

10 najlepszych praktyk dla Infrastructure-as-Code



Wojciech Barczyński | VPE | Spacelift.io

Co to jest OpenTofu/Terraform?

Demo:

```
variable "repository_name" {  
    type    = string  
    default = "my_app"  
}  
  
resource "github_repository" "my_repo" {  
    name          = var.repository_name  
    description  = "My Azure App repository created by OpenTofu!"  
    visibility   = "public"  
}
```

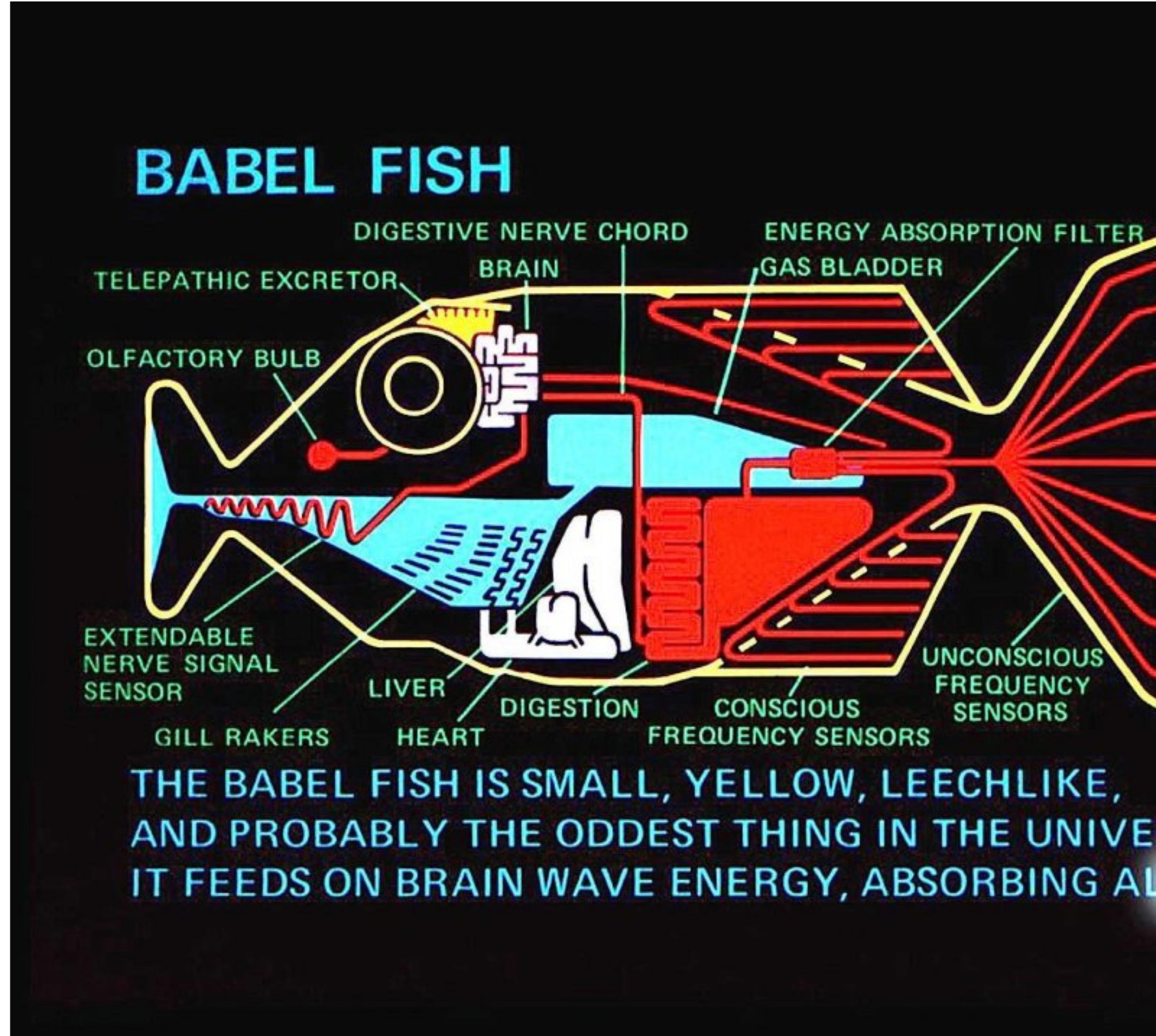
Dlaczego laC?

