

Terraform Infrastructure as Code

infoShare Academy



HELLO

Karol Kołodziejczyk

DevOps engineer @ 7N
Azure, Azure DevOps, Terraform, Data engineering

- Organizacja
- Trochę historii
- Infrastructure as code – a po co to komu?
 - Podstawowe zasady i założenia
 - Co vs. Jak?
 - Benefity i wyzwania
 - Narzędzia
- Terraform – a po co to komu?
 - Co to i dlaczego?
 - Główne założenia i budowa
 - CLI i Workflow
- Terraform – składnia
 - Składnia w szczegółach
 - Praca z kodem i dokumentacja



01. **Trochę historii**

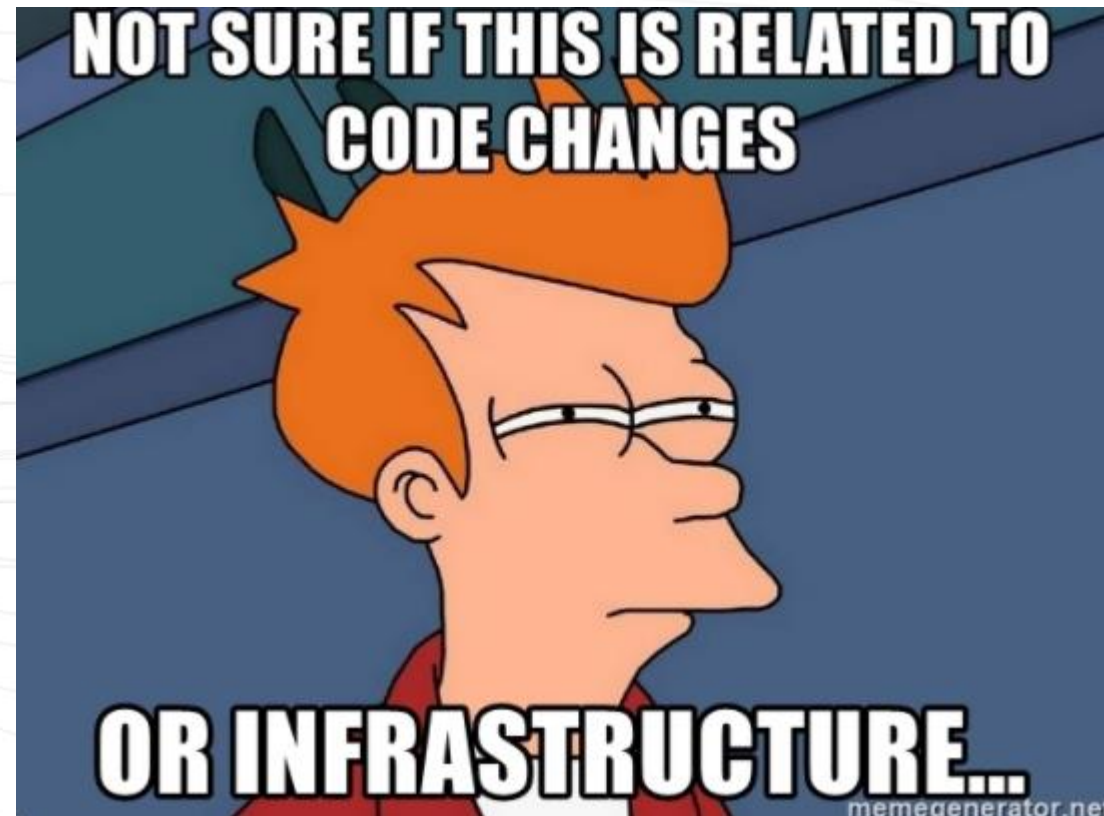
- Baremetal
 - Wysokie koszty (zakup, utrzymanie, administracja, licencje)
 - Trudne w skalowaniu, migracji i administracji
 - Ryzyko
- Wirtualizacja
 - Lepsza dystrybucja zasobów
 - Nadal pozostaje większość problemów z baremetal
- Chmura
 - „Współdzielenie” zasobów – pay-as-you-go
 - Scentralizowane zarządzanie i koszty (optymalizacja)
 - Zasoby (infrastruktura, platforma) jako serwis

