UNIWERSYTETY WSB MERITO

Programowanie aplikacji Java

Maciej Gowin

Zjazd 10 - dzień 2

Testy integracyjne

Testy integracyjne służą weryfikacji połączeń oraz interakcji pomiędzy poszczególnymi modułami lub innymi systemami.

W przypadku testów jednostkowych możemy sprawdzić, czy:

- zapis do bazy działa poprawnie
- metoda serwisu poprawnie przepisuje obiekty i wykonuje skomplikowaną logikę

Czego nie sprawdzamy:

- czy aplikacja w ogóle się włączy?
- czy zapytanie HTTP zakończy się poprawnym zapisem danych do bazy danych?

Na tego typu pytania odpowiadają testy integracyjne.

Testy integracyjne

Test integracyjny polega na faktycznym **uruchomieniu** aplikacji (lub jej części) oraz wykonaniu na niej testu (np. wykonanie zapytania HTTP).

Spring: testy

W przypadku użycia Springa jest możliwość uruchomienia całej aplikacji podczas wykonywania testu, włącznie z podłączeniem jej do bazy danych.

W zależności co chcemy przetestować, możemy uruchomić całość bądź skupić się jedynie na części aplikacji (np. test repozytorium).

- `@SpringBootTest`
  - uruchamia cały kontekst Spring
- `@WebMvcTest`
  - używany do testowania pojedynczej klasy @Controller
  - nie przeskanuje beanów takich jak `@Service`, `@Repository` (należy pamiętać, aby dostarczyć odpowiednie zależności)

Spring: testy

- `@DataJpaTest`
  - używany do testowania warstwy persistence (repozytoria, dao)
  - uruchomi jedynie część aplikacji w warstwie persistence (JPA)
  - uruchomi testową bazę in-memory H2 (należy dostarczyć zależność w `pom.xml`)
  - przestawi aplikację w tryb logowania SQL na konsoli

Spring: testy

Jednym z przykładów testu integracyjnego jest wykonanie zapytania HTTP do jednego z endpointów naszej aplikacji oraz sprawdzenie, czy wynik jest zgodny z oczekiwanym.

- Do wykonywania zapytań HTTP z poziomu testów będziemy używali klasy `MockMvc`.
- Do skonfigurowania `MockMvc` możemy użyć adnotacji `@AutoConfigureMockMvc`, która utworzy ziarno typu `MockMvc`.
- Wywołania na tej klasie zwracają obiekty pozwalające na wywołanie sprawdzeń na rezultatach.

Spring: testy

Przykład testu inicjalizującego kontekst Springa.

```
@SpringBootTest
@AutoConfigureMockMvc
public class ApplicationControllerTest {
    @Autowired
    private MockMvc mockMvc;

    @Test
    void shouldReturnExample() throws Exception {
        // given & when & then
        MvcResult result = mockMvc.perform(get("/example")) // actual HTTP GET invocation
                .andExpect(status().isOk())
                .andExpect(content().string("example-value"))
                .andDo(print()).andReturn();
    }
}
```

Spring: testy

Testy integracyjne mogą korzystać z:

- rzeczywistych implementacji
- mocków

Aby zamockować bean należy użyć adnotacji `@MockBean`:

```
@SpringBootTest
@AutoConfigureMockMvc
public class ApplicationControllerMockTest {
    @Autowired
    private MockMvc mockMvc;
    @MockBean
    private ExampleService exampleService;
```

Testowanie: przyklad-spring-integration-test

Przeanalizuj przykład oraz pełną inicjalizację kontekstu Springa.

Spring: zależności Maven

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>test</scope>
</dependency>
```

Warto zwrócić uwagę na *scope* zależności.

Testowanie: zadanie

Wykonaj zadania opisane na podstawie przykładu `przyklad-junit5-spring-boot`.
Szczegóły opisane w przykładzie.

Test Driven Development

W klasycznym podejściu rozwoju oprogramowania właściwy kod programu powstaje przed powstaniem testów, w tym testów jednostkowych.

Możliwe jest odwrócenie tego procesu, a podejście to jest opisywane jako Test Driven Development.

Test Driven Development

Test Driven Development (ang. TDD) to podejście do tworzenia oprogramowanie, w którym głównym nacisk kładzie się na rozwijanie testów.

Opiera się ono na tworzeniu testów przed stworzeniem właściwego kodu programu.

Test Driven Development

Cykl tworzenia opiera się na kilku krokach:

- dodaj test,
- uruchom test, który powinien zwrócić błąd,
- zaimplementuj funkcjonalność w najprostszy możliwy sposób,
- wszystkie testy powinny zakończyć się sukcesem,
- zrefaktoruj kod w razie potrzeby, upewniając się, że wszystkie testy zawracają sukces.

Test Driven Development

Oczywiście podejście TDD może być używane nie tylko przy okazji pisania testów jednostkowych, ale też testów integracyjnych.

Definiuje ono jedynie model pracy i tworzenia oprogramowania.

Behavioral Driven Development

BDD (behavior-driven development) jest rozszerzeniem podejścia TDD. Podobnie jak w przypadku TDD rozwój oprogramowania opiera się na początkowym tworzeniu testów. Główną różnicą jest samo podejście do opisu testów.

W podejściu BDD testy opisują zachowanie i oczekiwane efekty tego zachowania, w przeciwieństwie do prostych testów jednostkowych, które opisuje mały wycinek funkcjonalności, która jest przedmiotem testów.

Behavioral Driven Development

Istnieje szereg narzędzi, które pozwalają na opisywanie zachowań oraz testów przy pomocy podejścia BDD w sposób zrozumiały dla użytkowników nietechnicznych takich jak analitycy biznesu czy testerzy manualni.

Dzięki temu możemy oddzielić definicję zachowań aplikacji od samej implementacji testów oraz umożliwić definicję testów behawioralnych nie tylko na poziomie kodu.

Cucumber

Narzędzie `Cucumber` jest narzędziem wspierającym rozwój oprogramowania na podstawie koncepcji BDD. Testy definiowane są za pomocą specjalnego języka `Gherkin`.

Gherkin pozwala na definiowanie zachowań oprogramowania w sposób zrozumiały nie tylko dla programistów, ale też użytkowników. Dzięki temu opis testów może posłużyć jako definicja funkcjonalności danego oprogramowania.

W wielu przypadkach Cucumber jest używany wraz z innymi narzędziami wspierającymi testowania oprogramowania. Dodatkowo wspiera wiele języków programowania w połączeniu, z którymi może zostać użyty.

Cucumber: Gherkin

Testy definiujemy w plikach `.feature`, które opisują daną funkcjonalność. Format pliku oraz słowa kluczowe są definiowane przez język Gherkin.

