  |  |--rhlab02
  |  |  |--rhlab03
```

1. Rozsyłamy klucz SSH na trzy węzły zarządzane i dajemy kontu `tomek` na nich `sudo` bez hasła, a potem sprawdzamy `ping` do wszystkich.

Odpowiedź

```
for h in rhlab02 rhlab03 rhlab04; do
    ssh-copy-id -i ~/.ssh/id_ed25519.pub tomek@"$h"
    ssh -t tomek@"$h" 'echo "tomek ALL=(ALL) NOPASSWD: ALL" | sudo tee
/etc/sudoers.d/tomek'
done
ansible all -m ansible.builtin.ping
ansible all -m ansible.builtin.command -a "id" -b
```

Drugie polecenie ssh zadaje pytanie o hasło `sudo` jeden raz na węzeł, bo opcja `-t` przydziela terminal. Ostatnia komenda ma pokazać `uid=0(root)` na każdym węźle, co potwierdza działanie `become`.

1. Konfigurujemy `ansible-navigator`: plik `ansible-navigator.yml` w katalogu projektu ma wyłączać kontener, wypisywać wynik jak zwykły `ansible-playbook` i nie zapisywać artefaktów. Uruchamiamy nim playbook ze sprawdzeniem łączności.

Odpowiedź

```
python3 -m pip install --user ansible-navigator
tee ansible-navigator.yml <<'EOF'
---
ansible-navigator:
  execution-environment:
    enabled: false
  mode: stdout
  playbook-artifact:
    enable: false
EOF
tee ping.yml <<'EOF'
---
- name: Sprawdzenie polaczenia
  hosts: all
  gather_facts: false
  tasks:
    - name: Ping
      ansible.builtin.ping:
EOF
ansible-navigator run ping.yml
ansible-playbook ping.yml
```

Gdy kontener jest włączony, w pliku podajemy nazwę obrazu w polu `execution-environment: image:` i polityki pobierania w `pull: policy:`. Obraz środowiska wykonawczego pochodzi z rejestru Red Hat i wymaga konta, więc dokładną nazwę sprawdzamy w dokumentacji AAP. Na egzaminie zwykle dostajemy gotowy.

Dzień 19: inwentarz, polecenia ad hoc, ansible-doc i kolekcje

> **Z ŻYCIA WZIĘTE**
>
> Środa, 11:30. Marek pyta, czy dałoby się sprawdzić w pięć sekund, na którym z dwudziestu serwerów skaner nie ma zainstalowanego klienta czasu. Tomek zamiast pisać skrypt układa inwentarz z grupami i jedną komendą ad hoc pyta wszystkie maszyny naraz. Ola pokazuje mu jeszcze, że nazwę modułu do tego zadania da się znaleźć bez internetu, w samym `ansible-doc`.

Cele egzaminu

- statyczne inwentarze i grupy hostów
- polecenia ad hoc i znajdowanie modułów z kolekcji
- korzystanie z dokumentacji wbudowanej (`ansible-doc`)

Czytamy w bazie

- Ansible: podstawy operacyjne: sekcje o inwentarzu, grupach i zmiennych
- Instalacja Ansible i pierwszy inwentarz Zielonej

Krótko o tym, czego baza nie ma

> **Pojęcie: wzorzec hostów.** Napis, który wskazuje maszyny z inwentarza: `all` to wszystkie, `lab:!web` to grupa `lab` bez grupy `web`, `web:&dane` to część wspólna, `rhlab0*` to nazwy zaczynające się od wzorca. Wzorzec podajemy w poleceniu ad hoc, w polu `hosts:` playbooka albo w opcji `--limit`.

Zmienne grupy zapisujemy albo w sekcji `[NazwaGrupy:vars]` inwentarza, albo w osobnym pliku `group_vars/NazwaGrupy.yml` obok niego. Drugi sposób jest czytelniejszy, gdy zmiennych jest kilka, i jest tym, którego używamy w pierwszym ćwiczeniu.

Ćwiczenia

1. Dodajemy do inwentarza zmienną `http_port=8080` dla grupy `web` w pliku `group_vars/web.yml` i pokazujemy ją w drzewie inwentarza.

Odpowiedź

