Tydzień 4 do EX294: błędy, sekrety, role i automatyzacja zadań RHCSA

Poziom: średnio

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

Czas: 5 dni po 1,5 do 2 godzin

Dni 24 do 28 programu: obsługa błędów i diagnostyka, Ansible Vault, role, kolekcje i RHEL System Roles, a na końcu automatyzacja zadań RHCSA modułami oraz praca z Git i VS Code. Po tym tygodniu zostaje tylko egzamin próbny.

ZAŁOŻENIA SCENARIUSZA

pracujemy dalej w katalogu `~/ansible` na `rhlab01` z plikami z poprzedniej strony (Tydzień 3 do EX294: Ansible od instalacji do szablonów): `ansible.cfg`, `inventory` z grupami `web` (`rhlab04`) i `dane` (`rhlab02`, `rhlab03`) oraz konto `tomek` z `sudo` bez hasła na węzłach. W dniu 26 potrzebujemy repozytorium z pakietem `rhel-system-roles` (subskrypcja Red Hat Developer albo lustro systemu), a w dniu 28 wolnego dysku `/dev/sdb` na `rhlab03`, czyli jednego z dwóch dodatkowych dysków po 5 GB.

Skróty użyte na tej stronie (12)

Skrót	Co znaczy
EX200	numer egzaminu praktycznego RHCSA, na którym kandydat konfiguruje żywy system RHEL bez dostępu do internetu
EX294	numer egzaminu praktycznego RHCE w ścieżce Ansible: kandydat pisze playbooki, które Red Hat uruchamia na świeżych systemach
GB	Gigabyte: miliard bajtów, jednostka pojemności pamięci i plików, tu najczęściej rozmiar modelu albo pamięci karty graficznej
HTTP	Hypertext Transfer Protocol: protokół stron WWW
LVM	Logical Volume Manager: warstwa w Linuksie składająca dyski w elastyczne wolumeny

Skrót	Co znaczy
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
SSH	Secure Shell: szyfrowane zdalne logowanie do serwera
URL	Uniform Resource Locator: adres strony lub zasobu
WWW	World Wide Web: strony i aplikacje otwierane w przeglądarce po HTTP albo HTTPS
XFS	system plików stworzony przez Silicon Graphics, domyślny w Rocky Linux i RHEL

Fiszka

Komendy wykonujemy w bash jako `tomek` na węźle sterującym w katalogu `~/ansible`, chyba że przy komendzie napisano inaczej. Nazwy takie jak `NazwaPlaybooka`, `NazwaRoli` czy `NazwaZadania` zastępujemy własnymi. Cele pochodzą ze strony Red Hat o egzaminie EX294, stan na 10 października 2026.

□□ ZMIENIA SYSTEM

Pokazanie szczegółów wykonania zadań: -v

```
ansible-playbook NazwaPlaybooka.yml -v
```

Każde `v` więcej daje więcej szczegółów: `-v` pokazuje wyniki modułów, `-vvv` także parametry połączenia SSH. Pierwszy poziom wystarcza do większości diagnoz.

□□ **SSH** = Secure Shell: szyfrowane zdalne logowanie do serwera · **ansible-playbook** = uruchom scenariusz Ansible

□□ ZMIENIA SYSTEM

Wznowienie playbooka od wskazanego zadania: --start-at-task

```
ansible-playbook NazwaPlaybooka.yml --start-at-task "NazwaZadania"
```

Pomija wszystkie zadania przed wskazanym, więc po naprawie błędu w środku długiego playbooka nie czekamy na zadania, które już się udały. Nazwę zadania wypisuje `--list-`

`tasks`.

ansible-playbook = uruchom scenariusz Ansible

ZMIENIA SYSTEM

Wykonywanie zadań pojedynczo z pytaniem: --step

```
ansible-playbook NazwaPlaybooka.yml --step
```

Przed każdym zadaniem Ansible pyta, czy je wykonać (`y`), pominąć (`n`) czy dalej już bez pytań (`c`).

ansible-playbook = uruchom scenariusz Ansible

ZMIENIA SYSTEM

Uruchomienie tylko zadań z wybranym tagiem: --tags

```
ansible-playbook NazwaPlaybooka.yml --tags NazwaTagu
```

Tag przypisujemy zadaniu słowem `tags:`. Pomija wszystkie zadania bez tego tagu, a `--skip-tags` robi odwrotnie.

ansible-playbook = uruchom scenariusz Ansible

ZMIENIA SYSTEM

Ograniczenie wykonania do jednego hosta: --limit

```
ansible-playbook NazwaPlaybooka.yml --limit NazwaHosta
```

Playbook dotyczy tylko wskazanego hosta albo grupy z listy `hosts:`, więc nową zmianę próbujemy najpierw na jednej maszynie.

ansible-playbook = uruchom scenariusz Ansible

ZMIENIA SYSTEM

Utworzenie zaszyfrowanego pliku: ansible-vault create

```
ansible-vault create --vault-password-file ~/.vault_pass vars/tajne.yml
```

Otwiera edytor z `$EDITOR` i po zapisaniu szyfruje plik. Bez opcji z plikiem hasła komenda zapyta o hasło dwa razy.

ansible-vault = od Ansible i vault (skarbiec): polecenie do szyfrowania plików i pojedynczych wartości z hasłami używanymi w playbookach

TYLKO ODCZYT

Podgląd i edycja zaszyfrowanego pliku: view i edit

```
ansible-vault view --vault-password-file ~/.vault_pass vars/tajne.yml
ansible-vault edit --vault-password-file ~/.vault_pass vars/tajne.yml
```

`view` tylko wypisuje odszyfrowaną treść, a `edit` otwiera ją w edytorze i po zapisaniu szyfruje z powrotem.

ansible-vault = od Ansible i vault (skarbiec): polecenie do szyfrowania plików i pojedynczych wartości z hasłami używanymi w playbookach

TYLKO ODCZYT

Zaszyfrowanie pojedynczej wartości: ansible-vault encrypt_string

```
ansible-vault encrypt_string --vault-password-file ~/.vault_pass 'Tajne123' --name
'haslo_db'
```

Wynik wklejamy do zwykłego pliku zmiennych, gdzie obok jawnych wartości stoi jedna zaszyfrowana. Wartość w apostrofach zostaje w historii powłoki, więc w praktyce wczytujemy ją z `--stdin-name`.

ansible-vault = od Ansible i vault (skarbiec): polecenie do szyfrowania plików i pojedynczych wartości z hasłami używanymi w playbookach

ZMIENIA SYSTEM

Uruchomienie playbooka z hasłem do sejfu: --vault-password-file

```
ansible-playbook NazwaPlaybooka.yml --vault-password-file ~/.vault_pass
```

Zamiast pliku z hasłem można dać `--ask-vault-pass`, które pyta interaktywnie. Ścieżkę pliku można też wpisać na stałe do `ansible.cfg` jako `vault_password_file`.

ansible-playbook = uruchom scenariusz Ansible

TYLKO ODCZYT

Zmiana hasła do zaszyfrowanego pliku: ansible-vault rekey

```
ansible-vault rekey --ask-vault-pass vars/tajne.yml
```

Pyta o stare hasło, a potem o nowe, i szyfruje plik od nowa. Do trwałego usunięcia szyfrowania służy `ansible-vault decrypt`.

ansible-vault = od Ansible i vault (skarbiec): polecenie do szyfrowania plików i pojedynczych wartości z hasłami używanymi w playbookach

TYLKO ODCZYT

Utworzenie szkieletu roli: ansible-galaxy role init

```
ansible-galaxy role init roles/NazwaRoli
```

Tworzy katalog roli ze standardowym układem: `tasks`, `handlers`, `templates`, `files`, `defaults`, `vars` i `meta`. Szkielet zawiera też pliki przykładowe, które wolno usunąć.