```
Feature: bank account operations

    Back account operations

    Scenario: Customer wants to add money to his bank account
        Given account balance is 100
        When customer adds 50
        Then account balance is 150

    Scenario: Customer wants to withdraw money from his bank account
        ...
```

Cucumber: Gherkin

Do słów kluczowych zaliczamy:

`Feature` - każda z funkcjonalności może zawierać jeden lub więcej scenariuszy oraz opisuje daną funkcjonalność oprogramowania.
`Scenario` - opisuje konkretny test, który zostanie wykonany. Scenariusze składają się z kroków, które sprawdzają dane zdarzenie.

Oraz kroki (ang. steps):

`Given` - opisuje warunki początkowe oraz stan przed uruchomieniem testów
`When` - opisuje akcje wykonane przez użytkownika podczas testu
`Then` - opisuje rezultaty testów

Cucumber: Gherkin

Kroki budujące scenariusze mogą być rozumiane jako wywołania funkcji. Aby mechanizm Cucumber mógł wykonać dany scenariusz, należy zdefiniować sposób, w jaki krok powinien zostać przetworzony.

Definicja kroków odbywa się na poziomie kodu źródłowego testów.

Cucumber: Gherkin

```
    @Given("account balance is set to {value}")
    public void account_balance_is_set_to(double value) {
        /* ... */
    }

    @When("customer adds {value}")
    public void customer_adds(double value) {
        /* ... */
    }

    @Then("account balance is {value}")
    public void account_balance_is(double value) {
        /* ... */
    }
```

Programowanie: przyklad-cucumber

Implementacja testów przy pomocy biblioteki Cucumber. Do wykonania testów użyj standardowego kroku:

```
mvn test
```

Testy manualne

Testowanie manualne opierają się na manualnym sprawdzeniu aplikacji.

Testy te najczęściej dotyczą stron internetowych oraz aplikacji mobilnych, czyli interfejsów użytkownika. Podczas ich wykonywania implicite testowane są też rozwiązania backendowe, które są konsumowane przez aplikacje frontendowe.

Błędy pojawiające się podczas testów manualnych mogą wynikać z błędów na poziomie FE oraz BE.

Testy manualne: problemy

Wyobraźmy sobie aplikacje do logowania użytkowników.

Przy dowolnej zmianie ekranu logowania lub też systemu wspierającego zapytania logowania wymagane jest przetestowanie aplikacji w celu sprawdzenia poprawności jej działania.

Krok ten powinien być wykonany przy każdej nowej wersji aplikacji. W przypadku testów manualnych musimy wykonać powtarzalną czynność, której kroki będą identyczne w każdej iteracji rozwoju oprogramowania.

Testy automatyczne

Aby rozwiązać problem powtarzalności manualnych testów, wprowadzamy testy automatyczne.

Testy automatyczne mają na celu zautomatyzowanie czynności, które musielibyśmy wykonać przy każdorazowej weryfikacji oprogramowania.

Testy te są oparte na skryptach, które imitują czynności wykonywane przez testera manualnego i mogą zostać uruchomione w dowolnym momencie.

Testy automatyczne

Aby zdefiniować test automatyczny, w większości przypadków definiujemy scenariusz testu, czyli kroki, które test powinien wykonać.

Skrypt definiujący kroki imituje zachowanie testera manualnego oraz stara się wykonać analogiczne czynności.

Uruchomienie skryptu równoznaczne jest z wykonaniem testu.

Testy automatyczne

Istnieje szereg systemów ułatwiających wykonywanie testów automatycznych.

Dla przykładu: do tej pory używaliśmy narzędzia Cucumber do testów jednostkowych aplikacji. Równie dobrze moglibyśmy użyć tego narzędzia do stworzenia scenariuszy testów automatycznych dla istniejącej aplikacji backend. Kroki odpowiadałyby wywołaniom API zdefiniowanego przez aplikację webową.

Testy automatyczne

Testowanie systemu backendowego takiego jak serwis RESTful wydaje się stosunkowo proste. Wywołania API mogą zostać wykonane z poziomu kodu programu.

Istnieje też możliwość automatycznego testowania aplikacji webowych, w tym stron internetowych. Testy te imitują zachowanie użytkownika przechodzącego po kolejnych stronach oraz wykonującego działania na stronie.

Testy automatyczne: Selenium

Przykładem narzędzia, które wspiera automatyczne testowanie aplikacji webowych, takich jak strony internetowe, jest narządzie Selenium.

Selenium pozwala na definiowanie testów przy pomocy wielu języków, w tym Java.

Programowanie: przyklad-selenium

Przykładowa automatyzacja akcji na stronie internetowej.

Testy automatyczne: łączenie rozwiązań

Jak widać rozwiązania i narzędzia testujące łączą się w łatwy sposób. Używając poznanych bibliotek, moglibyśmy zdefiniować scenariusze testów przy pomocy Cucumber, gdzie kroki testów wykorzystywałyby Selenium do wykonywania akcji na stronie internetowej.

Testy wydajnościowe

Aplikacje backendowa uruchomiona na serwerze jest w stanie obsłużyć ograniczoną ilość zapytań.

Ograniczenia te mogą wynikać z różnych czynników takich jak:

- zasobów serwera (procesor, pamięć),
- wymogów pamięciowych aplikacji,
- złożoności obliczeniowej aplikacji,
- ilości przypisanych wątków
- czasu przetworzenia zapytania
- wydajności zależności.

Testy wydajnościowe

Testy sprawdzające zachowanie aplikacji dla określonego obciążenia to testy wydajnościowe.

Testy wydajnościowe

Testy wydajnościowe przyjmują formę eksperymentowania i sprawdzania aplikacji w różnych warunkach.

Ich analiza opera się na 3 głównych aspektach:

- zachowanie w czasie: zdolność systemu do reagowania na dane wejściowe użytkownika lub systemu w określonym czasie i w określonych warunkach,
- wykorzystanie zasobów: ilość zasobów wymaganych przez system do poprawnego działania,
- przepustowość: ograniczenie systemu w zakresie limitów wydajnościowych (np. ograniczenia w ilości zapytań).

Testy wydajnościowe: Gatling

Gatling jest narzędziem pozwalającym na łatwe tworzenie testów wydajnościowych oraz ich automatyzację.

Testy opierają się na symulacjach wywołań aplikacji oraz zapisywaniu rezultatów dla danych warunków wykonywania.

Programowanie: przyklad-gatling

Test wydajnościowy dla RESTful API.

Uruchomienie:

