Bezpieczeństwo infrastruktury krytycznej i systemów sterowania przemysłowego IoT – Laboratorium 2 – Infrastructure as a Code (Config management) - Ansible

Forma: praca w samodzielna / małe zespoły (2 - 4 osoby)

Wstęp teoretyczny

Infrastruktura przed IaC

Zanim zdefiniujemy pojęcie *Infrastructure as Code (IaC)*, warto zrozumieć, dlaczego jest potrzebna innymi słowy, jakie problemy rozwiązuje.

Przyjrzyjmy się, jak wyglądało konfigurowanie i zarządzanie infrastrukturą przed pojawieniem się IaC. Oto przykładowe zadania, które należały do obowiązków różnych zespołów (w zależności od struktury firmy), takich jak DevOps/DevSecOps, Platform, Infrastructure, Network czy Systems Administration:

- Ręczne przygotowywanie serwerów (fizyczna instalacja serwerów lub ręczne wdrażanie maszyn wirtualnych),
- Konfiguracja sieci (tworzenie sieci dla infrastruktury, w tym przypisywanie adresów IP, podsieci, tablic routingu oraz wszystkich komponentów sieciowych potrzebnych infrastrukturze, takich jak routery, przełączniki lub zapory),
- Instalacja systemu operacyjnego,
- Instalacja, konfiguracja i aktualizacja oprogramowania (np. bazy danych, serwera WWW itp.),
- Odzyskiwanie po awarii (Disaster Recovery, DR) obejmujące utrzymywanie zapasowej infrastruktury w gotowości, przygotowanie szczegółowego planu awaryjnego i opracowanie procedur na wypadek awarii głównego centrum danych.

Wyjaśnienie pojęcia Infrastructure as Code

Zamiast ręcznie przygotowywać i zarządzać infrastrukturą, jak opisano powyżej, Infrastructure as Code pozwala zautomatyzować te zadania przy użyciu kodu.
Kod ten może być użyty do zdefiniowania „pożądanego stanu" infrastruktury, stanowiąc wzorzec dla narzędzi IaC, które tworzą infrastrukturę. Na przykład można w kodzie zdefiniować zasoby, takie jak komponenty sieciowe, maszyny wirtualne czy baza danych. Narzędzie IaC utworzy tę infrastrukturę zgodnie z definicją i zapewni, że zawsze pozostaje ona zgodna z opisem w kodzie. Jeśli chcesz wprowadzić zmianę w infrastrukturze np. zwiększyć liczbę maszyn wirtualnych wystarczy zmienić kod, a po zastosowaniu tej zmiany infrastruktura zostanie automatycznie zaktualizowana.vW porównaniu do tradycyjnego podejścia, w którym trzeba było ręcznie składać wniosek o utworzenie nowej maszyny wirtualnej i czekać, aż administrator ją przygotuje, korzyści z IaC stają się oczywiste.

Tworzenie infrastruktury z kodu oznacza również, że staje się ona procesem zespołowym i wspólną odpowiedzialnością. Wiedza o konfiguracji infrastruktury nie jest ograniczona do jednej osoby, co eliminuje zależności i wspiera ciągłą integrację (Continuous Integration, CI). IaC wspiera również ciągłe wdrażanie (Continuous Deployment, CD) dzięki możliwości tworzenia spójnej infrastruktury w wielu środowiskach. Oprócz zmniejszenia liczby ręcznych działań i ryzyka błędów, podejście IaC pozwala na stworzenie infrastruktury niezawodnej, skalowalnej, wersjonowanej i powtarzalnej.

Skalowalność

Technologia ta nabiera jeszcze większego sensu, gdy weźmiemy pod uwagę powszechne wykorzystanie chmury obliczeniowej i jej elastyczną naturę, umożliwiającą zwiększanie lub zmniejszanie liczby zasobów w zależności od zapotrzebowania, przy jednoczesnym rozliczaniu za sekundy, minuty lub godziny.

Dzięki IaC można skalować zasoby jednym poleceniem lub całkowicie zautomatyzować ten proces, podczas gdy wcześniej wymagało to tygodni pracy (np. zgłaszania wniosków i ręcznego przygotowywania lub usuwania zasobów).

Wersjonowanie

Infrastrukturę jako kod należy traktować jak każdy inny kod powinna być wersjonowana przy użyciu systemu kontroli wersji.

Wersjonowanie infrastruktury sprawia, że jest ona samodokumentująca się wszystkie zmiany są zapisywane i możliwe do śledzenia. Jest to korzystne dla wielu zespołów pracujących z infrastrukturą, np. podczas audytów lub rozwiązywania problemów.

Wersjonowanie umożliwia także powrót do ostatniej działającej wersji infrastruktury w przypadku wystąpienia problemów to przykład tego, jak IaC zapewnia niezawodność.

Ponadto wersjonowanie infrastruktury pozwala testować aplikacje w oparciu o wcześniejsze wersje środowisk.

Powtarzalność

W środowisku deweloperskim często pracuje się z wieloma środowiskami. Programiści tworzą kod w środowisku deweloperskim, następnie testują go w środowisku akceptacyjnym/testowym (odwzorowującym produkcję), a na końcu wdrażają w środowisku produkcyjnym.

Dawniej każde z tych środowisk wymagało ręcznego przygotowania i konfiguracji infrastruktury. Dzięki IaC można użyć tego samego kodu i konfiguracji, aby stworzyć identyczne środowiska jednym poleceniem.

Oszczędza to ogromną ilość czasu i zapewnia spójność między środowiskami, co ma kluczowe znaczenie podczas testowania oprogramowania a było bardzo trudne do osiągnięcia przy ręcznym tworzeniu infrastruktury.

IaC w praktyce

IaC to potężna technologia, która przy właściwym użyciu może usprawnić i zoptymalizować procesy kluczowe dla biznesu.

Aby pokazać jej praktyczne zastosowanie, przyjrzyjmy się jednemu z przykładów wspomnianych wcześniej: **Disaster Recovery (odzyskiwanie po awarii)**.

Odzyskiwanie po awarii polega na odtworzeniu infrastruktury po katastrofalnym lub zakłócającym działaniu. Przykładowo, jeśli firma posiada fizyczną infrastrukturę lokalną w centrum danych, awaria może wynikać zarówno z krótkiej przerwy w zasilaniu, jak i z katastrof naturalnych, takich jak pożary czy huragany, które powodują niedostępność głównego centrum danych.

Ręczne przełączenie systemu z głównego centrum danych na zapasowe jest czasochłonne, stresujące i obarczone ryzykiem błędów. IaC może pomóc w tym procesie na wiele sposobów:

- **Automatyzacja infrastruktury:** IaC umożliwia automatyczne tworzenie i konfigurowanie infrastruktury, co znacznie przyspiesza jej odtworzenie w zapasowym centrum danych.
- **Automatyczne kopie zapasowe i odzyskiwanie danych:** IaC może definiować i uruchamiać procedury tworzenia i odtwarzania kopii zapasowych, aby zminimalizować ryzyko utraty danych.
- **Automatyzacja failoveru:** IaC może automatycznie przełączać usługi na zapasową infrastrukturę w przypadku awarii, skracając czas przestoju.

