Ansible - nie taka trudna technologia

Michał Przyłucki

2025-05-25

I. Wstęp

Wiele osób, zwłaszcza w Polsce, kojarzy Ansible z ogromnymi, skomplikowanymi wdrożeniami w korporacjach, gdzie setki maszyn są konfigurowane przez doświadczonych DevOpsów. Tymczasem to narzędzie może być przydatne również w mniejszych, a nawet prywatnych projektach. Jego prostota i czytelna składnia YAML sprawiają, że każdy - nawet osoba, która nie czuje się jeszcze ekspertem w dziedzinie DevOps - może zacząć automatyzować np. deployment aplikacji czy zarządzanie serwerami.

Wbrew pozorom, Ansible nie wymaga wielkiego zaplecza technologicznego, skomplikowanego środowiska czy specjalistycznego sprzętu. Wystarczy dostęp SSH do serwera i kilka podstawowych informacji o tym, co chcesz zautomatyzować, aby w krótkim czasie osiągnąć konkretne rezultaty. Dzięki temu, że Ansible jest agentless — nie wymaga instalowania dodatkowego oprogramowania na serwerach docelowych — możesz zacząć działać praktycznie od razu, niezależnie od tego, czy zarządzasz jednym VPS-em, czy kilkoma środowiskami stagingowymi w chmurze. zawartość...

Co więcej, Ansible pozwala łatwo skalować się w miarę rozwoju Twojego projektu. Gdy w Twoim środowisku pojawia się kolejny serwer lub instancja w chmurze, wystarczy dopisać go do pliku inventory i uruchomić Playbook — Ansible zadba o to, żeby konfiguracja była spójna i zgodna z tym, co już wdrożyłeś wcześniej. Dzięki temu unikniesz chaosu i tzw. „konfiguracyjnego spaghetti", które często towarzyszy ręcznemu wdrażaniu kolejnych komponentów w dynamicznie rosnącym projekcie.

W tym artykule pokażę Ci, jak w praktyce wykorzystać Ansible do deploymentu aplikacji na przykładzie AWS EC2. Przekonasz się, że stworzenie Playbooka nie jest wcale takie trudne, a sam proces wdrożenia aplikacji — od pobrania kodu źródłowego, przez instalację zależności, aż po restart usług — można opisać w kilku prostych zadaniach YAML. Dzięki temu nawet w małym projekcie, bez rozbudowanej infrastruktury DevOps, możesz poczuć się jak prawdziwy inżynier automatyzacji.

II. Czym jest Ansible?

Ansible to nowoczesne narzędzie do automatyzacji zadań, które pozwala w prosty sposób zarządzać infrastrukturą, konfigurować serwery i wdrażać aplikacje. W odróżnieniu od wielu innych rozwiązań tego typu, Ansible nie wymaga instalowania agentów na maszynach docelowych — wystarczy dostęp SSH i Python (dostępny praktycznie na każdej współczesnej dystrybucji Linuksa). Dzięki temu można zacząć pracę niemal od razu, bez skomplikowanego przygotowania środowiska.

Sercem Ansible jest Playbook — plik w formacie YAML, który opisuje, co i jak ma zostać wykonane na wskazanych serwerach. Każdy Playbook składa się z zadań (tasks), które są uruchamiane w kolejności, umożliwiając krok po kroku instalowanie oprogramowania, konfigurowanie usług czy wdrażanie aplikacji. Dzięki swojej idempotentności (czyli powtarzalności bez skutków

ubocznych), Ansible zapewnia, że środowisko będzie zawsze w tym samym, pożądanym stanie — niezależnie od tego, ile razy uruchomisz ten sam Playbook.

Kolejnym ważnym elementem jest inventory — plik tekstowy (np. inventory.ini), w którym definiujesz, na jakich hostach (czyli maszynach lub instancjach) mają być wykonywane zadania. Możesz w nim określić grupy serwerów, takie jak webservers, dbservers czy test, co pozwala na łatwe skalowanie Twoich projektów — od pojedynczej maszyny aż po setki czy tysiące serwerów w środowiskach produkcyjnych.