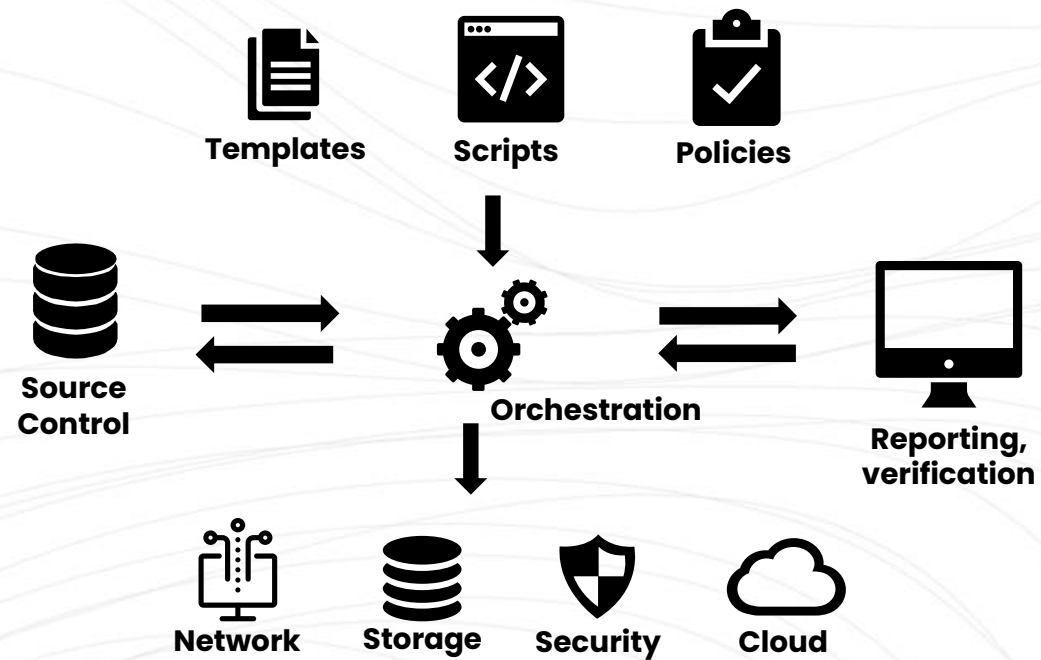
02. **Infrastructure as Code**

A po co to komu?



- Tworzenie i zarządzanie infrastrukturą za pomocą kodu
- Infrastruktura traktowana jak software
- Opis komponentów i zależności zamiast fizycznej konfiguracji hardware'u
- Hardware i wirtualizacja nadal istnieje
- Rozwiązuje realny problem





- Idempotentność – zawsze ten sam stan
- Powtarzalny proces – powielanie środowisk (infrastruktury)
- Samodokumentujące
- Ułatwienie częstych zmian
- Możliwość ciągłego ulepszania
- Szybkie przywracanie po awariach
- Wszystko trzymane w kodzie i kontroli wersji – brak zmian „manualnych”

Immutable vs. Mutable

Immutable infrastructure

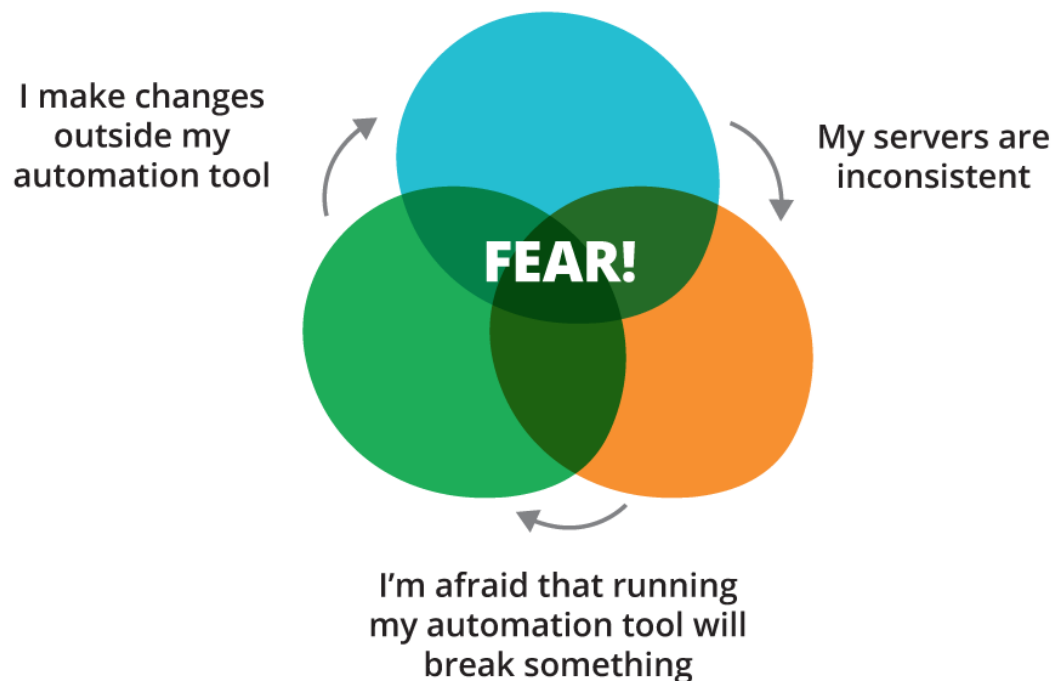
- Niezmienna po wdrożeniu
- Każda nowa zmiana oznacza wdrożenie i nową wersję
- Zmiany przewidywalne i niezawodne
- Zredukowany „configuration drift”

