Szkolenie Terraform — Automatyzacja wdrożeń (Infrastructure as Code)

Intensywny kurs, który umożliwi Wam opanowanie jednego z najnowocześniejszych narzędzi w dziedzinie automatyzacji infrastruktury IT. Terraform to potężne narzędzie typu Infrastructure as Code (IaC), które pozwala na definiowanie, wdrażanie i zarządzanie infrastrukturą poprzez kod.

O trenerze

Imię i nazwisko: Juliusz Marciniak

Zawód: Programista, DevOps, TechLead

Doświadczenie: 10 lat

Technologie:

- Java, Spring
- Jenkins, GitLab, Nexus
- Docker, Kubernetes

Jestem programistą Java z 10-letnim stażem. W codziennej pracy zajmuję się tematami cloudowymi i Kubernetesowymi. Oprócz tego doglądam architektury aplikacji w swoich projektach. Jestem entuzjastą nowych rozwiązań i technologii. Zwracam dużą uwagę, by rzeczy wdrażane przeze mnie były przemyślane i spójne. Czasami też programuję… i rekrutuję.

Agenda

Dzień 1

1. Infrastructure as Code
    a. Podejście tradycyjne
    b. Co to jest Infrastructure as Code?
    c. Zalety podejścia IaC
    d. Narzędzia IaC
2. Wprowadzenie do Terraform
    a. Co to jest Terraform?

   b. Jak działa Terraform?
   c. Etapy pracy z Terraform
   d. Elementy Terraform
3. Poznanie języka HCL
   a. Najważniejsze elementy języka HCL i ich wykorzystanie w pracy z Terraform
4. Pierwsze kroki z Terraform
   a. Instalacja Terraform
   b. Komendy Terraform
   c. Stan infrastruktury w Terraform
   d. Providerzy
   e. Data sources
   f. Ćwiczenia

Dzień 2

1. Terraform Cloud
   a. Co to jest Terraform Cloud?
   b. Konfiguracja narzędzia
   c. Ćwiczenia
2. Przykład użycia Terraform w ramach chmury AWS
   a. Konfiguracja konta AWS
   b. Przedstawienie funkcjonalności AWS (AWS provider) dostępnej w Terraform
   c. Zarządzanie uwierzytelnieniem i autoryzacją
   d. Provisioning elementów chmury AWS za pomocą Terraform
3. Jak projektować infrastrukturę systemów w Terraform
   a. Moduły — wprowadzenie (parametry wejściowe, sposób wersjonowania oraz parametry wyjściowe)
   b. Jak zarządzać stanem optymalnie?
   c. Kwestie bezpieczeństwa
   d. Jak pracować w zespole nad projektami Terraform?

Zasady

- Staramy się nie spóźniać
- Telefon odbieramy poza salą
- Jesteśmy aktywni

- Jeśli utknęliśmy na problemie podczas realizacji zadań — natychmiast informujemy
- Nie śmiejemy się z innych i nie krytykujemy pomysłów innych
- Każdy ma prawo do wyrażania swojej opinii
- Przerwa na kawę co 1-1,5h
- Przerwa obiadowa o 13:00

Podstawy teoretyczne

Czym jest Infrastructure as Code

Infrastructure as Code (IaC) to podejście do zarządzania infrastrukturą informatyczną, w którym infrastruktura jest definiowana, konfigurowana i zarządzana za pomocą kodu, zamiast tradycyjnych, ręcznych procesów konfiguracyjnych. Głównym celem IaC jest automatyzacja procesów związanych z wdrażaniem i zarządzaniem infrastrukturą, co przynosi szereg korzyści, takich jak zwiększona elastyczność, skalowalność, powtarzalność oraz zminimalizowane ryzyko ludzkiego błędu.

Kluczowe elementy Infrastructure as Code to:

1. Deklaratywność: W odróżnieniu od imperatywnego podejścia, w IaC definiuje się zamierzenia dotyczące infrastruktury, a nie kroki, które należy podjąć, aby osiągnąć pożądany stan. Oznacza to, że opisuje się, co chcemy osiągnąć, a nie jak to zrobić.
2. Kod źródłowy: Infrastruktura jest reprezentowana za pomocą kodu, najczęściej w formie plików tekstowych. Ten kod jest przechowywany w systemie kontroli wersji, co umożliwia śledzenie zmian, wersjonowanie oraz współpracę między zespołami.
3. Automatyzacja: Za pomocą IaC można automatyzować procesy wdrażania i zarządzania infrastrukturą, co eliminuje potrzebę ręcznego wprowadzania zmian. Automatyzacja zwiększa skuteczność, redukuje błędy i przyspiesza cykl życia infrastruktury.
4. Idempotencja: Procesy wdrażania za pomocą IaC są idempotentne, co oznacza, że można je wielokrotnie stosować, a wynik będzie zawsze taki sam, niezależnie od ilości wywołań.
5. Wsparcie dla różnych środowisk: Dzięki IaC można jednocześnie zarządzać infrastrukturą w różnych środowiskach, takich jak testowe, deweloperskie, stagingowe czy produkcyjne.