- **Spójność konfiguracji:** IaC zapewnia jednolitą konfigurację i dokładną dokumentację infrastruktury, co pomaga uniknąć błędów konfiguracyjnych podczas przełączania.

- **Skalowanie zasobów:** IaC umożliwia dynamiczne zwiększanie zasobów podczas procesu odzyskiwania i ich późniejsze zmniejszenie po zakończeniu awarii.

Skoro wiemy już, czym jest *Infrastructure as Code (IaC)*, możemy przyjrzeć się narzędziom wykorzystywanym w tym podejściu. Pod pojęciem IaC mieści się wiele technologii, takich jak **Terraform**, **AWS CloudFormation**, **Google Cloud Deployment Manager**, **Ansible**, **Puppet**, **Chef**, **SaltStack** czy **Pulumi**.

Warto zrozumieć, że każde z tych narzędzi ma inne zastosowania i własne zalety. Ze względu na różnorodność przypadków użycia, aby w pełni zautomatyzować proces tworzenia, konfiguracji i wdrażania infrastruktury, często konieczne jest połączenie kilku narzędzi IaC zamiast korzystania z jednego rozwiązania.

Aby lepiej zrozumieć te narzędzia, przyjrzymy się ich najważniejszym cechom, które pozwalają odróżnić je od siebie i dobrać odpowiednie narzędzie do konkretnego zastosowania. Warto pamiętać, że zachowanie niektórych narzędzi może różnić się w zależności od sposobu ich użycia, co często prowadzi do dyskusji na temat ich charakterystyki. Z tego powodu poniżej omówiono kluczowe pojęcia i sposób, w jaki poszczególne narzędzia są zazwyczaj wykorzystywane.

Deklaratywne vs. Imperatywne

Na początek ustalmy różnicę między narzędziami IaC **deklaratywnymi** i **imperatywnymi** (nazywanymi też *funkcyjnymi* i *proceduralnymi*).

Deklaratywne

W tym podejściu definiuje się pożądany stan infrastruktury np. minimalne i maksymalne zasoby, komponenty systemu itp. Narzędzie IaC wykonuje działania na podstawie tego, co zostało określone. Stan infrastruktury jest zapisywany w tzw. *state file*, aby upewnić się, że aktualny stan odpowiada stanowi docelowemu. Innymi słowy, podejście deklaratywne skupia się na **„co”**, a nie **„jak”**. Narzędzia deklaratywne są zazwyczaj idempotentne, co oznacza, że wielokrotne ich uruchomienie przynosi ten sam rezultat. Dzieje się tak dlatego, że przed wykonaniem zmian porównują aktualny stan infrastruktury z zapisanym stanem. Jeśli infrastruktura jest już utworzona, nie zostaną wprowadzone żadne zmiany.

Przykłady narzędzi deklaratywnych:
Terraform, AWS CloudFormation, Pulumi, Puppet (Ansible również wspiera podejście deklaratywne).

Imperatywne

Podejście imperatywne polega na definiowaniu konkretnych poleceń, które mają zostać wykonane w celu osiągnięcia pożądanego stanu. Komendy muszą być wykonywane w określonej kolejności. Zazwyczaj narzędzia imperatywne nie są idempotentne ponowne ich uruchomienie może doprowadzić do innego rezultatu lub błędów w konfiguracji. Wynika to z faktu, że polecenia są wykonywane niezależnie od aktualnego stanu infrastruktury. Uruchomienie takich komend na już skonfigurowanym środowisku może spowodować utworzenie dodatkowych, niepożądanych zasobów lub uszkodzenie istniejącej konfiguracji (w zależności od środowiska i zestawu poleceń).

Przykłady narzędzi imperatywnych:
Najlepszym przykładem jest **Chef**, choć również **SaltStack** i **Ansible** obsługują programowanie w stylu imperatywnym.

Można to zobrazować w ten sposób:

- W podejściu **imperatywnym** otrzymujesz listę kroków prowadzących do punktu X na mapie – do celu dojdziesz, tylko jeśli zaczynasz w określonym miejscu (np. na świeżym, niekonfigurowanym serwerze). Jeśli jednak punkt początkowy jest inny (np. częściowo skonfigurowany serwer), trafisz w inne miejsce.
- W podejściu **deklaratywnym** narzędzie analizuje, gdzie aktualnie się znajdujesz, i samo wyznacza, jakie kroki należy podjąć, aby dotrzeć do punktu X.

Wybór podejścia zależy od potrzeb infrastruktury. Deklaratywne IaC jest prostsze w utrzymaniu i lepiej sprawdza się w długofalowych projektach, natomiast imperatywne daje większą elastyczność i kontrolę, pozwalając dokładnie określić, w jaki sposób ma być zarządzana infrastruktura.

Z agentem vs. Bez agenta

Podział **agent-based** i **agentless** wykracza poza IaC występuje również w narzędziach monitorujących i zabezpieczających. W kontekście IaC oznacza to sposób komunikacji z zarządzanymi zasobami.

Z agentem (Agent-based)

W tym modelu na serwerze, którym zarządza narzędzie IaC, musi być zainstalowany **agent**. Działa on w tle i pełni rolę kanału komunikacyjnego między narzędziem IaC a zarządzanymi zasobami. Agent odpowiada za wykonywanie zadań i raportowanie stanu infrastruktury.

Agent musi być zainstalowany na każdym systemie lub węźle, który ma być zarządzany. Takie narzędzia mogą działać nawet przy ograniczonej łączności lub w trybie offline, co czyni je odpornymi i niezawodnymi w automatyzacji. Wymagają jednak utrzymania należy monitorować stan agenta i restartować go w razie awarii.

Niektóre narzędzia agentowe wymagają również otwarcia portów sieciowych dla ruchu przychodzącego i wychodzącego, aby agent mógł wymieniać informacje konfiguracyjne. Z perspektywy bezpieczeństwa zwiększa to ryzyko, jednak zapewnia bardzo szczegółową kontrolę nad zarządzanymi zasobami i umożliwia gromadzenie dokładniejszych danych.

Przykłady narzędzi agentowych: Puppet, Chef, SaltStack.

Bez agenta (Agentless)

Narzędzia bezagentowe nie wymagają instalacji żadnego oprogramowania na systemach docelowych. Zamiast tego wykorzystują istniejące protokoły komunikacyjne, takie jak **SSH**, **WinRM** lub **API chmurowe**, aby zarządzać zasobami.

Narzędzia tego typu są łatwiejsze w użyciu ich prostota przy konfiguracji oznacza szybsze wdrożenie, a brak agenta pozwala stosować je w różnych środowiskach bez konieczności dostosowywania instalacji. To czyni je elastycznym wyborem.

W porównaniu z narzędziami agentowymi wymagają mniej utrzymania i nie wiążą się z ryzykiem związanym z zabezpieczeniem agenta. Mają jednak mniejszy poziom kontroli nad systemami docelowymi.