- Clickops 
- skrypty CLI z if/else - 
- trudno śledzić zmiany
- ciężko pracować w zespole

Dlaczego OpenTofu/Terraform FOSS?

1. `tofu plan` – co się zmieni;
2. `tofu apply` – wykonanie planu;
3. plik stanu – korzyści ([demo 2](#)):
 - shift left X;
 - polityki, lintery, skanery;
 - detekcja driftu.

Common
Language
Artifacts
Platform



Dlaczego OpenTofu/Terraform FOSS?

- Doskonała dokumentacja ([tf/ot](#)),
- Bogaty ecosystem.

Infrastructure-as-Code

Cel/gwiazda północy projektu IaC:

- zaangażowanie najszerzej grupy inżynierów;
- zaangażowanie = blisko zespołów produktowych;
- zaangażowanie = można zajrzeć do środka i kontrybuować.

Infrastructure-as-Code

Błędy na początku drogi, mszczą się później.

1/10: Remote state

Implementacje ze wsparciem dla blokad:

- AWS S3 + DynamoDB;
- Azure storage ([docs](#));
- Google Cloud Storage ([docs](#)).

koniecznie włącz szyfrowanie

1/10: Remote state

Nie zapomnij o:

- `.gitignore`
- `.dockerignore`

2/10: Struktura projektu

Mono-repo (per env/app):

```
infrastructure.git/
|- modules/
|   |- network
|   \- service-a/
|
|- production/
|   \- service-a/
|
...
```

2/10: Struktura projektu

Close to the application:

```
my_app.git/
|- app/
|- infra/
|   \- ...tf
|
... .
```

2/10: Struktura projektu

- Copy&paste jest najłatwiejszym długiem do spłacenia;
- Nie musisz od pierwszego dnia mieć własne moduły;
- Wykorzystuj dostępne moduły z community;
- Jak z bibliotekami, często lepiej później niż wcześniej.

3/10: Wiele mniejszych stack/stanów

- Najmniejszy możliwy, na przykład, per komponent;
- Unikamy czytania wartości z innych stanów;
- Bardzo łatwo o rigid infrastructure .

3/10: Rigid infrastructure?

- Wymagane wysokie uprawnienia do zarządzania,
- Twarde zależności między stanami,
- Odczytywanie stanów innych stacków 
- Zbyt wcześnie i za bardzo opinionated moduły,
- Cykliczne zależności.

3/10: Rozwiążanie

- **data** - odczytywanie stanu zasobów z chmury
- **key/value store** (Azure Config / AWS SSM / HC Consul);
- Spacelift ([TACOS](#)): **context** oraz **stack dependencies**.

4/10: Konwencje nazewnictwa

1. Używaj podkreśleń(_) jako seperatorów w nazwach.
2. Pisz nazwy małymi literami.
3. Nie dodawaj typu zasobu do nazw zasobów.
4. Zawsze używaj opisowych nazw dla zmiennych wejściowych (`variables`) i wyjściowych (`outputs`).
Opisz je (`description`).

4/10: Konwencje nazewnictwa

Więcej na:

- [Developer Terraform Docs](#);
- [Cloudposse best practices](#);
- Według: [AWS](#), [Azure](#), [GCP](#);
- [Spacelift blog](#);
- np., [terraform-null-label](#).

5/10: Pinning wersji dla wszystkiego

```
terraform {  
    required_version = ">= 0.12.26"  
  
    backend "s3" {}  
}
```

- Providers – minimum version
- Modules i wersje OpenTofu/Terraform - exact

6/10: Narzędzia

1. **tofu plan** - daje nam możliwość sprawdzenia zmian przed ich zaaplikowaniem;
2. shift-left X, gdzie X - najlepsze praktyki, bezpieczeństwo, polityki.

6/10: Narzędzia

Warto dodać do CI/CD albo git .pre-commit:

- Lintery: `tofu fmt & tflint`;
- Bezpieczeństwo: `tfsec`, `kics` i `checkov`;
- Estymacja kosztów: `tf-cost-estimation` & `infracost`.