Popularnymi narzędziami do implementacji IaC są Terraform, Ansible, Chef i Puppet. Dzięki nim organizacje mogą efektywnie zarządzać swoją infrastrukturą, zwłaszcza w kontekście chmur obliczeniowych, gdzie wymagana jest elastyczność i dynamiczne dostosowywanie zasobów do potrzeb biznesowych.

Korzyści

- Automatyzacja procesów: IaC umożliwia automatyzację procesów wdrożeniowych i zarządzania infrastrukturą, co przyspiesza cykle dostarczania oprogramowania.
- Powtarzalność: Dzięki IaC można powtarzać te same operacje wdrożeniowe wielokrotnie, co

eliminuje błędy wynikające z ręcznych interwencji i zapewnia spójność infrastruktury.

- Skalowalność: IaC ułatwia skalowanie infrastruktury w zależności od potrzeb biznesowych, umożliwiając elastyczne dostosowywanie zasobów do wymagań obciążeniowych.
- Szybkie odtworzenie środowiska: Zastosowanie IaC ułatwia szybkie odtworzenie i zrekonstruowanie środowiska produkcyjnego, co jest istotne w przypadku awarii lub potrzeby migracji.
- Łatwe zarządzanie zmianami: IaC ułatwia śledzenie i zarządzanie zmianami w infrastrukturze, dzięki czemu zespoły mogą skutecznie wprowadzać aktualizacje i poprawki.
- Zwiększenie efektywności: Automatyzacja za pomocą IaC redukuje nakłady czasowe i ludzkie błędy, co przekłada się na zwiększenie efektywności operacyjnej.
- Współpraca zespołów: Kod IaC jest przechowywany w systemie kontroli wersji, co ułatwia współpracę między zespołami, a także umożliwia łatwe monitorowanie i zarządzanie historią zmian.
- Zgodność z przepisami: Stosowanie IaC ułatwia utrzymanie zgodności z przepisami i standardami bezpieczeństwa poprzez precyzyjne definiowanie konfiguracji infrastruktury.
- Łatwość wdrażania w chmurze: Narzędzia IaC są często zoptymalizowane do współpracy z chmurami obliczeniowymi, umożliwiając łatwe zarządzanie infrastrukturą w różnych środowiskach chmurowych.

Kim jest DevOps Engineer

DevOps Engineer — osoba, która jednocześnie jest programistą i administratorem. Działa na przecięciu tych funkcji.

DevOps zajmuje się tworzeniem pipelinów CI/CD, monitoringiem aplikacji i infrastruktury, tworzeniem i przygotowywaniem infrastruktury, automatyzacją procesów.

Narzędzia do IaC

1. Typ narzędzia:
    a. Terraform: Narzędzie typu Infrastructure as Code (IaC), zaprojektowane do zarządzania infrastrukturą w sposób deklaratywny i niezależny od dostawcy.
    b. Chef, Puppet, Ansible: Narzędzia konfiguracji zarządzania systemem, skupione na automatyzacji konfiguracji oprogramowania na maszynach.
    c. AWS CloudFormation: Usługa chmurowa do zarządzania zasobami w chmurze AWS.
2. Język konfiguracji:
    a. Terraform: Korzysta z własnego języka konfiguracji HCL (HashiCorp Configuration Language).
    b. Chef: Wykorzystuje język Ruby.
    c. Puppet: Posługuje się własnym językiem Puppet DSL (Domain Specific Language).
    d. Ansible: Korzysta z języka YAML.

e. AWS CloudFormation: Posługuje się językami JSON i YAML.

3. Architektura:
   a. Terraform: Jest narzędziem agentless, co oznacza brak stałego agenta na zarządzanych maszynach.
   b. Chef, Puppet: Wykorzystują architekturę agent-server, gdzie agenty są zainstalowane na zarządzanych maszynach.
   c. Ansible: Jest agentless, co oznacza brak potrzeby stałego agenta na zarządzanych maszynach.
   d. AWS CloudFormation: Jest usługą chmurową, co oznacza, że nie ma potrzeby instalacji agentów na maszynach.
4. Przenośność i wieloplatformowość:
   a. Terraform: Jest niezależne od dostawcy, co oznacza, że może zarządzać zasobami na różnych platformach.
   b. Chef, Puppet, Ansible: Są wieloplatformowe, ale wymagają odpowiednich modułów do obsługi różnych systemów operacyjnych.
   c. AWS CloudFormation: Jest specyficzne dla chmury AWS.
5. Zastosowanie:
   a. Terraform: Idealny dla organizacji, które chcą zarządzać infrastrukturą jako kodem na różnych platformach, w tym na chmurze i lokalnie.
   b. Chef, Puppet, Ansible: Skoncentrowane na konfiguracji systemów, są używane do automatyzacji zarządzania konfiguracją i instalacją oprogramowania.
   c. AWS CloudFormation: Optymalne dla organizacji korzystających z chmury AWS, które chcą wdrażać i zarządzać zasobami na tym konkretnym środowisku.