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

ZMIENIA SYSTEM

Instalacja ról z pliku wymagań: ansible-galaxy role install

```
ansible-galaxy role install -r requirements.yml -p ./roles
```

Czyta listę ról z pliku i pobiera je z wskazanych źródeł: serwera Galaxy, repozytorium Git albo adresu URL archiwum. Opcja `-p` wskazuje katalog docelowy.

URL = Uniform Resource Locator: adres strony lub zasobu · **ansible-galaxy** = od Ansible i galaxy: polecenie do tworzenia szkieletów ról i kolekcji oraz ich instalowania ze źródeł i z archiwów

TYLKO ODCZYT

Wypisanie dostępnych ról: ansible-galaxy role list

```
ansible-galaxy role list
```

Pokazuje role ze wszystkich katalogów wskazanych w `roles_path`. Brak roli na liście oznacza, że Ansible jej nie znajdzie.

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

ZMIENIA SYSTEM

Instalacja kolekcji: ansible-galaxy collection install

```
ansible-galaxy collection install NazwaPrzestrzeni.NazwaKolekcji -p ./collections
```

Kolekcja z katalogu `collections` obok playbooka jest widoczna bez dodatkowej konfiguracji. Zamiast nazwy można podać ścieżkę do archiwum `.tar.gz`.

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

TYLKO ODCZYT

Przeglądanie kolekcji w ansible-navigator: collections

```
ansible-navigator collections --mode stdout --ee false
```

Wypisuje kolekcje i moduły, które narzędzie widzi, a bez opcji `--mode stdout` otwiera interaktywny ekran, w którym można przejść do modułu i jego dokumentacji. Tym sposobem znajdujemy nowe moduły, o co prosi cel egzaminu.

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

TYLKO ODCZYT

Dokumentacja modułu w ansible-navigator: doc

```
ansible-navigator doc ansible.posix.firewalld --mode stdout --ee false
```

Daje tę samą treść co `ansible-doc`, więc wybieramy to narzędzie, do którego jesteśmy przyzwyczajeni.

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

ZMIENIA SYSTEM

Instalacja ról systemowych RHEL: dnf install rhel-system-roles

```
sudo dnf install rhel-system-roles
```

Pakiet dostarcza gotowe role do typowych zadań (czas, sieć, magazyn, SELinux, zapora, logowanie), a dokumentację z przykładami kładzie w `/usr/share/doc/rhel-system-roles/`.

RHEL = Red Hat Enterprise Linux: komercyjna dystrybucja Linuksa · **SELinux** = Security-Enhanced Linux: dodatkowa warstwa uprawnień w jądrze RHEL · **dnf** = Dandified YUM: menedżer pakietów RHEL/Rocky

TYLKO ODCZYT

Sklonowanie repozytorium Git: git clone

```
git clone ssh://tomek@rhlab02/home/tomek/repo/ansible.git ~/ansible-git
```

Tworzy lokalną kopię repozytorium razem z całą historią. Adres zależy od tego, gdzie repozytorium stoi: serwer Git, GitHub albo zwykły katalog na innej maszynie.

git = system kontroli wersji (nazwa to brytyjski slang: nieznośny typ)

TYLKO ODCZYT

Dodanie plików do repozytorium: git add

```
git add NazwaPliku.yml
```

Dodaje plik do poczekalni, czyli listy zmian przygotowanych do zapisania. Cały katalog dodaje `git add .`, a to, czego nie chcemy w repozytorium, wpisujemy do `.gitignore`.

git = system kontroli wersji (nazwa to brytyjski slang: nieznośny typ)

□□ ZMIENIA SYSTEM

Zapisanie zmian w historii: git commit

```
git commit -m "Opis zmiany"
```

Zapisuje poczekalnię jako migawkę w lokalnej historii. Przy pierwszym użyciu Git wymaga ustawienia `user.name` i `user.email`.

□□ **git** = system kontroli wersji (nazwa to brytyjski slang: nieznośny typ)

□□ ZMIENIA SYSTEM

Wysłanie zmian do zdalnego repozytorium: git push

```
git push origin main
```

Wysyła lokalne zapisy z gałęzi `main` do repozytorium oznaczonego jako `origin`. Przy pierwszym wysłaniu nowej gałęzi dodajemy `-u`, które zapamiętuje powiązanie.

□□ **git** = system kontroli wersji (nazwa to brytyjski slang: nieznośny typ)

□□ TYLKO ODCZYT

Utworzenie pustego repozytorium na serwerze: git init --bare

```
git init --bare --initial-branch=main ~/repo/ansible.git
```

Repozytorium bez katalogu roboczego służy tylko do wymiany zmian, więc nadaje się na wspólny punkt zapisu w laboratorium.

□□ **git** = system kontroli wersji (nazwa to brytyjski slang: nieznośny typ)

Jak to działa: role, kolekcje i sejf

Pojęcie: rola. Rola to playbook pocięty na standardowe katalogi, żeby dało się go użyć w wielu projektach: zadania w `tasks/main.yml`, handlery w `handlers/main.yml`, szablony w `templates/`, pliki w `files/`, wartości domyślne w `defaults/main.yml`, a zmienne o wysokim priorytecie w `vars/main.yml`. Playbook wywołuje rolę słowem `roles:`, a Ansible sam wczytuje wszystkie te katalogi.

Pojęcie: kolekcja. Kolekcja to paczka z modułami, wtyczkami i rolami pod wspólną nazwą w postaci `przestrzen.nazwa`, na przykład `ansible.posix` albo `community.general`. Moduł z kolekcji wywołujemy pełną nazwą (`ansible.posix.firewalld`), a rolę z kolekcji nazwą `przestrzen.nazwa.rola`.

Pojęcie: Ansible Vault. Mechanizm szyfrowania haseł i kluczy trzymanych w plikach ze zmiennymi, żeby można je było zapisać razem z playbookiem. Plik zaszyfrowany zaczyna się wierszem `$ANSIBLE_VAULT;1.1;AES256`. Hasło do sejfu nie leży w repozytorium, tylko w osobnym pliku poza nim albo jest wpisywane ręcznie.

Pojęcie: RHEL System Roles. Zestaw ról utrzymywanych przez Red Hat, które ustawiają typowe elementy systemu przez zmienne zamiast przez moduły: `timesync`, `selinux`, `firewall`, `storage`, `network` i inne. Dostarcza je pakiet `rhel-system-roles`, a ich dokumentacja leży lokalnie w `/usr/share/doc/rhel-system-roles/`, więc jest dostępna także bez internetu.

Dzień 24: obsługa błędów i diagnostyka

◫ Z ŻYCIA WZIĘTE

Środa, 9:40. Playbook z konfiguracją serwera WWW zatrzymuje się na czwartym zadaniu na jednej z trzech maszyn, a pozostałe dwie dobiegają do końca. Tomek patrzy na czerwony komunikat i nie wie, czy ma poprawiać playbook, czy maszynę. Ola odpowiada krótko: „Najpierw przeczytaj całe zdanie błędu, potem szukaj po nazwie zadania. Zwykle pisze tam, co jest nie tak”.