Ansible jest narzędziem, które z powodzeniem wykorzystuje się w dużych firmach — do zarządzania setkami maszyn i integracji z systemami CI/CD — ale jego prosta, czytelna składnia i brak konieczności instalowania dodatkowego agenta sprawiają, że jest także doskonałym wyborem do mniejszych projektów czy nawet prywatnych serwerów. Możesz w łatwy sposób przygotować sobie automatyczne wdrożenie swojej aplikacji Django czy Node.js, zainstalować Nginxa, przygotować bazy danych czy skonfigurować zaporę ogniową — wszystko w jednym pliku YAML.

Podsumowując, Ansible to:

- Proste narzędzie do automatyzacji (działa przez SSH, bez agentów)
- Playbooki w formacie YAML — czytelne i łatwe do pisania
- Inventory — definiujesz grupy serwerów i możesz skalować deployment
- Idempotentność — Playbook zawsze doprowadza środowisko do pożądanego stanu
- Wszechstronność — od małego projektu po środowiska korporacyjne

Dzięki tym cechom Ansible stał się jednym z najpopularniejszych narzędzi DevOps — i właśnie dlatego warto się go nauczyć, niezależnie od tego, czy jesteś początkującym programistą, administratorem czy DevOpsem w dużej firmie.

III. Przykładowe scenariusze

Jest bardzo wiele zastosowań dla technologii Ansible. Od aktualizacji serwerów po deploy całych aplikacji. Można go używać do prostych małych operacji na kilku serwerach oraz do aktualizowania wersji aplikacji webowej na każdym środowisku. Tworzenie prostych i skalowalnych rozwiązań sprowadza się do napisania plików, tzw. Playbooków w formacie YAML, a następnie ich uruchomienia na dowolnej liczbie serwerów/VMek.

Kilka przykładowych scenariuszy, w których użycie Ansible mogłoby mieć znaczący wpływ na prędkość wdrożenia:

- Automatyczne aktualizacje systemów - za pomocą Ansible można przygotować playbook, który zaktualizuje wszystkie serwery na liście, zapewniając, że każdy jest na tej samej wersji systemu i oprogramowania.
- Wykonanie backupów baz danych wielu środowisk naraz, np. środowiska developerskiego, środowiska QA, środowiska produkcyjnego, środowiska efemerycznego.
- Masowa konfiguracja pakietów i aplikacji, na przykład nginxa.
- Zarządzanie użytkownikami - utworzenie użytkownika z takimi samymi prawami na wielu środowiskach naraz.
- Wdrażanie aplikacji, czyli kopiowanie plików, przygotowanie środowisk.
- Konfiguracja sieci, w tym ustawienia firewalli.
- Restart usług

IV. Wymagania

Aby skorzystać z Ansible do automatyzacji infrastruktury, trzeba spełnić kilka podstawowych wymagań zarówno po stronie maszyny kontrolnej (tej, z której uruchamiamy Ansible), jak i hostów docelowych. Na szczęście Ansible jest stosunkowo lekkim i łatwym w konfiguracji narzędziem.

- System operacyjny: Ansible najlepiej działa na systemach Linux/Unix (np. Ubuntu, CentOS, Debian), ale może też zarządzać maszynami Windows (przy odpowiedniej konfiguracji).
- Python: Na maszynie kontrolnej oraz na hostach docelowych musi być zainstalowany Python (2.7 lub 3.x), aby umożliwić wykonywanie modułów Ansible.
- Dostęp SSH: Ansible łączy się z hostami docelowymi przez SSH (lub WinRM dla Windows), dlatego wymagany jest skonfigurowany dostęp SSH.
- Uprawnienia: Na maszynie kontrolnej oraz (opcjonalnie) na hostach docelowych wymagane są odpowiednie uprawnienia (np. sudo) do wykonywania operacji administracyjnych.
- Zainstalowany Ansible: Na maszynie kontrolnej należy zainstalować Ansible (np. pip install ansible lub przez menedżer pakietów).