Ostateczny wybór narzędzia zależy od konkretnych potrzeb organizacji, preferencji zespołu oraz wymagań związanych z zarządzaniem infrastrukturą i wdrażaniem aplikacji. W niektórych przypadkach organizacje decydują się na kombinację różnych narzędzi w zależności od konkretnej roli, jaką pełnią w ich środowisku.

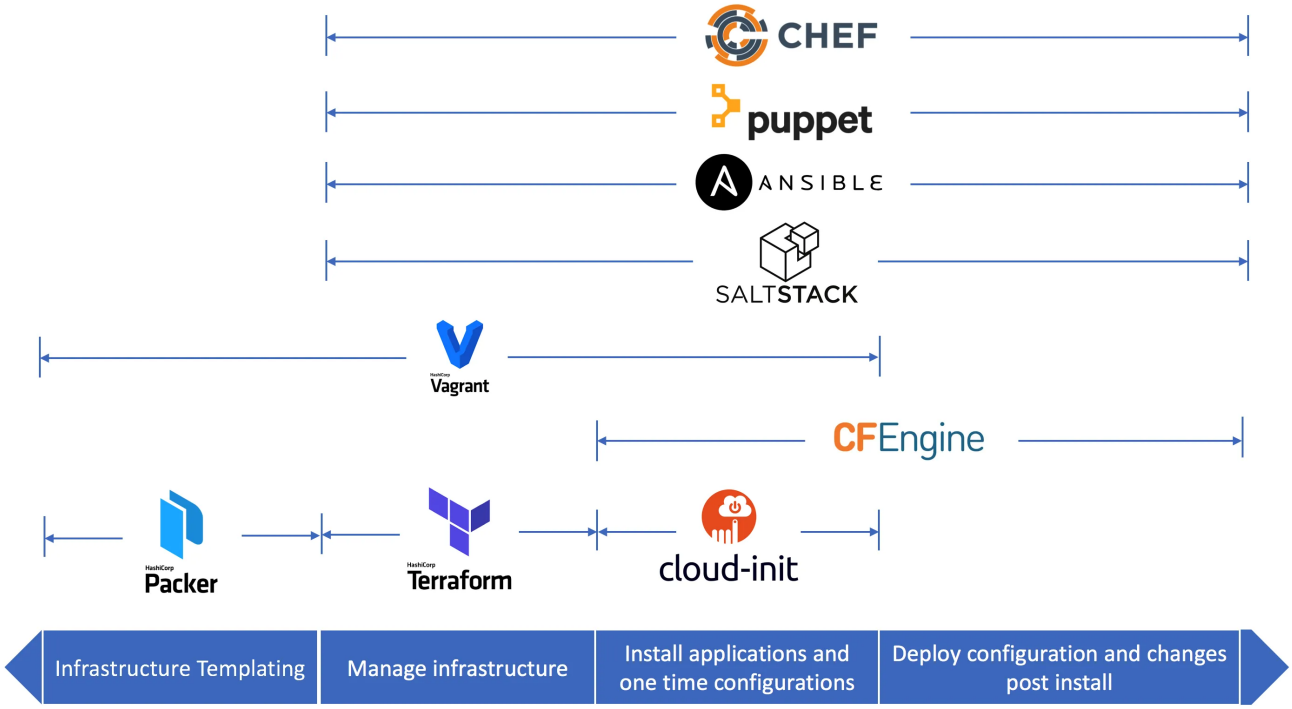

Figure 1. *https://medium.com/cloudnativeinfra/when-to-use-which-infrastructure-as-code-tool-665af289fbde*