```
mvn gatling:test
```

Rezultaty:

```
/target/gatling
```

Materiały dodatkowe: zawartość

- Wdrażanie i hosting aplikacji
- CI/CD
- Strategie integracji kodu
- Strategie wdrażania aplikacji
- Zadanie z CI/CD
- Bazy NoSQL
- Zadanie z NoSQL

Wdrażanie aplikacji

Do tej pory zajmowaliśmy się implementacją kodu naszej aplikacji w Springu oraz uruchamialiśmy ją lokalnie, na naszych komputerach.

Załóżmy, że skończyliśmy właśnie pracę nad jej pierwszą wersją.

Naszym kolejnym zadaniem jest dostarczenie jej do użytkowników.

Rozważymy teraz istniejące opcje dostarczenia naszej aplikacji do użytkowników oraz przejdziemy przez cały proces wdrażania aplikacji.

Hosting

Udostępnienie aplikacji z naszego własnego komputera wymaga dużo wysiłku związanego z administracją serwera. O ile nie jesteśmy firmą, która chce poświęcić wiele zasobów na zbudowanie i utrzymywanie własnej serwerowni, najlepszą opcją będzie skorzystanie z usług firm zewnętrznych.

Na rynku jest wiele firm, które posiadają serwerownie złożone z wielu komputerów, które udostępniają swoich klientom moc obliczeniową potrzebną do funkcjonowania aplikacji. Udostępnianie zasobów serwera klientowi nazywamy **hostingiem**.

Istnieje wiele rodzajów hostingu, które różnią się sposobem rozliczania płatności oraz podziałem odpowiedzialności za poszczególne czynności związane z funkcjonowaniem naszej aplikacji.