6/10: Narzędzia

Warto dodać do CI/CD albo git .pre-commit:

- Polityki: [conftest](#) & [OpenPolicyAgent](#):
 - Ostrzeż, że kasujemy lub odtwarzamy,
 - Etykiety na zasobach wymagane.

TACOS takie jak [Spacelift](#), Env0, Scalr mają gotowe integracje

7/10: YOLO from laptop



- Uff, luckily it was my test env
- ...
- ...
- 

7/10: Continuous Deployment

1. Warto zacząć od Continuous Integration, nawet kiedy odpalasz OpenTofu z laptopa;
2. Kiedy demokratyzujemy dostęp i skalujemy, Continuous Deployment jest niezbędne.

7/10: Continuous Deployment

- generic CI/CD;
- TACOS: Atlantis, Env0, Scalr, Spacelift.io.

8/10: Demokratyzacja

To tylko kwestia czasu:

- Zespoły produktowe rosną szybko;
- Trudno zatrudnić Platform czy System Cloud/DevOps inżynierów;
- Budget?
- CTO dlaczego ktoś czeka na X?;
- Product teams!

8/10: Demokratyzacja

Jak zacząć:

1. Zaczynamy od jednej osoby w zespole, np. infra/IaC champion;
2. Opcjonalnie - przydzielmy jedną osobę z zespołu DevOps/platformy (tymczasowo).

8/10: Demokratyzacja

Będzie łatwiej:

1. Jeśli nie mamy hard dependencies między stanami;
2. Nie potrzeba najwyższych uprawnień do postawienia zasobów chmurowych dla serwisów.

