Ansible: podstawy operacyjne

Poziom: podstawy **Dotyczy: Ansible Core 2.16+** **Czas: ~35 min**

Ansible łączy się po SSH i wykonuje moduły napisane w Pythonie — cała reszta (inwentarz, playbooki, role) to sposób na opisanie, na jakich maszynach i w jakiej kolejności ma to zrobić.

Komendy wykonujemy z CLI bash jako user z dostępem SSH do maszyn zarządzanych.

Skróty użyte na tej stronie (6)

Skrót	Co znaczy
API	Application Programming Interface: sposób, w jaki programy rozmawiają ze sobą
CLI	Command Line Interface: praca przez wpisywanie komend w terminalu
CMDB	Configuration Management Database: baza konfiguracji, czyli spis elementów, z których składają się usługi, i ich powiązań; w Zielonej rolę tę pełni NetBox
ICMP	Internet Control Message Protocol: protokół diagnostyczny sieci, którym działa między innymi ping
SSH	Secure Shell: szyfrowane zdalne logowanie do serwera
YAML	YAML Ain't Markup Language: format plików konfiguracyjnych oparty na wcięciach

Fiszka

TYLKO ODCZYT

Sprawdzenie łączności z maszynami: ansible -m ping

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

Wywołujemy moduł `ping` (nie ICMP — to moduł Ansible sprawdzający, że Python jest dostępny na maszynie docelowej i że SSH działa) na wszystkich maszynach z inwentarza.

Przykład:

```
app01.ipa.zielona.example | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
pg01.ipa.zielona.example | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

ICMP = Internet Control Message Protocol: protokół diagnostyczny sieci, którym działa między innymi ping · **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 pojedynczej komendy ad-hoc na grupie maszyn

```
ansible web -i inventory.ini -a "df -h /"
```

> **Pojęcie: komenda ad-hoc.** Wywołanie pojedynczego modułu Ansible bezpośrednio z linii poleceń, bez pisania playbooka — użyteczne do szybkiego sprawdzenia stanu albo jednorazowej zmiany na kilku maszynach naraz, bez tworzenia trwałego pliku.

Przykład:

```
app01.ipa.zielona.example | CHANGED | rc=0 >>
Filesystem      Size  Used Avail Use% Mounted on
/dev/mapper/vgOS-lvOS   60G   12G   48G  20% /
```

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

TYLKO ODCZYT

Instalacja pakietu na grupie maszyn ad-hoc

```
ansible web -i inventory.ini -b -m dnf -a "name=git state=present"
```

Flaga `-b` (become) podnosi uprawnienia do roota na czas wykonania modułu — instalacja pakietu wymaga uprawnień administracyjnych, których zwykłe konto SSH zazwyczaj nie ma.

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

ZMIENIA SYSTEM

Uruchomienie playbooka

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

ansible-playbook = uruchom scenariusz Ansible

TYLKO ODCZYT

Podgląd zmian bez ich wprowadzania: --check

```
ansible-playbook -i inventory.ini plik.yml --check
```

> **Pojęcie: tryb `--check` (dry-run).** Ansible przechodzi przez wszystkie zadania playbooka i raportuje, co by zmienił, ale nie wprowadza żadnej faktycznej zmiany na maszynach docelowych — nie każdy moduł wspiera ten tryb w 100%, zwłaszcza moduły wywołujące dowolne komendy powłoki, które nie mają jak przewidzieć swojego efektu bez faktycznego wykonania.

ansible-playbook = uruchom scenariusz Ansible

TYLKO ODCZYT

Podgląd treści zmian: --diff

```
ansible-playbook -i inventory.ini plik.yml --check --diff
```

Dokładamy do trybu `--check` wyświetlenie różnicy treści plików, które zostałyby zmienione — przydatne przy zadaniach modyfikujących pliki konfiguracyjne, żeby zobaczyć dokładnie, która linia by się zmieniła, zanim faktycznie się zmieni.

ansible-playbook = uruchom scenariusz Ansible

ZMIENIA SYSTEM

Ograniczenie wykonania do wybranej grupy lub maszyny: --limit