Mutable infrastructure

- Nieśledzone zmiany
- Zmiany „manualne”, bez wdrożenia
- Łatwiejsze i szybsze zmiany
- Podatna na błędy
- Podatna na „configuration drift”

Unikaj Automatyzacyjnej Spirali Strachu!

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



Co vs. Jak?

Co – podejście deklaratywne

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

Jak – podejście imperatywne

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

Co vs. Jak?

Co – podejście deklaratywne

Terraform

```
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"  
  }  
}
```

Jak – podejście imperatywne

Bash

```
for VM in ${VMS[@]}; do  
  echo "Creating VM $VM"  
  
  az vm create -n $VM -g MyResourceGroup --image UbuntuLTS  
done
```

Pros and Cons

Benefity

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

Ryzyka

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

Narzędzia

Zarządzanie infrastrukturą

- Terraform
- Cloud Formation (AWS)
- Azure Resource Manager
- Google Cloud Deployment manager
- Pulumi



Zarządzanie konfiguracją

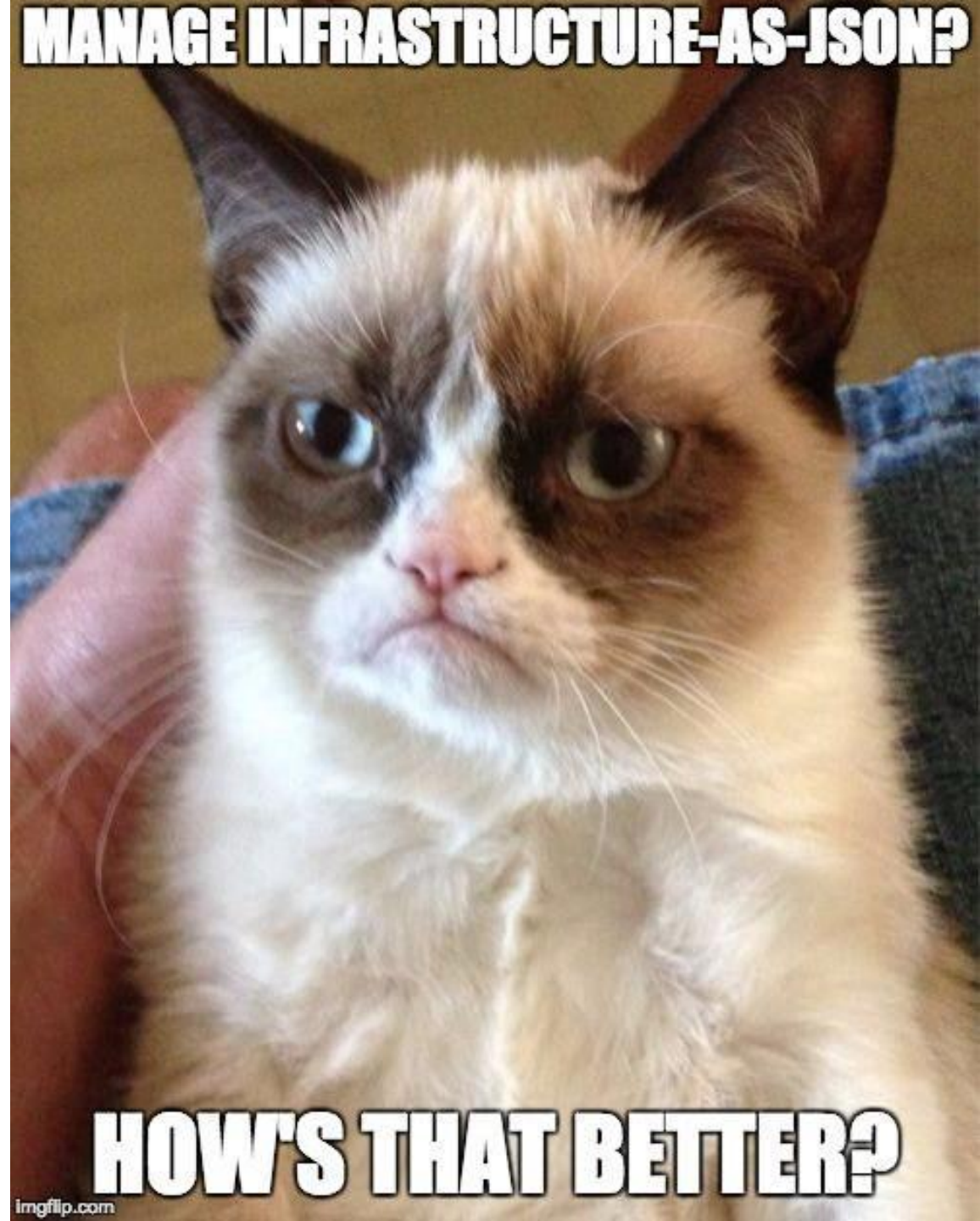
- Ansible
- Chef
- Puppet
- Saltstack



03. Terraform

A po co to komu?

MANAGE INFRASTRUCTURE-AS-JSON?



HOW'S THAT BETTER?