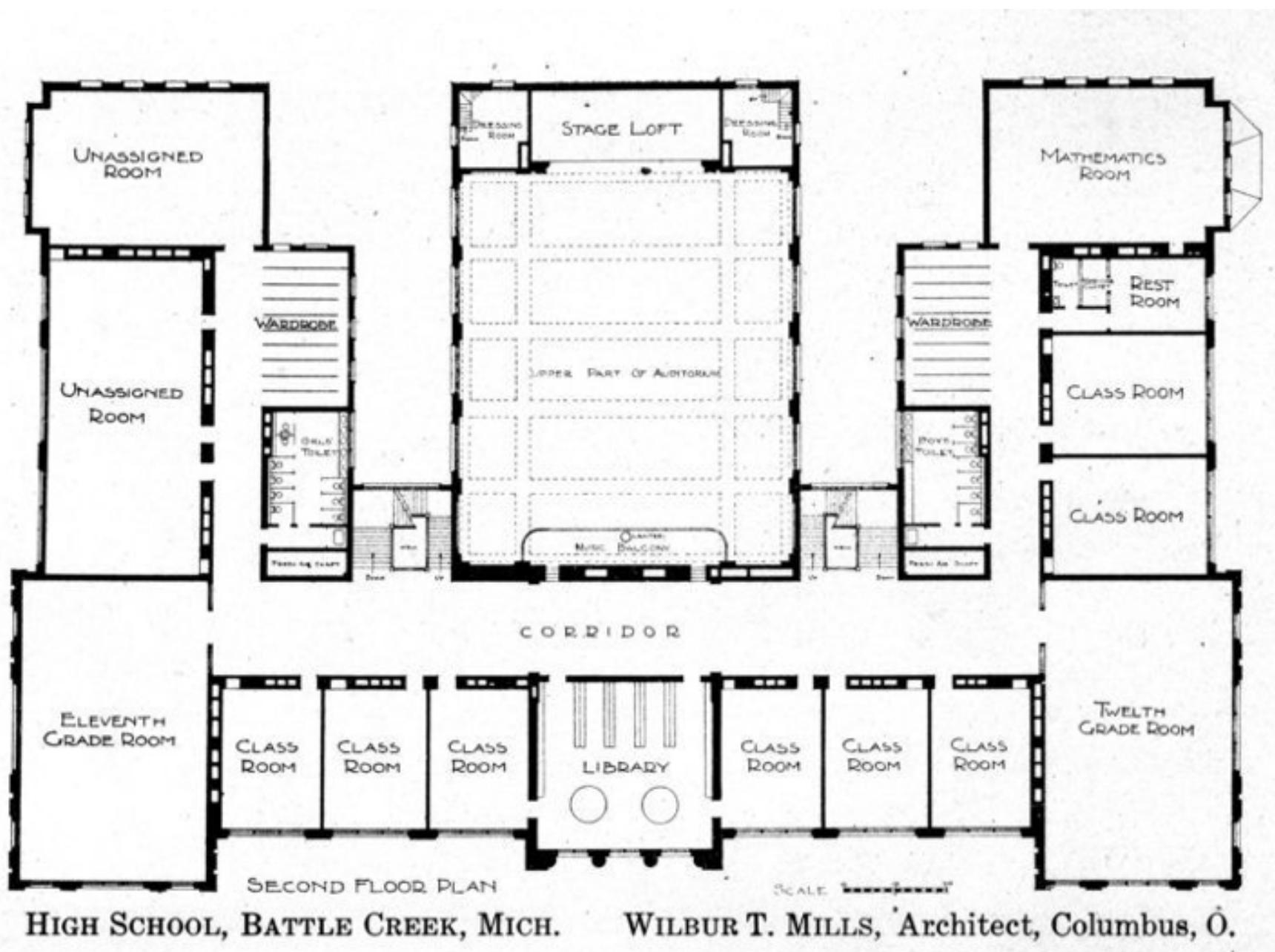
8/10: Demokratyzacja

Będzie łatwiej:

- Dobrze mieć CD na generycznym CI lub TACOS;
- Praca oparta o Pull Request'ach;
- W pierwszej iteracji, może być copy&paste planu w PR;
- Małe/generyczne wspólne moduły.

8/10: Demokratyzacja

- Zacząć od wspólnych konwencji... potem narzędzia/moduły



8/10: Demokratyzacja

TACOS - prosty workflow z PRem:

- Otwórz PR;
- **Spacelift** uruchomi:
 - `plan` i zwaliduje zmiany,
 - `workflow customization` (np., `tfsec`),
 - run polices against the changes,
 - trigger approval process.

8/10: Demokratyzacja

TACOS - prosty workflow z PRem:

- PR merged;
- [Spacelift](#) uruchomi:
 - co powyżej lub specificzna konfiguracja,
 - apply the changes,
 - uruchomi downstream stacks (dependencies).

9/10: Skalowanie

- Template-y dla nowych serwisów;
- Współdzielenie najlepszych praktyk i polityk;
- Więcej wspólnych modułów;
- Tymczasowe środowiska;
- Drift-detection (managed i unmanaged).

9/10: Skalowanie

Template-y dla nowych serwisów:

- Konwencje;
- No-black magic;
- Lekka warstwa HCL, YAML, czy JSON albo CRD;
- Self-service;
- Oczywiście dev musi mieć możliwość uruchomienia dev lokalnie;

9/10: Skalowanie

Drift detection:

- Zmiany w konsoli na chwilę;
- Upewnienie się, że rzadko dotykany moduł działa;
- Sprzątanie po eksperymentach;
- Śledzenie postępów w projektach clickops do IaC.

9/10: Skalowanie

Mono repozytorium czy repozytorium per komponent:

- To zależy od firmy.

10/10: Metryki

- Pierwsze sukcesy łatwo odczuć,
- Później trudniej mierzyć postępy,
- Co więc mierzyć?

High delivery performance

- Lead Time
- Deployment frequency
- Mean time to Recovery
- Change Fail Percent



10/10: Metryki

Deployment frequency:

1. Łatwe do uchwycenia przy gitops;
2. Zakończenie pipeline lub po prostu merge do
brancha produkcyjnego.

10/10: Metryki

Lead time:

1. Od utworzenia Pull Request jako DRAFT;
2. Do wprowadzenia na produkcję (merge do master/prod).

10/10: Metryki

Obecnie używam:

- Metabase + dbt + python experter z Github;
- Wcześniej Google Colab Notebook na Google BigQuery.

10/10: Metryki

Później:

- Ilość kontrybutorów spoza Platform/Infra/DevOps team;
- Pozostałe metryki **DORA**: MTTR i failure-rate.

Podsumowanie

- Celem jest zaangażowanie najszerszej grupy inżynierów;
- Opentofu i Terraform jest najbardziej solidnym narzędziem;
- Większość praktyk, stosuje się do od AWS CloudFormation, CDK, CDK-TF, po Kubernetes.