```
cd ~/ansible
mkdir -p group_vars
tee group_vars/web.yml <<'EOF'
---
http_port: 8080
motd_tekst: "Serwer WWW Zielona lab"
EOF
ansible-inventory --graph --vars
ansible-inventory --host rhlab04
```

Przykład wyniku `--graph --vars` dla grupy `web`:

```
  |  |--@web:
  |  |  |--rhlab04
  |  |  |  |--{http_port = 8080}
  |  |  |  |--{motd_tekst = Serwer WWW Zielona lab}
```

Nazwa pliku w `group_vars` musi być zgodna z nazwą grupy, a Ansible czyta go tylko wtedy, gdy katalog leży obok inwentarza albo obok playbooka.

1. Wypisujemy hosty z grupy `lab`, z wyłączeniem grupy `web`, oraz część wspólną grup `dane` i `lab`, bez uruchamiania żadnego zadania.

Odpowiedź

```
ansible 'lab:!web' --list-hosts
ansible 'dane:&lab' --list-hosts
ansible 'rhlab0[2-3]' --list-hosts
```

Cudzysłowy chronią wzorzec przed powłoką, bo znaki `!` i `&` mają w niej własne znaczenie. Przykład wyniku pierwszej komendy:

```
  hosts (2):
    rhlab02
    rhlab03
```

1. Poleceniami ad hoc instalujemy `tmux` na grupie `dane`, sprawdzamy czas pracy każdego węzła i odczytujemy z faktów nazwę dystrybucji oraz wersję.

Odpowiedź

```
ansible dane -b -m ansible.builtin.dnf -a "name=tmux state=present"
ansible lab -m ansible.builtin.command -a "uptime -p"
ansible lab -m ansible.builtin.setup -a "filter=ansible_distribution*"
```

Przy drugiej instalacji tego samego pakietu wynik pokazuje `changed: false`, bo moduł `dnf` zmienia stan tylko wtedy, gdy trzeba. Tę cechę nazywamy idempotentnością i opisuje ją strona Idempotentność: skrypt, który wolno uruchomić dwa razy.

1. Bez internetu znajdujemy moduł do zarządzania przełącznikami SELinux (boolean), wypisujemy jego opcje i przykład użycia oraz sprawdzamy, jakie kolekcje są zainstalowane.

Odpowiedź

```
ansible-doc -l | grep -i seboolean
ansible-doc ansible.posix.seboolean
ansible-doc -s ansible.posix.seboolean
ansible-galaxy collection list
```

Przykład wyniku pierwszej komendy:

```
ansible.posix.seboolean       Toggles SELinux booleans
```

Gdy `ansible-galaxy collection list` nie pokazuje `ansible.posix`, kolekcję doinstalowujemy z pakietu systemu (`dnf list 'ansible-collection-*'`) albo przez `ansible-galaxy collection install ansible.posix`. Na egzaminie kolekcje są już zainstalowane w środowisku, ale sprawdzamy to na początku.

Dzień 20: pierwsze playbooki

Z ŻYCIA WZIĘTE

> Czwartek, 9:15. Ola chce serwer testowy ze stroną startową do pokazania klientowi i nie chce czekać, aż Tomek zrobi go ręcznie po raz piąty. Tomek zapisuje cztery kroki w playbooku: instalacja pakietu, uruchomienie usługi, plik strony, otwarcie zapory. Pierwszy przebieg zmienia cztery rzeczy, a drugi niczego, i Tomek wreszcie rozumie, co znaczy „deklaracja stanu”.

Cele egzaminu

- pisanie playbooków z użyciem najczęściej stosowanych modułów
- doprowadzanie systemów do określonego stanu
- uruchamianie playbooków przez `ansible-playbook` i `ansible-navigator`

Czytamy w bazie

- Ansible: podstawy operacyjne: sekcja o strukturze playbooka
- Instalacja Ansible i pierwszy inwentarz Zielonej: pierwszy playbook
- Idempotentność: skrypt, który wolno uruchomić dwa razy

Ćwiczenia

1. Piszemy playbook `web.yml`, który na grupie `web` instaluje `httpd`, włącza i uruchamia usługę, wgrywa stronę startową i otwiera w zaporze usługę `http` na stałe. Sprawdzamy składnię, robimy próbę na sucho i uruchamiamy dwa razy.

Odpowiedź