Narzędzia bezagentowe dobrze sprawdzają się w środowiskach, które muszą dynamicznie reagować na zmiany obciążenia, tworząc lub usuwając zasoby w zależności od potrzeb. Nowo utworzone zasoby można wtedy zarządzać bez potrzeby instalowania agentów.

Przykłady narzędzi bezagentowych: Terraform, AWS CloudFormation, Pulumi, Ansible.

Niezmienna vs. Zmienna infrastruktura

Niezmienna (immutable) i zmienna (mutable) infrastruktura odnoszą się do tego, czy w infrastrukturze można wprowadzać zmiany. Aby zdefiniować oba podejścia, wyobraźmy sobie prostą infrastrukturę maszynę wirtualną z serwerem WWW i aplikacją. Jeśli chcesz zmienić tę infrastrukturę, na przykład wdrożyć nowy kod aplikacji, aktualizacja ta spowoduje, że aplikacja przejdzie z wersji 1 do wersji 2. Jak to wygląda w każdym z przypadków?

Zmienna infrastruktura (Mutable)

Zmienna infrastruktura oznacza, że można wprowadzać zmiany bezpośrednio w istniejącym środowisku. W tym przykładzie możemy zaktualizować aplikację na serwerze, na którym obecnie działa. Zaleta polega na tym, że nie trzeba tworzyć nowych zasobów wystarczy zmodyfikować te już istniejące.

Problem pojawia się, gdy aktualizacja składa się z kilku zmian, a jedna z nich się nie powiedzie. Jest to częste w procesie deweloperskim i może wynikać z różnych przyczyn, np. błędnej konfiguracji aplikacji lub problemu z siecią. W takiej sytuacji serwer będzie mieć wersję aplikacji, która zawiera tylko część zmian nie jest to już wersja 1, ale też nie w pełni wersja 2.

W środowisku produkcyjnym aplikacja jest testowana w środowiskach deweloperskim i testowym, ale taka „pośrednia" wersja nigdy nie była testowana, co może prowadzić do problemów. Jeśli aktualizacja jest wdrażana na wielu serwerach, z których niektóre zakończą się błędami, a inne nie, łatwo o niespójność różne wersje aplikacji na różnych serwerach.

Niezmienna infrastruktura (Immutable)

Niezmienna infrastruktura oznacza, że nie można wprowadzać w niej zmian. Po jej utworzeniu pozostaje niezmienna aż do momentu usunięcia. W naszym przykładzie, gdy wdrażana jest wersja 2 aplikacji, tworzona jest nowa infrastruktura z tą wersją. W efekcie istnieją dwa środowiska jedno z wersją 1, drugie z wersją 2.

Jeśli wszystkie zmiany zostaną wdrożone pomyślnie (i tylko wtedy), infrastruktura z wersją 1 zostaje usunięta. Jeśli wdrożenie się nie powiedzie, nowa infrastruktura jest niszczona, a proces powtarzany. Takie podejście wymaga więcej zasobów, ponieważ równocześnie działają dwa środowiska lub trzeba powtarzać proces, ale zapewnia spójność między serwerami wiadomo dokładnie, która wersja działa na którym serwerze.

Przykłady narzędzi IaC typu immutable: Terraform, AWS CloudFormation, Google Cloud Deployment Manager, Pulumi (niektóre z tych narzędzi można skonfigurować również do pracy w trybie mutable, ale najlepiej działają w podejściu immutable).

W podanym przykładzie sensowne jest użycie infrastruktury niezmiennej. Jednak oba podejścia mają swoje zastosowania. Na przykład, jeśli serwer jest krytyczną bazą danych wymagającą regularnych aktualizacji i poprawek, użycie infrastruktury niezmiennej oznaczałoby konieczność odbudowy całej bazy przy każdej zmianie, co może być ryzykowne dla danych. W takim przypadku bardziej odpowiednie będzie podejście mutable.

Provisioning vs. Configuration Management

Wszystkie omówione wcześniej cechy dotyczą zachowania sposobu działania narzędzia. Aby określić, czym dane narzędzie jest i kiedy się go używa, należy ustalić, czy służy do provisioning (tworzenia infrastruktury), czy do zarządzania konfiguracją.

Niektóre źródła rozróżniają te pojęcia jako infrastructure as code i configuration as code. Aby jednak określić, do której kategorii należy narzędzie, wystarczy zadać pytanie: „Co ono robi?".

Można wyróżnić cztery główne zadania:

1. Provisioning infrastruktury (tworzenie środowiska),
2. Zarządzanie infrastrukturą (wprowadzanie zmian),
3. Instalacja oprogramowania (pierwsze wdrożenie i konfiguracja aplikacji),
4. Zarządzanie oprogramowaniem (aktualizacje i zmiany konfiguracji).

Nie istnieje pojedyncze narzędzie, które wykonuje wszystkie te zadania. Dlatego powszechną praktyką jest łączenie dwóch narzędzi jednego do provisioning i drugiego do configuration management.

Przykłady narzędzi:

- **Provisioning:** Terraform, AWS CloudFormation, Google Cloud Deployment Manager, Pulumi
- **Configuration management:** Ansible, Chef, Puppet, SaltStack

Przykład z branży: Firma musi utworzyć nową infrastrukturę dla aplikacji internetowej. Poprzednia instalacja została zaatakowana, więc tym razem planowane jest zainstalowanie agenta monitorującego. Infrastruktura zostanie zdefiniowana i utworzona przy użyciu Terraform, a następnie Ansible zostanie wykorzystany do instalacji agenta monitorującego w tej infrastrukturze.

Tool	Declarative/ Imperative	Agent-Based	Type	Cloud-Agnostic
Terraform	Declarative	No	Infrastructure Provisioning	Yes
Ansible	Supports Both	No	Configuration Management	Yes
Pulumi	Declarative	No	Infrastructure Provisioning	Yes
AWS CloudFormation	Declarative	No	Infrastructure Provisioning	No (AWS only)
Google Cloud Deployment Manager	Declarative	No	Infrastructure Provisioning	No (Google Cloud only)
Chef	Imperative	Yes	Configuration Management	Yes
Puppet	Declarative	Yes	Configuration Management	Yes
SaltStack	Supports Both	Yes	Configuration Management	Yes

Cykl życia IaC (Infrastructure as Code Lifecycle – IaCLC)