Kubernetes and OpenTofu/Terraform

Często zadawane pytanie:

- OpenTofu i Terraform do postawienia K8S + ArgoCD, Crossplane, lub flux (last mile);
- ArgoCD manages apps on Kubernetes;
- Spacelift operator - last mile cloud resources.



Pytania

github.com/wojciech12/talks

ps. Hiring Frontend and Backend engineers:
spacelift.io/careers

Backup Slides

Rabbit holes everywhere...

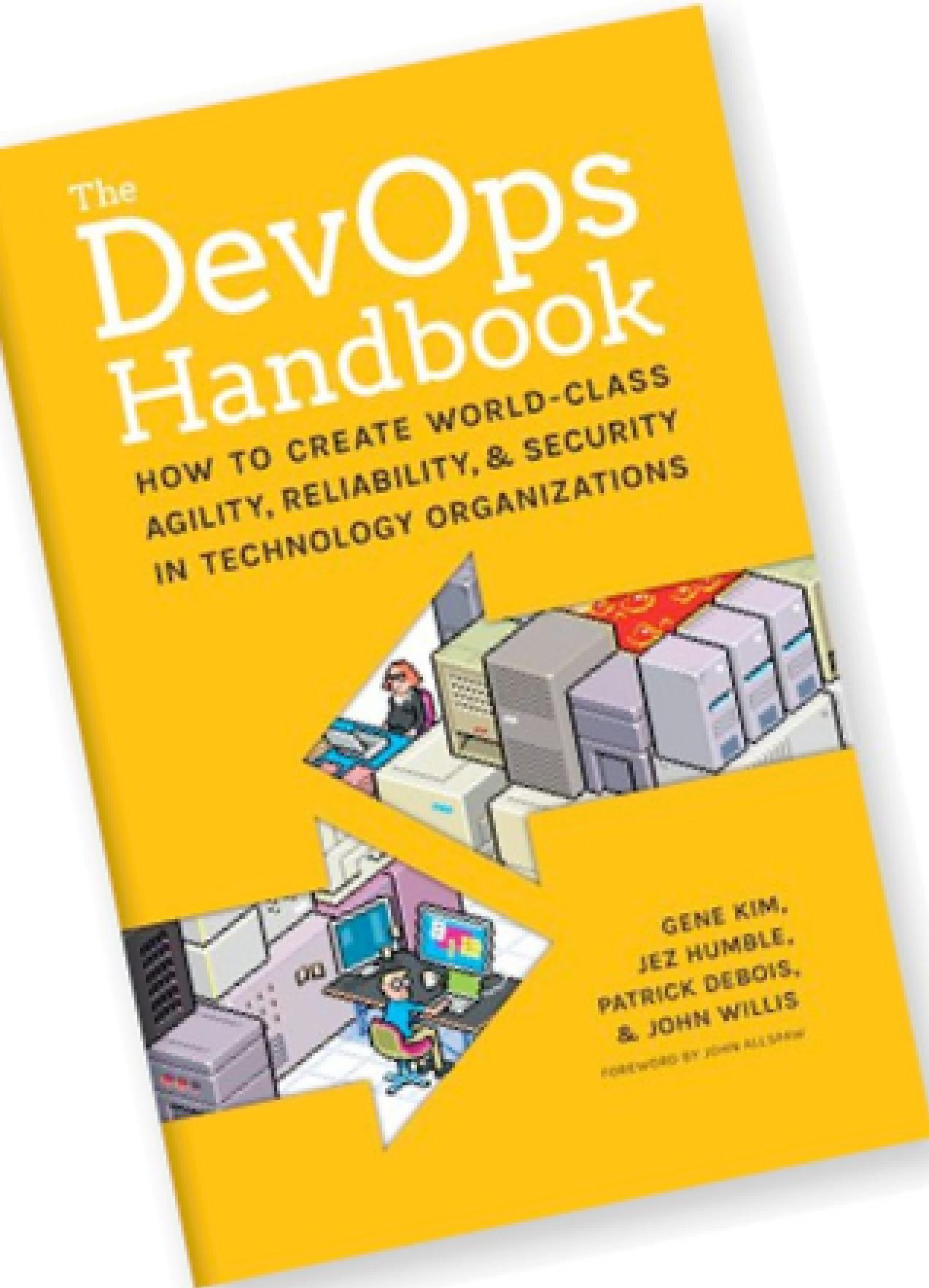
Approach:

- The iteration, decision, and deliver
- As soon as possible to get into the cycle Patch Patch Patch

Alternative take:

- Tracer bullet development,
- Lean v1/v2.

