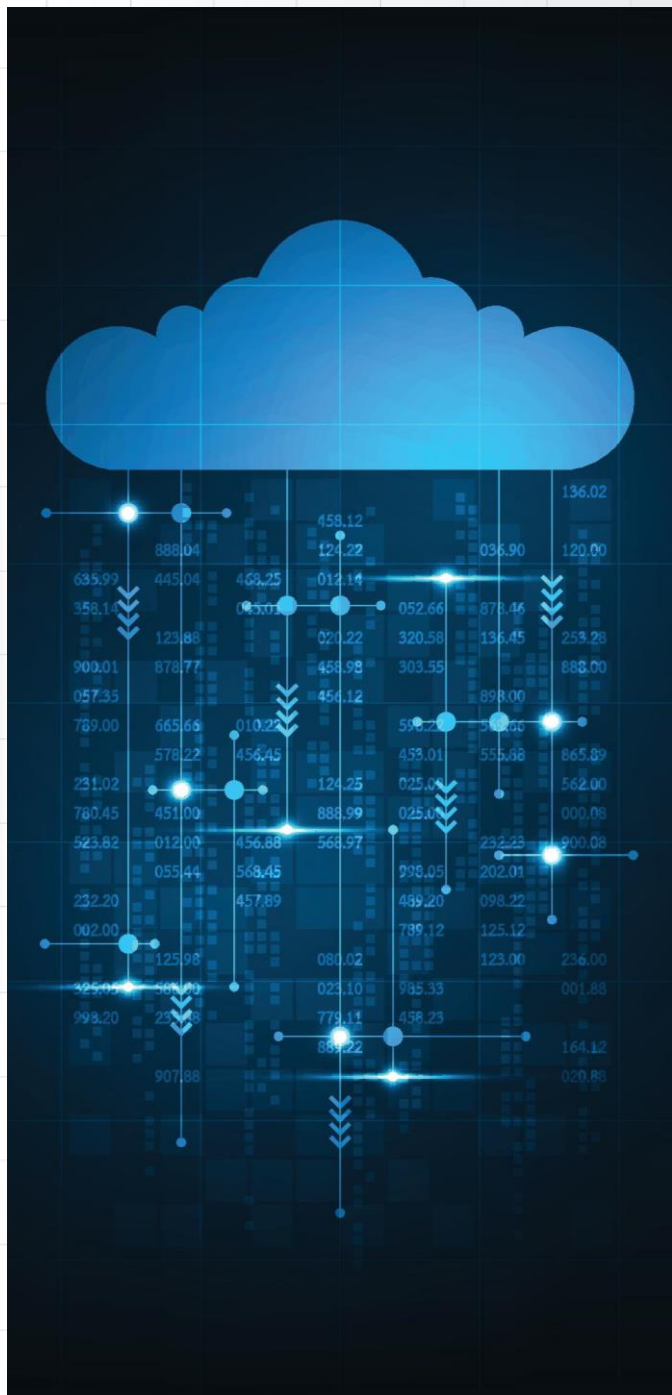




Wrocław
University
of Science
and Technology



Programowanie w chmurze

Rafał Palak

Politechnika Wrocławska

Stworzenie Prostej Konfiguracji

- Utwórz plik konfiguracyjny z rozszerzeniem .tf

<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/finding-an-ami.html>

```
provider "aws" {  
  region = "us-west-2"  
}  
  
resource "aws_instance" "example" {  
  # Uwaga: Sprawdź najnowsze AMI dla Twojego regionu https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/finding-an-ami.html  
  ami           = "ami-"  
  instance_type = "t2.micro"  
}
```

Pliki Konfiguracyjne

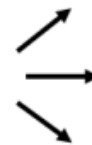
- Służą do **definiowania i konfiguracji zasobów**, które mają być zarządzane,
- Teraform używa języka konfiguracyjnego HashiCorp Configuration Language (HCL) lub JSON
- Pliki mają zazwyczaj rozszerzenie **.tf** i można je grupować w katalogach.



Terraform Configuration File (HCL)



Terraform CLI



Infrastructure Automation/Provisioning with Terraform

Struktura Pliku – Bloki

- Pliki HCL są **zorganizowane w bloki**.
- Elementy bloku ***resource***:
 - **aws_instance** - Typ zasobu. W tym przypadku, chodzi o instancję EC2 w AWS.
 - **Example** - Nazwa zasobu w konfiguracji Teraforma. Jest to identyfikator, który pozwala Teraformowi jednoznacznie zidentyfikować zasób w ramach danego projektu.
 - **ami = "ami-06dd92ecc74fdfb36"** - Argument ustawiający Amazon Machine Image (AMI), czyli obraz maszyny, który będzie użyty do uruchomienia instancji.
 - **instance_type = "t2.micro"** - Argument określający typ instancji EC2. W tym przypadku, to t2.micro, co jest jednym z mniejszych, tańszych typów instancji.

```
resource "aws_instance" "example" {  
  ami           = "ami-06dd92ecc74fdfb36"  
  instance_type = "t2.micro"  
}
```