V. Praktyczny przykład

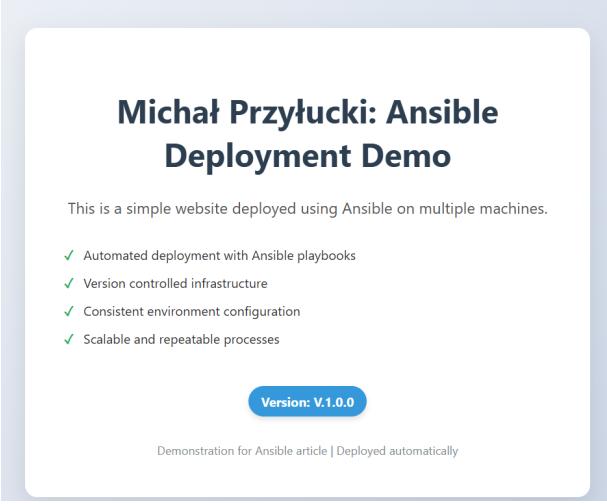
Nie sposób zliczyć wszystkich praktycznych zastosowań Ansible. Pokusiłem się na przedstawienie praktycznego scenariusza, do którego osobiście używam tej technologii.

Wdrożenie nowej wersji aplikacji/strony na usługę chmurową. W tym przypadku EC2 na AWS.

Uruchomiłem dwie instancje na AWS:

- 18.157.158.168
- 3.77.193.247

Stworzyłem również repozytorium na githubie. W repozytorium znajduje się statyczna strona www, sam plik index.html. Nie korzysta on z żadnego frameworka, backendu ani bazy danych - dzięki temu cały proces wdrożenia można maksymalnie uprościć i skupić się wyłącznie na kwestiach związanych z infrastrukturą oraz automatyzacją.



Przykładowa strona

Ansible do poprawnego działania potrzebuje dwóch rzeczy:

- zdefiniowanych hostów - czyli adresów, na których mają zostać wykonane zmiany/operacje
- pliku yaml, z tymi zmianami/operacjami

Pliki yaml ansible z założenia są deklaratywne, czyli opisują stan końcowy, nie to jak go osiągnąć. Chociaż pojawiają się pewne wyjątki, ponieważ można też znaleźć elementy imperatywne.

Plik `hosts.ini` wskazujący maszyny na których będą dokonywane zmiany:

```
[web]
18.157.158.168 ansible_user=ubuntu ansible_ssh_private_key_file=mp_webiste.pem
3.77.193.247 ansible_user=ubuntu ansible_ssh_private_key_file=mp_website2.pem
```

Plik `deploy_website.yml` wskazujący co chcemy osiągnąć (czyli playbook)

```
- name: Deploy website
  hosts: web
  become: true

  vars:
    repo_url: https://github.com/MichalPrzyl/simple_website_for_ansible
    clone_path: /tmp/webrepo
    web_root: /var/www/html

  tasks:
    - name: install nginx
      apt:
        name: nginx
        state: present
        update_cache: true

    - name: make sure dir exists
      file:
        path: "{{ web_root }}"
        state: directory

    - name: clone repo
      git:
        repo: "{{ repo_url }}"
        dest: "{{ clone_path }}"
        force: yes

    - name: copy files to /www
      copy:
        src: "{{ clone_path }}/"
        dest: "{{ web_root }}/"
        remote_src: yes

    - name: set owner and perms
      file:
        path: "{{ web_root }}"
        state: directory
        recurse: yes
        owner: www-data
        group: www-data
        mode: '0755'
```