Terraform – intro

- HCL – język deklaratywny
- Łatwy do czytania – samodokumentujący
- Niezależny od wykorzystywanej chmury
- Łatwy do przeniesienia na innych dostawców chmury (providers)
- Prosty w uruchomieniu (kilka komend)
- Duże wsparcie społeczności

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



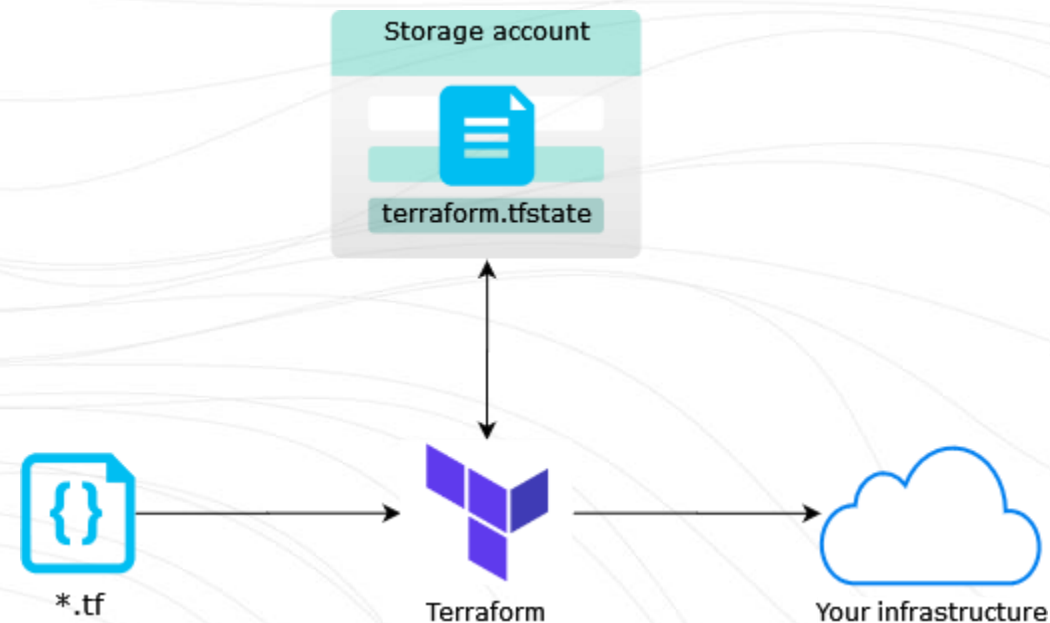
Terraform – pliki

- Rozszerzenie *.tf
- Możliwe konfiguracja w jednym lub wielu plikach
- Praktyka:
 - main.tf
 - outputs.tf
 - variables.tf
 - provider.tf

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

Terraform - state

- *.tfstate
- Plik JSON zarządzany przez Terraform
- Służy do mapowania zasobów, które definiujesz w kodzie, z działającymi serwisami na platformie (np. AWS)
- Stan jest sprawdzany przed i aktualizowany po wprowadzeniu zmian
- Pozwala na dodawanie/usuwanie zasobów
- Zawiera dane wrażliwe – powinien być traktowany jak sekret
- Zwykle przechowywany na zewnętrznym Storage'u typu AWS S3, Azure Storage Account



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



Terraform – providers

- Połączenie między Terraformem a API danego dostawcy/usługi (cloud providera)
- Niezbędne, żeby tworzyć/zarządzać zasobami