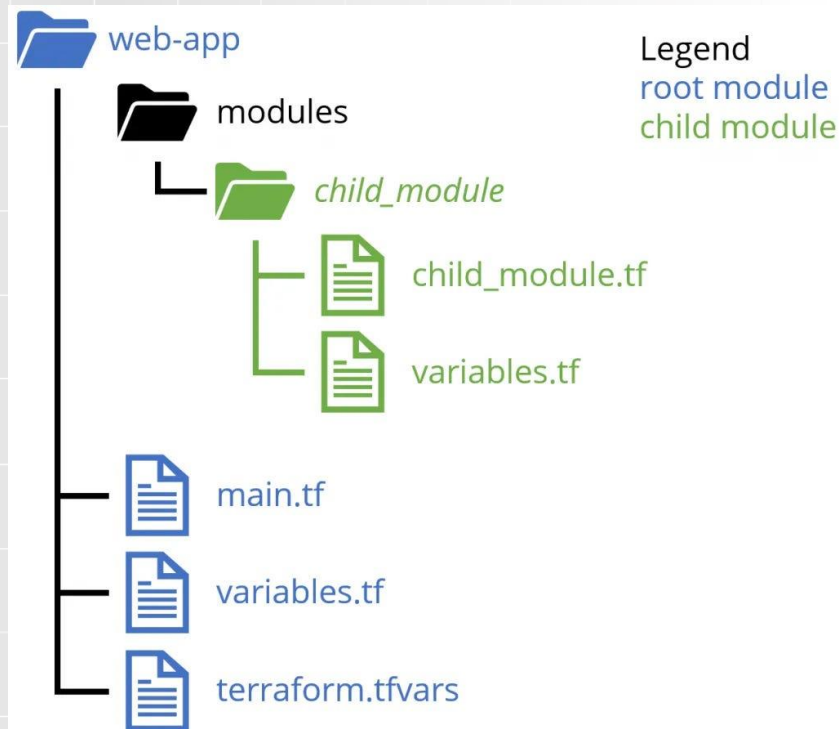
AMI - Amazon Machine Image

- **Gotowy do użycia obraz wirtualnej maszyny**, który można używać do szybkiego uruchamiania instancji (wirtualnych serwerów)
- Zawiera wszystkie niezbędne informacje do uruchomienia maszyny
 - **System operacyjny:** Podstawowy system operacyjny, który będzie uruchomiony na instancji.
 - **Oprogramowanie serwerowe:** Oprogramowanie serwerowe, takie jak serwer WWW lub baza danych, która jest prekonfigurowana w AMI.
 - **Ustawienia i konfiguracje:** Dodatkowe ustawienia i konfiguracje mogą być zapisane w AMI, co pozwala na szybkie wdrażanie prekonfigurowanych systemów.
- **Różne regiony AWS mogą mieć różne ID AMI dla tego samego oprogramowania**

```
provider "aws" {  
  region = "us-west-2"  
}  
  
resource "aws_instance" "example" {  
  # Uwaga: Sprawdź najnowsze AMI dla Twojego regionu https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/finding-an-ami.html  
  ami           = "ami-"  
  instance_type = "t2.micro"  
}
```

