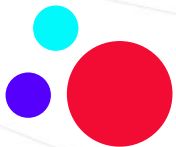


Infrastructure as a Code Terraform

infoShare Academy



HELLO

Maciej Małek

DevOps engineer
AWS, Terraform, Python





Agenda

- Wprowadzenie
- Infrastructure as a Code – co to jest ? kiedy stosujemy ?
- Terraform
 - podstawy
 - składnia
 - pliki
 - provider
 - Podstawowe komendy terraform cli
 - resource
 - data sources
 - variables
 - outputs
 - locals
 - modules



Baremetal

- Wysokie koszty (zakup, utrzymanie, administracja, licencje)
- Trudne w skalowaniu, migracji i administracji

Wirtualizacja

- Lepsza dystrybucja zasobów
- Nadal pozostaje większość problemów z baremetal

Chmura

- „Współdzielenie” zasobów pay as you go
- Scentralizowane zarządzanie i koszty (optymalizacja)



01. Infrastructure as a Code

Co to jest ?



infoShare
ACADEMY

Infrastruktura jako kod (IaC) używa metodologii DevOps i służy do przechowywania wersji z opisowym modelem do definiowania i wdrażania infrastruktury, takich jak sieci, maszyny wirtualne, itp.

Tak jak ten sam kod źródłowy zawsze buduje ten sam plik binarny tak model IaC generuje to samo środowisko za każdym razem, gdy jest wdrażany.

Dzięki IaC, zespoły DevOps mogą współpracować ze sobą.

Przykład:

- zespół security może przygotować moduły związane z np. skanowaniem serwerów
- centralne zespoły mogą przygotować gotowe do użycia moduły zgodne z dobrym praktykami w firmie



IaC – podstawowe zasady

- Idempotentność – zawsze ten sam stan
- Powtarzalny proces – możliwość powielania środowisk (infrastruktury)
dev -> staging -> prod
- Samodokumentujące
- Ułatwienie częstych zmian
- Możliwość ciągłego ulepszania
- Szybkie przywracanie po awariach
- Wszystko trzymane w kodzie. Zmiany manualne zostaną utracone przy kolejnym “deploymencie” lub jeżeli mają być zachowane, muszą być zakodowane.

Immutable vs. Mutable

Immutable infrastructure

Niezmienna po wdrożeniu

Każda nowa zmiana oznacza wdrożenie i nową wersję

Zmiany przewidywalne i niezawodne

Zredukowany „configuration drift”

Dobrze się sprawdza przy własnych aplikacjach.

Mutable infrastructure

Nieśledzone zmiany

Zmiany „manualne”

Łatwiejsze i szybsze zmiany

Podatna na błędy

Podatna na „configuration drift”

Często wymagane przy wdrożeniach aplikacji 3rd party

Co ? czy Jak ?

Co – podejście deklaratywne

- Definiujemy co chcemy osiągnąć, a nie jak to zrobić
- Idempotentność, utrzymujemy stan
- Nie ma efektów ubocznych
- Możemy łatwo wycofać zmiany

Przykłady:

CloudFormation

Terraform

Jak – podejście imperatywne

- Definiujemy proces/logikę jak coś chcemy osiągnąć/stworzyć
- Nie śledzimy stanu
- Możliwe efekty uboczne
- Nie możemy łatwo wycofać zmian

Przykłady:

Chef

AWS CLI / Bash

Co vs. Jak?

Terraform

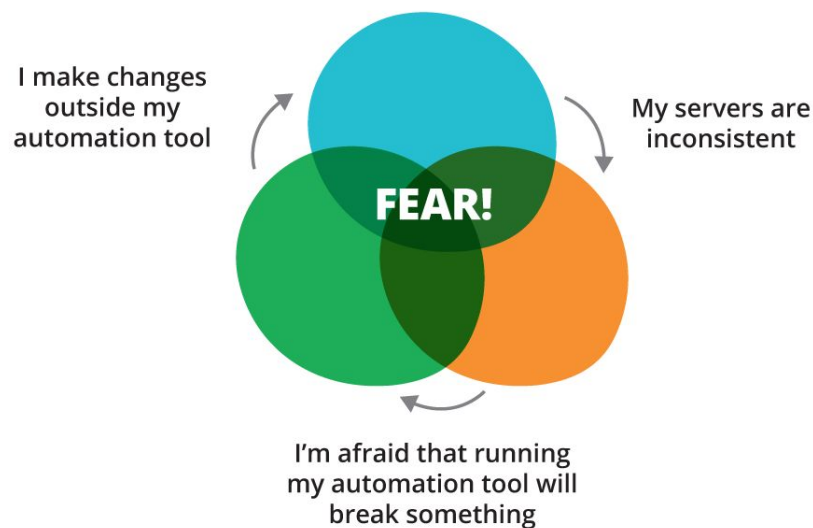
```
resource "aws_instance" "ec2" {  
  ami                = "ami-0abcdef1234567890"  
  instance_type      = "t2.micro"  
  vpc_security_group_ids = [sg-0b0384b66d7d692f9]  
  subnet_id          = "subnet-08fc749671b2d077c"  
  iam_instance_profile = "ec2-iam-role"  
  tags                = { Name = "test-ec2" }
```

