Automatyzacja w chmurze obliczeniowej na przykładzie narzędzi wsparcia dla IaC

mgr inż. Piotr Pudełko
Mariusz Kuświk

Skomplikowane zarządzanie infrastrukturą

Czym jest IaC?

Świat technologii nieustannie ewoluuje, a zarządzanie infrastrukturą nie jest wyjątkiem. Tradycyjne "ręczne" przygotowanie serwerów niesie ze sobą liczne trudności: brak powtarzalności, konieczność monitorowania zmian na serwerach, błędy ludzkie, brak kontroli wersji.

W tym kontekście doskonałym sprzymierzeńcem okazuje się podejście IaC (Infrastructure as Code).

"**Idea stojąca za infrastrukturą jako kodem (ang. infrastructure as code, IaC) polega na tworzeniu i wykonywaniu kodu w celu zdefiniowania, wdrożenia, uaktualnienia i usunięcia infrastruktury.**
To pokazuje ważną zmianę w nastawieniu, polegającą na tym, że wszystkie aspekty operacji są traktowane jako oprogramowanie — nawet te związane ze sprzętem (np. fizyczne przygotowanie serwera do pracy).

Za pomocą podejścia do infrastruktury jako kodu (IaC) zyskujemy zdolność do **efektywnego przygotowywania, konfigurowania i wdrażania aplikacji.**
utomatyczne wdrożenie IaC kontra wykorzystanie skryptów tymczasowych
Najprostszym podejściem do automatyzacji wdrożeń są skrypty tymczasowe. Języki takie jak a **bash, python czy powershell** dają bardzo szerokie możliwości.

W procesie automatyzacji jest to ich wielką zaletą, jak i wadą. Bez dobrej dokumentacji i przejrzyście napisanego kodu ciężko jest dojść do tego, za co dany skrypt lub ich zbiór odpowiada. Naprzeciw temu wychodzą **języki specjalizowane, takie jak HCL.**

Narzędzia korzystające z języków specjalizowanych wymuszają przewidywalną strukturę kodu, układ plików oraz zarządzanie poświadczeniami, co znacząco upraszcza współpracę nad projektem.
W artykule tym - pomimo obszernej tematyki zagadnienia - skoncentrujmy się głównie na roli **narzędzia Terraform.**

Terraform - narzędzie do automatyzacji infrastruktury

Terraform jest oprogramowaniem wydanym i wspieranym przez HashiCorp, korzystającym z języka HCL (HashiCorp Configuration Language). J**est to rozwiązanie umożliwiające budowanie, modyfikowanie i usuwanie infrastruktury poprzez deklaratywne definiowanie jej konfiguracji jako kodu.**

Język HCL

HCL, jest językiem deklaratywnym, narzędzie używa planu wykonawczego do analizy istniejącego stanu infrastruktury i porównania go z pożądanym stanem, a następnie generuje plan działania, decydując co zostanie utworzone, zmienione lub usunięte (kolejność deklaracji w kodzie nie jest brana pod uwagę) - to kontrastuje z podejściem proceduralnym, gdzie kroki wykonywane są sekwencyjnie (przykładem narzędzia proceduralnego jest Ansible).

Dzięki deklaratywnemu podejściu w Terraform możemy skupić się na tym co ma zostać wdrożone a nie w jaki sposób chcemy to osiągnąć.

Kolejnym z plusów terraform jest jego uniwersalność, **możemy użyć go dla większości popularnych dostawców chmury publicznej oraz prywatnej, takich jak AWS, Azure, czy VMWare**, wtyczki dostawców są rozwijane zarówno przez firmę HashiCorp jak i społeczność, listę wszystkich wspieranych dostawców znajdziemy tutaj.

Tutaj pojawia się największy minus terraform o którym trzeba pamiętać, **moduły terraform nie są uniwersalne, każdy dostawca dostarcza swoje własne**, z którymi użytkownicy mogą wdrażać zasoby specyficzne dla danej platformy,

Z tego powodu szczególnie warto jest korzystać z dokumentacji, całość jest dobrze opisana przez HashiCorp tutaj. Aby znaleźć dokumentację interesującego nas modułu wybieramy naszego dostawcę chmury a następnie Documentation.