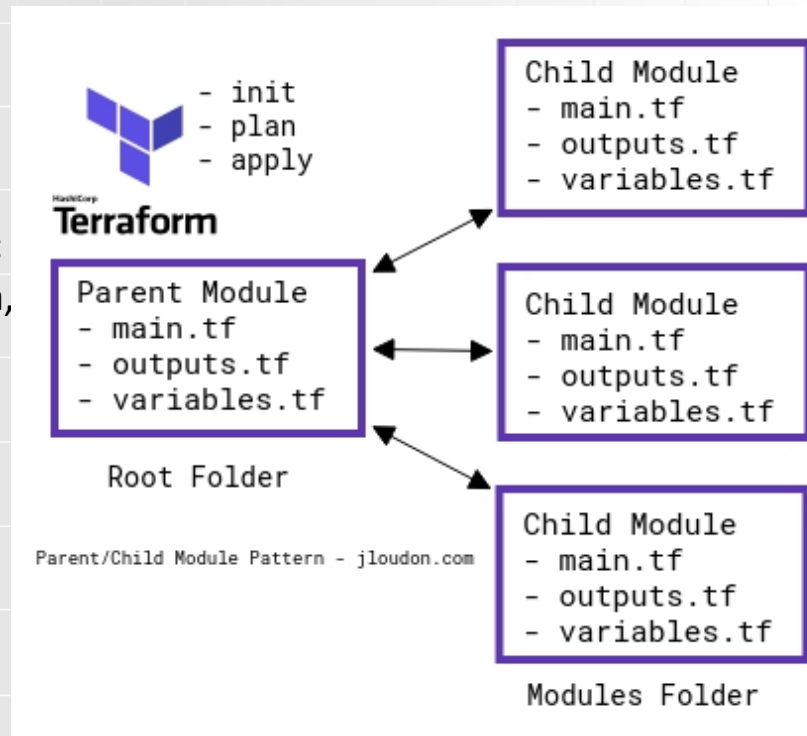
Moduły

- To **kontenery na wielokrotne użycie dla zasobów** Teraforma, które **pozwalają na grupowanie zasobów w logiczne jednostki**, które mogą być używane jako **budynki blokowe do budowania infrastruktury**. Dzięki temu **kod jest bardziej organizowany, łatwiejszy do zarządzania i można go łatwo ponownie użyć**.
- **Użycie modułów** pozwala na **uniknięcie powtarzania kodu, ułatwia zarządzanie złożonymi konfiguracjami, promuje dobre praktyki** w zakresie reużywalności kodu i ułatwia zarządzanie wersjami i aktualizacjami.



Moduły – jak używać

- **Definiowanie Modułu** - Moduł jest definiowany poprzez utworzenie nowego katalogu i umieszczenie w nim plików konfiguracyjnych Teraforma. Każdy moduł **powinien mieć swoje własne pliki konfiguracyjne**, które definiują zasoby, które moduł będzie zarządzać.
- **Użycie Modułu** - Aby użyć modułu, należy dodać blok module do głównej konfiguracji Teraforma, określając ścieżkę do katalogu modułu oraz ewentualne wejścia (inputy), które moduł wymaga.
- **Parametry Wejściowe i Wyjściowe** - Moduły mogą przyjmować parametry wejściowe (zmienne), które pozwalają na konfigurowanie ich działania. Moduły mogą również zwracać wartości wyjściowe, które mogą być używane przez inne części twojej konfiguracji Teraforma.
- **Moduły Publiczne i Prywatne** - Można tworzyć własne moduły lub korzystać z modułów dostępnych publicznie w Rejestrze Modułów Teraforma, gdzie społeczność udostępnia moduły dla różnych dostawców i usług.



Moduły – dobre praktyki

- **Dokładna Dokumentacja** - Każdy moduł powinien być dokumentowany, opisując jego cel, wymagane parametry wejściowe i wartości wyjściowe.
- **Zakres Modułu** - Każdy moduł powinien być odpowiedzialny za jedną logiczną część infrastruktury, co ułatwia ponowne użycie i zarządzanie.
- **Wersjonowanie** – Zaleca się użycie wersjonowania dla modułów, szczególnie gdy są udostępniane innym. Pozwala to na lepsze zarządzanie zależnościami i aktualizacjami.
- **Moduły są kluczowym elementem w budowaniu skalowalnych i łatwych do zarządzania konfiguracji w Teraformie.** Dzięki nim można budować bardziej złożone infrastruktury zarządzalne w sposób efektywny.



Moduły – przykład

```
resource "aws_instance" "example" {  
  ami          = var.ami_id  
  instance_type = var.instance_type  
  
  tags = {  
    Name = "ExampleInstance"  
  }  
}
```

```
variable "instance_type" {  
  description = "The type of EC2 instance"  
  type        = string  
}  
  
variable "ami_id" {  
  description = "The ID of the AMI for the EC2 instance"  
  type        = string  
}
```

```
# Specify the provider, in this case, AWS  
provider "aws" {  
  region = "us-west-2"  
}  
  
# Use the module to create an EC2 instance  
module "example_ec2_instance" {  
  source = "../modules/ec2_instance"  
  
  instance_type = "t2.micro"  
  ami_id        = "ami-0c55b159cbfafe1f0"  
}
```



Terraform – Zmienne i Wyjścia

Zmienne (variable) [1]

- **Zmienne w Teraformie służą do przechowywania wartości, które mogą być używane w wielu miejscach w konfiguracji. Umożliwiają parametryzację konfiguracji, co pozwala łatwo dostosować swoje zasoby bez konieczności zmiany całego kodu.**
- **Typy Zmiennych** - Teraform wspiera różne typy zmiennych, w tym **string**, **number**, **bool**, a także złożone typy takie jak **list**, **map**, i **object**.
- **Deklaracja Zmiennych** - Zmienne **deklaruje się w plikach konfiguracyjnych** za pomocą **bloku variable**, określając ich **nazwę** i **opcjonalnie domyślną wartość** oraz **opis**.
- **Przypisywanie Wartości:** Wartości zmiennych mogą być przypisywane w linii poleceń przy uruchomieniu Teraforma, w plikach konfiguracyjnych, lub w plikach zewnętrznych, jak pliki **.tfvars** lub zmienne środowiskowe.

```
variable "instance_type" {  
  description = "Typ instancji EC2"  
  type        = string  
  default     = "t2.micro"  
}
```