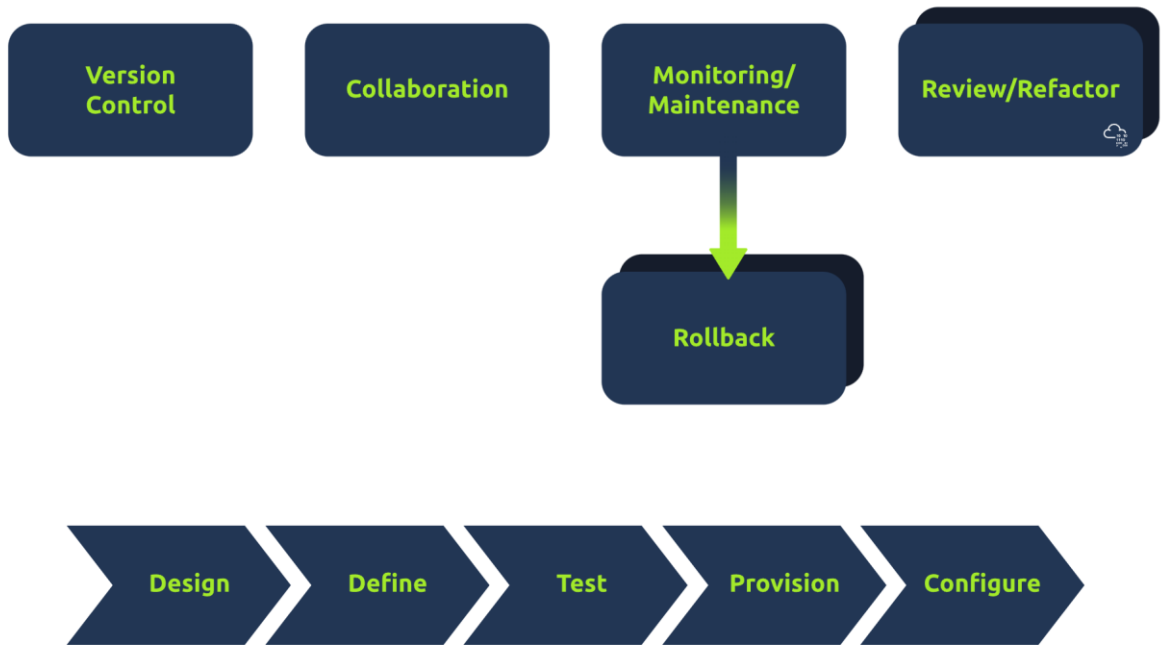
Skoro wiemy już, czym jest *Infrastructure as Code (IaC)* i jakich narzędzi używamy do jej wdrażania, czas zdefiniować **cykl życia IaC**.
W artykułach, publikacjach naukowych i wpisach na blogach można spotkać różne wersje tzw. „IaC lifecycle", jednak **nie istnieje jeden powszechnie uznany i ustandaryzowany model** tak jak w przypadku DevOps czy DevSecOps. Z tego względu opracowano model edukacyjny, który można nazwać **Infrastructure as Code Lifecycle (IaCLC)**.
Opisuje on działania i zadania wykonywane przez deweloperów oraz narzędzia IaC, dzieląc je na fazy, które razem tworzą cykl życia. Celem tego cyklu jest pokazanie, że *Infrastructure as Code* to proces **ciągły i powtarzalny**, a nie tylko jednokrotna sekwencja tworzenia i zarządzania infrastrukturą.

Cel i znaczenie IaCLC

IaCLC ma służyć jako przewodnik dla inżynierów DevOps (oraz innych osób zajmujących się infrastrukturą), pomagając w planowaniu, tworzeniu i utrzymaniu środowisk.
W ramach cyklu uwzględniono także kluczowe praktyki DevOps, takie jak **Continuous Integration (CI)** i **Continuous Deployment (CD)**, które stanowią integralną część najlepszych praktyk na każdym etapie. Ponieważ IaC jest procesem deweloperskim, można zauważyć pewne **podobieństwa do cyklu życia oprogramowania (SDLC)** oba obejmują etapy planowania/projektowania, wdrażania, testowania, uruchomienia i monitorowania.
Mimo tych podobieństw, cykle te nie są ze sobą tożsame.
SDLC dotyczy aplikacji, a IaCLC – infrastruktury, która musi często dostosowywać się do zmieniających się wymagań i środowisk.

Dlatego w IaCLC wyróżnia się **dwa typy faz**:

1. **Fazy ciągłe (Continual / Best Practice Phases)** – powtarzane w sposób ciągły, zapewniające zgodność z najlepszymi praktykami,
2. **Fazy powtarzalne (Repeatable / Infra Creation + Config Phases)** – wykonywane przy tworzeniu lub modyfikowaniu infrastruktury.



Podział cyklu IaCLC

1. Fazy ciągłe (Continual / Best Practice Phases)

Są to działania realizowane regularnie, aby utrzymać dobrą kondycję infrastruktury, spójność procesów oraz bezpieczeństwo. Czasem fazy te uruchamiają ponownie cykl faz powtarzalnych (np. gdy wykryty zostanie błąd lub potrzeba zmiany).

Fazy ciągłe obejmują:

- **Version Control (Kontrola wersji)**
  Obejmuje wersjonowanie kodu, w którym definiowana jest infrastruktura oraz wprowadzane są zmiany. Dzięki temu, w przypadku błędów, można powrócić do poprzedniej stabilnej wersji w fazie *rollback*.

- **Collaboration (Współpraca)**
  W pracy zespołowej nad infrastrukturą kluczowa jest komunikacja i koordynacja. Brak współpracy może prowadzić do niezgodności między modułami infrastruktury i ogólnego chaosu w zespole.

- **Monitoring / Maintenance (Monitorowanie i utrzymanie)**
  Po wdrożeniu infrastruktury należy ją stale monitorować – pod kątem wydajności, bezpieczeństwa, awarii czy ostrzeżeń. Automatyczne zadania konserwacyjne (np. czyszczenie dysków) pomagają zapobiegać problemom. W przypadku awarii ta faza może uruchomić proces *rollback*.

- **Rollback (Wycofanie zmian)**
  Jeśli po wdrożeniu wystąpi awaria lub błąd, infrastruktura musi zostać przywrócona do ostatniej znanej, działającej wersji.

- **Review + Change (Przegląd i zmiany)**
  Działająca infrastruktura nie zawsze jest optymalna. Nowe wymagania biznesowe lub odkryte luki bezpieczeństwa mogą wymagać modyfikacji. Dlatego infrastrukturę należy regularnie przeglądać i aktualizować w razie potrzeby.

2. Fazy powtarzalne (Repeatable / Infra Creation + Config Phases)

Fazy te są realizowane w trakcie **tworzenia lub modyfikowania infrastruktury**.
Mogą być wykonywane wiele razy – przy różnych zmianach lub dla różnych środowisk (np. dev, staging, production).

Fazy powtarzalne obejmują:

- **Design (Projektowanie)**
  Projektowanie infrastruktury na podstawie potrzeb organizacji.
  Na tym etapie należy uwzględnić bezpieczeństwo w całym procesie (np. niewłaściwie zaprojektowane reguły skalowania mogą prowadzić do problemów z dostępnością).

- **Define (Definiowanie)**
  Na podstawie projektu definiuje się infrastrukturę w kodzie, określając zasoby, sieci, maszyny wirtualne i komponenty.

- **Test (Testowanie)**
  Kod infrastruktury należy sprawdzić pod kątem błędów składniowych i logicznych, np. za pomocą *lintera*.
  Przed wdrożeniem do produkcji testy powinny być wykonane w środowisku stagingowym, które odwzorowuje produkcję.