Bash / AWS CLI

```
aws ec2 run-instances \  
  --image-id ami-0abcdef1234567890 \  
  --instance-type t2.micro \  
  --subnet-id subnet-08fc749671b2d077c \  
  --security-group-ids sg-0b0384b66d7d692f9 \  
  --iam-instance-profile ec2-iam-role \  
  --tag-specifications \  
  'ResourceType=instance,Tags=[{Key=Name,Value=test-ec2}]'
```

Unikaj Automatyzacyjnej Spirali Strachu!

W celu utrzymania niezmiennych (immutable) środowisk, musisz przezwyciężyć Automatyzacyjną Spiralę Strachu i unikać zmian „manualnych”



Source: <https://opensenselabs.com/blog/tech/infrastructurecode principles and practices>

Zalety i wady

Zalety

- Łatwość zarządzania
- Łatwość migracji
- Niższe koszty
- Szybkość i elastyczność
- Spójność
- Śledzenie zmian
- Dokumentacja
- Obniżenie ryzyka błędu ludzkiego
- Automatyzacja (CI/CD, testowanie)

Wady

- Configuration drift
- Przypadkowe awarie/usunięcie komponentów
- Wyższy poziom wejścia jeśli chodzi o wiedzę
- Trudniejsze odtworzenie błędów



Narzędzia

Zarządzanie infrastrukturą

- CloudFormation (AWS)
- Azure Resource Manager (Azure)
- Google Cloud Deployment manager (GCP)
- Terraform (niezależny)
- Pulumi (niezależny)



Zarządzanie konfiguracją

- Ansible
- Chef
- Puppet
- Saltstack







Terraform – podstawowe informacje

- HCL (HashiCorp Configuration Language) – język deklaracyjny
- Łatwy do czytania, samodokumentujący
- Niezależny od wykorzystywanej chmury
- Prosty w uruchomieniu (kilka komend)
- Duże wsparcie społeczności

Terraform (OSS) – <https://www.terraform.io/>

Terraform Enterprise

Terraform Cloud – <https://app.terraform.io/session>



Terraform – pliki

Rozszerzenie *.tf

Możliwa konfiguracja w jednym lub wielu plikach

Praktyka:

- main.tf
- outputs.tf
- variables.tf
- versions.tf / provider.tf

<https://www.terraform.io/language/files>



Terraform – state

- rozszerzenie “.tfstate”
- plik w formacie JSON zarządzany przez Terraform
- służy do mapowania zasobów – to co jest zapisane w kodzie vs. to działający serwis na platformie (np. AWS / Azure)
- stan sprawdzany jest przed i aktualizowany po wprowadzaniu zmian
- zawiera dane wrażliwe (np. hasła do bazy danych) – powinien być traktowany jako secret
 - nie umieszczamy w systemie wersjonowania)
 - zwykle przechowywany na zewnętrznym storage’u (AWS S3, Azure Storage Account)

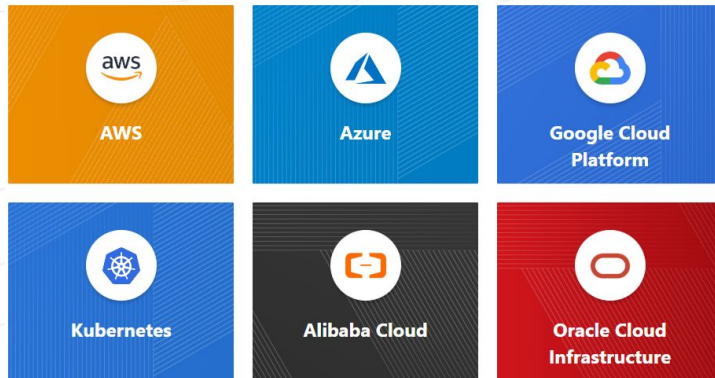
<https://developer.hashicorp.com/terraform/language/state>

Terraform – providers

Połączenie między terraform a API danego dostawcy usług (np. chmura)
Niezbędne aby tworzyć / zarządzać zasobami