```
---
- name: Serwer WWW na grupie web
  hosts: web
  become: true
  tasks:
    - name: Instalacja httpd
      ansible.builtin.dnf:
        name: httpd
        state: present

    - name: Wlaczenie i start httpd
```

```
      ansible.builtin.service:
        name: httpd
        state: started
        enabled: true

    - name: Strona startowa
      ansible.builtin.copy:
        content: "Zielona lab: rhlab04\n"
        dest: /var/www/html/index.html
        owner: root
        group: root
        mode: "0644"

    - name: Otwarcie zapory dla http
      ansible.posix.firewalld:
        service: http
        permanent: true
        immediate: true
        state: enabled
```

```
ansible-playbook web.yml --syntax-check
ansible-playbook web.yml --check --diff
ansible-playbook web.yml
ansible-playbook web.yml
curl -s http://rhlab04
```

Przy drugim uruchomieniu w podsumowaniu `changed=0`, bo wszystko jest już w żądanym stanie. Wynik ostatniej komendy to treść strony startowej. Przykład podsumowania pierwszego uruchomienia (kształt wyniku):

```
PLAY RECAP *********************************************************
rhlab04      : ok=5    changed=4    unreachable=0    failed=0
```

1. Dopisujemy do `web.yml` drugi `play`, który na grupie `dane` wgrywa plik `/etc/lab-info` z nazwą hosta, i uruchamiamy go od drugiego play bez ponownego przechodzenia przez pierwszy.

Odpowiedź

```
- name: Znacznik na grupie dane
  hosts: dane
  become: true
  tasks:
    - name: Plik z nazwa hosta
      ansible.builtin.copy:
        content: "{{ inventory_hostname }}\n"
        dest: /etc/lab-info
        mode: "0644"
```

Ten blok dopisujemy do `web.yml` pod pierwszym play i uruchamiamy zawężenie do nowej grupy:

```
ansible-playbook web.yml --limit dane
ansible-playbook web.yml --list-tasks
ansible-playbook web.yml --start-at-task "Plik z nazwa hosta" --limit dane
```

Zawężenie `--limit` ogranicza hosty, a `--start-at-task` pomija zadania przed wskazanym. Gdy po dopisaniu play Ansible zgłasza błąd składni, najczęściej winne jest wcięcie: `- name:` drugiego play musi stać w tej samej kolumnie co w pierwszym.

1. Uruchamiamy ten sam playbook przez `ansible-navigator` i porównujemy z `ansible-playbook`.

Odpowiedź