Cele egzaminu

- konfiguracja obsługi błędów w playbookach
- użycie zmiennych do przechowania wyniku polecenia (`register`)
- sterowanie wykonaniem zadań warunkami
- korzystanie z dokumentacji modułów i poleceń podczas pracy

Czytamy w bazie

- Obsługa błędów: ta sama myśl w skryptach powłoki, kody wyjścia i pułapki
- Kod wyjścia, $?, &&, ||, średnik
- Ansible: podstawy operacyjne: sekcja o częstych błędach

Krótko o tym, czego baza nie ma

> **Pojęcie: obsługa błędów w play.** Domyślnie błąd zadania zatrzymuje play na tym hoście, a pozostałe hosty idą dalej. Zmieniamy to czterema narzędziami: `ignore_errors: true` (błąd jest wypisany, ale play idzie dalej), `failed_when` (sami określamy, kiedy zadanie jest błędem), `changed_when` (sami określamy, kiedy zadanie coś zmieniło) oraz `block` z `rescue` i `always`. Blok działa jak `try`, `catch` i `finally`: zadania z `block` się wykonują, przy błędzie uruchamia się `rescue`, a `always` wykonuje się zawsze.

Najczęstsze polecenia diagnostyczne to zwiększenie szczegółowości (`-v`), zadanie `ansible.builtin.debug` z `msg` albo `var`, `--syntax-check` przed uruchomieniem oraz `--start-at-task` po poprawce. Komunikat błędu Ansible zawiera nazwę zadania, plik i wiersz, a także najczęściej tekst błędu z węzła, więc czytamy go w całości.

Ćwiczenia

1. Piszemy playbook, który na grupie `dane` próbuje uruchomić nieistniejącą usługę, a po błędzie zapisuje w `/var/tmp/blad-uslugi` nazwę hosta i na końcu zawsze zapisuje znacznik `/var/tmp/koniec-proby`. Play nie może się przerwać.

Odpowiedź

```
---
- name: Obsluga bledow
  hosts: dane
  become: true
  tasks:
    - name: Proba z obsluga bledu
      block:
        - name: Uruchomienie nieistniejacej uslugi
          ansible.builtin.service:
```

```
            name: nieistniejaca-usluga
            state: started
      rescue:
        - name: Zapis informacji o bledzie
          ansible.builtin.copy:
            content: "Blad na {{ inventory_hostname }}\n"
            dest: /var/tmp/blad-uslugi
            mode: "0644"
      always:
        - name: Znacznik konca
          ansible.builtin.copy:
            content: "koniec\n"
            dest: /var/tmp/koniec-proby
            mode: "0644"
```

```
ansible-playbook blok.yml
ansible dane -m ansible.builtin.command -a "cat /var/tmp/blad-uslugi /var/tmp/koniec-
proby"
```

W podsumowaniu play przy każdym hoście `failed=0` i `rescued=1`: błąd został obsłużony i nie liczy się jako niepowodzenie. Zadanie, które zawiodło, jest nadal widoczne w wyniku jako `fatal: [rhlab02]: FAILED!`, ale po nim wykonuje się `rescue` i play idzie dalej.

1. Sprawdzamy w jednym play, czy na węzłach istnieje konto `praktykant` (polecenie `id`), zapisujemy wynik w zmiennej, nie traktujemy braku konta jako błędu i wypisujemy po jednym zdaniu na host.

Odpowiedź

```
---
- name: Sprawdzenie konta
  hosts: lab
  gather_facts: false
  tasks:
    - name: Szukanie konta praktykant
      ansible.builtin.command: id praktykant
      register: konto
      changed_when: false
      failed_when: konto.rc not in [0, 1]
```

```
    - name: Wynik
      ansible.builtin.debug:
        msg: "{{ inventory_hostname }}: {{ 'konto jest' if konto.rc == 0 else 'brak konta'
}}"
```

```
ansible-playbook konto.yml
```

Polecenie `id` dla nieistniejącego konta kończy się kodem 1, a `failed_when` uznaje za błąd dopiero inny kod niż 0 i 1. `changed_when: false` jest potrzebne, bo moduł `command` zawsze zgłasza zmianę, choć samo czytanie niczego nie zmienia. Zmienna `konto` zawiera pola `rc`, `stdout` i `stderr`, które pokazuje `debug: var=konto`.

1. Dostajemy playbook, który nie działa. Znajdujemy oba błędy narzędziami Ansible i poprawiamy je: pierwszy widać zaraz po `--syntax-check`, drugi dopiero po uruchomieniu.

Odpowiedź

```
---
- name: Serwer z bledami
  hosts: web
  become: true
  tasks:
    - name: Instalacja httpd
      ansible.builtin.dnff:
        name: httpd
        state: present

    - name: Plik z tytulem
      ansible.builtin.copy:
        content: "{{ tytul_strony }}\n"
        dest: /var/www/html/index.html
```

```
ansible-playbook zepsuty.yml --syntax-check
ansible-doc -l | grep -i "dnf"
```

Pierwsza komenda kończy się komunikatem, który mówi, że nie znaleziono modułu `ansible.builtin.dnff`, i wskazuje zadanie `Instalacja httpd`. Poprawka to `ansible.builtin.dnf`, a nazwę potwierdza `ansible-doc -l`. Po poprawce składnia przechodzi, ale uruchomienie kończy

się komunikatem, że zmienna `tytul_strony` jest niezdefiniowana. Zmienną dodajemy do `group_vars/web.yml` albo do `vars:` play:

```
  vars:
    tytul_strony: "Zielona lab"
```

Błędy nazw modułów i wcięć wychodzą w `--syntax-check`, a błędy zmiennych dopiero w czasie pracy, bo zmienne są obliczane w trakcie wykonania. Szczegóły błędu połączenia pokazuje `-vvv`.

1. Do playbooka z ćwiczenia 3 dodajemy tagi: `pakiety` dla instalacji i `tresc` dla pliku strony. Po poprawce uruchamiamy tylko zadanie z treścią i wskazujemy jeden host.

Odpowiedź

```
---
- name: Serwer z tagami
  hosts: web
  become: true
  vars:
    tytul_strony: "Zielona lab"
  tasks:
    - name: Instalacja httpd
      ansible.builtin.dnf:
        name: httpd
        state: present
      tags: pakiety

    - name: Plik z tytulem
      ansible.builtin.copy:
        content: "{{ tytul_strony }}\n"
        dest: /var/www/html/index.html
        mode: "0644"
      tags: tresc
```

```
ansible-playbook tagi.yml --list-tasks
ansible-playbook tagi.yml --tags tresc --limit rhlab04
ansible-playbook tagi.yml --start-at-task "Plik z tytulem"
```

Pierwsza komenda wypisuje zadania razem z ich tagami. Druga wykonuje tylko zadanie z tagiem `tresc` na `rhlab04`, a trzecia ma ten sam efekt, ale wskazuje zadanie nazwą. Na egzaminie zwykle uruchamiamy cały playbook, ale po poprawce w środku długiego przebiegu oszczędzamy czas.

Dzień 25: Ansible Vault

> **Z ŻYCIA WZIĘTE**
>
> Czwartek, 10:20. Tomek ma w playbooku hasło praktykanta wpisane tekstem jawnym i chce je wysłać na Gitea, żeby Ola mogła obejrzeć zmiany. Ola zatrzymuje go słowami: „W repozytorium zostaje na zawsze, nawet po usunięciu pliku". Pokazuje, jak to samo hasło trafia do zaszyfrowanego pliku, który wolno bezpiecznie zapisać razem z resztą.

Cele egzaminu

- użycie Ansible Vault w playbookach do ochrony danych wrażliwych
- tworzenie i modyfikacja kont użytkowników modułami
- praca ze zmiennymi z kilku źródeł

Czytamy w bazie

- Ansible: podstawy operacyjne: sekcja o `ansible-vault`
- Hasła i wygasanie kont: chage, passwd, faillock

Krótko o tym, czego baza nie ma

Baza pokazuje dwie komendy: szyfrowanie istniejącego pliku i uruchomienie z pytaniem o hasło. Na egzaminie potrzeba całego cyklu: utworzenie pliku z sekretem, wczytanie go przez `vars_files`, uruchomienie playbooka z plikiem hasła oraz podgląd i zmiana. Hasło do sejfu trzymamy w pliku `~/.vault_pass` z prawami `600`, poza katalogiem repozytorium.