Przykład wdrożenia serwera http z użyciem Terraform

Ze względu na mnogość dostawców chmurowych dla których działa Terraform przykładowe wdrożenie, które przedstawię poniżej będzie realizowane tylko na jednym z nich - **Google Cloud Platform**.

Cloud Shell, który jest tam dostępny ma domyślnie zainstalowany Terraform, dokładny proces instalacji Terraform znajduje się pod tym linkiem.

Wykorzystanie GCP Cloud Shell pozwala również na pominięcie autoryzacji klienta CLI (całość opisana tutaj).

Rozpoczęcie pracy z Terraform

Przy rozpoczęciu pracy z Terraform pierwszym krokiem będzie jego instalacja lub potwierdzenie, że znajduje się w naszym systemie.

Dostępność terraform sprawdzimy poleceniem terraform –help, które podpowiada nam jak budować polecenia terraform.

terraform -help
Usage: terraform [global options] <subcommand> [args]

Mając zainstalowany Terraform możemy przystąpić do budowania pierwszego projektu, każdy powinien znajdować się w osobnym katalogu. Proponuję abyśmy my swój nazwali **gcp_apache_terraform**.

mkdir gcp_apache_terraform && **cd** gcp_apache_terraform

Podstawowa struktura projektów

Katalog, który właśnie utworzyliśmy będzie modułem głównym (root module). Pozostałe moduły mogą być tworzone przez nas i przechowywane lokalnie, pochodzić z repozytoriów społeczności, lub tworzone mogą być w ramach naszych własnych repozytoriów.

Podstawą każdego projektu jest plik **main.tf.** W nim możemy zawrzeć całość naszego kodu. Nie jest to jednak zalecane. Lepszym pomysłem będzie podział na osobne pliki, najczęściej stosuje się poniżej opisaną strukturę:

- **main.tf** - Importuje moduły, ustawienia lokalne i źródła danych.
- **variables.tf** - Zawiera deklaracje zmiennych używanych w pliku main.tf (zmienne dla Terraform mogą być też przechowywane w postaci zmiennych środowiskowych w postaci **TF_VAR_name**).
- **outputs.tf** - Zwraca dane wyjściowe z zasobów utworzonych w pliku main.tf
- **versions.tf** - Określa wymagania dotyczące wersji Terraform oraz dostawców.
- **providers.tf** - Deklaruje dostawców, ich konfigurację oraz ewentualne aliasy.

Aby utworzyć wszystkie wyżej wymienione pliki wykonujemy poniższą komendę

touch main.tf variables.tf outputs.tf versions.tf provider.tf

versions.tf
Na początek zajmiemy się edycją pliku **versions.tf**. Możemy otworzyć go w edytorze tekstowym dostępnym z poziomu terminala (np. Nano, Vim), lub w przystawce oferowanej przez Google Cloud. Dla uproszczenia skorzystamy tutaj z Nano.

nano versions.tf

Kod podany poniżej wklejamy do terminala i zapisujemy plik wybierając ctrl+o, następnie opuszczamy Nano, wybierając ctrl+x

```
terraform {
  required_providers {
    google = {
      source = "hashicorp/google"
      version = "5.7.0"
    }
  }
}
```

Przejdźmy do wytłumaczenia powyższego kodu:

Plik **versions.tf** zawiera konfigurację Terraform, w której określone są informacje dotyczące dostawcy Google Cloud Platform. W bloku Terraform znajduje się definicja dostawcy Google z podaniem źródła oraz wersji. Ten plik umożliwia Terraformowi skorzystanie z dostawcy GCP do zarządzania zasobami w chmurze Google Cloud.

1. Blok Terraform

- **terraform** - Jest to blok, wewnątrz którego definiowane są ustawienia globalne dla projektu.
- **required providers** - Zawiera definicję dostawców wymaganych do zarządzania infrastrukturą w danym projekcie Terraform.