- **Provision (Tworzenie infrastruktury)**
  Wdrażanie zdefiniowanej infrastruktury w środowisku produkcyjnym przy użyciu narzędzi provisioningowych, takich jak **Terraform**.

- **Configure (Konfiguracja)**
  Konfiguracja wdrożonej infrastruktury przy użyciu narzędzi do zarządzania konfiguracją, takich jak **Ansible**.

IaCLC w praktyce

W praktyce cykl życia IaC jest **ciągłym i powtarzalnym procesem**, w którym fazy wzajemnie się przeplatają.
Nowe zmiany mogą wynikać z monitoringu lub przeglądu, co prowadzi do ponownego przejścia przez fazy projektowania, definiowania, testowania i wdrożenia.

IaCLC stanowi więc ramy, które pomagają zespołom DevOps zachować **spójność, bezpieczeństwo i kontrolę nad infrastrukturą**, przy jednoczesnym zachowaniu elastyczności i możliwości automatyzacji całego cyklu zarządzania środowiskiem.

Wirtualizacja a Infrastructure as Code

W kontekście *Infrastructure as Code (IaC)*, **wirtualizacja** jest jednym z kluczowych pojęć. Technologie wirtualizacji, stosowane razem z IaC, umożliwiają wyjątkowo wydajne i elastyczne zarządzanie infrastrukturą.
Choć część tych zagadnień została już omówiona we wprowadzeniu do konteneryzacji, poniższy materiał przedstawia przegląd technologii wirtualizacji oraz ich powiązanie z IaC.

Poziomy wirtualizacji

Mówiąc o poziomach wirtualizacji, odnosimy się do tego, **co dokładnie jest wirtualizowane**.
Aby lepiej zrozumieć różnicę, przyjrzyjmy się dwóm głównym typom: **wirtualizacji na poziomie hypervisora** i **wirtualizacji na poziomie kontenera (container)**.

Wirtualizacja na poziomie hipernadzorcy

Przykład: **VMware**
Umożliwia uruchamianie wielu maszyn wirtualnych (VM) na jednym fizycznym serwerze.
Hipernadzorca (hypervisor) dzieli zasoby fizycznego serwera - CPU, pamięć, przestrzeń dyskową - i przydziela je osobnym maszynom wirtualnym.
Każda maszyna wirtualna działa w pełni niezależnie, posiadając własny system operacyjny (może się on różnić od systemu gospodarza), co pozwala uruchamiać różne OS-y na jednym serwerze.
Tworzenie takiej maszyny wirtualnej jest jednak procesem stosunkowo czasochłonnym.

Wirtualizacja na poziomie kontenera

Przykład: **Docker**
W tym przypadku wirtualizacja odbywa się na poziomie systemu operacyjnego.
Wszystkie kontenery współdzielą ten sam **kernel systemu operacyjnego**, ale działają jako odizolowane środowiska.
Dzięki temu kontenery są **lekkie, szybkie w uruchamianiu i skalowaniu**, co umożliwia błyskawiczne wdrożenia aplikacji.

Zastosowanie wirtualizacji w IaC

Wirtualizacja odgrywa w *Infrastructure as Code* kluczową rolę w wielu obszarach.
Można wyróżnić sześć głównych zastosowań:

1. Skalowalność (Scalability)

IaC wykorzystuje wirtualizację do automatycznego dostosowywania zasobów do aktualnego zapotrzebowania.
Jeśli ruch w systemie gwałtownie wzrośnie, zasady skalowania (zdefiniowane w kodzie) mogą automatycznie utworzyć dodatkowe maszyny wirtualne, które przejmą część obciążenia.

2. Izolacja zasobów (Resource Isolation)

Wirtualizacja pozwala na ścisłą izolację zasobów między różnymi maszynami wirtualnymi lub kontenerami.
Dzięki IaC można w kodzie precyzyjnie określić, które komponenty mają dostęp do jakich zasobów.
Jeśli jedna aplikacja zużywa całą dostępną moc CPU lub pamięć, nie wpłynie to na pozostałe elementy systemu.

3. Testowanie / Snapshoty / Rollbacki

Wirtualizacja umożliwia łatwe pakowanie systemów operacyjnych, konfiguracji i zależności w formie obrazów (VM lub kontenerów). Dzięki IaC można wdrożyć identyczne środowisko testowe lub deweloperskie. Dodatkowo możliwość tworzenia *snapshotów* (migawki stanu) pozwala na szybkie przywrócenie systemu do poprzedniego punktu w przypadku błędnego wdrożenia.

4. Szablony (Templates)

Wirtualizacja umożliwia tworzenie **szablonów maszyn wirtualnych lub kontenerów**, które służą jako gotowe wzorce wdrożenia. IaC może wykorzystywać te szablony do automatycznego tworzenia nowych instancji, co skraca czas provisioning i zapewnia spójność konfiguracji.

5. Multi-tenancy

Wirtualizacja wspiera środowiska wielodzierżawne - wielu klientów (tenantów) może korzystać z tego samego fizycznego serwera, mając jednocześnie odseparowane zasoby. Dzięki IaC możliwe jest zarządzanie zasobami dla wielu tenantów w ramach jednej wspólnej infrastruktury.

6. Przenośność (Portability)

Wirtualizacja ułatwia przenoszenie infrastruktury między dostawcami chmurowymi lub centrami danych.
Ta przenośność, wspierana przez IaC, przyspiesza procesy takie jak *Disaster Recovery* i umożliwia organizacjom w środowisku *hybrid cloud* odtwarzanie identycznych środowisk niezależnie od dostawcy usług chmurowych (CSP).

Technologie wirtualizacji a IaC

Jednym z najczęściej używanych rozwiązań łączących IaC i wirtualizację jest **Kubernetes (K8s)**.
Kubernetes to oprogramowanie do orkiestracji kontenerów - automatyzuje ich wdrażanie, skalowanie i zarządzanie.

Aby korzystać z Kubernetes, potrzebny jest **klaster K8s**, czyli infrastruktura składająca się z **noda głównego (master)** i jednego lub wielu **nodów roboczych (worker nodes)**, na których działają kontenery z aplikacjami.

Narzędzia IaC mogą:

- **tworzyć i konfigurować infrastrukturę** potrzebną dla klastra Kubernetes,
- **automatyzować instalację komponentów Kubernetes**,
- **definiować zasady skalowania** (np. automatyczne dodawanie lub usuwanie nodów/podsów w zależności od obciążenia).

Współdziałanie technologii – przykład praktyczny

Załóżmy, że inżynier DevOps ma za zadanie wdrożyć nową aplikację webową od podstaw, korzystając z technologii **Terraform** i **Kubernetes**.

Może on:

1. Użyć **Terraform**, aby automatycznie utworzyć nowy klaster Kubernetes oraz inne potrzebne zasoby, takie jak serwer Nginx czy load balancer.
2. Następnie z pomocą **Kubernetes** wdrożyć aplikację i zarządzać jej komponentami.