<https://developer.hashicorp.com/terraform/language/providers>

Różne „tier’y” providerów Official, Verified, Community



<https://registry.terraform.io/browse/providers>



Terraform – resource

Serwisy / zasoby dostępne do utworzenia u danego dostawcy

W przypadku chmury to np. subnet, serwer, baza danych

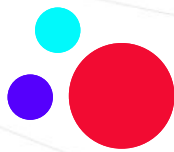
Każdy ma swoje parametry w zależności od prawdziwego zasobu

Serwer AWS

```
resource "aws_instance" "ec2" {  
  ami                = data.aws_ami.amazonlinux2.id  
  instance_type      = "t2.micro"  
  vpc_security_group_ids = [aws_security_group.sg.id]  
  subnet_id          = values(aws_subnet.private)[0].id  
  iam_instance_profile = var.iam_instance_profile  
  tags = merge(  
    var.common_tags,  
    {  
      Name = "${var.vpc_name}-ec2"  
    }  
  )  
}
```

Serwer Azure

```
resource "azurerm_windows_virtual_machine" "vm_example" {  
  name                = "vm-example"  
  location             = "westeurope"  
  resource_group_name = "rg-example"  
  network_interface_ids = ["sample_id"]  
  size                = "Standard_DS1_v2"  
  enable_automatic_updates = true  
  os_disk {  
    caching              = "ReadWrite"  
    storage_account_type = "Premium_LRS"  
  }  
}
```



Terraform - data sources

Umożliwiają Terraform korzystanie z informacji zdefiniowanych poza Terraform, zdefiniowanych przez inną oddzielną konfigurację Terraform lub zmodyfikowanych przez funkcje.

Przykład:

```
data "aws_ami" "amazonlinux2" {
  most_recent = true
  filter {
    name     = "name"
    values   = ["amzn2-ami-hvm-*-gp2"]
  }
  filter {
    name     = "virtualization-type"
    values   = ["hvm"]
  }
  filter {
    name     = "architecture"
    values   = ["x86_64"]
  }
  owners    = ["amazon"]
}
```

```
data "template_file" "env_vars" {
  template = file("env_vars.json")
}
```

```
data "aws_iam_policy_document" "cloudwatch-policy" {
  statement {
    actions = [
      "cloudwatch:PutMetricData",
      "cloudwatch:GetMetricData",
      "cloudwatch:ListMetrics",
      "cloudwatch:PutMetricAlarm",
      "cloudwatch:EnableAlarmActions",
      "cloudwatch:Describe*"
    ]
    resources = ["*"]
  }
}
```




Terraform – składnia

Słowa kluczowe:

- provider
- resource
- data
- variable
- terraform
- module

Argumenty np. `instance_type = "t3.micro"`

Bloki (`network_interface { ... }`)

Komentarze:

`# jednolinijkowe`

`/* i */ wieloliniowe`

```
load_balancer {  
  target_group_arn = aws_lb_target_group.app_tg.arn  
  container_name   = "${var.vpc_name}-container"  
  container_port   = 5000  
}  
  
depends_on = [aws_lb_listener.app_lb_listener]  
}
```

<https://developer.hashicorp.com/terraform/language/syntax/configuration>



Terraform – konfiguracja providera (AWS)

- Potrzebne konto AWS i dostęp do API
- Konfiguracja uwierzytelniania:

Environment variables

```
provider "aws" {}
```

```
$ export AWS_ACCESS_KEY_ID="anaccesskey"  
$ export AWS_SECRET_ACCESS_KEY="asecretkey"  
$ export AWS_REGION="us-west-2"  
$ terraform plan
```

```
provider "aws" {  
  shared_config_files      = ["/Users/tf_user/.aws/conf"]  
  shared_credentials_files = ["/Users/tf_user/.aws/creds"]  
  profile                  = "customprofile"  
}
```

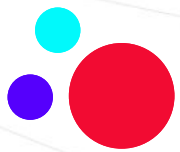
<https://registry.terraform.io/providers/hashicorp/aws/latest/docs>



Terraform – provider konfiguracja cd.

- Konfiguracja wersji providera w kodzie
- Konfiguracja wersji terraform w kodzie

```
1 terraform {  
2     required_version = ">= 1.2.1"  
3  
4     required_providers {  
5         aws = {  
6             source = "hashicorp/aws"  
7             version = ">= 3.34"  
8         }  
9     }  
10 }  
11  
12 provider "aws" {  
13     profile = "infoshare"  
14     region = "eu-west-1"  
15 }
```