Recommended read

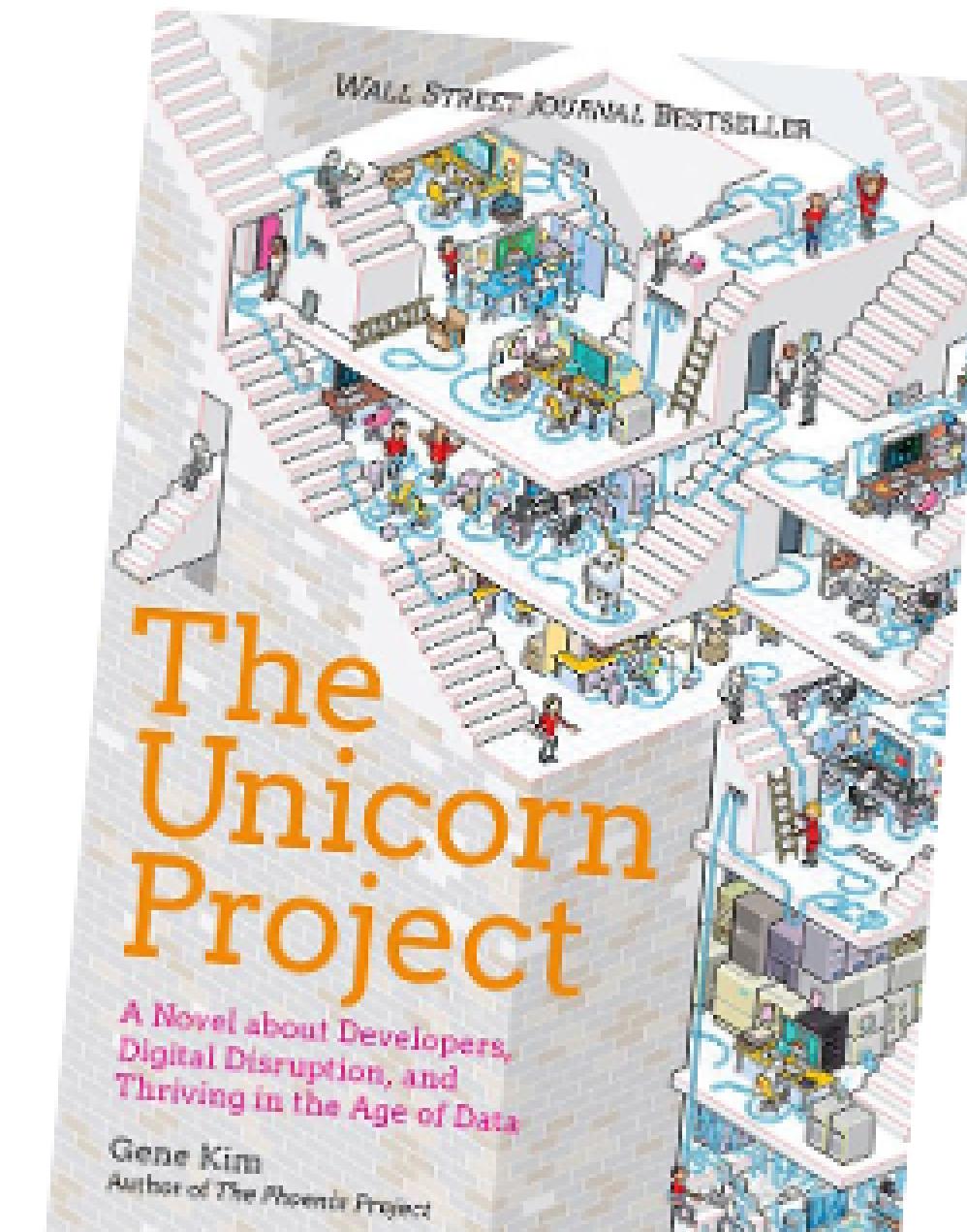


1

Concrete, do A, do B,
because C
(DevOps here as culture ☺)

2

The first 4 chapters,
the rest if you have time.



3

A story,
not as straight forward
as (1)



OODA