Choć zarządzanie klastrem byłoby możliwe za pomocą samego narzędzia **kubectl**, połączenie go z IaC, jak Terraform, zapewnia znacznie więcej korzyści:

- Terraform umożliwia **wersjonowanie i powtarzalność wdrożeń** – stan klastra zawsze odpowiada stanowi zdefiniowanemu w kodzie.
- Terraform rozpoznaje **zależności między zasobami** – jeśli utworzenie jednego komponentu się nie powiedzie, nie próbuje wdrożyć zależnych elementów, co oszczędza czas i ogranicza błędy.

W dużych organizacjach, gdzie infrastruktura musi być powielana między projektami i klientami, Terraform pozwala tworzyć moduły (np. dla deploymentów, usług, config map) wielokrotnego użytku, co znacznie przyspiesza pracę.

Inne przykłady integracji wirtualizacji i IaC

- **Vagrant** – narzędzie używane przez deweloperów do uruchamiania i zarządzania maszynami wirtualnymi lub kontenerami na potrzeby lokalnego testowania. Dzięki IaC proces tworzenia takich środowisk może być w pełni zautomatyzowany i powtarzalny.
- **OpenStack** – otwartoźródłowa platforma chmurowa pozwalająca tworzyć własne środowiska chmurowe. Narzędzia IaC mogą automatyzować provisioning i zarządzanie zasobami w ramach OpenStacka.

Infrastructure as code przypadki użycia

On-premises IaC oraz **cloud-based IaC**. Najpierw ustalmy kluczowe różnice między nimi. Poniżej przedstawiono główne różnice między on-prem IaC a cloud-based IaC w pięciu kategoriach: **Location, Tech, Resources, Scalability, Cost**.

Porównanie

Location

On-prem: Jak sama nazwa wskazuje, jeśli Twoja infrastruktura jest on-prem, kod będzie konfigurował serwery, urządzenia sieciowe, pamięć masową i oprogramowanie znajdujące się w Twojej lokalizacji. Może to oznaczać fizyczną infrastrukturę w siedzibie organizacji lub wynajętą w centrum danych.

Cloud-based: Oznacza to przygotowywanie (provisioning) i konfigurowanie zasobów infrastruktury w środowisku chmurowym przy użyciu zasobów chmurowych. Odbywa się to za pośrednictwem dostawcy usług chmurowych (CSP), takiego jak AWS, Microsoft Azure, GCP itp.

Tech

On-prem: Zazwyczaj, pracując z infrastrukturą on-premises, używa się narzędzi IaC takich jak Ansible, Chef i Puppet (choć inne również można do tego skonfigurować) do zarządzania i przygotowywania infrastruktury na serwerach fizycznych lub maszynach wirtualnych w on-prem data center.

Cloud-based: Istnieje wiele narzędzi do przygotowywania i zarządzania infrastrukturą w chmurze, zaprojektowanych tak, aby wykorzystywać elastyczną naturę przetwarzania w chmurze; niektórzy dostawcy chmury mają własne narzędzie/usługę IaC. Przykłady narzędzi stosowanych w cloud-based IaC: Terraform, AWS CloudFormation, Azure Resource Manager (ARM) templates oraz Google Cloud Deployment Manager.

Resources

On-prem: W przypadku on-prem IaC najpewniej będziesz pracować ze sprzętem fizycznym, co wiąże się z kwestiami zgodności sprzętowej, utrzymania fizycznego i ograniczonych zasobów fizycznych. Oznacza to konieczność konfiguracji tych zasobów (serwery fizyczne, urządzenia pamięci masowej, sprzęt sieciowy itd.). On-prem IaC może być używane przez firmy polegające na infrastrukturze on-premises z powodu systemów legacy, wymagających wsparcia w efektywnym zarządzaniu tą infrastrukturą.

Cloud-based: W chmurze pracujesz z zasobami wirtualnymi dostarczanymi przez CSP (AWS, GCP, Azure itp.). Te zasoby mogą być przygotowywane/zarządzane przez narzędzia cloud-based IaC. Infrastruktura bazowa jest utrzymywana przez CSP, a nie przez użytkownika. Cloud-based IaC idealnie sprawdzi się w firmach oferujących streaming wideo: ze względu na charakter tej usługi zapotrzebowanie na zasoby obliczeniowe może się wahać i osiągać szczyty. Cloud-based IaC ułatwia uzyskanie dodatkowych maszyn wirtualnych lub przestrzeni składowania w zależności od zużycia zasobów, aby zapewnić stałą wydajność w czasie pików.

Scalability

On-prem: Skalowanie zasobów przy użyciu on-prem IaC może być procesem powolnym i uciążliwym. Może obejmować zmiany manualne/fizyczne i zakupy sprzętu. Ponieważ skalowanie jest tak wolne, wymaga planowania, a ograniczenia sprzętowe muszą wystarczyć do obsługi obciążeń szczytowych. To bywa trudne, np. gdy infrastruktura on-premises zaczyna otrzymywać zwiększony ruch (pomyśl o okresie rekrutacji lub sesji egzaminacyjnej na uczelni).

Cloud-based: Ze względu na elastyczną naturę chmury, skalowanie zasobów jest znacznie bardziej elastyczne niż on-premises. Narzędzia cloud-based IaC mogą definiować polityki skalowania, tak aby zasoby były zwiększane lub zmniejszane w zależności od popytu i zużycia, wykorzystując funkcje auto-scaling. Zapewnia to opłacanie wyłącznie potrzebnych zasobów. Cloud-based IaC usprawnia skalowanie infrastruktury w okresach szczytowych. Wyobraź sobie infrastrukturę wspierającą grę online ruch rośnie przy wydaniu nowej aktualizacji lub podczas wydarzenia w grze.

Cost

On-prem: Wydatki na infrastrukturę on-premises oznaczają konieczność płacenia za sprzęt fizyczny (lub leasing serwera w zewnętrznym data center w zależności od przypadku użycia); to także koszty utrzymania fizycznego (naprawy/ wymiany części itd.), koszty operacyjne i wydatki związane z rozbudową infrastruktury. Sama infrastruktura on-prem i użycie on-prem IaC nie przynoszą z definicji korzyści kosztowych. Wybór on-prem wynika zwykle z kombinacji innych czynników niż koszt.

Cloud-based: W przypadku cloud-based IaC masz do czynienia z infrastrukturą rozliczaną przez dostawców w modelu pay-as-you-go. Oznacza to płacenie wyłącznie za to, co faktycznie zużywasz (co może być bardziej opłacalne przy efektywnym użyciu auto-scalingu). Cloud-based IaC to idealne i opłacalne rozwiązanie w scenariuszach, gdzie infrastruktura musi być automatycznie skalowana w górę i w dół na żądanie. Na przykład sprzedawca online oczekujący wzrostu ruchu w Black Friday lub w okresie świątecznym może chwilowo zwiększyć zasoby, ale nie płacić za nie, gdy ruch wróci do normy.