Ekosystem narzędzi DevOps

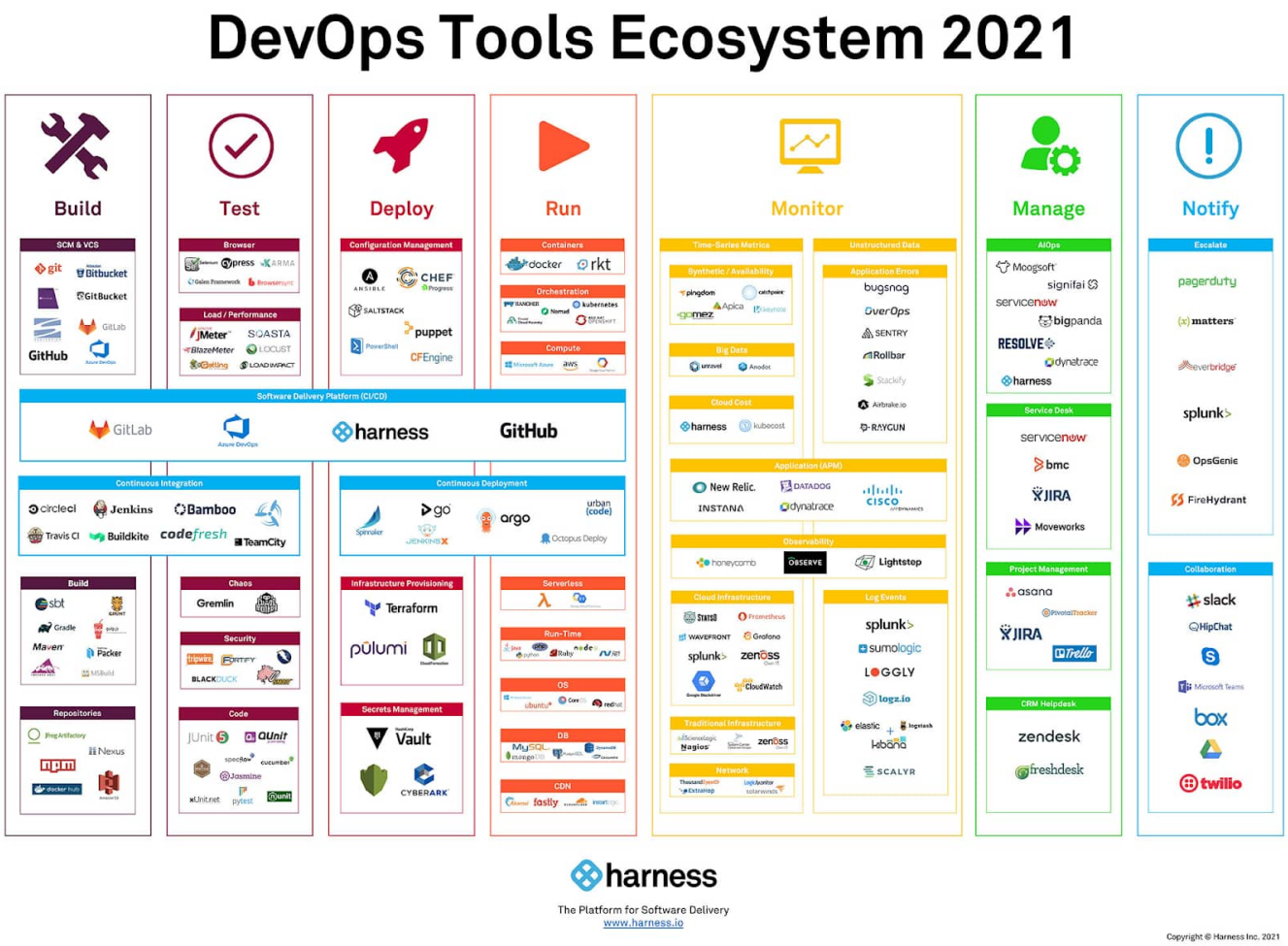

Figure 2. *https://www.harness.io/blog/continuous-delivery-tools*

Terraform

Co to jest Terraform?

Terraform to potężne narzędzie do zarządzania infrastrukturą jako kodem, które umożliwia programistom i administratorom skonfigurowanie i wdrożenie zasobów IT w sposób zautomatyzowany. Dzięki deklaratywnemu językowi konfiguracji, znacznie ułatwia tworzenie plików opisujących pożądany stan infrastruktury. Terraform jest niezależny od dostawcy, co oznacza, że można go używać zarówno w chmurze publicznej (np. AWS, Azure, Google Cloud), jak i w środowiskach lokalnych. Narzędzie to utrzymuje plik stanu, który śledzi aktualny status infrastruktury, umożliwiając efektywne wdrażanie zmian i automatyzację procesów. Dzięki możliwości paralelnego wdrażania oraz modułom, Terraform ułatwia zarządzanie infrastrukturą na dużą skalę i zapewnia elastyczność w dostosowywaniu zasobów do potrzeb organizacji. Aktywna społeczność i obszerny ekosystem dostawców sprawiają, że Terraform jest powszechnie stosowany w branży IT, przyczyniając się do efektywniejszego i spójniejszego zarządzania środowiskiem IT.

Jak działa Terraform?

Terraform działa na zasadzie deklaratywnego zarządzania infrastrukturą, umożliwiając programistom i administratorom definiowanie i wdrażanie zasobów IT w sposób zautomatyzowany.