Zmienne (variable) [2]

- **Variable** - definiuje zmienne, które można używać w konfiguracji, umożliwiając jej parametryzację.
- Elementy bloku ***variable***:
 - **region** - Nazwa zmiennej, którą można używać w innych częściach konfiguracji. Dzięki temu **można parametryzować różne elementy konfiguracji**.
 - **default = "us-west-2"** - Wartość **domyślna dla zmiennej**. Jeśli podczas wywoływania terraform apply lub terraform plan nie zostanie podana wartość dla tej zmiennej, **używana będzie wartość domyślna**.
 - **Type** - Określa typ zmiennej (np. string, list, map itd.).
 - **Description** - Dodaje opis zmiennej, który jest wyświetlany, gdy użytkownik jest proszony o jej wartość.
 - **Validation** - Pozwala na dodanie reguł walidacji dla zmiennej.

```
variable "region" {  
  type          = string  
  description = "The AWS region where resources will be created"  
  default       = "eu-central-1"  
  validation {  
    condition     = length(var.region) > 0  
    error_message = "The region variable must not be empty."  
  }  
}
```

Zmienne (variable) [3]

- Można używać tej **zmiennej** w **innych miejscach konfiguracji**, co ułatwia zmiany.
- **Zmiana wartości** - Wartość zmiennej można zmienić na kilka sposobów, np. **przez użycie flagi -var w linii poleceń, przez plik terraform.tfvars lub poprzez zmienne środowiskowe.**
- **Planowanie i Zastosowanie** – Przy użyciu terraform plan i terraform apply, Terraform zastępuje wystąpienia `var.region` rzeczywistą wartością zmiennej, co **pozwalą na dynamiczne generowanie konfiguracji.**
- **Zalety:**
 - **Elastyczność** - Umożliwia zmianę konfiguracji bez potrzeby zmiany samego kodu, co jest szczególnie przydatne w środowiskach wielostanowiskowych.
 - **Czytelność i utrzymanie** - Użycie zmiennych sprawia, że kod jest łatwiejszy do zrozumienia i utrzymania.
 - **Wielokrotne użycie** - Możesz używać tych samych zmiennych w różnych modułach i projektach, co czyni je bardzo reużywalnymi.



```
provider "aws" {  
    region = var.region  
}
```

Zmienne (variable) - walidacja

- Typy Warunków Walidacji:
 - **Zakres wartości** - Można określić **minimalne, maksymalne wartości dla liczbowych zmiennych wejściowych** lub **konkretne wartości**, które są **dozwolone dla zmiennych typu string**.
 - **Format łańcucha** - Używając **wyrażeń regularnych**, można wymagać, aby **wartości string spełniały określony format**, np. adresu email czy identyfikatora UUID.
 - **Długość łańcucha** - Sprawdzanie **minimalnej lub maksymalnej długości łańcuchów znakowych**.
 - **Weryfikacja list i map** - Możliwość sprawdzania, czy **listy czy mapy zawierają określone elementy**, czy ich **rozmiar spełnia określone wymagania**.

- Sposoby Wyrażenia Warunków:
 - **Wykorzystanie funkcji wbudowanych** - Terraform oferuje szereg funkcji, które mogą być używane **do formułowania warunków walidacji**, takie jak **length, regex, contains** itp.
 - **Logika warunkowa** - Możesz **używać operatorów logicznych** (takich jak **&&, ||, !**) do tworzenia **bardziej złożonych warunków walidacji**.

- Współpraca i Dokumentacja:
 - **Dokumentuj wymagania walidacji** - Pomaga to innym członkom zespołu zrozumieć, **jakie wartości są akceptowalne i dlaczego pewne ograniczenia zostały wprowadzone**.
 - **Współpraca przy definiowaniu walidacji** - **Włączanie członków zespołu do procesu definiowania walidacji** może pomóc w **uchwyceniu różnych przypadków użycia i potrzeb biznesowych**.

```
variable "instance_type" {  
  type      = string  
  description = "EC2 instance type"  
  
  validation {  
    condition     = length(var.instance_type) > 4  
    error_message = "The instance_type value must be longer than 4  
characters."  
  }  
}
```

Zmienne lokalne (locals)[1]