Rodzaje hostingu

- Serwer dedykowany
- Hosting współdzielony
- Serwer VPS (Virtual Private Server)
- Hosting zarządzany
- Kolokacja
- Hosting w chmurze

Serwer dedykowany

Serwer dedykowany jest dostępny tylko dla nas i nasza aplikacja jest jedyną uruchomioną na nim.

W przypadku hostingu na serwerze dedykowanym mamy największą kontrolę nad serwerem, na którym uruchomiona jest nasza aplikacja. Mamy pełen dostęp administratora do wszystkich zasobów serwera.

Serwer dedykowany wymaga najwięcej wysiłku oraz wiedzy z zakresu instalacji i zarządzania nim.

Hosting współdzielony

W przypadku hostingu współdzielonego mamy bardzo ograniczone możliwości administracji serwerem, ponieważ na jednym serwerze uruchomione może być wiele aplikacji różnych klientów.

Serwery współdzielone pozwalają na lepsze wykorzystanie zasobów poprzez uruchamianie wielu aplikacji na jednej maszynie.
Dzięki temu są znacznie tańsze od serwerów dedykowanych.

Oprócz ograniczonej kontroli nad serwerem, wadą tego typu hostingu jest możliwy wzajemny wpływ aplikacji uruchomionych na tym samym serwerze.

Serwer VPS

Serwer VPS jest rozwiązaniem pomiędzy serwerem dedykowanym i hostingiem współdzielonym.