```
ansible-navigator run web.yml
ansible-navigator run web.yml --mode stdout --ee false
```

Przy pliku `ansible-navigator.yml` z dnia 18 pierwsza komenda wypisuje wynik tak jak `ansible-playbook`. Bez pliku konfiguracyjnego navigator próbuje użyć kontenera i otwiera interaktywny ekran, z którego wychodzimy klawiszem Esc, a wynik pojedynczego zadania otwieramy numerem w dwukropku. Na egzaminie wybieramy to narzędzie, które znamy lepiej, o ile zadanie nie wskazuje któregoś z nich.

Dzień 21: zmienne, fakty i register

Z ŻYCIA WZIĘTE

Piątek, 10:40. Pan Zbigniew prosi o zestawienie, ile pamięci i jakiego systemu mają serwery testowe, „najlepiej na jednej kartce”. Tomek wie, że Ansible i tak zbiera te dane przed każdym zadaniem, więc pisze playbook, który je tylko wypisuje. Ola dopytuje, skąd zmienna `ansible_facts`, a Tomek odpowiada, że z fazy zbierania faktów na początku każdego play.

Cele egzaminu

- zmienne i fakty
- zapamiętywanie wyniku polecenia w zmiennej (`register`)
- praca z modułami `command` i `debug`

Czytamy w bazie

- Ansible: podstawy operacyjne: sekcje o inwentarzu, grupach i zmiennych
- Zmienne, parametry i argumenty

Krótko o tym, czego baza nie ma

> **Pojęcie: fakty.** Dane o węźle, które Ansible zbiera przed pierwszym zadaniem play: nazwa i wersja systemu, adresy, pamięć, dyski, urządzenia. Dostępne są w zmiennej `ansible_facts`, na przykład `ansible_facts['distribution']`. Zbieranie można wyłączyć przez `gather_facts: false`, co przyspiesza playbook, ale odbiera te dane.

Zmienne mogą pochodzić z wielu miejsc: sekcja `vars` play, plik `vars_files`, `group_vars`, `host_vars`, polecenie `-e` i `register`. Przy konflikcie wygrywa najbardziej szczegółowe źródło: zmienne z `-e` mają najwyższy priorytet, a domyślne zmienne roli najniższy. Tę kolejność, bez zapamiętywania całej listy, wystarczy znać w ogólnym zarysie.

Ćwiczenia

1. Piszemy playbook `raport.yml`, który na grupie `lab` wypisuje nazwę hosta, nazwę i wersję systemu oraz ilość pamięci w megabajtach.

Odpowiedź

```
---
- name: Raport o systemie
  hosts: lab
  gather_facts: true
  tasks:
    - name: Opis wezla
      ansible.builtin.debug:
        msg: >-
          {{ inventory_hostname }}:
          {{ ansible_facts['distribution'] }} {{ ansible_facts['distribution_version'] }},
          pamiec {{ ansible_facts['memtotal_mb'] }} MB
```

```
ansible-playbook raport.yml
```

Przykład wyniku (kształt):

```
ok: [rhlab04] => {
    "msg": "rhlab04: RedHat 9.6, pamiec 1770 MB"
}
```

Zapis `>-` w YAML łączy kolejne wiersze w jeden bez końcowego znaku nowej linii.

1. Zapamiętujemy w zmiennej wynik komendy `uptime -p` i wypisujemy tylko jej standardowe wyjście, a moduł `command` oznaczamy tak, by nie zgłaszał zmiany.

Odpowiedź

```
---
- name: Czas pracy wezlow
  hosts: lab
  gather_facts: false
  tasks:
    - name: Pobranie czasu pracy
      ansible.builtin.command: uptime -p
      register: czas_pracy
      changed_when: false

    - name: Wypisanie czasu pracy
```

```
    ansible.builtin.debug:
      var: czas_pracy.stdout
```

Zadanie z `command` zawsze zgłasza `changed`, jeżeli nie dodamy `changed_when: false`, więc podsumowanie wprowadzałoby w błąd. W zmiennej `czas_pracy` są też pola `rc`, `stderr` i `stdout_lines`.

1. Dodajemy do `group_vars/web.yml` zmienną `motd_tekst` (mamy ją z dnia 19) oraz zmienną `wlasciciel` w pliku `host_vars/rhlab04.yml` i wypisujemy obie na hoście `rhlab04`, sprawdzając, która z nich skąd pochodzi.

Odpowiedź

```
mkdir -p host_vars
tee host_vars/rhlab04.yml <<'EOF'
---
wlasciciel: ola
EOF
```

```
---
- name: Zmienne z inwentarza
  hosts: web
  gather_facts: false
  tasks:
    - name: Wypisanie zmiennych
      ansible.builtin.debug:
        msg: "{{ motd_tekst }}, wlasciciel {{ wlasciciel }}, port {{ http_port }}"
```

Zmienna `wlasciciel` pochodzi z `host_vars`, a pozostałe dwie z `group_vars`. Do podejrzenia wszystkich zmiennych hosta służy `ansible-inventory --host rhlab04`.

1. Tworzymy fakt własny na hoście `rhlab04`: plik `/etc/ansible/facts.d/zielona.fact`, z którego Ansible odczyta wartość jako `ansible_local`.

Odpowiedź