- **Locals** - służy do definiowania zmiennych lokalnych, które są dostępne tylko w ramach danego pliku konfiguracyjnego lub modułu. Są one **użyteczne dla uproszczenia konfiguracji** i dla **zwiększenia czytelności kodu**.
- Elementy bloku *locals*:
 - ***common_tags*** - Jest to **nazwa zmiennej lokalnej**. Możesz nazwać ją dowolnie.
 - ***Owner = "devops team"*** - Jest to **klucz-wartość dla mapy**, którą przypisywanej do *common_tags*. Można mieć **wiele takich par klucz-wartość** w **jednej zmiennej lokalnej**.
- Zmienna lokalna **common_tags** jest **dostępna w ramach tego samego pliku konfiguracyjnego** i można ją używać w różnych blokach, jak **resource, module, provider**, itd.
- **Zlety**:
 - **Czytelność**: Umożliwia grupowanie powtarzających się elementów konfiguracji, co sprawia, że kod jest bardziej zorganizowany.
 - **Reużywalność**: Dzięki zmiennym lokalnym łatwiej jest zarządzać kodem, który ma być używany w wielu miejscach w obrębie jednej konfiguracji.
 - **Zachowanie Kontekstu**: Zmienne lokalne są widoczne tylko w ramach pliku/modułu, co pomaga w zachowaniu kontekstu i ogranicza zasięg zmiennych do miejsc, w których są rzeczywiście potrzebne.



```
locals {  
    common_tags = {  
        Owner = "devops team"  
    }  
}
```

Zmienne lokalne (locals) [2]

- Zlety:
 - **Czytelność** - Umożliwia grupowanie powtarzających się elementów konfiguracji, co sprawia, że kod jest bardziej zorganizowany.
 - **Reużywalność** - Dzięki zmiennym lokalnym łatwiej jest zarządzać kodem, który ma być używany w wielu miejscach w obrębie jednej konfiguracji.
 - **Zachowanie Kontekstu** - Zmienne lokalne są widoczne tylko w ramach pliku/modułu, co pomaga w zachowaniu kontekstu i ogranicza zasięg zmiennych do miejsc, w których są rzeczywiście potrzebne.
 - **Kompozycja** - Umożliwia składanie różnych elementów konfiguracji z mniejszych, łatwiejszych do zarządzania kawałków, takich jak wspólne tagi.



```
locals {  
    common_tags = {  
        Owner = "devops team"  
    }  
}
```


locals vs variable

Locals

- **Zakres** - Zmienne lokalne są **dostępne tylko w ramach jednego pliku konfiguracyjnego lub modułu**, w którym są zdefiniowane.
- **Brak Modyfikacji** - Nie można ich modyfikować za pomocą argumentów linii komend ani plików zmiennych (*.tfvars).
- **Czytelność** - Używane głównie do uproszczenia kodu i poprawy czytelności w jednym pliku lub module.
- **Brak Wartości Domyślnych i Walidacji** - Nie można określić wartości domyślnych ani używać walidacji.
- **Brak Interakcji z Użytkownikiem** - Użytkownik końcowy nie interaguje z nimi bezpośrednio.

Variable

- **Zakres** - Mogą być używane przez cały projekt i mogą być przekazywane między modułami.
- **Modyfikacja** - Można je modyfikować za pomocą argumentów linii komend, plików zmiennych (*.tfvars) lub zmiennych środowiskowych.
- **Reużywalność** - Używane do tworzenia konfiguracji, które można łatwo dostosować do różnych środowisk lub ustawień.
- **Wartości Domyślne i Walidacja** - Można określić wartości domyślne, typy danych, i nawet wprowadzić walidację.
- **Interakcja z Użytkownikiem** - Użytkownik końcowy często dostarcza wartości dla zmiennych, co pozwala na dostosowanie zachowania Teraforma.

Wyjścia (output) [1]

- **Wyjścia** w Teraformie służą do zwracania informacji o zasobach utworzonych przez Teraform, co może być użyteczne dla użytkownika lub innych części konfiguracji Teraforma.
- **Wyjścia** deklaruje się używając bloku **output**, określając ich **nazwę** i **wartość**.
- **Wyjścia** mogą być użyte do uzyskania ważnych informacji po zastosowaniu konfiguracji, jak adresy IP, identyfikatory zasobów, itp. Są one szczególnie użyteczne w modułach, gdzie **wyjście z jednego modułu** może być przekazane jako **wejście do innego**.

```
output "instance_id" {  
  value      = aws_instance.example.id  
  description = "The ID of the created instance."  
}
```

```
module "source_module" {  
  source          = "../modules/source_module"  
  instance_name   = "ExampleInstance"  
}  
  
module "target_module" {  
  source          = "../modules/target_module"  
  source_instance_id = module.source_module.instance_id  
}
```

```
variable "source_instance_id" {  
  type      = string  
  description = "The source instance ID to be used in this module."  
}
```

Wyjścia (output) [2]