2. Dostawca Google Cloud Platform

- **google** - Alias dla dostawcy GCP. Aliasu używamy w definicjach zasobów, aby wskazać, który dostawca ma obsługiwać dany zasób.
- **source** - Wskazuje z jakiego źródła będą pobrane informacje wymagane dla danego dostawcy.
- **version** - Określa używaną wersję dostawcy. Nie jest to wymagane, ale zaleca się jego używanie.

providers.tf
Podobnie jak w przypadku pliku **versions.tf** wklejamy kod podany poniżej do pliku **providers.tf**

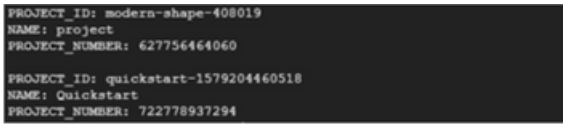
nano providers.tf

```
provider "google" {
  # credentials = file("sciezka/do/pliku/poswiadczen.json")
  #
  project     = "ID Projektu"
  region      = "us-central1"
  zone        = "us-central1-a"
  # inne ustawienia dostawcy GCP
}
```

ctrl o + ctrl + x

Wymagane jest tutaj podmienienie “ID Projektu” na realne ID naszego projektu. ID naszych projektów podejrzymy poleceniem:

gcloud projects list

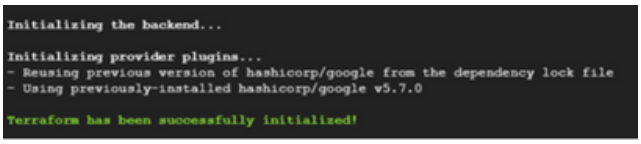


Fragment kodu wklejony do pliku **providers.tf** określa dostawcę jako Google Cloud Platform (GCP) i zawiera niezbędne konfiguracje, takie jak projekt, region oraz obszar (zone) z którego będziemy korzystać przy zarządzaniu zasobami w chmurze.
Tak jak wspomniałem na początku, Google Cloud Shell nie wymaga uwierzytelniania, dlatego też parametr credentials pozostaje zakomentowany.

Inicjalizacja konfiguracji Terraform
Po zadeklarowaniu danych wymaganych dla dostawcy, przeprowadzamy inicjalizację projektu.

terraform init

Terraform pobierze wtyczkę dostawcy z wcześniej podanego źródła. Jeżeli wszystko przebiegnie pomyślnie, powinniśmy zobaczyć poniższy komunikat.



main.tf
Przechodzimy teraz do deklaracji samej infrastruktury. Aby skorzystać z reguły firewall dla portu 80 (którą dodamy jako osobny zasób w późniejszej części), przypisujemy tag http-server do maszyny.

Dodatkowo, w celu maksymalnego uproszczenia procesu wdrożenia maszyny, umieszczamy wewnątrz bloku prosty skrypt bash. Skrypt ten zainstaluje aplikację Apache i uruchomi ją jako usługę zarówno teraz, jak i po restarcie systemu.

```
resource "google_compute_instance" "google" {
  name         = "terraform"
  machine_type = "e2-micro"

  tags = ["http-server", "inne", "tagi"]

  boot_disk {
    initialize_params {
      image = "debian-cloud/debian-11"
    }
  }

  metadata_startup_script = <<-SCRIPT
    #!/bin/bash
    echo "Poczatek skryptu startowego!"
    apt update -y
    apt install apache2 -y
    systemctl enable --now apache2
  SCRIPT

    network_interface {
     network = "default"
  }
}
```

Poniżej przedstawiam bardziej schematyczną wersję wdrożenia zasobu:

```
resource "<DOSTAWCA>_<TYP>" "<NAZWA>" {
  [KONFIGURACJA ...]
}
```

- <DOSTAWCA>_<TYP> lub w naszym przypadku google_compute_instance - to moduł udostępniany przez dostawcę, który będziemy wykorzystywać (lista modułów udostępniana przez HashiCorp tutaj), zazwyczaj będą nosiły nazwę którą wskazałem - <DOSTAWCA>_<TYP>

- <NAZWA> "google" - wykorzystanie aliasu dostawcy który zadeklarowaliśmy w pliku providers.tf w postaci provider "google"

- [KONFIGURACJA ...] - W przypadku maszyn wirtualnych tworzonych dla GCP (moduł google_compute_instance) wymagane jest podanie parametrów **name, machine_type oraz network_interface,** reszta jest opcjonalna,