Jeszcze jedna pułapka dotyczy haseł kont. Moduł `user` przyjmuje w polu `password` skrót hasła, a nie samo hasło, więc zamieniamy je filtrem `password_hash('sha512')`. Filtr losuje sól przy każdym uruchomieniu, więc bez dodatkowego zabezpieczenia zadanie zgłaszałoby zmianę za każdym razem. Opcja `update_password: on_create` ustawia hasło tylko przy tworzeniu konta i usuwa ten problem.

Ćwiczenia

1. Zakładamy plik hasła do sejfu, tworzymy zaszyfrowany plik `vars/tajne.yml` ze zmienną `haslo_praktykanta`, sprawdzamy, jak wygląda zaszyfrowany, a potem go podglądamy.

Odpowiedź

```
cd ~/ansible
mkdir -p vars
echo 'HasloSejfuLab1' > ~/.vault_pass
chmod 600 ~/.vault_pass
ansible-vault create --vault-password-file ~/.vault_pass vars/tajne.yml
```

W edytorze wpisujemy jedną linię i zapisujemy plik:

```
haslo_praktykanta: "Praktyk4nt!Lab"
```

```
head -2 vars/tajne.yml
ansible-vault view --vault-password-file ~/.vault_pass vars/tajne.yml
```

Pierwszy wiersz zaszyfrowanego pliku to `$ANSIBLE_VAULT;1.1;AES256`, a reszta to cyfry szesnastkowe. Komenda `view` wypisuje wpisaną linię. Plik z hasłem do sejfu dopisujemy do `.gitignore`, żeby przypadkiem go nie wysłać.

1. Piszemy playbook, który na wszystkich węzłach zakłada konto `praktykant` z hasłem z sejfu, ustawia hasło tylko przy tworzeniu konta i nie pokazuje skrótu w wyniku. Uruchamiamy go z plikiem hasła i potem logujemy się na konto poleceniem `su`.

Odpowiedź

```
---
- name: Konto praktykanta z haslem z sejfu
  hosts: lab
  become: true
  vars_files:
    - vars/tajne.yml
  tasks:
    - name: Konto praktykant
      ansible.builtin.user:
```

```
        name: praktykant
        password: "{{ haslo_praktykanta | password_hash('sha512') }}"
        update_password: on_create
        state: present
      no_log: true
```

```
ansible-playbook praktykant.yml --vault-password-file ~/.vault_pass
ansible-playbook praktykant.yml --vault-password-file ~/.vault_pass
```

Przy drugim uruchomieniu zadanie zgłasza `changed: false` dzięki `update_password: on_create`. Opcja `no_log: true` ukrywa wynik zadania, bo w wyniku `user` jest skrót hasła, a w razie błędu także wartości parametrów. Gdy filtr `password_hash` zgłasza brak biblioteki `passlib` albo `crypt`, doinstalowujemy ją na węźle sterującym.

1. Szyfrujemy pojedynczą wartość `haslo_db` i wklejamy ją do `group_vars/dane.yml` obok zwykłej zmiennej `baza_port`, a potem wypisujemy ją zadaniem `debug` na `rhlab02`.

Odpowiedź

```
ansible-vault encrypt_string --vault-password-file ~/.vault_pass 'Tajne123' --name
'haslo_db'
```

Wynik wygląda tak (cyfry szesnastkowe będą inne):

```
haslo_db: !vault |
          $ANSIBLE_VAULT;1.1;AES256
          653038303963313039343231663235356133313438326231376164663439316561376163653630333
          366330346564353336303938313537313965643433323034 0a663130333962346632343938656432
```

Wklejamy go do pliku razem z jawną zmienną:

```
---
baza_port: 5432
haslo_db: !vault |
          $ANSIBLE_VAULT;1.1;AES256
          653038303963313039343231663235356133313438326231376164663439316561376163653630333
          366330346564353336303938313537313965643433323034 0a663130333962346632343938656432
```

```
ansible rhlab02 -m ansible.builtin.debug -a "var=haslo_db" --vault-password-file
~/.vault_pass
```

Ostatnia komenda wypisuje odszyfrowaną wartość, bo `debug` pokazuje wynik. W prawdziwym playbooku z sekretem używamy `no_log: true`. Zaleta tej metody to jawne nazwy zmiennych w repozytorium przy zaszyfrowanych wartościach. Wada to nieczytelny blok w pliku, więc przy wielu sekretach wygodniejszy jest cały zaszyfrowany plik.

1. Wpisujemy ścieżkę do pliku hasła w `ansible.cfg`, żeby nie dodawać opcji przy każdym uruchomieniu, a potem zmieniamy hasło sejfu dla pliku `vars/tajne.yml`.

Odpowiedź

W sekcji `[defaults]` pliku `ansible.cfg` dopisujemy:

```
vault_password_file = ~/.vault_pass
```

```
ansible-config dump --only-changed | grep -i vault
ansible-playbook praktykant.yml
echo 'NoweHasloSejfu2' > ~/.vault_pass.nowe
chmod 600 ~/.vault_pass.nowe
ansible-vault rekey --vault-password-file ~/.vault_pass --new-vault-password-file
~/.vault_pass.nowe vars/tajne.yml
mv ~/.vault_pass.nowe ~/.vault_pass
ansible-playbook praktykant.yml
```

Po zmianie hasła stary plik z hasłem przestaje działać, więc podmieniamy go na nowy, a bez tego playbook zgłosi błąd odszyfrowania. Jeżeli w `ansible.cfg` jest plik hasła, a na egzaminie podany jest inny sposób, opcja z linii poleceń wygrywa z konfiguracją.

Dzień 26: role, kolekcje i RHEL System Roles

> **□□ Z ŻYCIA WZIĘTE**

> Piątek, 10:10. Ola prosi, żeby serwer WWW z dnia 20 postawić jeszcze raz na innym projekcie i dodać wspólną konfigurację czasu. „Kopiujemy playbook?”, pyta Tomek. „Nie. Robisz z niego rolę, a czas bierzesz z roli, którą Red Hat już napisał”. Po godzinie Tomek widzi komunikat, że rola `rhel-system-roles.timesync` nie została znaleziona, choć pakiet jest zainstalowany. Winny jest jego własny `roles_path`.

Cele egzaminu

- tworzenie i używanie ról
- instalowanie ról i używanie ich w playbookach
- instalowanie kolekcji i używanie ich w playbookach
- pozyskiwanie zestawu powiązanych ról, dodatkowych modułów i innej zawartości z kolekcji

Czytamy w bazie

- Ansible: podstawy operacyjne: sekcja o rolach
- vim: minimum operacyjne dla administratora: edycja wielu plików roli w jednej sesji

Krótko o tym, czego baza nie ma

Baza opisuje rolę jednym zdaniem i jednym fragmentem playbooka. Tutaj potrzebny jest cały układ: `ansible-galaxy role init` tworzy katalogi, a w nich wpisujemy zadania, handlery, szablony i domyślne wartości. Rola dostaje zmienne z `defaults/main.yml` (najniższy priorytet, łatwo je nadpisać) albo z `vars/main.yml` (wysoki priorytet), a playbook może je zmienić w wpisie `roles:`.