- **Output** - Definiuje wyjścia, które będą wyświetlane po zastosowaniu konfiguracji. Służą do ekstrakcji wartości z utworzonej infrastruktury.
- Elementy bloku *output*:
 - **instance_ip** - To jest nazwa wartości wyjściowej. Jest ona używana do identyfikacji tej konkretnej wartości wyjściowej w wynikach komendy **terraform apply** oraz do odwołań do niej w innych miejscach.
 - **value = aws_instance.my_instance.public_ip** - Określa wartość, która będzie przypisana do **instance_ip**.
 - **Description** - Opis wartości wyjściowej, może być pomocny dla innych osób korzystających z konfiguracji.
 - **Sensitive** - Ustawienie, które określa, czy wartość wyjściowa zawiera wrażliwe dane i czy powinna być ukrywana w logach.

```
output "instance_ip" {  
  value      = aws_instance.my_instance.public_ip  
  description = "The public IP address of the instance"  
  sensitive   = false  
}
```

```
resource "aws_instance" "my_instance" {  
  ami           = "ami-06dd92ecc74fdfb36"  
  instance_type = "t2.micro"  
}
```

Zmienne i Wyjścia – dobre praktyki

- **Nazewnictwo Zmiennych** – Zaleca się użycie **znaczących nazw** dla zmiennych, które **jasno określają, co zawierają i do czego służą**.
- **Organizacja** - Grupuj powiązane zmienne i wyjścia, aby ułatwić zarządzanie konfiguracją, szczególnie w większych projektach.
- **Modularyzacja** - **zmienne i wyjścia** mogą być pomocne **do tworzenia modułów**, które można łatwo ponownie używać w różnych projektach i środowiskach.
- **Bezpieczeństwo** – Zaleca się **zachować ostrożność przy wyprowadzaniu wrażliwych danych jako wyjść**, szczególnie w środowiskach wielodostępnych lub publicznych.

Any `-var` and `-var-file` options on the command line, in the order they are provided.

Any `*.auto.tfvars` or `*.auto.tfvars.json` files, processed in lexical order of their filenames

`terraform.tfvars.json`

`terraform.tfvars`

Environment variables

Precedence



Terraform – najważniejsze typy bloków

Bloki – module [1]

- **Module** - służy do wykorzystania modułów, czyli reużywalnych, samodzielnych fragmentów kodu Teraforma.
- **Moduły** to jednostki, które można parametryzować, używać w różnych projektach, a nawet udostępniać dla społeczności. Są one szczególnie przydatne w dużych projektach i zespołach, gdzie różne elementy infrastruktury są tworzone w podobny sposób.

```
module "my_vpc" {  
  source = "terraform-aws-modules/vpc/aws"  
  version = "2.64.0"  
  
  name = "my-vpc"  
  cidr = "10.0.0.0/16"  
  
  azs = ["us-west-2a", "us-west-2b", "us-west-2c"]  
  private_subnets = ["10.0.1.0/24", "10.0.2.0/24", "10.0.3.0/24"]  
  public_subnets = ["10.0.4.0/24", "10.0.5.0/24", "10.0.6.0/24"]  
  
  enable_nat_gateway = true  
  enable_vpn_gateway = true  
  
  tags = {  
    Terraform = "true"  
    Environment = "dev"  
  }  
  
  providers = {  
    aws = aws.custom  
  }  
  
  depends_on = [  
    aws_iam_role.example  
  ]  
}
```

Bloki – module [2]

- **source = "terraform-aws-modules/vpc/aws"** - Wskazuje, że kod modułu będzie pobrany z repozytorium terraform-aws-modules dla VPC w AWS.
- **version = "2.64.0"** - Określa wersję modułu, co zapewnia spójność i stabilność.
- **name = "my-vpc"** - Nazwa VPC, która zostanie utworzona.
- **cidr = "10.0.0.0/16"** - Zakres CIDR dla VPC. Określa, jakie adresy IP będą dostępne w tej chmurze prywatnej.
- **azs = ["us-west-2a", "us-west-2b", "us-west-2c"]** - Lista stref dostępności (Availability Zones), w których będą umieszczone podsieci.
- **private_subnets = ["10.0.1.0/24", "10.0.2.0/24", "10.0.3.0/24"]** - Lista zakresów CIDR dla prywatnych podsieci.
- **public_subnets = ["10.0.4.0/24", "10.0.5.0/24", "10.0.6.0/24"]** - Lista zakresów CIDR dla publicznych podsieci.
- **enable_nat_gateway = true**: Opcja ta sprawia, że w VPC zostanie utworzona brama NAT (Network Address Translation)

```
module "my_vpc" {  
  source = "terraform-aws-modules/vpc/aws"  
  version = "2.64.0"  
  
  name = "my-vpc"  
  cidr = "10.0.0.0/16"  
  
  azs = ["us-west-2a", "us-west-2b", "us-west-2c"]  
  private_subnets = ["10.0.1.0/24", "10.0.2.0/24", "10.0.3.0/24"]  
  public_subnets = ["10.0.4.0/24", "10.0.5.0/24", "10.0.6.0/24"]  
  
  enable_nat_gateway = true  
  enable_vpn_gateway = true  
  
  tags = {  
    Terraform = "true"  
    Environment = "dev"  
  }  
  
  providers = {  
    aws = aws.custom  
  }  
  
  depends_on = [  
    aws_iam_role.example  
  ]  
}
```