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

- Różne „tier’y” providerów – Official, Verified, Community
- AWS, Azure, Kubernetes, CloudFlare i wiele innych

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



Terraform – resources

- Serwisy/zasoby dostępne u danego dostawcy chmury (providera)
- Mogą być jakimkolwiek serwisem u danego providera – VM, VNET, DNS record
- Każdy ma swoje atrybuty/parametry zależne od prawdziwego zasobu
- Data sources

<https://www.terraform.io/language/resources/syntax>

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

<https://www.terraform.io/language/data-sources>



Terraform – składnia podstawy

- Jak zmodyfikowany JSON
- Słowa kluczone (provider, resource, data, variable itp.)
- Argumenty (image_id = "abc123")
- Bloki (network_interface { ... })
- Komentarze:
 - # - jednolinijkowe
 - /* i */ - wieloliniowe

<https://www.terraform.io/language/syntax/configuration>



Terraform – provider konfiguracja 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
```

- Credential files

```
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 AWS

- Konfiguracja samej wersji providera w kodzie

```
terraform {  
  required_providers {  
    aws = {  
      source  = "hashicorp/aws"  
      version = "~> 3.0"  
    }  
  }  
}
```

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



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
- validate
- plan
- apply

- fmt
- destroy
- import
- state

<https://learn.hashicorp.com/tutorials/terraform/init?in=terraform/cli>

<https://learn.hashicorp.com/tutorials/terraform/infrastructure-as-code?in=terraform/aws-get-started>



Terraform – ćwiczenie

- Konta
- WoW
- Konfiguracja środowiska
- Logowanie do AWS
- Utworzyć plik z providerem i zainicjalizować



01. Terraform – składnia





01. Terraform – variables





Input variables – intro

- Definiujemy w bloku variable {}
- Zwykle w osobnym pliku: variables.tf
- Zmienna może posiadać: nazwę, typ, opis, wartość domyślną, a nawet walidację

```
variable "image_id" {  
  type      = string  
  description = "The id of the machine image (AMI) to use for the server."  
  
  validation {  
    condition     = length(var.image_id) > 4 && substr(var.image_id, 0, 4) == "ami-"  
    error_message = "The image_id value must be a valid AMI id, starting with \"ami-\"."  
  }  
}
```

<https://www.terraform.io/language/values/variables>



Input variables – przekazywanie wartości

- Jeżeli zmienna nie ma wartości – terraform zapyta przy apply
- Możemy zdefiniować wartości:
 - Przez wartość domyślną
 - Przy terraform apply
 - Z pliku *.tfvars (np. różne pliki dla różnych środowisk)
- terraform apply -var-file=„terraform.tfvars”

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

<https://www.terraform.io/language/values/variables>



Input - typy

- string - ciąg znaków
- number - liczba
- bool - prawda / fałsz
- list(<TYPE>) (lub tuple) - lista (np. ["us-west-1a", "us-west-1c"])
- map(<TYPE>) (lub object) - grupa par klucz/wartość (np. {name = "Mabel", age = 52})

<https://www.terraform.io/language/expressions/types>



Input - przykład

```
variable "image_id" {  
  type = string  
}  
  
variable "availability_zone_names" {  
  type      = list(string)  
  default   = ["us-west-1a"]  
}  
  
resource "aws_instance" "example" {  
  instance_type = "t2.micro"  
  ami           = var.image_id  
}
```




01. Terraform – output





Output values

- 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://www.terraform.io/language/values/outputs>



01. Terraform – locals





Local values

- „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://www.terraform.io/language/values/locals>



01. Terraform – modules





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://www.terraform.io/language/modules>

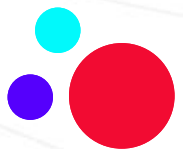


Modules - registry

- „Biblioteka” gotowych modułów z popularnymi usługami
- Tworzone i rozwijane przez społeczność lub providerów

<https://registry.terraform.io/namespaces/terraform-aws-modules>

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



Struktura plików – przykład

```

└─ envs
  └─ dev
    ├── core
    ├── databricks
    ├── gateway
    └─ network
      ├── .terraform
      ├── .terraform.lock.hcl
      ├── main.tf
      ├── outputs.tf
      ├── plan
      ├── operational
      ├── vms
      ├── production
      ├── uat
      └─ modules
        ├── databricks
        ├── datafactory
        ├── gateway
        ├── network
        ├── operational
        ├── ops_storage
        ├── services
        └─ vm
```

<https://www.terraform-best-practices.com/>

<https://github.com/shuaibiyy/awesome-terraform>



01. Terraform – kodowanie!



THANK YOU FOR YOUR ATTENTION

infoShareAcademy.com