Ścieżki szukania ról i kolekcji ustawiają dwa klucze `ansible.cfg`: `roles_path` i `collections_path`. Gdy wpiszemy do nich tylko własny katalog, Ansible przestaje widzieć role i kolekcje z systemu, w tym `/usr/share/ansible/roles` z pakietu `rhel-system-roles`. Dlatego oba katalogi wypisujemy po dwukropku.

Ćwiczenia

1. Zamieniamy serwer WWW w rolę `www` ze zmienną `www_tytul`, szablonem strony i handlerem restartu, a potem wywołujemy ją w playbooku na grupie `web`.

Odpowiedź

```
cd ~/ansible
ansible-galaxy role init roles/www
```

Plik `roles/www/defaults/main.yml`:

```
---
www_tytul: "Zielona lab"
```

Plik `roles/www/tasks/main.yml`:

```
---
- name: Instalacja httpd
  ansible.builtin.dnf:
    name: httpd
    state: present

- name: Strona startowa z szablonu
  ansible.builtin.template:
    src: index.html.j2
    dest: /var/www/html/index.html
    owner: root
    group: root
    mode: "0644"
  notify: Restart httpd

- name: Wlaczenie i start httpd
  ansible.builtin.service:
    name: httpd
    state: started
    enabled: true

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

Plik `roles/www/handlers/main.yml`:

```
---
- name: Restart httpd
  ansible.builtin.service:
    name: httpd
    state: restarted
```

Plik `roles/www/templates/index.html.j2`:

```
<h1>{{ www_tytul }}</h1>
<p>Host: {{ inventory_hostname }}, system: {{ ansible_facts['distribution'] }} {{
ansible_facts['distribution_version'] }}</p>
```

Playbook `www.yml`:

```
---
- name: Serwer WWW z roli
  hosts: web
  become: true
  roles:
    - role: www
      vars:
        www_tytul: "Zielona: serwer testowy"
```

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

Ostatnia komenda wypisuje stronę z tytułem z playbooka, a nie z wartości domyślnej roli. Usuwamy z szkieletu niepotrzebne pliki (`tests`, `vars`, `README.md`), jeśli nie będą używane, ale puste katalogi nie przeszkadzają.

1. Z katalogu na `rhlab01` serwujemy archiwum roli `lab_motd` przez HTTP, wpisujemy je do `requirements.yml` jako rolę z adresu URL i instalujemy do `./roles`. Potem używamy jej w playbooku.

Odpowiedź

Najpierw przygotowujemy rolę i archiwum:

```
cd ~/ansible
ansible-galaxy role init roles/lab_motd
tee roles/lab_motd/tasks/main.yml <<'EOF'
---
- name: Komunikat powitalny
  ansible.builtin.copy:
    content: "Serwer Zielona, {{ inventory_hostname }}\n"
    dest: /etc/motd
    mode: "0644"
EOF
mkdir -p ~/serwowane
tar czf ~/serwowane/lab_motd.tar.gz -C roles lab_motd
rm -rf roles/lab_motd
```

W drugim terminalu uruchamiamy prosty serwer HTTP:

```
cd ~/serwowane && python3 -m http.server 8000
```

W pierwszym terminalu piszemy plik `requirements.yml`, instalujemy rolę i używamy jej:

```
---
- name: lab_motd
  src: http://localhost:8000/lab_motd.tar.gz
```

```
ansible-galaxy role install -r requirements.yml -p ./roles
ansible-galaxy role list
tee motd-rola.yml <<'EOF'
---
- name: Komunikat z roli
  hosts: lab
  become: true
  roles:
    - lab_motd
EOF
ansible-playbook motd-rola.yml
```

Klucz `src` może też wskazywać repozytorium Git (adres kończący się na `.git`) albo rolę z serwera Galaxy podaną jako `przestrzen.rola`. W miejscu `src` na egzaminie dostajemy adres źródła, więc nazwę roli podajemy przez `name`. Serwer HTTP zatrzymujemy `Ctrl+C` po instalacji.

1. Budujemy kolekcję `zielona.lab` z jedną rolą `motd`, instalujemy ją z archiwum do katalogu `collections` w projekcie i wywołujemy rolę po pełnej nazwie.

Odpowiedź

```
cd ~
ansible-galaxy collection init zielona.lab
mkdir -p zielona/lab/roles/motd/tasks
tee zielona/lab/roles/motd/tasks/main.yml <<'EOF'
---
- name: Komunikat powitalny z kolekcji
  ansible.builtin.copy:
    content: "Zielona: {{ inventory_hostname }}\n"
    dest: /etc/motd
    mode: "0644"
EOF
tee zielona/lab/galaxy.yml <<'EOF'
---
namespace: zielona
name: lab
version: 1.0.0
readme: README.md
authors:
  - Tomek <tomek@zielona.example>
description: Kolekcja laboratoryjna z jedna rola
license:
  - GPL-2.0-or-later
tags: []
dependencies: {}
EOF
cd zielona/lab && ansible-galaxy collection build --output-path ~
cd ~/ansible
ansible-galaxy collection install ~/zielona-lab-1.0.0.tar.gz -p ./collections
ansible-galaxy collection list -p ./collections
```

Playbook `motd-kolekcja.yml`:

```
---
- name: Komunikat z kolekcji
```

```
  hosts: lab
  become: true
  roles:
    - zielona.lab.motd
```

```
ansible-playbook motd-kolekcja.yml --syntax-check
ansible-playbook motd-kolekcja.yml
```

Archiwum kolekcji można wymienić w `requirements.yml` pod kluczem `collections:` z `type: file`, a kilka kolekcji naraz instaluje `ansible-galaxy collection install -r requirements.yml`. Katalog `collections` obok playbooka jest przeszukiwany automatycznie, a dla innych miejsc ustawiamy `collections_path` w `ansible.cfg`. Przeglądanie kolekcji w narzędziu interaktywnym: `ansible-navigator collections`.

1. Instalujemy `rhel-system-roles`, odnajdujemy rolę czasu i jej dokumentację bez internetu, poprawiamy `ansible.cfg`, a potem synchronizujemy czas na `rhlab01`, `rhlab03` i `rhlab04` z serwerem czasu `rhlab02`.

Odpowiedź

```
sudo dnf install rhel-system-roles
rpm -ql rhel-system-roles | grep -i timesync | head
ls /usr/share/doc/rhel-system-roles/timesync/
ansible-galaxy role list
ansible-galaxy collection list | grep -i rhel
```

W `ansible.cfg` przepisujemy dwie ścieżki, tak żeby obejmowały też katalogi systemowe:

```
roles_path = ./roles:/usr/share/ansible/roles
collections_path = ./collections:/usr/share/ansible/collections
```

Playbook `czas.yml` w wersji z rolą w katalogu `/usr/share/ansible/roles`:

```
---
- name: Czas z roli systemowej
  hosts: lab:!rhlab02
  become: true
  vars:
    timesync_ntp_servers:
```

```
      - hostname: rhlab02
        iburst: true
  roles:
    - rhel-system-roles.timesync
```

Gdy `ansible-galaxy collection list` pokazuje kolekcję `redhat.rhel_system_roles`, wołamy tę samą rolę po pełnej nazwie `redhat.rhel_system_roles.timesync`. Układ zależy od wersji pakietu, więc o wyborze decyduje to, co pokazują dwie komendy `list`.