Etapy pracy z Terraform

1. Definicja infrastruktury jako kodu: Użytkownicy tworzą pliki konfiguracyjne w języku HCL (HashiCorp Configuration Language), opisując pożądany stan infrastruktury. W tym etapie definiuje się zasoby, takie jak maszyny wirtualne, sieci, bazy danych itp.
2. Inicjalizacja: Po utworzeniu plików konfiguracyjnych, użytkownicy wykonują komendę `terraform init`. Terraform pobiera odpowiedniego dostawce (provider) dla zdefiniowanych zasobów i inicjalizuje plik stanu, który będzie śledził aktualny stan infrastruktury.
3. Planowanie: Użytkownicy wykonują komendę `terraform plan`, która generuje plan wdrożenia. Terraform porównuje zdefiniowany kod z aktualnym stanem infrastruktury i określa, które zasoby trzeba utworzyć, zmodyfikować lub usunąć, aby osiągnąć pożądany stan.
4. Wdrażanie: Po zaakceptowaniu planu, użytkownicy wykonują komendę `terraform apply`. Terraform automatycznie wdraża zdefiniowane zasoby, aktualizuje plik stanu i raportuje wyniki.
5. Śledzenie stanu: Terraform utrzymuje plik stanu, który zawiera informacje o aktualnym stanie infrastruktury. Plik ten jest istotny, ponieważ pozwala narzędziu na identyfikację różnic między zdefiniowanym a aktualnym stanem oraz na monitorowanie wprowadzanych zmian.
6. Modyfikacje i utrzymanie: W miarę ewolucji infrastruktury, użytkownicy mogą modyfikować pliki konfiguracyjne, a Terraform automatycznie przewiduje i wdraża zmiany zgodnie z zaktualizowanym kodem.

Elementy Terraform

- Pliki Konfiguracyjne (Configuration Files): Pliki z rozszerzeniem `.tf` zawierają kod konfiguracyjny Terraforma napisany w HashiCorp Configuration Language (HCL). W tych plikach definiuje się zasoby i ich właściwości.
- Dostawcy (Providers): Dostawcy to pluginy, które pozwalają Terraformowi komunikować się z różnymi platformami chmurowymi lub lokalnymi środowiskami. Przykłady to dostawcy dla AWS, Azure, Google Cloud, Docker, itp.
- Zasoby (Resources): Zasoby to elementy infrastruktury, takie jak maszyny wirtualne, sieci, bazy danych itp. W plikach konfiguracyjnych określa się, jakie zasoby chcemy utworzyć lub zarządzać.
- Zmienne (Variables): Zmienne pozwalają na dynamiczne parametryzowanie plików konfiguracyjnych. Używane są do przechowywania wartości, które mogą być różne w różnych środowiskach.
- Moduły (Modules): Moduły to zorganizowane struktury kodu, które mogą być wielokrotnie używane w różnych projektach. Pozwalają na ponowne użycie kodu i organizację konfiguracji.
- Pliki Stanu (State Files): Pliki stanu zawierają informacje o aktualnym stanie infrastruktury, utworzonej lub zarządzanej przez Terraform. Te pliki są istotne do śledzenia, co zostało utworzone, zmienione lub usunięte.
- Pliki Wyjściowe (Output Files): Pliki wyjściowe pozwalają na definiowanie wartości, które mogą być zwracane po pomyślnym wdrożeniu, co ułatwia korzystanie z wyników działania Terraforma w innych skryptach czy narzędziach.

Poznanie języka HCL

Struktura języka HCL (HashiCorp Configuration Language) obejmuje kilka kluczowych elementów, które umożliwiają programistom i administratorom definiowanie konfiguracji w sposób czytelny i zrozumiały. Całość dokumentacji znajduje się tutaj. Kilka przykładów poniżej

- Zmienne Blokowe (Block Variables): Zmienne blokowe są jednym z podstawowych elementów HCL. Są używane do definiowania zmiennych w kontekście konkretnego bloku. Na przykład, w Terraform, zmienne blokowe są często używane do parametryzowania konfiguracji zasobów.

```
variable "instance_type" {
    type    = string
    default = "t2.micro"
}
```

- Blok Konfiguracyjny (Configuration Block): Blok konfiguracyjny zawiera instrukcje dotyczące konkretnego aspektu konfiguracji. Bloki te są zagnieżdżone i strukturalne, co pozwala na hierarchiczne organizowanie kodu.

```
resource "aws_instance" "example" {
  ami           = "ami-0c55b159cbfafe1f0"
```

```
  instance_type = var.instance_type
}
```

- Interpolacja Zmiennych (Variable Interpolation): Interpolacja zmiennych pozwala na dynamiczne wstawianie wartości zmiennych do konkretnych miejsc w konfiguracji.

```
tag = {
  Name = "web-server-${var.environment}"
}
```

- Typy Danych (Data Types): HCL obsługuje różne typy danych, takie jak liczby, ciągi znaków, listy, mapy itp. Pozwala to na elastyczne modelowanie różnych rodzajów danych w konfiguracji.

```
variable "subnet_ids" {
  type    = list(string)
  default = ["subnet-12345678", "subnet-87654321"]
}
```

- Atrybuty (Attributes): Atrybuty określają właściwości zasobów i są definiowane wewnątrz bloków konfiguracyjnych. Na przykład, atrybuty instancji AWS mogą obejmować ami, instance_type, itp.

```
resource "aws_instance" "example" {
    ami           = "ami-0c55b159cbfafe1f0"
    instance_type = var.instance_type
}
```

Zmienne

```
# Deklaracja zmiennej "region" o typie "string" i wartości domyślnej "us-east-1"
variable "region" {
  type    = string
  default = "us-east-1"
}
```

- `variable` to słowo kluczowe do deklaracji zmiennej.
- `"region"` to nazwa zmiennej.
- `{ ⋯ }` to blok definiujący właściwości zmiennej.
- `type = string` określa, że zmienna "region" jest typu string.
- `default = "us-east-1"` ustawia wartość domyślną zmiennej na "us-east-1".