```
ansible-playbook -i inventory.ini plik.yml --limit app01.ipa.zielona.example
```

Zawężamy wykonanie playbooka do jednej maszyny albo grupy, mimo że plik inwentarza i sam playbook obejmują znacznie więcej — użyteczne przy testowaniu zmiany na jednej maszynie przed wypuszczeniem jej na całą flotę.

ansible-playbook = uruchom scenariusz Ansible

ZMIENIA SYSTEM

Wykonanie tylko wybranych zadań: --tags

```
ansible-playbook -i inventory.ini plik.yml --tags "konfiguracja,restart"
```

Wykonujemy wyłącznie zadania oznaczone jedną ze wskazanych etykiet, pomijając resztę playbooka — pozwala trzymać wiele powiązanych czynności w jednym pliku, ale wywoływać je selektywnie zależnie od potrzeby (np. sama zmiana konfiguracji bez pełnego wdrożenia aplikacji).

ansible-playbook = uruchom scenariusz Ansible

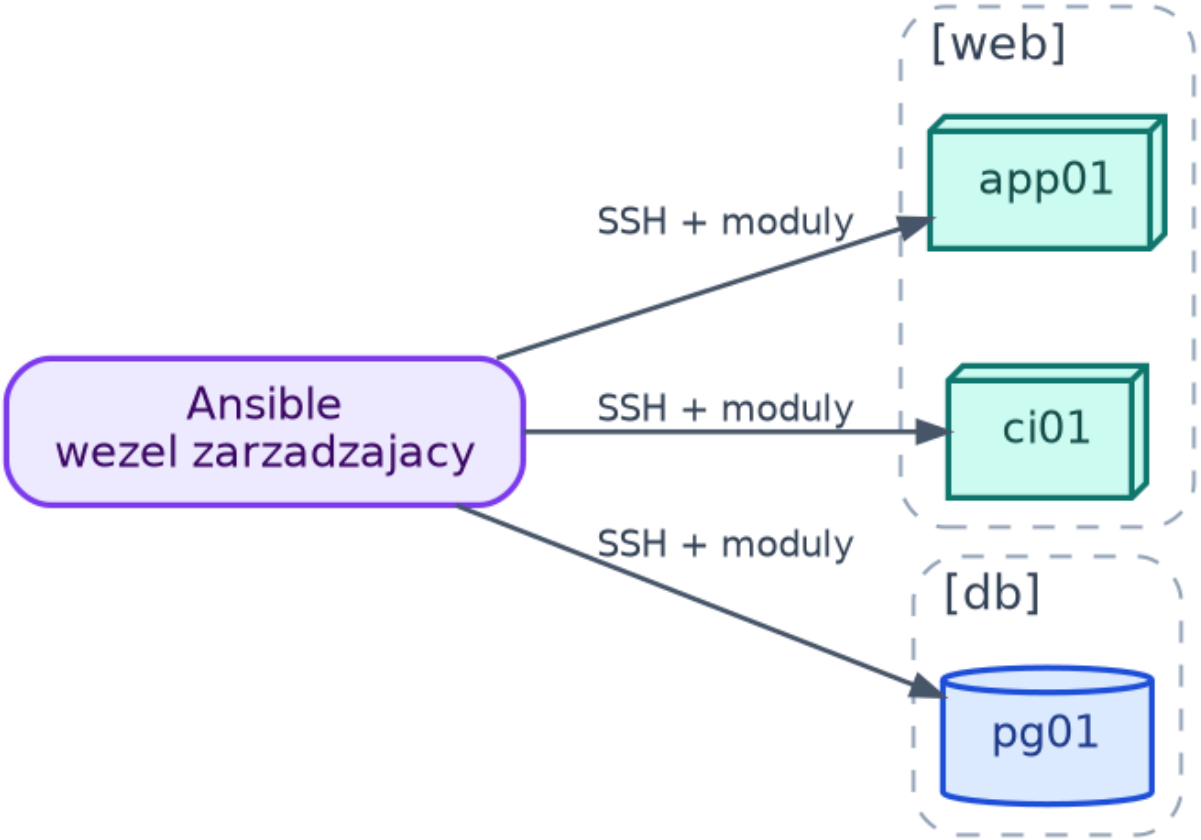
Z życia wzięte

Z ŻYCIA WZIĘTE

W środę rano Tomek uruchamia playbook konfigurujący nginksa od razu na całym inwentarzu, bez `--limit`, bo „to tylko drobna zmiana". Literówka w nazwie pakietu psuje konfigurację na wszystkich serwerach z grupy `web` naraz, zamiast na jednym testowym. Naprawa zajmuje dłużej niż samo pisanie playbooka, a Tomek od tej pory zawsze zaczyna od jednej maszyny.

Inwentarz

> **Pojęcie: inwentarz.** Lista maszyn, którymi Ansible zarządza, razem z ich przypisaniem do grup i zmiennymi specyficznymi dla każdej z nich. Bez inwentarza Ansible nie wie, do czego się łączyć — nawet pojedyncza maszyna podana ad-hoc w linii poleceń jest technicznie inwentarzem złożonym z jednego wpisu.



Ansible łączy się po SSH z każdą maszyną z grupy web i db osobno, bez żadnego trwałego agenta po ich stronie (kliknięcie otwiera pełny rozmiar)

Inwentarz statyczny

Najprostsza forma to plik tekstowy w formacie INI:

```
[web]
app01.ipa.zielona.example
ci01.ipa.zielona.example
```

```
[db]
pg01.ipa.zielona.example

[web:vars]
http_port=8080
```

Grupy (`[web]`, `[db]`) pozwalają kierować playbook na cały zestaw maszyn naraz, a nie na każdą z osobna. Sekcja `[nazwa_grupy:vars]` ustawia zmienne wspólne dla wszystkich maszyn danej grupy.

Inwentarz dynamiczny

> **Pojęcie: inwentarz dynamiczny.** Zamiast statycznej listy w pliku, skrypt albo wtyczka odpytująca zewnętrzne źródło prawdy (API dostawcy chmury, CMDB, wynik `terraform output`) i budująca listę maszyn oraz grup w locie, przy każdym uruchomieniu. Ma sens tam, gdzie zestaw maszyn zmienia się częściej, niż ktokolwiek nadążyłby ręcznie aktualizować plik statyczny — typowo przy maszynach tworzonych i niszczonych automatycznie w chmurze.

Grupy i zmienne

Grupy mogą się zagnieżdżać — grupa grup (`[grupa_nadrzedna:children]`) pozwala budować hierarchię zamiast płaskiej listy. Zmienne przypisane maszynie mają pierwszeństwo przed zmiennymi grupy, a zmienna grupy bardziej szczegółowej — przed zmienną grupy nadrzędnej; ta hierarchia pierwszeństwa pozwala ustawić wartość domyślną na poziomie ogólnym i nadpisać ją punktowo tam, gdzie to konieczne, bez duplikowania całej reszty konfiguracji.

ansible.cfg

> **Pojęcie: `ansible.cfg`.** Plik konfiguracyjny czytany przez Ansible w określonej kolejności lokalizacji (najpierw katalog bieżący, potem katalog domowy, potem `/etc/ansible/ansible.cfg`) — ustawia domyślną ścieżkę do pliku inwentarza, użytkownika SSH, liczbę równoległych połączeń (`forks`) i dziesiątki innych parametrów, żeby nie trzeba było powtarzać ich jako flag przy każdym wywołaniu.

```
[defaults]
inventory = ./inventory.ini
remote_user = nazwa_usera
forks = 10
host_key_checking = False
```

Struktura playbooka

> **Pojęcie: playbook.** Plik YAML opisujący jeden lub więcej `play` — każdy `play` przypisuje zestaw zadań (`tasks`) do konkretnej grupy maszyn z inwentarza:

```
- name: Konfiguracja serwerów web
  hosts: web
  become: true
  vars:
    http_port: 8080
  tasks:
    - name: Instalacja nginksa
      ansible.builtin.dnf:
        name: nginx
        state: present

    - name: Uruchomienie i włączenie usługi
      ansible.builtin.systemd:
        name: nginx
        state: started
        enabled: true
```

Każde zadanie deklaruje pożądany stan, nie kolejność komend — moduł `dnf` z `state: present` nic nie robi, jeśli pakiet już jest zainstalowany, zamiast bezwarunkowo wywoływać instalację od nowa.

To właśnie czyni playbook idempotentnym, temat rozwinięty na stronie [Idempotentność: skrypt, który wolno uruchomić dwa razy](#).

Role

>

> **Pojęcie: rola.** Standardowa struktura katalogów grupująca zadania, pliki, szablony i zmienne dotyczące jednej odpowiedzialności (np. „serwer nginks", „klient PostgreSQL") w wielokrotnego użytku pakiet, który da się dołączyć do dowolnego playbooka jedną linią:

```
roles/
  nginx/
    tasks/main.yml
    handlers/main.yml
    templates/nginx.conf.j2
    defaults/main.yml
```

```
- name: Konfiguracja serwerów web
  hosts: web
  roles:
    - nginx
```

Role rozwiązują problem powtarzania tych samych zadań w wielu playbookach różnych projektów — ta sama rola `nginx` używana jest identycznie niezależnie od tego, jaki playbook ją wywołuje.

ansible-vault

> **Pojęcie: `ansible-vault`.** Narzędzie do szyfrowania plików ze zmiennymi zawierającymi sekrety (hasła, klucze API) hasłem albo kluczem zewnętrznym, żeby mogły bezpiecznie trafić do tego samego repozytorium co reszta playbooka, bez ujawniania treści w postaci jawnej.

Zaszyfrowanie pliku ze zmiennymi

```
ansible-vault encrypt zmienne_tajne.yml
```

Uruchomienie playbooka z odszyfrowaniem w locie

```
ansible-playbook -i inventory.ini plik.yml --ask-vault-pass
```

Połączenie SSH i eskalacja uprawnień

> **Pojęcie: `become`.** Mechanizm podnoszenia uprawnień na maszynie docelowej — odpowiednik wpisania `sudo` przed komendą, tylko zadeklarowany w playbooku (`become: true`) albo w linii poleceń (`-b`), a nie wpisywany ręcznie za każdym razem. Domyślną metodą eskalacji jest `sudo`, ale Ansible obsługuje też inne (`su`, `doas`), konfigurowalne przez `become_method`.

Połączenie samo w sobie odbywa się zwykłym SSH kontem systemowym, zdefiniowanym w `ansible.cfg` (`remote_user`) albo per host w inwentarzu (`ansible_user=nazwa_usera`) — Ansible nie wymaga żadnego dodatkowego agenta zainstalowanego na maszynie docelowej poza samym klientem SSH i interpreterem Pythona, których w praktyce niemal każda dystrybucja Linuksa ma domyślnie.

Częste błędy

Pisanie zadań jako sekwencji komend zamiast deklaracji stanu. Zadanie `ansible.builtin.command: dnf install -y nginx` działa, ale traci idempotentność modułu `dnf` — wykona się (i zgłosi zmianę) przy każdym uruchomieniu, niezależnie od tego, czy pakiet już jest zainstalowany.

Testowanie zmiany od razu na całym inwentarzu. Brak `--limit` przy pierwszym uruchomieniu nowego albo zmodyfikowanego playbooka oznacza, że błąd w logice zadania trafia od razu na każdą maszynę z grupy, zamiast zostać złapanym na jednej testowej.

Trzymanie haseł w zwykłych zmiennych zamiast w `ansible-vault`. Plik ze zmiennymi trafia zwykle do tego samego repozytorium co reszta playbooka — hasło w postaci jawnej zostaje w historii commitów na stałe.

□□ Ćwiczenia

> Zadania sprawdzające.

1. Piszemy plik inwentarza statycznego z dwiema grupami — `web` (dwie maszyny) i `db` (jedna maszyna) — i zmienną `http_port` wspólną dla grupy `web`.

2. Wyjaśniamy różnicę między komendą ad-hoc a playbookiem i podajemy sytuację, w której ad-hoc wystarcza, a playbook byłby przesadą.
3. Piszemy playbook instalujący i uruchamiający usługę `nginx`, wykonujący się z podniesionymi uprawnieniami, wyłącznie na grupie `web`.
4. Chcemy sprawdzić, co dokładnie zmieniłby playbook na jednej konkretnej maszynie, zanim uruchomimy go naprawdę. Piszemy odpowiednie wywołanie `ansible-playbook` z właściwymi flagami.

Powiązane: Ansible, AWX, Semaphore, Rudder, Terraform: co do czego · AWX i Semaphore · Idempotentność: skrypt, który wolno uruchomić dwa razy

---

Revision #37
Created 2026-09-25 07:40:05 UTC by Tomasz Konieczny
Updated 2026-10-26 14:13:28 UTC by Tomasz Konieczny