Bloki – module [3]

- **enable_nat_gateway = true** - Opcja ta sprawia, że w VPC zostanie utworzona brama NAT (Network Address Translation).
- **enable_vpn_gateway = true** - Opcja ta umożliwia utworzenie bramy VPN.
- **tags = { ... }** - Mapa tagów, które będą dołączone do zasobów utworzonych przez ten moduł.
- **providers = { aws = aws.custom }** - Ustala, że moduł będzie używał niestandardowego providera AWS o nazwie aws.custom.
- **depends_on = [aws_iam_role.example]** - Zapewnia, że moduł nie zostanie uruchomiony, dopóki nie zostanie utworzona określona rola IAM (aws_iam_role.example).

```
module "my_vpc" {  
  source = "terraform-aws-modules/vpc/aws"  
  version = "2.64.0"  
  
  name = "my-vpc"  
  cidr = "10.0.0.0/16"  
  
  azs          = ["us-west-2a", "us-west-2b", "us-west-2c"]  
  private_subnets = ["10.0.1.0/24", "10.0.2.0/24", "10.0.3.0/24"]  
  public_subnets  = ["10.0.4.0/24", "10.0.5.0/24", "10.0.6.0/24"]  
  
  enable_nat_gateway = true  
  enable_vpn_gateway = true  
  
  tags = {  
    Terraform = "true"  
    Environment = "dev"  
  }  
  
  providers = {  
    aws = aws.custom  
  }  
  
  depends_on = [  
    aws_iam_role.example  
  ]  
}
```


Bloki – zasoby (resource)

- **Resource** - służy do **definiowania zasobów**, które mają być zarządzane przez Teraform.



```
resource "aws_instance" "example" {  
    ami          = "ami-06dd92ecc74fdfb36"  
    instance_type = "t2.micro"  
}
```

Zasoby

- **Zasób** to jednostka infrastruktury, taka jak maszyna wirtualna, grupa zabezpieczeń czy baza danych.
- **Deklaracja** - Zasoby są deklarowane w plikach konfiguracyjnych Teraforma.
- **Atrybuty** - Każdy zasób ma różne atrybuty, które można konfigurować. Na przykład, dla maszyny wirtualnej w **AWS (EC2)**, można ustawiać **typ instancji**, **klucz SSH**, **obraz** itd.
- **Zależności** - Zasoby mogą mieć **zależności między sobą**. Na przykład, zasób reprezentujący **bazę danych** może **wymagać**, aby zasób reprezentujący **sieć VPC** został najpierw utworzony.
- **Zmienne** - Można używać **zmiennych do parametryzacji zasobów**, co zwiększa elastyczność i ponowne użycie konfiguracji.
- **Stan** - Informacje o zasobach są **przechowywane w plikach stanu**, co pozwala Teraformowi zarządzać ich cyklem życia.

```
provider "aws" {  
  region = "eu-central-1"  
}  
  
resource "aws_instance" "moja_instancja" {  
  ami           = "ami-06dd92ecc74fdfb36"  
  instance_type = "t2.micro"  
  
  tags = {  
    Name = "moja_instancja"  
  }  
}
```

Zasoby - przykład

- **Zasób** to instancja EC2 w Amazon Web Services (AWS)
- **Provider** - provider "aws" definiuje, że użycie **AWS** jako **dostawcy** usług chmurowych.
- **region** = "us-west-2" określa **region**, w którym zasoby będą utworzone.
- **resource** "aws_instance" "moja_instancja" - zasób, który chcemy utworzyć. Definiuje **zasób** typu **aws_instance** z nazwą **moja_instancja**. To jest właśnie.
- **Atrybuty** - Wewnątrz bloku zasobu, różne atrybuty są ustawione:
- **ami** = "ami-0abcdef1234567890" - Określa ID obrazu AMI, który będzie używany do utworzenia instancji.
- **instance_type** = "t2.micro" - Typ tworzonej instancji.
- **tags** = { Name = "moja_instancja" } - Etykiety przypisane do instancji.

```
provider "aws" {  
    region = "eu-central-1"  
}  
  
resource "aws_instance" "moja_instancja" {  
    ami           = "ami-06dd92ecc74fdfb36"  
    instance_type = "t2.micro"  
  
    tags = {  
        Name = "moja_instancja"  
    }  
}
```