- boot_disk - debian-11 jest obecnie domyślną opcją, jednak postanowiłem zadeklarować go ręcznie w celu lepszego zobrazowania dostępnych opcji, wykorzystany tutaj będzie czysty obraz Debiana w wersji 11

- metadata_startup_script - skrypt, który tutaj podamy zostanie uruchomiony przy pierwszorazowym uruchomieniu systemu, wykorzystamy tę możliwość do instalacji aplikacji Apache2 oraz uruchomienia jej jako usługa, innym przydatnym zastosowaniem mogłaby być automatyczna aktualizacja serwera

- network_interface - Deklarujemy tutaj interfejs sieciowy, a następnie wybieramy sieć, do której będzie podpięty, w naszym przypadku będzie to sieć domyślna default.

- tags - tagi deklarujemy w formie listy, tag "http-server" zostanie przez nas wykorzystany przy okazji deklarowania reguły sieciowej, dającej dostęp do portu 80 dla sieci zewnętrznej

Mając wciąż otwarty plik **main.tf** przystępujemy do zadeklarowania zapotrzebowania na zewnętrzny adres IP, do tego zadania użyjemy modułu google_compte_address.

Jedynym wymaganym parametrem jest tutaj **name**, tak jak wspomniałem wcześniej, kolejność deklaracji nie jest brana pod uwagę, więc poniższy kod możemy dodać na początku lub na końcu pliku, Terraform sam decyduje co utworzyć pierwsze.

ZA POMOCĄ PODEJŚCIA DO INFRASTRUKTURY JAKO KODU (IAC) ZYSKUJEMY ZDOLNOŚĆ DO EFEKTYWNEGO PRZYGOTOWYWANIA, KONFIGUROWANIA I WDRAŻANIA APLIKACJI

```
resource "google_compute_address" "ip_deb_apache" {
  name = "ipv4-address"
}
```

Chcąc wykorzystać wcześniej zadeklarowany adres ip (**ip_deb_apache**) odwołujemy się kolejno do: modułu który odpowiada za stworzenie danego zasobu, następnie do aliasu który przypisaliśmy do zasobu (ip_deb_apache) i na koniec jego atrybutu (address).

<WYKORZYSTANY_MODUŁ>.<NAZWA>.<ATRYBUT>

Nasze odwołanie będzie miało więc następującą postać:

google_compute_address.ip_deb_apache.address

Do deklaracji network_interface dodajemy access_config wraz z wcześniej wspomnianym odwołaniem. Poniżej przedstawiam pełną treść deklaracji, która powinna znaleźć się w pliku **main.tf**:

```
network_interface {
  network = "default"
  access_config {
    nat_ip = google_compute_address.ip_deb_apache.address
   }
 }
```

Ostatnim krokiem, który musimy zadeklarować w pliku **main.tf**, jest reguła firewall dla wcześniej przypisanego tagu maszyny **http-server**. Do tego celu skorzystamy z modułu google_compute_firewall.
Konieczne jest nadanie regule nazwy (name) oraz określenie sieci, dla której ma ona działać (network).

W naszym przypadku umożliwiamy ruch TCP na porcie 80 oraz w zakresie portów 85-90 (zakres portów jest zadeklarowany w celach prezentacyjnych i nie będzie wykorzystany). Adresy IP, z których serwer będzie dostępny, określamy jako source_ranges = ["0.0.0.0/0"], co oznacza, że ruch może pochodzić z dowolnego adresu IP.

```
resource "google_compute_firewall" "rules" {
  name        = "http-servers"
  network     = "default"
  source_ranges = ["0.0.0.0/0"]
  allow {
    protocol  = "tcp"
    ports     = ["80", "85-90"]
  }

  target_tags = ["http-server"]
}
```

outputs.tf
Chcąc wyświetlić przypisane nam IP, umieszczamy poniższy kod w pliku **outputs.tf**

```
output "ip" {
  value = google_compute_address.ip_deb_apache.address
}
```

Składnia odwołująca się do adresu jest taka sama jak przy przypisywaniu adresu do karty sieciowej, o czym wspomniałem wcześniej (przypisanie noszące nazwę network_interface znajduje się w pliku **main.tf**).