```
---
- name: Fakt wlasny
```

```
hosts: web
become: true
tasks:
  - name: Katalog faktow
    ansible.builtin.file:
      path: /etc/ansible/facts.d
      state: directory
      mode: "0755"

  - name: Plik faktu
    ansible.builtin.copy:
      content: |
        [lokalizacja]
        miejsce=Pamiatkowo
      dest: /etc/ansible/facts.d/zielona.fact
      mode: "0644"

  - name: Odswiezenie faktow
    ansible.builtin.setup:
      filter: ansible_local

  - name: Wypisanie faktu
    ansible.builtin.debug:
      var: ansible_local.zielona.lokalizacja.miejsce
```

Plik z rozszerzeniem `.fact` może mieć format INI albo JSON. Gdyby fakt miał być czytany bez dodatkowego `setup`, wystarczy, że plik istnieje przed zbieraniem faktów.

Dzień 22: pętle, warunki i handlery

> **□□ Z ŻYCIA WZIĘTE**
>
> Poniedziałek, 13:00. Praktykant dostaje konto na czterech serwerach, a Tomek ma w notatniku cztery linie z `useradd`. Ola pyta, co zrobi przy dziesiątej maszynie, i pokazuje mu zadanie z `loop`. Potem dokłada drugie pytanie: skąd Ansible ma wiedzieć, że po zmianie konfiguracji trzeba przeładować usługę, ale tylko wtedy, gdy zmiana naprawdę zaszła.

Cele egzaminu

- pętle i warunki w playbookach
- handlery, czyli zadania uruchamiane po zmianie
- praca z wieloma hostami i grupami

Czytamy w bazie

- Warunki i pętle
- Idempotentność: skrypt, który wolno uruchomić dwa razy

Krótko o tym, czego baza nie ma

> **Pojęcie: handler.** Zadanie, które wykonuje się dopiero na końcu play i tylko wtedy, gdy któreś zadanie je wywołało przez `notify` i naprawdę coś zmieniło. Dzięki temu serwer nie restartuje się przy każdym uruchomieniu playbooka, tylko po faktycznej zmianie konfiguracji. Handler uruchamia się raz, nawet jeśli wywołało go kilka zadań.

Pętla (`loop`) powtarza zadanie dla każdego elementu listy, a bieżący element jest dostępny jako `item`. Warunek (`when`) pomija zadanie, gdy wyrażenie jest fałszem; wyrażenie piszemy bez nawiasów klamrowych, bo `when` już jest wyrażeniem Jinja2.

Ćwiczenia

1. Piszemy playbook, który na grupie `dane` tworzy grupę `zespol` oraz dwa konta z listy (`ola` o UID 3101 i `marek` o UID 3102), oba w tej grupie, zapisane w zmiennej i obsłużone jedną pętlą.

Odpowiedź

```
---
- name: Konta zespolu
  hosts: dane
  become: true
  vars:
    konta:
```

```
      - { nazwa: ola, uid: 3101 }
      - { nazwa: marek, uid: 3102 }
  tasks:
    - name: Grupa zespolu
      ansible.builtin.group:
        name: zespol
        state: present

    - name: Konta w petli
      ansible.builtin.user:
        name: "{{ item.nazwa }}"
        uid: "{{ item.uid }}"
        groups: zespol
        append: true
        state: present
      loop: "{{ konta }}"
```

```
ansible-playbook konta.yml
ansible dane -m ansible.builtin.command -a "id ola"
```

Drugie uruchomienie nie zmienia niczego, bo konta już istnieją z tymi samymi UID. Dla listy prostych wartości zamiast słowników piszemy `loop: [a, b, c]` i odwołujemy się do `{{ item }}`.

1. Instalujemy `tmux` tylko na węzłach z co najmniej 1500 MB pamięci i drugi pakiet, `tree`, tylko na węzłach z RHEL 9.

Odpowiedź