Zalety

Przyjrzyjmy się, jak powyższe różnice przekładają się na korzyści w zależności od konkretnych przypadków użycia.

On-prem IaC: Główną zaletą infrastruktury jako kodu w modelu on-premises jest **pełna kontrola**. Bywa ona wymagana przez regulacje/ bezpieczeństwo i wymagania zgodności. To częste w dużych instytucjach bankowych lub firmach przetwarzających dane wrażliwe klientów czy dane rządowe. Regulacje mogą wymagać suwerenności danych (przetwarzanie i przechowywanie danych w granicach kraju). Choć dostawcy chmury oferują specjalne środowiska „government cloud" dla części takich przypadków (np. AWS US Gov Cloud), są one dostępne tylko w wybranych regionach; dla regionów takich jak Afryka brak dedykowanej chmury rządowej oznacza konieczność użycia on-prem IaC, aby spełnić rygorystyczne wymagania bezpieczeństwa i zgodności.

Cloud-based IaC: Ma wiele zalet dla firm rozwijających się dynamicznie. **Skalowalność** pozwala szybko zwiększać lub zmniejszać zasoby, aby sprostać wahaniom popytu bez ograniczeń fizycznej infrastruktury on-premises. Infrastruktura chmurowa może być wdrażana globalnie w wielu regionach, zmniejszając **latencję** kluczowe dla firm świadczących usługi na całym świecie przy minimalnych opóźnieniach, jak gry online. Dzięki rozliczeniu **pay-as-you-go** firmy płacą tylko za faktycznie potrzebne/zużyte zasoby, bez inwestycji w sprzęt fizyczny i jego utrzymanie.

Materiały dodatkowe:

- https://docs.ansible.com/ansible/latest/getting_started/index.html
- https://www.redhat.com/en/interactive-labs/ansible
- https://kodekloud.com/free-labs/ansible
- https://labex.io/tutorials/ansible-how-to-display-output-of-shell-commands-in-ansible-playbooks-415017
- https://gitlab.com/eckholm/ansible-labs
- https://github.com/spurin/diveintoansible-lab
- https://github.com/labex-labs/ansible-free-tutorials?tab=readme-ov-file
- https://github.com/devopsjourney1/ansible-labs?tab=readme-ov-file
- https://github.com/mpsOxygen/ansible
- https://github.com/ansible/workshops
- https://github.com/LMtx/ansible-lab-docker
- https://github.com/spurin/diveintoansible-lab
- https://diveinto.com/playgrounds/ansible-lab
- https://www.youtube.com/watch?v=goclfp6a2IQ&list=PL2_OBreMn7FqZkvMYt6ATmgC0KAGGJNAN&index=1
- https://www.youtube.com/watch?v=KuiAiUyuDY4&list=PLnFWJCugpwfzTlIJ-JtuATD2MBBD7_m3u
- https://www.youtube.com/watch?v=3RiVKs8GHYQ&list=PLT98CRl2KxKEUHie1m24-wkyHpEsa4Y70
- https://www.youtube.com/watch?v=1id6ERvfozo&t=580s
- https://www.youtube.com/watch?v=5hycyr-8EKs
- https://spacelift.io/blog/ansible-tutorial
- https://spacelift.io/blog/ansible-playbooks

Przykłady – konfiguracja lokalna

https://docs.ansible.com/ansible/latest/getting_started/index.html

Instalacja:

```
sudo apt update
sudo apt install ansible -y

On CentOS/Redhat:
sudo yum install epel-release -y
sudo yum install ansible -y


Verify after install:
ansible --version
```

Struktura plików:

```
mkdir -p ~/ansible-labs/{inventories,playbooks,templates,files,roles}


inventory/      # Contains the list of hosts
playbooks/      # Directory for storing your playbooks
roles/          # Directory for Ansible roles
ansible.cfg     # Configuration file
```

Utworzenie inventory z połączeniem lokalnym

```
~/ansible-labs/inventories/local.ini

[local]
localhost ansible_connection=local
```

Pierwsze polecenia:

```
ansible -i inventories/local.ini all -m ping
ansible -i inventories/local.ini all -m command -a "uname -a"
ansible -i inventories/local.ini all -m shell -a "uptime"
ansible -i inventories/local.ini all -m copy -a "src=/etc/hosts dest=/tmp/hosts.copy"

ls -l /tmp/hosts.copy
```

Pierwszy playbook

Cel: utworzyć użytkownika, katalog i plik konfiguracyjny.

playbooks/lab2.yml:

```
- name: Lab 2 - Local configuration
  hosts: local
  connection: local
  become: true

  tasks:
    - name: Utwórz grupę app
      group:
        name: app
        state: present

    - name: Utwórz użytkownika app
      user:
        name: app
        group: app
        create_home: yes
        shell: /bin/bash
        state: present

    - name: Utwórz katalog aplikacji
      file:
        path: /opt/app
        state: directory
        owner: app
        group: app
        mode: '0755'

    - name: Utwórz plik konfiguracyjny
      copy:
        dest: /opt/app/config.ini
        content: |
          [app]
          env=dev
          port=8080
        owner: app
        group: app
        mode: '0644'
```

Wykonaj i sprawdź wynik (pamiętaj o podniesieniu uprawnień do root / użycie sudo):

```
ansible-playbook -i inventories/local.ini playbooks/lab2.yml

ls -l /opt/app
cat /opt/app/config.ini
```

Szablony Jinja2 i zmienne

Cel: utworzenie dynamicznego pliku HTML.

Utwórz plik templates/index.html.j2

```
<!doctype html>
<html>
 <head><title>{{ title }}</title></head>
 <body>
  <h1>{{ title }}</h1>
  <p>Środowisko: {{ env }}</p>
  <p>Data generacji: {{ ansible_date_time.date }}</p>
 </body>
</html>
```

Utwórz plik playbooks/lab3.yml:

```
- name: Lab 3 - Template
 hosts: local
 connection: local
 vars:
  title: "Moja strona Ansible"
  env: "test"
 tasks:
  - name: Wygeneruj stronę index.html
   template:
    src: ../templates/index.html.j2
    dest: /opt/app/index.html
```

Uruchom I sprawdź działanie:

```
ansible-playbook -i inventories/local.ini playbooks/lab3.yml

cat /opt/app/index.html
```

Instalacja pakietów, warunki i pętle

Cel: zainstalować wybrane programy, pokazać pętle i warunki.