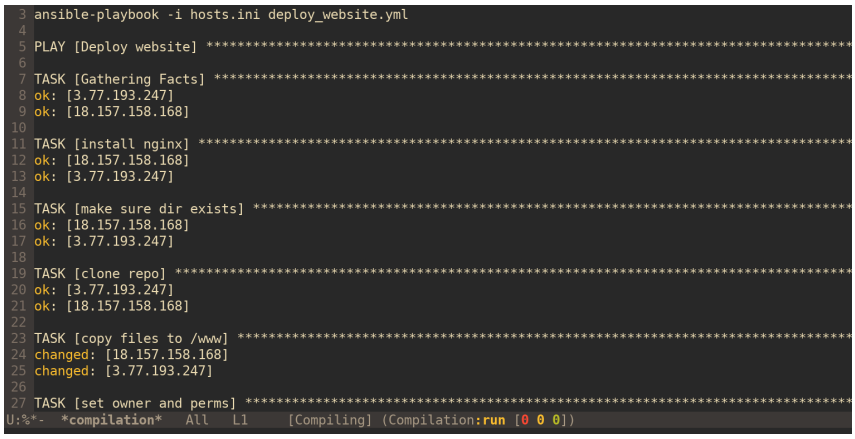
```
42
43
44      - name: restart nginx
45        service:
46          name: nginx
47          state: restarted
```

Komenda uruchamiająca playbooka:

```
1  ansible-playbook -i hosts.ini deploy_website.yml
```

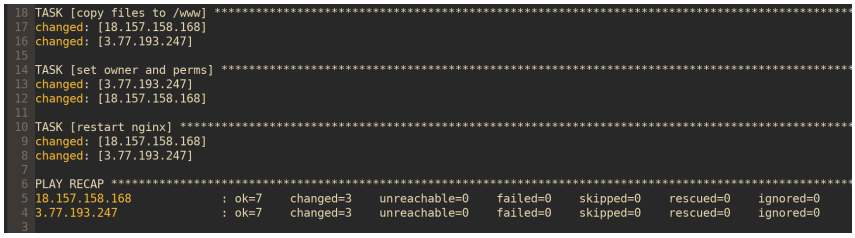
Jak widać, bezpośrednio wskazujemy plik z hostami oraz playbook, który ma zostać wykonany.

Przykładowy widok w trakcie wykonywania playbooka na dwóch hostach:



Widok w trakcie wykonywanie playbooka

Zakończenie sukcesem. Jeśli naszym oczom okaże się podobny wynik, możemy być spokojni. Wszystko co zaplanowaliśmy wykonało się poprawnie.



Widok zakończonego procesu

W celu udowodnienia prostoty takiego deployu zmienimy teraz wersję naszej aplikacji i jedną komendą wdrożymy nasze zmiany na dwa serwery naraz.

Wprowadzenie zmian w kodzie:

```
    <ul class="features">
        <li>Automated deployment with Ansible playbooks</li>
        <li>Version controlled infrastructure</li>
        <li>Consistent environment configuration</li>
        <li>Scalable and repeatable processes</li>
    </ul>

    <div class="version-badge">Version: V.2.0.0</div>

    <div class="footer">
        Demonstration for Ansible article | Deployed automatically
    </div>
</div>
```

Fragment HTML - zmiana wersji aplikacji

Commit oraz wypushowanie zmian do repozytorium:

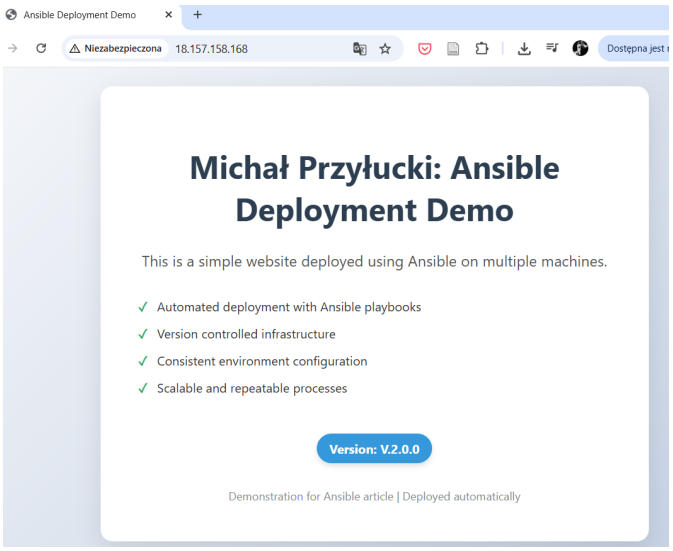
```
commit 920814105920128 52ed79e7e3c8a5f1b44469397 (HEAD -> main, origin/main)
Author: michalPrzyl <michal.przyl@gmail.com>
Date:   Fri Aug 1 22:22:34 2025 +0200

    deploy v2
```