1. Zaplanowanie wdrożenia
Przechodzimy do generowania planu wdrożenia. Komenda **terraform plan** sprawdza połączenie do dostawcy, a następnie tworzy wykres tego, co zostanie utworzone.

terraform plan

Schematyczny wynik przykładowej komendy przedstawiam poniżej.

```
# google_compute_instance.google will be created
+ resource "google_compute_instance" "google" {
    + ... (informacje o zasobie)
  }

Plan: 3 to add, 0 to change, 0 to destroy.
```

Komenda terraform plan nie wprowadza żadnych zmian, informuje nas jedynie co zostanie **dodane, zmienione lub usunięte**:
- **+** - dodanie zasobu
- **~** - zmiana danego zasobu
- **–** - usunięcie zasobu

W naszym przypadku zostaną dodane trzy zasoby (maszyna wirtualna, reguła firewall oraz zewnętrzny adres ip), nic nie zostanie zmienione, ani usunięte.

Komenda terraform plan jest dobra min. do sprawdzania poprawności naszych plików, dopiero wykonanie terraform apply wprowadza faktyczne zmiany.

2. Wdrożenie infrastruktury
Kolejnym krokiem jest przeprowadzenie faktycznego wdrożenia za pomocą polecenia:

terraform apply

Terraform ponownie przedstawi plan wdrożenia, aby potwierdzić wpisujemy **yes** w terminalu.

```
Plan: 3 to add, 0 to change, 0 to destroy.

Changes to Outputs:
  + ip = (known after apply)

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes
```

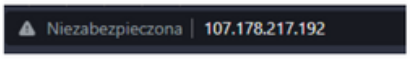
Jeżeli wszystko przebiegnie poprawnie to wyświetlony zostanie komunikat Apply complete!, ilość zasobów, które zostały dodane, zmienione lub usunięte oraz adres IP, który wywołaliśmy w pliku **outpus.tf**

```
Apply complete! Resources: 3 added, 0 changed, 0 destroyed.

Outputs:

ip = "107.178.217.192"
```

3. Test wdrożonego serwera http
Adres IP, który został wyświetlony przy wdrożeniu naszego serwera wklejamy w oknie przeglądarki.



Zostanie wyświetlona strona aplikacji Apache z komunikatem "**It works!**", potwierdzająca działanie naszego serwera.

4. Usunięcie wdrożenia
Podczas przeprowadzania testowych wdrożeń w chmurze publicznej dobrym zwyczajem jest usuwanie tworzonej infrastruktury po zakończeniu testów. Ta praktyka pozwala uniknąć naliczania niepotrzebnych kosztów.

Chcąc usunąć utworzoną infrastrukturę wykonujemy:

trraform destroy

Polecenie terraform destroy działa podobnie do terraform apply. Terraform destroy usuwa wszystkie zasoby **zadeklarowane w konfiguracji** Terraform.

Podobnie jak w przypadku tworzenia infrastruktury, zostanie przedstawiony schemat usuwania, a następnie zostaniemy poproszeni o potwierdzenie. Jeżeli zgadzamy się z usunięciem wszystkich zadeklarowanych zasobów infrastruktury, wpisujemy **yes**.

```
Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value:
```

Usunięcie zasobów zostanie potwierdzone poniższym komunikatem:

```
Destroy complete! Resources: 3 destroyed.
```

Na koniec przedstawiam ogólny schemat przeprowadzonego przez nas wdrożenia

1. Instalacja Terraform lub potwierdzenie jego posiadania.
2. Utworzenie katalogu projektu i podstawowych plików.
3. Deklaracja dostawcy w pliku konfiguracyjnym.
4. **terraform init** - Inicjalizacja projektu, pobranie wtyczek dostawców i modułów.
5. Deklaracja wdrażanej infrastruktury.
6. **terraform plan** - Generowanie wykresu wdrożenia (opcjonalne, ale zalecane).
7. **terraform apply** - Wdrożenie na podstawie zdefiniowanej konfiguracji.
8. **terraform destroy** - Usunięcie dodanej infrastruktury.