Terraform – inicjalizacja providera

Terraform automatycznie pobiera potrzebne moduły i biblioteki providerów
Zgodnie z konfiguracją, którą mamy w bloku provider { }

```
Initializing modules...

Initializing the backend...

Initializing provider plugins...
- Finding hashicorp/aws versions matching ">= 3.63.0, 4.0.0"...
- Finding latest version of hashicorp/local...
- Installing hashicorp/aws v4.0.0...
- Installed hashicorp/aws v4.0.0 (self-signed, key ID 34365D9472D7468F)
- Installing hashicorp/local v2.2.2...
- Installed hashicorp/local v2.2.2 (self-signed, key ID 34365D9472D7468F)

Partner and community providers are signed by their developers.
If you'd like to know more about provider signing, you can read about it here:
https://www.terraform.io/docs/plugins/signing.html

Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!
```



Terraform CLI – podstawowe komendy

init – pobranie providera i ewentualnie modułów. Inicjalizacja backendu.

validate – sprawdzanie składni

plan – terraform porównuje zapisany stan (terraform.tfstate) z zapisaną konfiguracją w kodzie i wyświetla przewidywane zmiany

apply – terraform aplikuje zmiany przedstawione przez terraform plan

fmt – “upiększanie kodu” czyli terraform formatuje pliki z kodem

destroy – usuwanie wszystkie obiektów zarządzanych przez kod terraform

import – importowanie to zasobów utworzonych ręcznie do terraform

state [show | list] – pokazuje atrybuty danego zasobu lub wyświetla wszystkie zasoby

<https://developer.hashicorp.com/terraform/cli/commands>



Input variables – przekazywanie wartości

- Jeżeli zmienna nie ma wartości terraform zapyta przy apply
- Możemy zdefiniować wartości:
 - przez wartość domyślną
 - interaktywnie przy “terraform apply”
 - z pliku .tfvars

```
#terraform.tfvars
image_id = "ami-abc123"
availability_zone_names = [
    "us-east-1a",
    "us-west-1c",
]
```

<https://developer.hashicorp.com/terraform/language/values/variables>



Terraform - typy

- string ciąg znaków

```
variable "vpc_name" {  
  type      = string  
  description = "Name for the VPC"  
}
```

- number liczba

```
variable "rds_port" {  
  type = number  
  description = "RDS port that database listen on"  
}
```

- bool prawda / fałsz

```
variable "enabled" {  
  type = bool  
  description = "If true resource will be created"  
}
```

- list(<TYPE>) (lub tuple) lista (np. ["us-west-1a", "us-west-1c"])

```
variable "availability_zones" {  
  type = list(string)  
  description = "List of Availability Zones"  
  default = [ "eu-west-1a", "eu-west-1b" ]  
}
```

- map(<TYPE>) (lub object) grupa par klucz/wartość (np. { Name = "Tom", age = 52 })

```
variable "tags" {  
  type = map(any)  
  default = {  
    Name      = "IAC_DOR5_Maciej"  
    training  = "DOR5"  
    terraform = "true"  
  }  
}
```




Terraform – output

Przekazywanie informacji do dalszego użytku w terraformie

Podobne do wartości zwracanych np. przez funkcje w językach programowania

```
output "instance_ip_addr" {  
  value = aws_instance.server.private_ip  
}
```

<https://developer.hashicorp.com/terraform/language/values/outputs>



Terraform – locals

„Zmienna” lokalna – wartość, którą możemy używać wielokrotnie, bez konieczności powtarzania wyrażenia

```
locals {  
  service_name = "forum"  
  owner       = "Community Team"  
}  
  
locals {  
  # Ids for multiple sets of EC2 instances, merged together  
  instance_ids = concat(aws_instance.blue.*.id, aws_instance.green.*.id)  
}  
  
locals {  
  # Common tags to be assigned to all resources  
  common_tags = {  
    Service = local.service_name  
    Owner   = local.owner  
  }  
}
```

<https://developer.hashicorp.com/terraform/language/values/locals>



Terraform – modules

Fragmenty kodu (np. grupa resource'ów), które mogą być wielokrotnie używane, bez powielania kodu

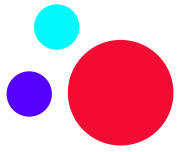
Wywoływanie używając bloku module {}

```
module "module_example" {  
  source = "../modules/awesome_instance_module"  
  
  ami      = var.ami  
  name     = "from-module"  
}
```

Nie mamy dostępu do zmiennych/resource'ów poza modułem

- Input variables – parametry modułu
- Resources
- Outputs – informacje, które chcemy użyć dalej, poza modułem

<https://developer.hashicorp.com/terraform/language/modules>



Struktura plików – przykład

```
part-2\terraform
├── .terraform
├── env_vars
├── modules
├── src
├── .terraform.lock.hcl
├── alb.tf
├── asg.tf
├── data.tf
├── main.tf
├── README.md
├── terraform.tfstate
├── variables.tf
├── versions.tf
```





THANK YOU FOR YOUR ATTENTION

infoShareAcademy.com