Wyrażenia

```
# Przykładowe wyrażenie w HCL
resource "aws_instance" "example" {
  ami           = "ami-0c55b159cbfafe1f0"
  instance_type = "t2.micro"

  tags = {
    Name        = "ExampleInstance"
    Environment = var.environment
  }
}

# Zmienna "environment" użyta w wyrażeniu
variable "environment" {
  type    = string
  default = "development"
}
```

- `resource` definiuje zasób instancji EC2 w AWS.
- `ami` i `instance_type` to właściwości tego zasobu.
- `tags` to mapa, która zawiera tagi przypisane do instancji
    - `var.environment` to dynamiczna wartość zmiennej.

Pętle

```
# Zmienna zawierająca mapę instancji EC2
variable "ec2_instances" {
  type = map(object({
    ami           = string
    instance_type = string
  }))
  default = {
    example_instance1 = {
      ami           = "ami-0c55b159cbfafe1f0"
      instance_type = "t2.micro"
    },
    example_instance2 = {
      ami           = "ami-0123456789abcdef0"
      instance_type = "t2.small"
    },
  }
}

# Dynamiczna iteracja z użyciem for_each
resource "aws_instance" "example" {
  for_each = var.ec2_instances
```

```
  ami           = each.value.ami
  instance_type = each.value.instance_type

  tags = {
    Name = each.key
  }
}
```

- `variable "ec2_instances"` zawiera mapę instancji EC2, gdzie klucz to unikalna nazwa instancji, a wartość to mapa z właściwościami instancji.
- W `resource "aws_instance" "example"` używamy `for_each = var.ec2_instances` do dynamicznej iteracji przez mapę.
- `each.key` to aktualny klucz iteracji, a `each.value` to mapa z właściwościami instancji dla danego klucza.

Zadania

Instalacja

Zadanie 1. Instalacja Terraform

1. Zainstaluj i uruchom Terraform

Instalacja Terraform w systemie Windows

1. Zainstaluj za pomocą tutoriala z oficjalnej strony https://developer.hashicorp.com/terraform/tutorials/aws-get-started/install-cli

Instalacja Terraform w systemie MacOS

1. Zainstaluj za pomocą tutoriala z oficjalnej strony https://developer.hashicorp.com/terraform/tutorials/aws-get-started/install-cli

Instalacja Terraform w systemie Linux

1. Upewnij się, że system jest aktualny i ma zainstalowane potrzebne oprogramowanie

```
sudo apt-get update && sudo apt-get install -y gnupg software-properties-common
```

2. Zainstaluj odpowiedni klucz GPG

```
wget -O- https://apt.releases.hashicorp.com/gpg | \
gpg --dearmor | \
sudo tee /usr/share/keyrings/hashicorp-archive-keyring.gpg
```

3. Zweryfikuj odcisk palca klucz

```
gpg --no-default-keyring \
--keyring /usr/share/keyrings/hashicorp-archive-keyring.gpg \
--fingerprint
```

4. Powinieneś otrzymać następujący wynik

```
/usr/share/keyrings/hashicorp-archive-keyring.gpg
-------------------------------------------------
pub   rsa4096 XXXX-XX-XX [SC]
AAAA AAAA AAAA AAAA
uid           [ unknown] HashiCorp Security (HashiCorp Package Signing)
<security+packaging@hashicorp.com>
sub   rsa4096 XXXX-XX-XX [E]
```

5. Zainstaluj odpowiednie repozytorium

```
echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] \
https://apt.releases.hashicorp.com $(lsb_release -cs) main" | \
sudo tee /etc/apt/sources.list.d/hashicorp.list
```

6. Zainstaluj Terraform

```
sudo apt update
sudo apt-get install terraform
```

Zadanie 2. Instalacja shell autocomplete

1. Zainstaluj shell autocomplete

```
terraform -install-autocomplete
```

2. Zrestartuj konsolę

Zadanie 3. Sprawdzenie wersji Terraform

1. Sprawdź wersję Terraform komendą

```
terraform version
```

Zadanie 4. Instalacja Dockera

1. Zainstaluj Dockera zgodnie z instrukcją

Podstawowe użycie

Zadanie 1. Skonfigurowanie pierwszych zasobów

1. Stwórz katalog `terraform-docker`
2. Przejdź do tego katalogu
3. Stwórz plik `main.tf` z zawartością

```
terraform {
  required_providers {
    docker = {
      source  = "kreuzwerker/docker"
      version = "~> 3.0.1"
    }
  }
}

provider "docker" {}

resource "docker_image" "nginx" {
  name         = "nginx"
  keep_locally = false
}

resource "docker_container" "nginx" {
  image = docker_image.nginx.image_id
  name  = "tutorial"

  ports {
    internal = 80
    external = 8000
  }
}
```