```
---
- name: Pakiety zalezne od faktow
  hosts: lab
  become: true
  tasks:
    - name: tmux tylko przy wiekszej pamieci
      ansible.builtin.dnf:
        name: tmux
        state: present
      when: ansible_facts['memtotal_mb'] >= 1500
```

```
    - name: tree tylko na RHEL 9
      ansible.builtin.dnf:
        name: tree
        state: present
      when: ansible_facts['distribution_major_version'] == "9"
```

W wyniku pominięte zadania mają status `skipping`. Wersję główną porównujemy jako napis w cudzysłowie, bo fakt `distribution_major_version` jest tekstem. Gdy warunków jest kilka, piszemy listę pod `when`, a Ansible łączy je operatorem i.

1. Wgrywamy plik `/etc/ssh/sshd_config.d/20-lab.conf` z wyłączonym logowaniem roota i przeładowujemy `sshd` tylko wtedy, gdy plik się zmienił. Uruchamiamy playbook dwa razy.

Odpowiedź

```
---
- name: Ustawienia SSH
  hosts: lab
  become: true
  tasks:
    - name: Wylaczenie logowania roota przez SSH
      ansible.builtin.copy:
        dest: /etc/ssh/sshd_config.d/20-lab.conf
        content: |
          PermitRootLogin no
        owner: root
        group: root
        mode: "0644"
      notify: Przeladowanie sshd

  handlers:
    - name: Przeladowanie sshd
      ansible.builtin.service:
        name: sshd
        state: reloaded
```

Przy pierwszym uruchomieniu w wyniku widać wykonanie handlera, a przy drugim go nie ma, bo plik się nie zmienił. Gdy handler ma się wykonać wcześniej, w środku play, wstawiamy

zadanie `ansible.builtin.meta: flush_handlers`.

1. Pętlą z warunkiem tworzymy katalogi `/srv/a`, `/srv/b` i `/srv/c`, ale `/srv/c` tylko na `rhlab02`.

Odpowiedź

```
---
- name: Katalogi z warunkiem
  hosts: dane
  become: true
  tasks:
    - name: Katalogi w srv
      ansible.builtin.file:
        path: "/srv/{{ item }}"
        state: directory
        mode: "0755"
      loop:
        - a
        - b
        - c
      when: item != "c" or inventory_hostname == "rhlab02"
```

Warunek w pętli oceniany jest osobno dla każdego elementu. Zmienna `inventory_hostname` to nazwa hosta z inwentarza, dostępna zawsze bez zbierania faktów.

Dzień 23: szablony Jinja2 i moduły plikowe

Z ŻYCIA WZIĘTE

Wtorek, 15:20. Każdy serwer Zielonej ma w komunikacie powitalnym inny tekst, napisany ręcznie przez inną osobę w innym roku. Ola chce, żeby wszystkie wyglądały tak samo, tylko z nazwą hosta i systemem w środku, a Tomek wie, że to jest dokładnie zadanie dla szablonu. Pan Wojtek przy okazji pyta, czy komunikat mówi, że plik jest zarządzany centralnie, i dostaje

> odpowiedź, że szablon dopisuje to sam.

Cele egzaminu

- szablony Jinja2 do tworzenia plików konfiguracyjnych
- moduły do zarządzania zawartością plików: `copy`, `file`, `lineinfile`, `blockinfile`, `template`
- praca z plikami i katalogami przez Ansible

Czytamy w bazie

- vim: minimum operacyjne dla administratora
- Komunikat powitalny: MOTD i banery

Krótko o tym, czego baza nie ma

> **Pojęcie: szablon Jinja2.** Plik tekstowy z rozszerzeniem `.j2`, w którym oprócz zwykłego tekstu stoją wstawki: `{{ zmienna }}` wstawia wartość, `{% for x in lista %}` powtarza fragment, a `{% if warunek %}` go warunkowo pomija. Moduł `template` renderuje szablon osobno dla każdego węzła i zapisuje wynik. Filtr po kresce pionowej zmienia wartość, na przykład `{{ nazwa | upper }}` zmienia litery na wielkie, a `{{ x | default("brak") }}` podaje wartość domyślną.

Zmienna `ansible_managed` zawiera napis o centralnym zarządzaniu plikiem, a filtr `comment` zamienia go w komentarz. Dzięki temu każdy wygenerowany plik zaczyna się ostrzeżeniem dla człowieka, który zechciałby go edytować ręcznie.

Ćwiczenia

1. Piszemy szablon `templates/motd.j2` z nazwą hosta, nazwą i wersją systemu, pamięcią i listą administratorów z zmiennej, oraz playbook wgrywający go jako `/etc/motd` na wszystkie węzły.

Odpowiedź

Szablon:

```
{{ ansible_managed | comment }}
Witaj na {{ inventory_hostname }}
System: {{ ansible_facts['distribution'] }} {{ ansible_facts['distribution_version'] }}
Pamiec: {{ ansible_facts['memtotal_mb'] }} MB
{% for adm in administratorzy %}
Opiekun: {{ adm }}
{% endfor %}
```

Playbook:

```
---
- name: Komunikat powitalny z szablonu
  hosts: lab
  become: true
  vars:
    administratorzy:
      - tomek
      - ola
  tasks:
    - name: Wgranie motd
      ansible.builtin.template:
        src: templates/motd.j2
        dest: /etc/motd
        owner: root
        group: root
        mode: "0644"
```

```
mkdir -p templates
ansible-playbook motd.yml --check --diff
ansible-playbook motd.yml
ansible lab -m ansible.builtin.command -a "cat /etc/motd"
```

Opcja `--check --diff` pokazuje, jak plik będzie wyglądał, zanim go zapiszemy. Po zalogowaniu na dowolny węzeł powinniśmy zobaczyć komunikat.

1. Na węzłach `dane` ustawiamy w pliku `/etc/selinux/config` linię `SELINUX=enforcing` jednym zadaniem, a do `/etc/hosts` dopisujemy blok z adresami laboratorium z oznaczeniem, żeby można go było później zaktualizować.

Odpowiedź

```
---
- name: Pliki konfiguracyjne
  hosts: dane
  become: true
  tasks:
    - name: Tryb SELinux w pliku konfiguracyjnym
      ansible.builtin.lineinfile:
        path: /etc/selinux/config
        regexp: '^SELINUX='
        line: SELINUX=enforcing

    - name: Blok z adresami laboratorium
      ansible.builtin.blockinfile:
        path: /etc/hosts
        marker: "# {mark} ZARZADZANE PRZEZ ANSIBLE: lab"
        block: |
          10.20.150.11 rhlab01
          10.20.150.12 rhlab02
          10.20.150.13 rhlab03
          10.20.150.14 rhlab04
```

Zadanie `lineinfile` zmienia istniejącą linię pasującą do wyrażenia regularnego albo dopisuje nową, jeśli żadna nie pasuje. Znacznik w `blockinfile` pozwala przy kolejnym uruchomieniu zastąpić blok, a nie dopisywać go jeszcze raz.

1. Tworzymy na `rhlab04` katalog `/srv/dane` z prawami 2775, plik `/srv/dane/README` z treścią i dowiązanie symboliczne `/srv/dane/biezacy` wskazujące na ten plik, a na końcu sprawdzamy plik modułem `stat`.

Odpowiedź

```
---
- name: Pliki i dowiazania
  hosts: web
  become: true
  tasks:
    - name: Katalog z setgid
```

```
    ansible.builtin.file:
      path: /srv/dane
      state: directory
      owner: root
      group: root
      mode: "2775"

  - name: Plik README
    ansible.builtin.copy:
      content: "Dane Zielonej lab\n"
      dest: /srv/dane/README
      mode: "0644"