Na jednej maszynie uruchomione jest wiele aplikacji, ale każda z nich jest uruchomiona w odizolowanej przestrzeni (maszynie wirtualnej).

Dzięki temu możliwa jest kontrola nad serwerem zbliżona do serwera dedykowanego, przy jednoczesnym wykorzystaniu zasobów podobnym do hostingu współdzielonego.

Maszyna wirtualna pozwala na uruchomienie więcej niż jednego systemu operacyjnego na tym samym serwerze, jednak oznacza to również, konieczność alokacji oddzielnych zasobów dla każdego z nich.

Hosting zarządzany

W przypadku hostingu zarządzanego oprócz serwera otrzymujemy również wsparcie techniczne związane z konfiguracją sprzętu oraz oprogramowania.

Wsparcie techniczne może obejmować monitorowanie czy instalacje aktualizacji poprawiających bezpieczeństwo serwera.

W zależności od dostawcy, pakiety usług wchodzących w skład hostingu zarządzanego mogą się różnić.

Kolokacja

Kolokacja polega na wynajęciu fizycznej przestrzeni na serwery.

Dzięki temu możemy sami zdecydować, z jakiego rodzaju sprzętu chcemy korzystać.

Rola firmy zewnętrznej ogranicza się do utrzymania dostępu do energii, internetu oraz chłodzenia naszego serwera.

Hosting w chmurze

Hosting w chmurze charakteryzuje się możliwością uruchomienia aplikacji z użyciem połączonych zasobów wielu komputerów.