```
ansible-playbook czas.yml --syntax-check
ansible-playbook czas.yml
ansible lab -b -m ansible.builtin.command -a "chronyc sources"
```

Ostatnia komenda pokazuje `rhlab02` jako źródło na trzech węzłach. Nazwy zmiennych roli (`timesync_ntp_servers`, `iburst`) czytamy z pliku `README.md` w katalogu dokumentacji roli, który jest jedynym źródłem na egzaminie.

Dzień 27: automatyzacja zadań RHCSA, część 1

> **□□ Z ŻYCIA WZIĘTE**
>
> Poniedziałek, 9:00. Ola kładzie przed Tomkiem listę z tygodni 1 i 2: pakiety, usługi, zapora, konta, cron, SELinux. „Każdy punkt z tej listy robiłeś ręcznie. Teraz robisz go playbookiem i uruchamiasz dwa razy”. Tomek zauważa, że każde ćwiczenie ma w Ansible odpowiednik w jednym module, i że moduł sam wie, czy coś trzeba zmienić.

Cele egzaminu

- automatyzacja zadań RHCSA modułami: pakiety i repozytoria, usługi, reguły zapory, treść plików, zadania zaplanowane, bezpieczeństwo, użytkownicy i grupy
- tworzenie playbooków doprowadzających system do zadanego stanu
- konfiguracja trwała po restarcie, bez interwencji

Czytamy w bazie

- dnf: instalacja, aktualizacja, historia, wycofywanie zmian i Repozytoria: konfiguracja, EPEL, priorytety
- firewalld: strefy, porty, usługi, rich rules
- SELinux: podstawy, tryby, konteksty
- Użytkownicy, grupy i pliki, w których system ich trzyma i sudo i /etc/sudoers: konfiguracja i audyt
- cron i crontab: zadania cykliczne

Krótko o tym, czego baza nie ma

Poniższa tabela łączy zadanie z tygodni 1 i 2 z modułem Ansible, w którym robimy to samo. Przy modułach spoza `ansible.builtin` sprawdzamy w `ansible-doc`, czy kolekcja jest na węźle sterującym.

Zadanie ręczne	Moduł	Kolekcja
instalacja pakietu, grupy pakietów	`dnf`	`ansible.builtin`
plik `.repo`	`yum_repository`	`ansible.builtin`
usługa włączona i uruchomiona	`service` albo `systemd`	`ansible.builtin`
usługa i port w zaporze	`firewalld`	`ansible.posix`
przełącznik SELinux (boolean)	`seboolean`	`ansible.posix`
tryb SELinux	`selinux`	`ansible.posix`
etykieta portu SELinux	`seport`	`community.general`
etykieta pliku SELinux	`sefcontext`	`community.general`
konto, grupa	`user`, `group`	`ansible.builtin`
klucz SSH konta	`authorized_key`	`ansible.posix`
plik z regułą `sudo`	`copy` z `validate`	`ansible.builtin`
zadanie cron	`cron`	`ansible.builtin`
zadanie jednorazowe `at`	`at`	`ansible.posix`
nazwa hosta	`hostname`	`ansible.builtin`

Przy etykiecie pliku `sefcontext` zmienia tylko regułę, a `restorecon` stosuje ją do plików, więc po `sefcontext` dodajemy zadanie `command: restorecon -Rv /ścieżka` albo używamy `file` z polem `setype`.

Ćwiczenia

1. Piszemy playbook, który na `rhlab04` dodaje repozytorium (wyłączone, żeby nie psuło instalacji), instaluje `httpd` i `tmux`, usuwa `telnet` i instaluje grupę pakietów `Development Tools`.

Odpowiedź

```
---
- name: Oprogramowanie na web
  hosts: web
  become: true
  tasks:
    - name: Repozytorium laboratorium
      ansible.builtin.yum_repository:
        name: zielona-lab
        description: Repozytorium laboratorium Zielonej
        baseurl: http://rhlab01/repo
        gpgcheck: false
        enabled: false
        state: present

    - name: Pakiety potrzebne
      ansible.builtin.dnf:
        name:
          - httpd
          - tmux
        state: present

    - name: Usuniecie telnet
      ansible.builtin.dnf:
        name: telnet
        state: absent

    - name: Grupa pakietow Development Tools
      ansible.builtin.dnf:
        name: "@Development Tools"
        state: present
```

```
ansible-playbook oprogramowanie.yml
ansible web -m ansible.builtin.command -a "dnf repolist --all"
```

Moduł `yum_repository` zapisuje plik `/etc/yum.repos.d/zielona-lab.repo`. Pole `enabled: false` pozwala przećwiczyć moduł bez konieczności stawiania prawdziwego repozytorium, a w zadaniu egzaminacyjnym wpisujemy adres ze zlecenia. Grupy pakietów podajemy z `@` na początku.

1. Piszemy playbook, który na `rhlab04` uruchamia `httpd` na porcie 8888: zmienia `Listen`, nadaje portowi etykietę SELinux `http_port_t`, otwiera port w zaporze na stałe, włącza przełącznik `httpd_can_network_connect` i startuje usługę.

Odpowiedź

```
---
- name: Httpd na porcie 8888
  hosts: web
  become: true
  vars:
    port_www: 8888
  tasks:
    - name: Pakiety
      ansible.builtin.dnf:
        name:
          - httpd
          - policycoreutils-python-utils
        state: present

    - name: Port nasluchu
      ansible.builtin.lineinfile:
        path: /etc/httpd/conf/httpd.conf
        regexp: '^Listen '
        line: "Listen {{ port_www }}"
      notify: Restart httpd

    - name: Etykieta portu dla httpd
      community.general.seport:
        ports: "{{ port_www }}"
        proto: tcp
        setype: http_port_t
        state: present

    - name: Port w zaporze
```

```
      ansible.posix.firewalld:
        port: "{{ port_www }}/tcp"
        permanent: true
        immediate: true
        state: enabled

    - name: Przelacznik SELinux dla httpd
      ansible.posix.seboolean:
        name: httpd_can_network_connect
        state: true
        persistent: true

    - name: Wlaczenie i start httpd
      ansible.builtin.service:
        name: httpd
        state: started
        enabled: true

  handlers:
    - name: Restart httpd
      ansible.builtin.service:
        name: httpd
        state: restarted
```

```
ansible-playbook port8888.yml
curl -s -o /dev/null -w "%{http_code}\n" http://rhlab04:8888
```

Moduł `seport` wymaga na węźle zarządzanym pakietu z bibliotekami Pythona do SELinux (`python3-libsemanage`), który wchodzi w skład `policycoreutils-python-utils`, więc instalujemy go w pierwszym zadaniu. Gdy port 8888 ma już inny typ w polityce, moduł zgłosi to w błędzie i wtedy sprawdzamy `semanage port -l | grep 8888`. Po restarcie usługa musi wstać sama, co potwierdza `systemctl is-enabled httpd`.

1. Tworzymy grupę `serwis`, konta `ola` i `marek` w tej grupie z kluczem SSH z pliku, dajemy grupie `serwis` prawa `sudo` bez hasła w pliku sprawdzanym przez `visudo` i ustawiamy wygaśnięcie hasła po 90 dniach.

Odpowiedź

```
---
- name: Konta serwisowe
  hosts: lab
  become: true
  vars:
    konta:
      - ola
      - marek
  tasks:
    - name: Grupa serwis
      ansible.builtin.group:
        name: serwis
        state: present

    - name: Konta w grupie serwis
      ansible.builtin.user:
        name: "{{ item }}"
        groups: serwis
        append: true
        password_expire_max: 90
        state: present
      loop: "{{ konta }}"

    - name: Klucz SSH dla kont
      ansible.posix.authorized_key:
        user: "{{ item }}"
        key: "{{ lookup('file', lookup('env', 'HOME') + '/.ssh/id_ed25519.pub') }}"
        state: present
      loop: "{{ konta }}"