Zadanie 2. Zainicjowanie providera

1. Zainicjuj providera komendą

```
terraform init
```

1. Sprawdź, co się zmieniło w katalogu, poprzeglądaj pliki (z uwzględnieniem ukrytych)

Zadanie 3. Walidacja zasobów

1. Zwaliduj zasoby komendą

```
terraform validate
```

2. Zmień plik, wprowadzając jakąś głupotę
3. Ponownie zwaliduj plik
4. Sprawdź wyniki

Zadanie 4. Formatowanie plików

1. Sformatuj pliki komendą

```
terraform fmt
```

2. Zmień plik, zmieniając formatowanie
3. Ponownie sformatuj pliki
4. Sprawdź wyniki

Zadanie 5. Zaplanowanie wdrożenie aplikacji

1. Wykonaj

```
terraform plan
```

2. Przeanalizuj co zostało wypisane na ekran

Zadanie 6. Wdrożenie kontenerów i inspekcja stanu

1. Wykonaj

```
terraform apply
```

2. Sprawdź, czy nginx odpowiada na porcie 8000
3. Przeanalizuj, jakie pliki zostały utworzone
4. Sprawdź stan komendą

```
terraform state show docker_container.nginx
```

5. Przeanalizuj wyniki
6. Zobacz jakimi zasobami zarządza Terraform komendą

```
terraform state list
```

Zadanie 7. Usunięcie wszystkich zasobów

1. Usuń zasoby

```
terraform destroy
```

2. Sprawdź, czy zasoby zostały usunięte
3. Sprawdź jak wyglądają obecnie pliki

Zadanie 8. Uaktualnienie zasobów

1. Ponownie wgraj zasoby (Zadanie 6.)
2. Wyedytuj zasoby w pliku `main.tf`, możesz np. zmienić port z 8000 na 8080
3. Ponownie wgraj zasoby
4. Zobacz jak wygląda output
5. Sprawdź, czy zmiany zostały zastosowane

Zadanie 9. Dodanie nowego zasobu

1. Dodaj nowy zasób w pliku `main.tf`, możesz np. dodać kolejny kontener (pamiętaj o zmianie nazwy)
2. Wgraj zasoby
3. Zobacz jak wygląda output
4. Sprawdź, czy zmiany zostały zastosowane

Zadanie 10. Usunięcie zasobu

1. Usuń zasób w pliku `main.tf`
2. Wgraj zasoby (za pomocą apply)
3. Zobacz jak wygląda output
4. Sprawdź, czy zmiany zostały zastosowane

Zadanie 11. Dodajmy zmienne

1. Dodaj plik `variables.tf`
2. Z zawartością

```
variable "container_name" {
  description = "Value of the name for the Docker container"
  type        = string
  default     = "ExampleNginxContainer"
}
```

3. Użyj tej zmiennej jako nazwy kontenera (`var.container_name`)
4. Wgraj zmiany podając nazwę zmiennej jako argument `apply`

Zadanie 12. Output

1. Dodaj plik `outputs.tf`
2. Z zawartością

```
output "container_id" {
  description = "ID of the Docker container"
  value       = docker_container.nginx.id
}

output "image_id" {
  description = "ID of the Docker image"
  value       = docker_image.nginx.id
}
```

3. Wgraj zmiany
4. Podejrzyj output

```
terraform output
```

Zadanie 13. Pętle

1. Skorzystaj z bazowego `main.tf` z zadania 1.
2. Dodaj zmienną, która będzie mapą — nazwa obrazu i nazwa kontenera (key, value)
3. Dodaj do tej zmiennej:
   a. nginx, nginx
   b. httpd, httpd
4. Pamiętaj o usunięciu portów
5. Zmień dodawanie zasobów, żeby obsługiwał tę mapę w pętli

Trzymajmy to w Git

Zadanie 1. Utwórz repozytorium na GitLab

1. Stwórz konto na GitLab (jeśli jeszcze nie masz)
2. Stwórz repozytorium, na którym będziesz trzymać kod

Zadanie 2. Wrzucenie gitignore

1. Wygeneruj plik `.gitignore` na stronie gitignore.io

2. Wrzuć ten plik do repozytorium

Zadanie 3. Wrzucenie projekt Terraform do repozytorium

1. Wrzuć swój projekt do repozytorium
2. Ściągnij projekt lokalnie
3. Uruchom to w IDE

Rozbudowane użycie

Zadanie 1. Użycie wielu providerów

1. Dodaj providera `random` i skorzystaj z jego możliwości
2. Nadaj losowe nazwy stworzonym kontenerem

Zadanie 2. Tworzenie repozytoriów w GitLab

1. Stwórz konto na GitLab (jeśli jeszcze nie masz)
2. Przy pomocy providera `gitlab` zarządzaj repozytoriami
3. Stwórz repozytorium `terraform`
4. Stwórz w nim brancha `develop`