Dostawcy chmury dostarczają narzędzia developerskie oraz usługi takie jak np. bazy danych, które mogą zostać wdrożone bez konieczności ręcznej instalacji oraz pozwalają ograniczyć czynności związane z ich utrzymaniem.

Dzięki temu ułatwione jest budowanie i zarządzanie infrastrukturą naszej aplikacji. Pozwala to poprawić niezawodność naszej aplikacji.

Rodzaje hostingu w chmurze

- **Chmura publiczna** - zasoby chmury są współdzielone przez użytkowników
  - **IaaS** (Infrastructure as a Service) - otrzymujemy serwer wirtualny, ale jesteśmy odpowiedzialni za jego konfigurację oraz zarządzanie środowiskiem dla naszej aplikacji
  - **PaaS** (Platform as a Service) - oprócz serwera wirtualnego otrzymujemy gotowe środowisko do uruchomienia naszej aplikacji
  - **FaaS** (Function as a Service) - pozwala uruchomić nasz kod na żądanie, bez rezerwowania żadnego serwera (serverless). Płacimy tylko za zużyte przez naszą aplikację zasoby
  - **SaaS** (Software as a Service) - otrzymujemy gotowe do użycia oprogramowanie

Rodzaje hostingu w chmurze

- **Chmura prywatna** - pozwala na korzystanie z usług chmurowych na sprzęcie, który posiadamy na własność
- **Chmura hybrydowa** - nasze zasoby składają się zarówno z chmury publicznej, jak i prywatnej
- **Multicloud** - nasze zasoby składają się z usług więcej niż jednego dostawcy chmury

Podejścia do chmury