    - name: Sudo dla grupy serwis
      ansible.builtin.copy:
        content: "%serwis ALL=(ALL) NOPASSWD: ALL\n"
        dest: /etc/sudoers.d/serwis
        owner: root
        group: root
        mode: "0440"
        validate: /usr/sbin/visudo -cf %s
```

```
ansible-playbook konta-serwis.yml
ansible lab -b -m ansible.builtin.command -a "chage -l ola"
ansible lab -b -m ansible.builtin.command -a "visudo -c"
```

Klucz z pliku na węźle sterującym wczytuje wyszukiwarka `lookup('file', ...)`. Parametr `validate` sprawdza plik przed zapisaniem, więc błąd w składni `sudoers` nie zablokuje konta administratora. Prawa `0440` są wymagane przez `sudo` dla plików w `/etc/sudoers.d/`.

1. Dodajemy zadanie `cron`, które co 15 minut zapisuje czas w `/var/tmp/cron-test`, zadanie `at` na jednorazowy zapis za pięć minut oraz ustawiamy nazwę hosta zgodną z inwentarzem.

Odpowiedź

```
---
- name: Zadania zaplanowane
  hosts: lab
  become: true
  tasks:
    - name: Cron co 15 minut
      ansible.builtin.cron:
        name: Zapis czasu
        minute: "*/15"
        job: "date >> /var/tmp/cron-test"
        user: root
        state: present

    - name: Zadanie at za piec minut
      ansible.posix.at:
        command: "date >> /var/tmp/at-test"
        count: 5
        units: minutes
        unique: true

    - name: Nazwa hosta
      ansible.builtin.hostname:
        name: "{{ inventory_hostname }}.lab.zielona.example"
```

```
ansible-playbook zaplanowane.yml
ansible lab -b -m ansible.builtin.command -a "crontab -l"
ansible lab -b -m ansible.builtin.command -a "atq"
```

Pole `name` zadania cron jest etykietą, po której moduł rozpoznaje własny wpis, więc drugie uruchomienie niczego nie dubluje. Usunięcie wpisu to `state: absent` z tym samym `name`. Moduł `at` wymaga na węźle zainstalowanego i uruchomionego `atd`.

Dzień 28: automatyzacja zadań RHCSA, część 2, Git i VS Code

> **Z ŻYCIA WZIĘTE**
>
> Wtorek, 11:00. Ola chce, żeby wszystko, co Tomek napisał przez trzy tygodnie, leżało w repozytorium, a nie w katalogu domowym na jednej maszynie. „Co się stanie, jeśli `rhlab01` padnie?”, pyta. Tomek najpierw dopisuje do playbooka magazyn danych na LVM, potem archiwum, a na końcu uczy się zapisać wszystko jednym `git push`.

Cele egzaminu

- automatyzacja zadań RHCSA: systemy plików, urządzenia magazynowe, archiwizacja
- podstawowe operacje Git: klonowanie repozytorium i dodawanie plików
- praca w VS Code: tworzenie playbooków, wysyłanie ich do repozytorium Git, konfiguracja `ansible-navigator`, uruchamianie w kontenerze deweloperskim

Czytamy w bazie

- LVM: dysk fizyczny, grupa wolumenów, wolumen logiczny i Montowanie: mount, /etc/fstab, zasoby sieciowe
- Powiększanie przestrzeni dyskowej bez restartu
- Pierwsze repozytorium: od pustego projektu do pierwszego pusha

Krótko o tym, czego baza nie ma

Baza opisuje Git na przykładzie Gitea z pierwszego repozytorium. Na egzaminie wystarczy znać cztery czynności: sklonować, dodać plik, zapisać i wysłać. Dochodzi do tego `.gitignore`, w którym wpisujemy plik hasła do sejfu i katalogi z pobranymi plikami, żeby nie trafiły do repozytorium.

> **Pojęcie: kontener deweloperski.** Obraz kontenera z zainstalowanym Ansible, `ansible-navigator` i narzędziami deweloperskimi, w którym VS Code otwiera projekt, edytuje playbooki i je uruchamia. Strona Red Hat o egzaminie EX294 wymienia VS Code i kontener deweloperski wśród używanych narzędzi, ale nie opisuje szczegółów, więc w laboratorium wiernie tego środowiska nie odtworzymy. Sprawdzamy to na stronie egzaminu i w jego opisie przy rezerwacji.

Do magazynu w Ansible używamy modułów z kolekcji `community.general`: `lvg` (grupa wolumenów), `lvol` (wolumen logiczny), `filesystem` (system plików) i `parted` (partycje), a do montowania `ansible.posix.mount`. Sprawdzamy w `ansible-doc`, czy są dostępne. Alternatywą jest rola systemowa `storage` z pakietu `rhel-system-roles`.

Ćwiczenia

1. Na `rhlab03` zakładamy z dysku `/dev/sdb` grupę wolumenów `vgDane` i wolumen `lvDane` o rozmiarze 2 GB, tworzymy system plików XFS i montujemy go w `/srv/dane` na stałe. Zadania mają się pominąć, gdy dysku nie ma.

Odpowiedź

```
---
- name: Magazyn danych na LVM
  hosts: rhlab03
  become: true
  tasks:
    - name: Sprawdzenie dysku sdb
      ansible.builtin.debug:
        msg: "Dysk sdb: {{ 'jest' if ansible_facts['devices']['sdb'] is defined else
'brak' }}"

    - name: Magazyn na sdb
      when: ansible_facts['devices']['sdb'] is defined
      block:
        - name: Grupa wolumenow vgDane
          community.general.lvg:
```

```
      vg: vgDane
      pvs: /dev/sdb
      state: present

  - name: Wolumen lvDane
    community.general.lvol:
      vg: vgDane
      lv: lvDane
      size: 2g
      state: present

  - name: System plikow XFS
    community.general.filesystem:
      fstype: xfs
      dev: /dev/vgDane/lvDane

  - name: Montowanie w srv dane
    ansible.posix.mount:
      path: /srv/dane
      src: /dev/vgDane/lvDane
      fstype: xfs
      state: mounted
```

```
ansible-playbook magazyn.yml
ansible rhlab03 -b -m ansible.builtin.command -a "lsblk /dev/sdb"
ansible rhlab03 -b -m ansible.builtin.command -a "findmnt /srv/dane"
ansible rhlab03 -b -m ansible.builtin.command -a "grep dane /etc/fstab"
```

Stan `mounted` montuje system plików od razu i zapisuje wpis w `/etc/fstab`, więc konfiguracja przetrwa restart. Nazwy `vgDane` i `lvDane` podążają za konwencją Zielonej (`vg<Przeznaczenie>`, `lv<Przeznaczenie>`). Gdy w wyniku pierwszego zadania widzimy `brak`, sprawdzamy w `lsblk` nazwę dysku, bo kontroler VirtIO nazywa go `vdb`.

1. Powiększamy wolumen `lvDane` do 3 GB razem z systemem plików, zmieniając tylko wartość `size`, a potem bez zmian uruchamiamy playbook drugi raz.

Odpowiedź

W zadaniu `Wolumen lvDane` zmieniamy rozmiar i dodajemy `resizefs`:

```
        - name: Wolumen lvDane
          community.general.lvol:
            vg: vgDane
            lv: lvDane
            size: 3g
            resizefs: true
            state: present
```