  - name: Dowiazanie symboliczne
    ansible.builtin.file:
      src: /srv/dane/README
      dest: /srv/dane/biezacy
      state: link

  - name: Odczyt informacji o pliku
    ansible.builtin.stat:
      path: /srv/dane/README
    register: wynik_stat

  - name: Wypisanie rozmiaru
    ansible.builtin.debug:
      msg: "Rozmiar {{ wynik_stat.stat.size }} bajtow"
```

Moduł `file` obsługuje stany `directory`, `touch`, `link`, `hard` i `absent`. Prawa w postaci czterech cyfr piszemy w cudzysłowie, bo YAML bez niego odczytałby `0644` jako liczbę ósemkową i zmieniłby ją.

1. Kopiujemy plik `/etc/hostname` z każdego węzła na węzeł sterujący do katalogu `~/pobrane`, zachowując osobny plik dla każdego hosta.

Odpowiedź

```
---
- name: Pobranie plikow z wezlow
  hosts: lab
  gather_facts: false
  tasks:
    - name: Pobranie hostname
      ansible.builtin.fetch:
        src: /etc/hostname
        dest: "{{ lookup('env', 'HOME') }}/pobrane/"
```

```
ansible-playbook pobierz.yml
find ~/pobrane -type f
```

Moduł `fetch` tworzy pod katalogiem docelowym podkatalog dla każdego hosta (`pobrane/rhlab02/etc/hostname`), więc pliki z różnych węzłów się nie nadpisują.

Częste błędy

Objaw	Przyczyna	Co robimy
`Permission denied (publickey)` przy ping	klucz nie trafił na węzeł albo ma złe prawa	`ssh-copy-id`, na węźle `chmod 700 ~/.ssh` i `chmod 600 ~/.ssh/authorized_keys`
`Missing sudo password`	brak `sudo` bez hasła dla konta `tomek`	plik w `/etc/sudoers.d/` albo `--ask-become-pass`
`couldn't resolve module/action 'ansible.posix.firewalld'`	brak kolekcji w środowisku	`ansible-galaxy collection list`, dopiero potem instalacja
Błąd `mapping values are not allowed here`	złe wcięcie albo dwukropek w nieujętym w cudzysłów napisie	`--syntax-check`, napisy z dwukropkiem w cudzysłowie
Zadanie `command` zawsze zgłasza `changed`	moduł nie wie, czy polecenie coś zmieniło	`changed_when: false` albo moduł, który zna stan
Handler się nie wykonał	zadanie nie zmieniło niczego albo `notify` ma inną nazwę	nazwę w `notify` piszemy dokładnie jak nazwę handlera
`ansible_facts['xxx']` nie istnieje	wyłączone zbieranie faktów	`gather_facts: true` albo `setup` w zadaniu
Szablon nie widzi zmiennej	zmienna nie istnieje w tym play	`ansible-inventory --host NazwaHosta` i `debug`

Co dalej

Dni 24 do 28: obsługa błędów, Ansible Vault, role, kolekcje i System Roles oraz automatyzacja zadań RHCSA opisuje strona Tydzień 4 do EX294: błędy, sekrety, role i automatyzacja zadań RHCSA.

Źródła: strona Red Hat o egzaminie EX294 (cele, narzędzia: AAP, ansible-navigator, ansible-playbook, Git i VS Code, stan na 10 października 2026); Red Hat, Considerations in adopting RHEL 10, rozdział System roles (ansible-core 2.16, węzły zarządzane RHEL 9 i 10); strona cyklu życia AAP (domyślny ansible-core 2.16 w wersjach 2.5 do 2.7); dokumentacja Ansible (moduły ansible.builtin, ansible.posix; polecenia sprawdzone na ansible-core 2.19, składnia na przykładach).

Revision #8
Created 2026-10-10 13:47:38 UTC by Tomasz Konieczny
Updated 2026-10-26 14:25:23 UTC by Tomasz Konieczny