- **Cloud native** - używamy usług specyficznych dla danej chmury
  - Trudniejsza zmiana dostawcy chmury (vendor locking)
  - Lepsze wykorzystanie zasobów chmury poprzez wykorzystanie zoptymalizowanych dla niej serwisów
- **Cloud agnostic** - używamy usług, które mogą zostać wdrożone na różnych chmurach
  - Łatwiejsza zmiana dostawcy chmury
  - Niekorzystanie z zoptymalizowanych przez dostawcę usług, powoduje zazwyczaj gorszą wydajność oraz wyższy koszt rozwiązania

CI/CD

Do tej pory zaimplementowaliśmy naszą aplikację, wybraliśmy hosting i dostarczyliśmy jej pierwszą wersję do naszych użytkowników. Jako zespół pracujący nad tą aplikacją planujemy teraz dodanie nowych funkcjonalności do jej kolejnej wersji.

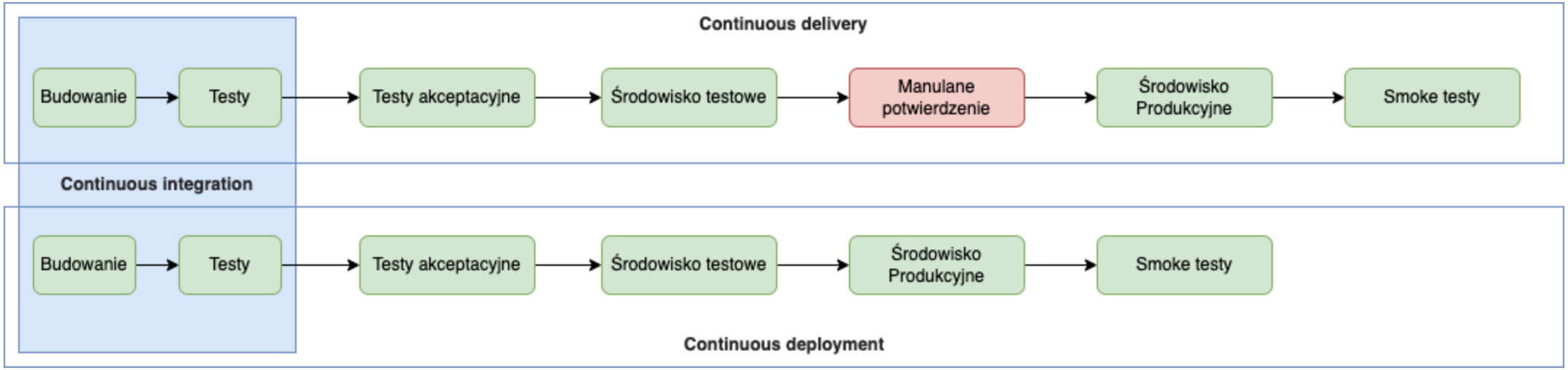
Aby zaoszczędzić nasz czas, chcemy zautomatyzować process wdrażania kolejnych wersji aplikacji. W tym celu zamierzamy stworzyć process CI/CD.

CI/CD

CI/CD oznacza:

- **CI** (Continuous Integration) - regularne budowanie i testowanie zmian w kodzie różnych członków zespołu pracujących nad aplikacją.
- **CD** (Continuous Delivery) - zmiany są automatycznie wdrażane i testowane na środowisku testowym. Przed dostarczeniem nowej wersji dla użytkownika konieczne jest manualne potwierdzenie.