```
ansible-playbook magazyn.yml
ansible rhlab03 -b -m ansible.builtin.command -a "df -h /srv/dane"
ansible-playbook magazyn.yml
```

Pierwsze uruchomienie zgłasza zmianę tylko w zadaniu z wolumenem, a drugie nie zmienia niczego. Moduł `lvol` nie zmniejsza wolumenu bez opcji `force: true`, a XFS w ogóle nie da się zmniejszyć, więc rozmiar tylko zwiększamy.

1. Tworzymy archiwum `/etc/ssh` na węzłach `dane` w pliku `/var/tmp/ssh-kopia.tar.gz`, rozpakowujemy je do `/var/tmp/ssh-test` i pobieramy archiwum z `rhlab02` na węzeł sterujący.

Odpowiedź

```
---
- name: Archiwum konfiguracji ssh
  hosts: dane
  become: true
  tasks:
    - name: Archiwum etc ssh
      ansible.builtin.archive:
        path: /etc/ssh
        dest: /var/tmp/ssh-kopia.tar.gz
        format: gz

    - name: Katalog na rozpakowanie
      ansible.builtin.file:
        path: /var/tmp/ssh-test
        state: directory
        mode: "0755"
```

```
    - name: Rozpakowanie archiwum
      ansible.builtin.unarchive:
        src: /var/tmp/ssh-kopia.tar.gz
        dest: /var/tmp/ssh-test
        remote_src: true

    - name: Pobranie archiwum
      ansible.builtin.fetch:
        src: /var/tmp/ssh-kopia.tar.gz
        dest: "{{ lookup('env', 'HOME') }}/pobrane/"
      when: inventory_hostname == 'rhlab02'
```

```
ansible-playbook archiwum.yml
ansible dane -b -m ansible.builtin.command -a "ls /var/tmp/ssh-test/ssh"
```

Opcja `remote_src: true` oznacza, że archiwum leży już na węźle zarządzanym, a bez niej `unarchive` szuka go na węźle sterującym. Formaty `bz2` i `xz` wybieramy polem `format`, a `zip` jest alternatywą dla Windows.

1. Zakładamy repozytorium bare na `rhlab02`, klonujemy je na `rhlab01`, dodajemy playbooki i `.gitignore` z plikiem hasła sejfu, zapisujemy zmiany i wysyłamy je.

Odpowiedź

Na `rhlab02`:

```
mkdir -p ~/repo
git init --bare --initial-branch=main ~/repo/ansible.git
```

Na `rhlab01`:

```
git config --global user.name "Tomek"
git config --global user.email "tomek@zielona.example"
git clone ssh://tomek@rhlab02/home/tomek/repo/ansible.git ~/ansible-git
cd ~/ansible-git
cp ~/ansible/*.yml ~/ansible/ansible.cfg ~/ansible/inventory .
cp -r ~/ansible/roles ~/ansible/group_vars .
tee .gitignore <<'EOF'
.vault_pass
```

```
*.retry
pobrane/
EOF
git add .
git status --short
git commit -m "Playbooki z tygodni 3 i 4"
git push -u origin main
git log --oneline
```

Klon pustego repozytorium pokazuje ostrzeżenie o pustym repozytorium, co jest normalne. Plik `.vault_pass` leży poza katalogiem projektu, więc zostaje na maszynie, a wpis w `.gitignore` chroni przed przypadkowym skopiowaniem go do repozytorium. Zaszyfrowany `vars/tajne.yml` wolno zapisać, bo bez hasła jest bezużyteczny. Na koniec sprawdzamy w `git log`, że zapis jest w historii.

1. Otwieramy katalog projektu w VS Code na własnej stacji, instalujemy rozszerzenia do Ansible i kontenerów deweloperskich, opisujemy kontener w pliku `.devcontainer/devcontainer.json` i uruchamiamy z niego `ansible-navigator`. Zapisujemy, czym ta praca różni się od pracy w terminalu.

Odpowiedź

Tej części nie da się w pełni sprawdzić bez środowiska egzaminacyjnego, więc traktujemy ją jako zapoznanie się z narzędziem. Na stacji z zainstalowanym VS Code i Dockerem albo Podmanem:

1. Instalujemy rozszerzenia: Ansible (Red Hat) i Dev Containers.
2. Otwieramy katalog `~/ansible-git` (`File`, `Open Folder`).
3. Dodajemy plik `.devcontainer/devcontainer.json` w kształcie:

```
{
  "name": "Zielona Ansible",
  "image": "ghcr.io/ansible/community-ansible-dev-tools:latest"
}
```

1. Wybieramy z palety poleceń `Dev Containers: Reopen in Container`. VS Code pobiera obraz i otwiera projekt w kontenerze.
2. W terminalu kontenera uruchamiamy `ansible-navigator run ping.yml --mode stdout`.

Nazwa obrazu to przykład, który sprawdzamy w dokumentacji narzędzi Ansible, bo egzamin dostarcza własny obraz. Pracę kończymy tak samo jak w terminalu: `git add`, `git commit` i `git`

`push`, które VS Code ma także w panelu kontroli źródła. Ważne, że playbook uruchomiony w kontenerze musi mieć dostęp do klucza SSH i inwentarza, więc w kontenerze sprawdzamy `ansible --version` i `ansible-inventory --graph` jako pierwsze.

Częste błędy

Objaw	Przyczyna	Co robimy
`Decryption failed` albo `Attempting to decrypt but no vault secrets found`	brak hasła do sejfu albo błędne hasło	`--vault-password-file`, `--ask-vault-pass` albo `vault_password_file` w `ansible.cfg`
Hasło praktykanta zmienia się przy każdym uruchomieniu	filtr `password_hash` losuje sól	`update_password: on_create`
`the role 'rhel-system-roles.timesync' was not found`	`roles_path` bez `/usr/share/ansible/roles` albo zła nazwa	`ansible-galaxy role list`, dopisanie ścieżki
Nie widać zainstalowanej kolekcji	katalog poza `collections_path`	`-p ./collections` albo `collections_path` w `ansible.cfg`
`lvol` nie zmniejsza wolumenu	moduł nie zmniejsza bez `force`	w XFS nie zmniejszamy, tylko zwiększamy
`seport` zgłasza brak bibliotek	na węźle brak `python3-libsemanage`	zadanie z `dnf` na pakiecie `policycoreutils-python-utils`
Plik w `sudoers.d` blokuje `sudo`	błąd składni zapisany bez sprawdzenia	`validate: /usr/sbin/visudo -cf %s`
Plik hasła sejfu trafił do repozytorium	brak wpisu w `.gitignore`	wpis w `.gitignore`, zmiana hasła sejfu i usunięcie z historii
Rolę z adresu URL widać pod złą nazwą	brak `name` w `requirements.yml`	`name:` obok `src:`

Co dalej

Dni 29 i 30: próbny egzamin EX294 na czas i strategię na dzień egzaminu opisuje strona Próbny egzamin EX294 i strategia na dzień egzaminu. Przed nim warto raz wrócić do stron o tygodniach z RHCSA, bo egzamin EX294 zakłada także umiejętności z EX200.

Źródła: strona Red Hat o egzaminie EX294 (cele: obsługa błędów, role i kolekcje, zadania RHCSA modułami, Vault, szablony, Git, VS Code z kontenerem deweloperskim, stan na 10 października 2026); dokumentacja Ansible (Vault, role, kolekcje, moduły z `ansible.builtin`, `ansible.posix` i `community.general`); Red Hat, Considerations in adopting RHEL 10, rozdział System roles. Polecenia `ansible-vault`

, `ansible-galaxy` (role z URL, kolekcje z archiwum), opcje playbooków i składnię przykładów sprawdzono na ansible-core 2.19; układ pakietu `rhel-system-roles` i zachowanie `seport` na RHEL nie były testowane.

Revision #8
Created 2026-10-10 13:47:40 UTC by Tomasz Konieczny
Updated 2026-10-26 14:25:25 UTC by Tomasz Konieczny