Utwórz plik playbooks/lab4.yml:

```
- name: Lab 4 - Packages and loops
  hosts: local
  connection: local
  become: true
  vars:
    common_packages:
      - curl
      - jq
      - tree
  tasks:
    - name: Zainstaluj podstawowe pakiety
      apt:
        name: "{{ item }}"
        state: present
        update_cache: yes
      loop: "{{ common_packages }}"

    - name: Pokaż system
      debug:
        msg: "Dystrybucja: {{ ansible_distribution }} {{ ansible_distribution_version }}"

    - name: Zainstaluj htop tylko na Ubuntu 22.04
      apt:
        name: htop
        state: present
      when: ansible_distribution_version is version('22.04', '==')
```

Uruchom playbook:

```
ansible-playbook -i inventories/local.ini playbooks/lab4.yml
```

Vault i sekrety

Cel: zaszyfrować hasło i użyć go w pliku.

```
mkdir group_vars
echo 'db_password: "SuperSecret123"' > group_vars/local.yml
ansible-vault encrypt group_vars/local.yml
```

Utwórz katalog playbooks/lab5.yml:

```
- name: Lab 5 - Vault example
  hosts: local
  connection: local
  become: true
  vars_files:
    - ../group_vars/local.yml
  tasks:
    - name: Utwórz plik z hasłem
      copy:
        dest: /opt/app/secret.txt
        content: "Hasło do bazy: {{ db_password }}"
        owner: app
        mode: '0600'
```

Uruchom playbook:

```
ansible-playbook -i inventories/local.ini playbooks/lab5.yml --ask-vault-pass
```

Dla chętnych:

Automatyczne sprzątanie

1. Utwórz playbook, który:
    - usuwa pliki starsze niż 7 dni z /tmp,\
    - zapisuje log w /var/log/cleanup.log,
    - po wykonaniu wyświetla wiadomość o liczbie usuniętych plików.
2. Użyj modułów find, file, lineinfile i debug.

Zabezpieczenie SSH

1. Stwórz rolę ssh_hardening, która:
    - w pliku /etc/ssh/sshd_config ustawia PermitRootLogin no i PasswordAuthentication no,
    - restartuje usługę SSH po zmianach.
2. Uruchom z tagiem --tags ssh.

Zadania praktyczne

Zadanie 1 Killercoda: Ansible Labs

https://killercoda.com/het-tanis/course/Ansible-Labs

Czym jest:
Killercoda to interaktywna platforma online z wbudowanymi terminalami Linux, która pozwala ćwiczyć Ansible bez instalacji lokalnej. Kurs „Ansible Labs" autorstwa *het-tanis* to zestaw krótkich, praktycznych scenariuszy uruchamianych w przeglądarce.

Co można tam zrobić:

- Uruchamiać komendy Ansible w prawdziwym środowisku (VM w przeglądarce).
- Poznać podstawy — od prostych pingów i playbooków po prace z plikami YAML i modułami.
- Każdy etap ma osobny lab (np. konfiguracja inventory, playbooki, moduły copy, apt, template, service).
- Zobaczysz, jak działa Ansible **bez żadnej lokalnej instalacji** — wszystko dzieje się w sesji terminalowej przeglądarki.

Zadanie 2 GitLab: Intro to Ansible

https://gitlab.com/eckholm/ansible-labs/-/tree/main/Lab1%20-%20Intro%20to%20Ansible

Czym jest:
Repozytorium GitLab zawierające serię laboratoriów Ansible, przygotowanych jako kurs dla studentów DevOps / SysAdmin. Pierwszy moduł – *Lab1: Intro to Ansible* – to wprowadzenie do konfiguracji środowiska, podstawowych komend i tworzenia pierwszego playbooka.

Co można tam zrobić:

- Przećwiczyć konfigurację środowiska Ansible na Linuxie (inventory, playbook, moduły).
- Zrozumieć strukturę projektu Ansible: katalogi, pliki hosts, group_vars, site.yml.
- Uruchomić przykładowy playbook tworzący użytkowników, instalujący pakiety i edytujący pliki.
- Zobaczyć dobre praktyki (komentarze, czytelny YAML, podział na role).

Zadanie 3 LabEx: Ansible Free Tutorials

https://github.com/labex-labs/ansible-free-tutorials
https://labex.io/tutorials/category/ansible

Czym jest:
LabEx to platforma szkoleniowa oferująca **mini-lekcje i symulacje terminalowe**, które prowadzą użytkownika przez konkretne zadania Ansible.
Repozytorium GitHub zawiera listę wszystkich darmowych laboratoriów dostępnych online w LabEx.

Co można tam zrobić:

- Uruchamiać krótkie, praktyczne zadania (np. „Using the ping module", „Managing cron jobs", „Using vars_files", „Working with hostvars").
- Każdy tutorial otwiera się w **webowym terminalu z instrukcjami krok po kroku**.
- Można zobaczyć efekty działania komend i playbooków w czasie rzeczywistym.
- Zadania dotyczą kluczowych elementów Ansible: zmiennych, ról, szablonów Jinja2, zarządzania pakietami, faktów systemowych i SSH.

Zadanie 4 DevOpsJourney1: Ansible Labs – Dla chętnych

https://github.com/devopsjourney1/ansible-labs

Czym jest:
To bardzo znane repozytorium GitHub prowadzone przez DevOpsJourney1 (autor YouTube „DevOps Journey").
Zawiera kompletny zestaw laboratoriów od podstaw do poziomu profesjonalnego.

Co można tam zrobić:

- Uruchomić wirtualne środowisko Vagrant z hostem kontrolnym i zarządzanymi maszynami.
- W każdym katalogu (ansible-lab1, ansible-lab2 itd.) są gotowe przykłady:
    - konfiguracja SSH,
    - instalacja pakietów,
    - wdrażanie aplikacji,
    - praca z rolami,
    - dynamiczne inventory,
    - ansible-vault i automatyzacja.

Zadanie 5 Dive Into Ansible Lab - Dla chętnych

https://github.com/spurin/diveintoansible-lab

Czym jest:
Rozbudowane środowisko demonstracyjne Ansible w oparciu o **Docker Compose** stworzone przez Jamesa Spurina (autor kursu „Dive Into Ansible"). Repozytorium zawiera gotowy system wielu maszyn (kontroler + nody) uruchamiany jednym poleceniem.

Co można tam zrobić:

- Uruchomić interaktywne laboratorium w przeglądarce pod http://localhost:1000.
- Praktykować wszystkie funkcje Ansible w kontrolowanym środowisku:
    - inventory,
    - playbooki,
    - role,
    - moduły sieciowe,
    - aktualizacje systemu,
    - automatyzację.
- Dodatkowo uczyć się poprzez wizualny interfejs i interaktywne przykłady.
- Całość działa lokalnie (Docker Compose) lub w chmurze (np. Google Cloud Shell).

Zadanie 6 DiveInto.com: Ansible Playground - Dla chętnych

https://diveinto.com/playgrounds/ansible-lab

Czym jest:
Interaktywny **playground online**, będący uproszczoną wersją środowiska z kursu „Dive Into Ansible".
Nie wymaga żadnej konfiguracji działa w przeglądarce jak terminal.

Co można tam zrobić:

- Natychmiast uruchomić sesję z preinstalowanym Ansible.
- Testować komendy i playbooki w czystym środowisku Linux.
- Otworzyć wiele mini-labów z gotowymi przykładami (np. konfiguracja Nginx, użytkownicy, kopie plików).
- Ćwiczyć Ansible bez instalacji lokalnej, co jest świetne do nauki podstaw i testów.