Widok commita

I to wszystko. Teraz wystarczy ponownie uruchomić playbooka, tak samo jak wcześniej, czyli:

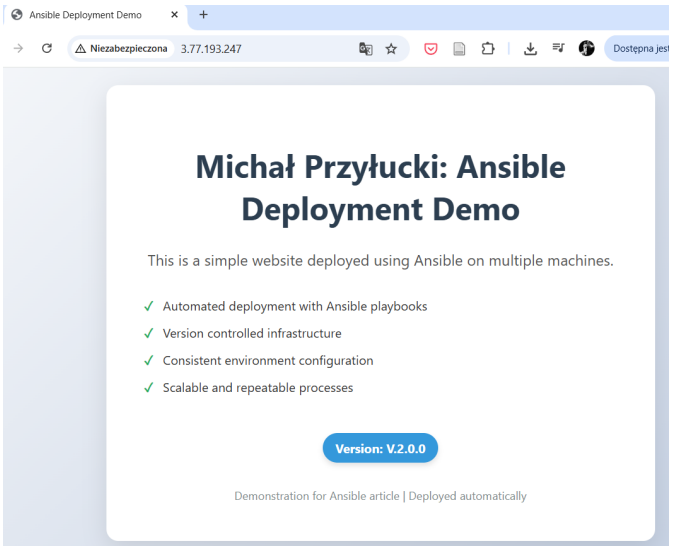
```
ansible-playbook -i hosts.ini deploy_website.yml
```

Pierwszy serwer:



Zaktualizowana aplikacja na serwerze 1.

Drugi serwer:



Zaktualizowana aplikacja na serwerze 2.

Cały deploy został uproszony do jednej komendy, niezależnie od ilości instancji/serwerów, na których wdrażamy aplikację.

Jedną z kluczowych cech Ansible, którą warto tutaj podkreślić, jest jego idempotencja Oznacza to, że wielokrotne uruchomienie tego samego playbooka nie spowoduje niepotrzebnych zmian ani błędów, o ile żądany stan już został osiągnięty. Przykładowo: jeśli ‘nginx‘ jest już zainstalowany, to zadanie po prostu zostanie pominięte. To sprawia, że administrator/devops nie musi martwić się zbędnym narzutem operacji i może uruchamiać playbooki Ansible nawet cyklicznie, np. co godzinę. Zmiany zostaną wprowadzone tylko tam gdzie są rzeczywiście potrzebne.

VI. Rozszerzenie

Mam nadzieję, że ten przerysowany w swojej prostocie przykład zdołał pozytywnie zobrazować koncepcję zawartą w tym artykule.

Oczywiście taki przykład można łatwo rozbudować.

W bardziej zaawansowanej aplikacji częste konieczne byłoby np. uruchomienie migracji bazy danych, instalacja zależności części frontendowej (`npm install`), backendowej (`pip install`) czy integracja z systemem zarządzania procesami, np. `systemd` .

Warto zaznaczyć, że taki playbook Ansible łatwo zintegrować z pipelinem CI/CD, na przykład GitHub Actions lub GitLab CI. W ten sposób można w łatwy sposób osiągnąć pełną automatyzację deploymentu automatycznie przy pushowaniu zmian na odpowiednie branche, np. `release-1-2-3` .