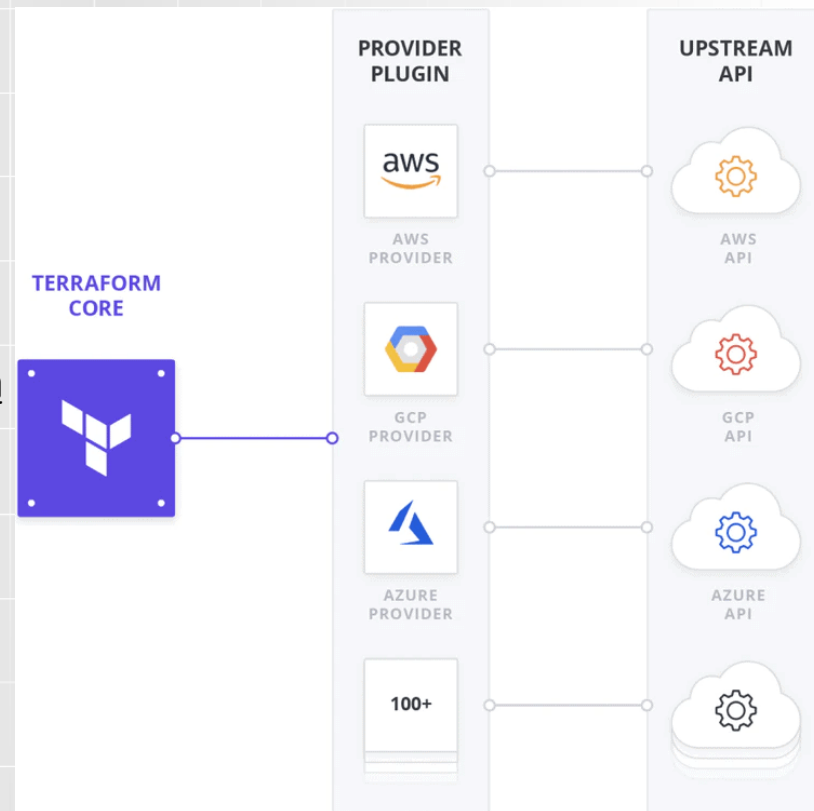
Bloki – provider

- **Provider** - określa dostawcę usług, z którym Teraform będzie interagował, np. AWS, Azure, Google Cloud.
- Elementy bloku *provider*:
 - **aws**: Jest to typ **dostawcy**, w tym przypadku Amazon Web Services. Istnieje wiele dostępnych dostawców, takich jak Azure, Google Cloud, DigitalOcean itp.
 - **region = "us-west-2"**: Jest to argument konfiguracyjny określający **region AWS**, w którym Teraform będzie zarządzał zasobami.
 - **profile**: Określa profil **AWS CLI**, który ma być użyty dla uwierzytelnienia.
 - **access_key** i **secret_key**: Klucze do uwierzytelnienia, jeśli nie używasz profilu AWS CLI czy roli IAM.
 - **version**: Możesz zdefiniować **wersję dostawcy**, z którą chcesz pracować.

```
provider "aws" {  
  region      = "eu-central-1"  
  profile     = "my-aws-profile"  
  version     = "~> 2.0"  
}
```

Dostawcy [1]

- **Definicja** - Dostawcy to serwisy lub platformy, z którymi Teraform interaguje w celu zarządzania zasobami. Najpopularniejsi to **AWS, Azure i Google Cloud**, ale Teraform obsługuje wiele innych, włączając w to **dostawców on-premises** jak **VMware**.
- **Wielu Dostawców** - W jednym projekcie można używać wielu dostawców, co pozwala na zarządzanie zasobami w różnych chmurach lub serwisach.
- **Konfiguracja** - Każdy dostawca ma własny zestaw wymaganych i opcjonalnych argumentów konfiguracyjnych. Te argumenty są zazwyczaj używane do autoryzacji i konfiguracji różnych ustawień.
- **Zasoby i Atrybuty** - Każdy dostawca oferuje zestaw zasobów i ich atrybutów, które można zarządzać. Na przykład, dostawca AWS oferuje zasoby takie jak **aws_instance** dla **EC2** czy **aws_s3_bucket** dla **S3**.



Dostawcy [2]

- **Dokumentacja** - Każdy dostawca ma własną dokumentację, która opisuje dostępne zasoby, ich atrybuty i jak je konfigurować. Jest to **niezbędne dla efektywnego wykorzystania Teraforma z danym dostawcą**.
- **Inicjalizacja** - Aby korzystać z konkretnego dostawcy, należy go zainicjalizować w projekcie za pomocą polecenia **terraform init**.



imgflip.com

JHFE-CLARK.TUMBLR

Dostawcy - przykład

- provider "aws" mówi Teraformowi, aby używać **dostawcy Amazon Web Services (AWS)** do zarządzania zasobami.

```
provider "aws" {  
    region = "eu-central-1"  
}  
  
resource "aws_instance" "moja_instancja" {  
    ami          = "ami-06dd92ecc74fdfb36"  
    instance_type = "t2.micro"  
  
    tags = {  
        Name = "moja_instancja"  
    }  
}
```



Wrocław
University
of Science
and Technology

Dziękuję za uwagę