Zadanie 3. Projekt

1. Na podstawie wiedzy z poprzednich zadań stwórz rozbudową infrastrukturę
2. Jeżeli nie masz pomysłu, możesz zrobić coś takiego:
    a. Stwórz repozytorium kodu w GitLab, nazwę, którego zawrzesz w zmiennych
    b. Stwórz sieć w Docker
    c. Stwórz kontener z MariaDB albo MySQL w Docker, pamiętaj, że baza danych potrzebuje wolumenu
    d. Stwórz kontener z Wordpress, który będzie miał randomową nazwę
    e. Zwróć nazwę kontenera z Wordpress jako output

AWS

Zadanie 1. Stworzenie darmowe konto na AWS

1. Załóż darmowe konto na AWS

Zadanie 2. Wygenerowanie ACCESS_KEY i SECRET_ACCESS_KEY

1. Wejdź w ustawienia konta
2. Wygeneruj ACCESS_KEY i SECRET_ACCESS_KEY
3. Zapisz te wartości

Zadanie 3. Stworzenie pierwszego zasobu na AWS

1. Skonfiguruj providera do AWSa
2. Pamiętaj o ustawieniu regionu
3. Stwórz pierwszy zasób

```
resource "aws_vpc" "example" {
  cidr_block = "10.0.0.0/16"
}
```

Zadanie 4. Tworzenie instancji EC2

1. Stwórz instancję EC2 typu t2.micro
2. Nadaj jej tagi

Zadanie 5. Użycie data source do wyszukania AMI

1. Stwórz data source do wyszukania AMI
    a. Właściciel: Amazon
    b. Nazwa zaczynająca się na: al2023-ami-2023
    c. Typ wirtualizacji: hvm
2. Użyj data source do wyszukania obrazu do instancji EC2

Zadanie 6. Tworzenie security groups

1. Stwórz security group (i obiekty powiązane)
2. Security group ma dopuszczać cały ruch wychodzący i dopuszczać ruch przychodzący z dowolnego IP na porcie 22 (SSH)
3. Dołącz ją do instancji EC2

Zadanie 7. Tworzenie kluczy logowania

1. Stwórz klucz ssh
2. Dodaj go do instancji EC2

Zadanie 8. Projekt

1. Utwórz instancję EC2, która
    a. Będzie miała publiczne IP
    b. Będzie dopuszczać ruch po SSH
    c. Będzie miała skonfigurowany dostęp po kluczu do SSH
2. Wykonaj na niej komendę za pomocą SSH, np. `echo 123`

Zadanie 9. Tworzenie bucketu w S3

1. Utwórz bucket w S3

Zadanie 10. Włączenie wersjonowania w bucket

1. Włącz wersjonowanie we wcześniej stworzonym bucket

Zadanie 11. Użycie S3 jako backend

1. Użyj wcześniej stworzonego bucketu jako backend dla Terraforma

Zadanie 12. Współdzielenie stanu z zespołem

1. Dobierzcie się w pary
2. Zróbcie wspólne repozytorium na GitLabie
3. Wrzućcie tam pliki Terraforma (jednej z osób)
4. Pamiętajcie o zawarciu `gitignore`
5. Skonfigurujcie S3 jako backend
6. Spróbujcie współdzielić pracę
7. Jakie problemy i zagrożenia widzicie?

Terraform Cloud

Zadanie 1. Tworzenie konta

1. Stwórz konto na Terraform Cloud

Zadanie 2. Konfiguracja Terraform Cloud

1. Skonfiguruj Terraform Cloud zgodnie z tutorialem na stronie
2. Pobierz przykładowy projekt
3. Uruchom go

Zadanie 3. Konfiguracja nowego Workspace

1. Stwórz nowy Workspace oparty o GitLaba
2. W repozytorium skonfiguruj zasoby do AWS z poprzednich zadań
3. Spróbuj doprowadzić pipeline do działania
4. Stwórz instancję EC2 i Bucket w S3

Zadanie 4. Praca wspólna

1. Dobierzcie się w 5 osobowe zespoły
2. Spróbujcie popracować na Terraform Cloud wspólnie

3. Zwróćcie szczególną uwagę na lockowanie stanu

Moduły

Zadanie 1. Tworzenie nowego modułu

1. Stwórz nowy moduł
2. Użyj tworzenie EC2 (z Zadania 8. Projekt, z sekcji AWS) do stworzenia modułu

Zadanie 2. Zmienne wejściowe

1. W module z poprzedniego zadania stwórz zmienną wejściową, która będzie użyta jako nazwa Bucketa

Zadanie 3. Wyjście

1. W module z poprzedniego zadania zwróć publiczne IP z modułu jako output

Zadanie 4. Wydzielanie modułów — Projekt

1. Wydziel 3 moduły
    a. Tworzący Wordpressa
    b. Tworzący EC2
    c. Tworzący Bucket w S3
2. Skonfiguruj odpowiednio providery
3. Zadeklaruj odpowiednio zmienne wejściowe
4. Zadeklaruj odpowiednio wyjście