- **CD** (Continuous Deployment) - w porównaniu do Continuous Delivery, po przejściu przez środowisko testowe zmiany są automatycznie dostarczane do użytkowników aplikacji.

Continuous Integration vs Continuous Delivery vs Continuous Deployment

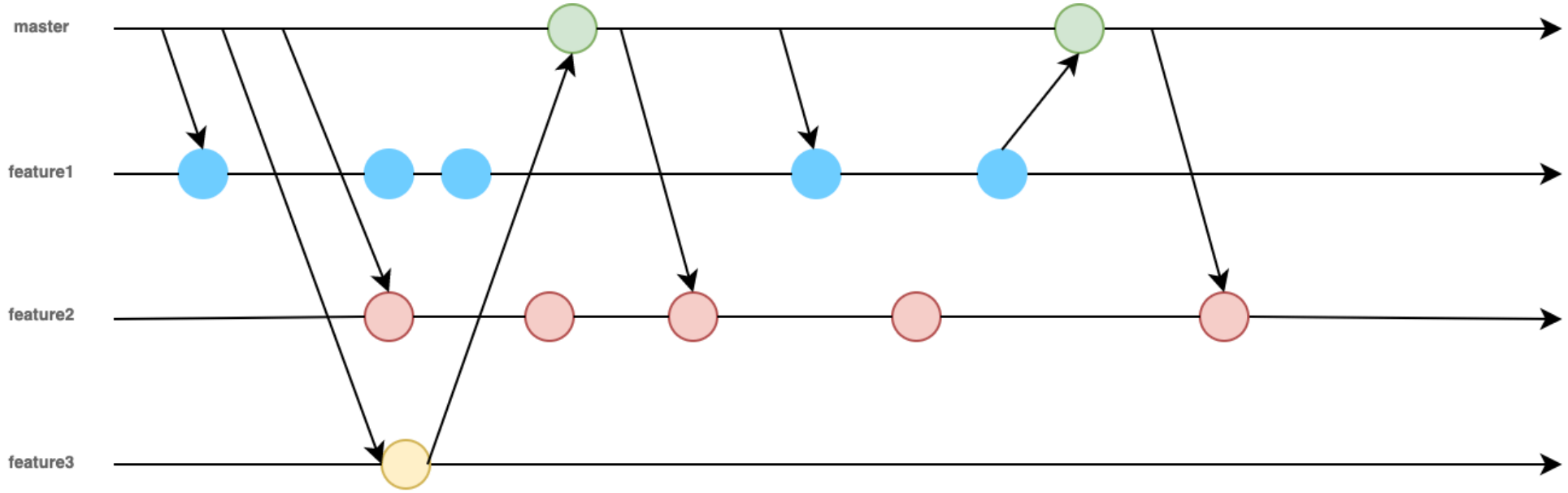


Strategie integracji kodu

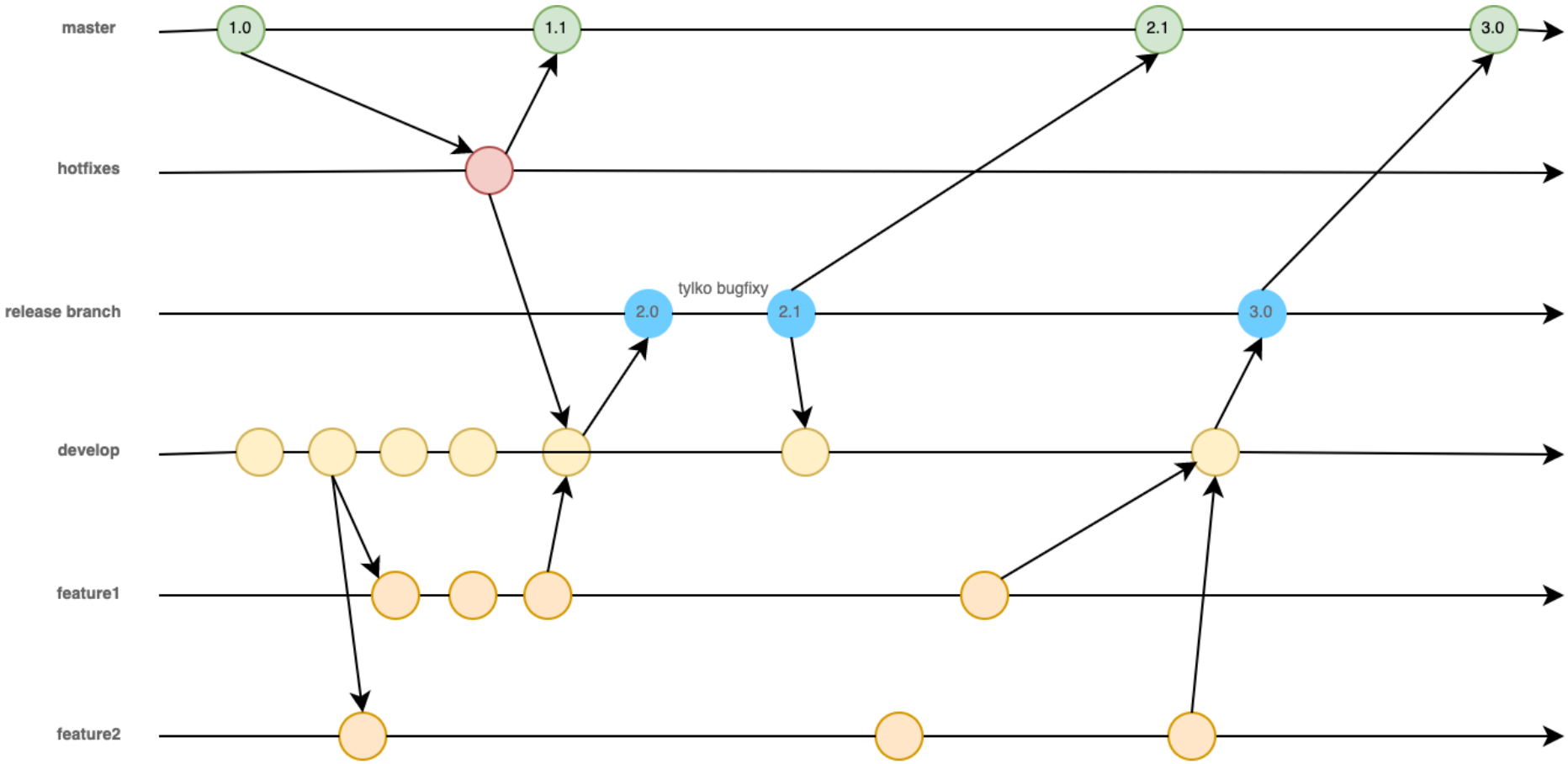
Najpopularniejsze strategie integracji kodu to:

- Trunk-based development
- Git-flow
- Github-flow
- Gitlab-flow

Trunk-based development



Git-flow

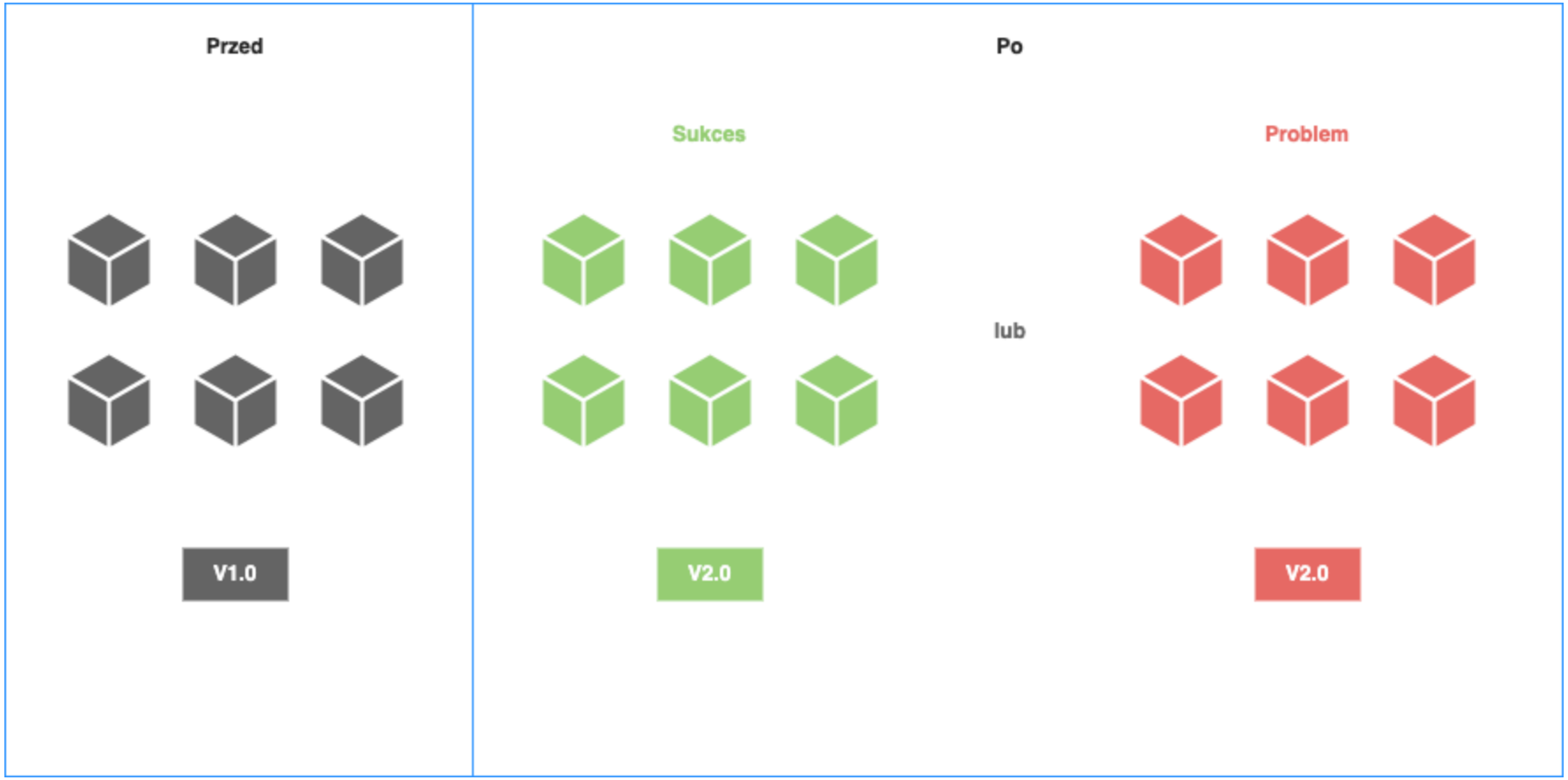

master
hotfixes
release branch
develop
feature1
feature2
1.0
1.1
2.1
3.0
2.0
tylko bugfixy
2.1
3.0

Strategie dostarczania kolejnych wersji aplikacji

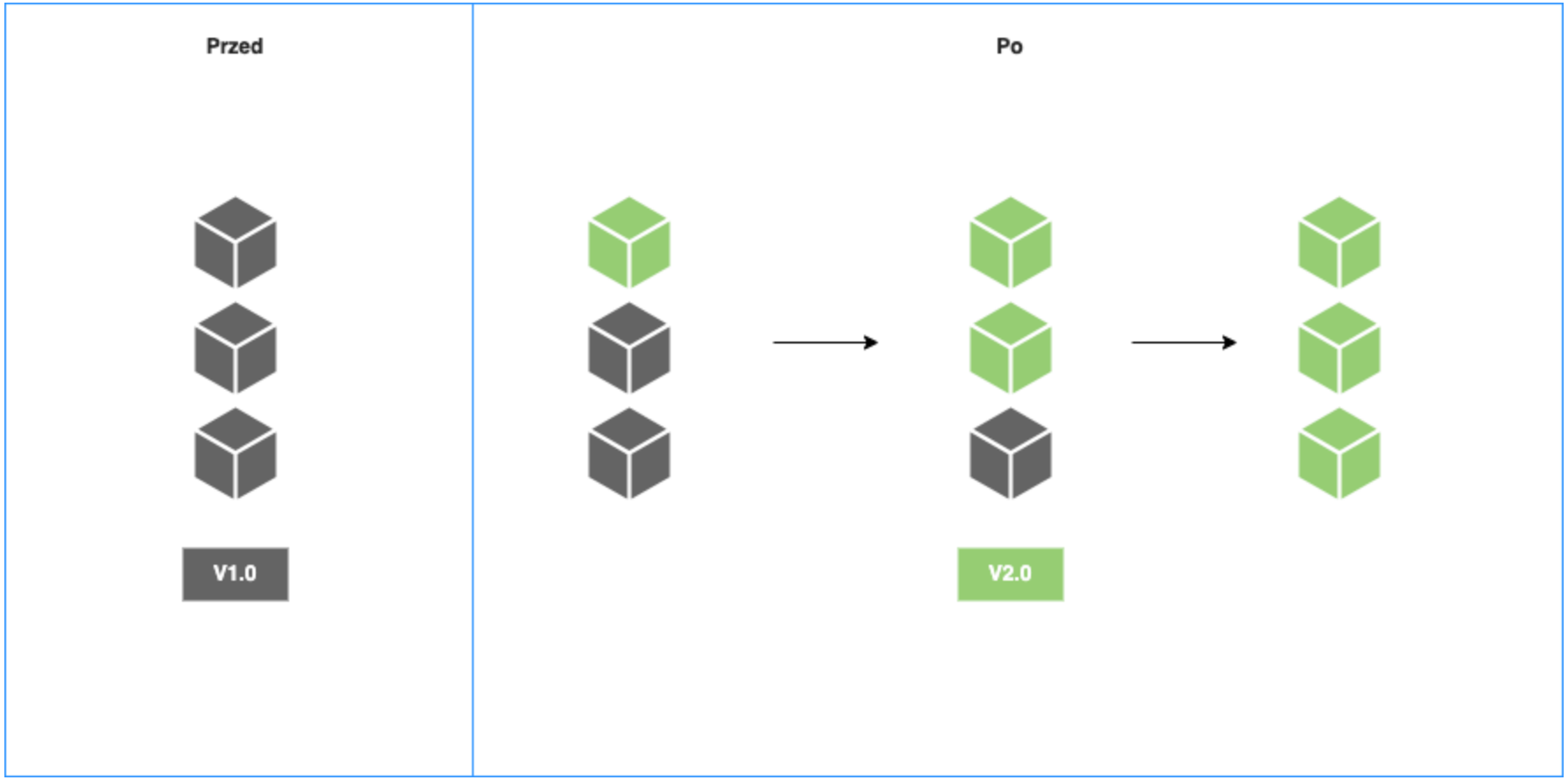
Najczęściej używanymi strategiami dostarczania nowych wersji aplikacji do użytkownika są:

- Basic Deployment
- Rolling update
- Blue-Green deployment
- Canary deployment
- A/B Testing

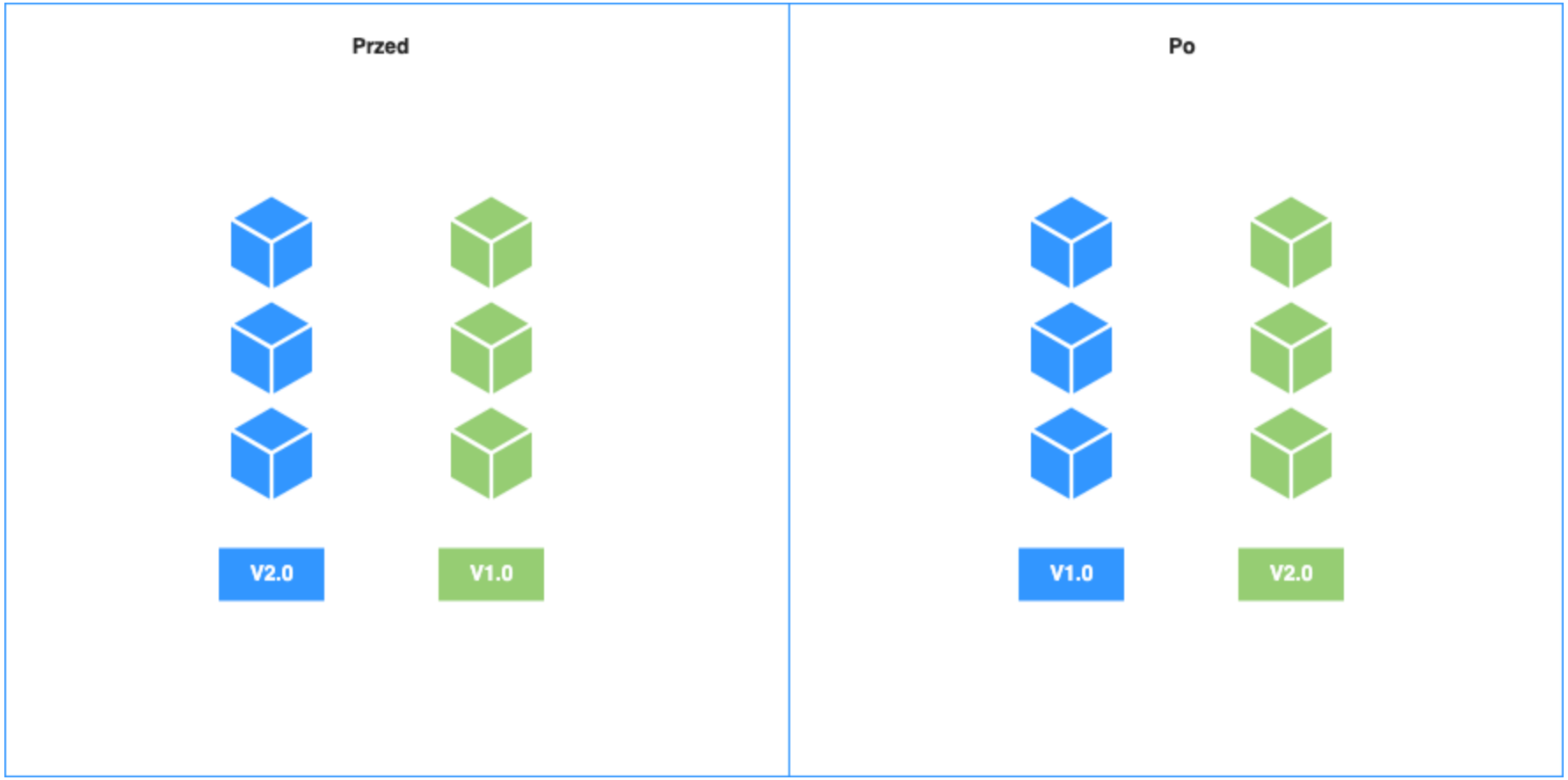
Basic deployment



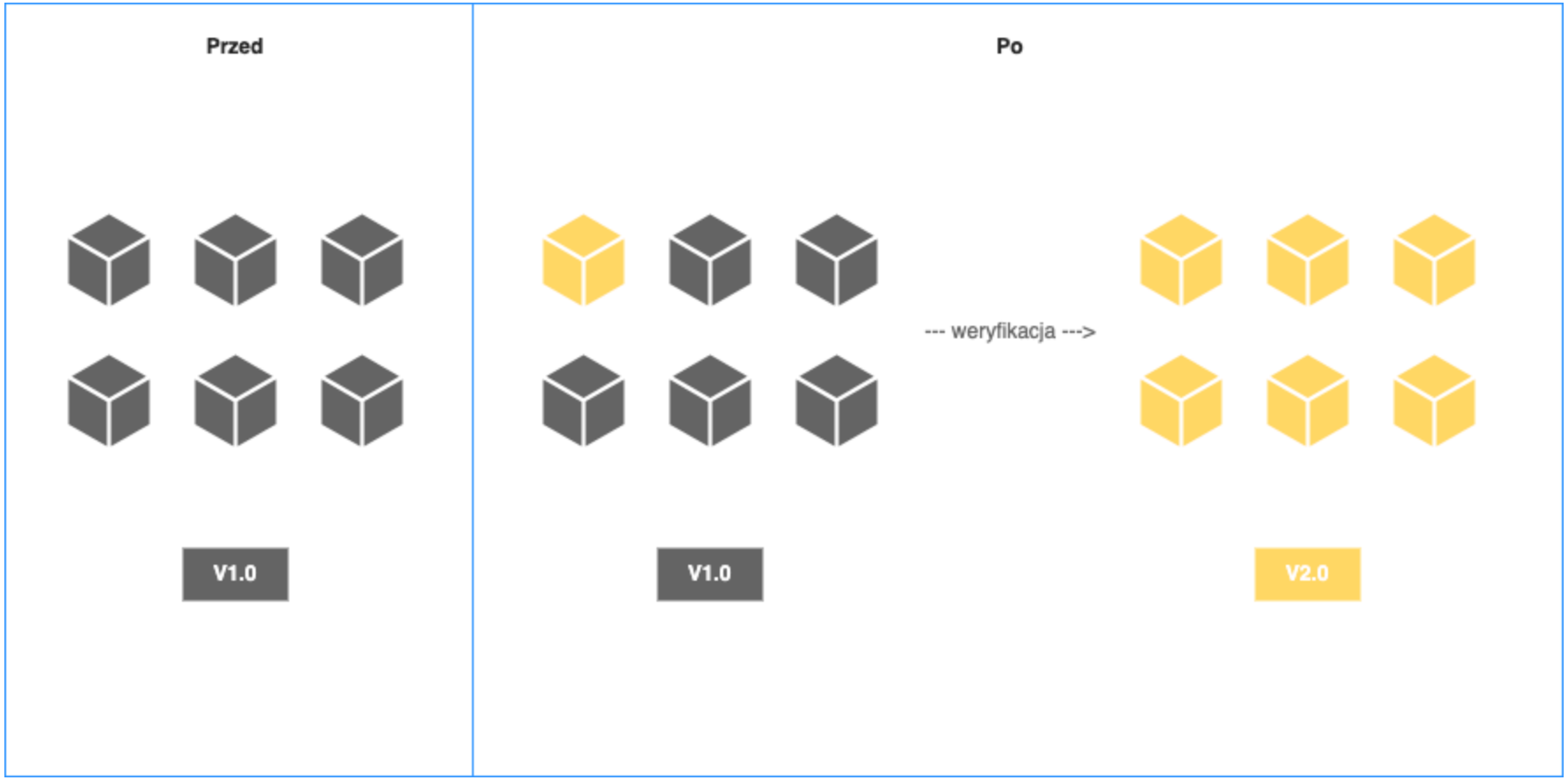
Rolling update



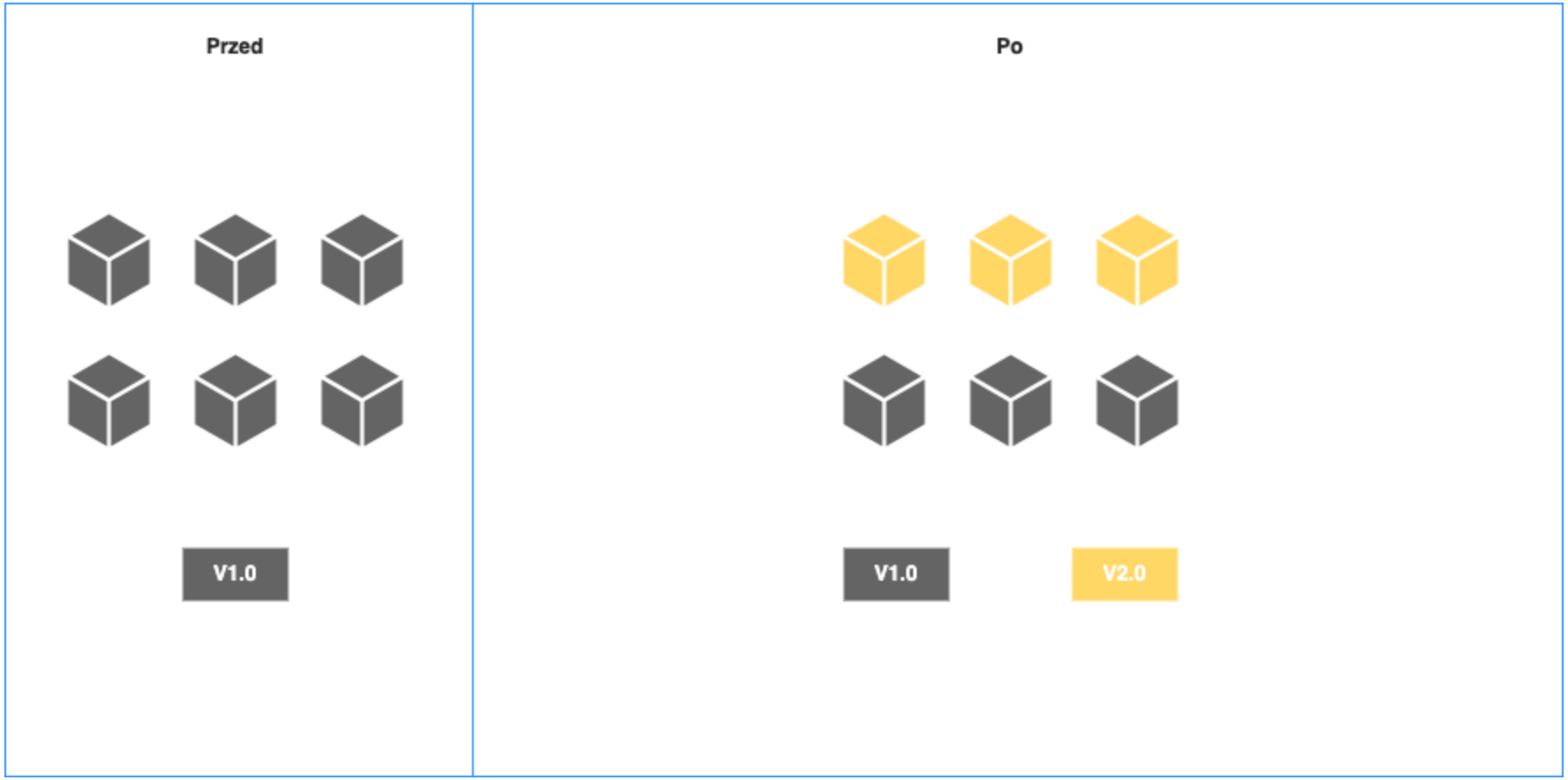
Blue-Green deployment



Canary deployment



A/B Testing



Narzędzia

Do implementacji naszego procesu CI/CD możemy użyć jednego z popularnych narzędzi, np.

- GitHub Actions
- Gitlab CI/CD
- Jenkins
- AWS CodePipeline
- Atlassian Bamboo
- Circle CI
- Buddy

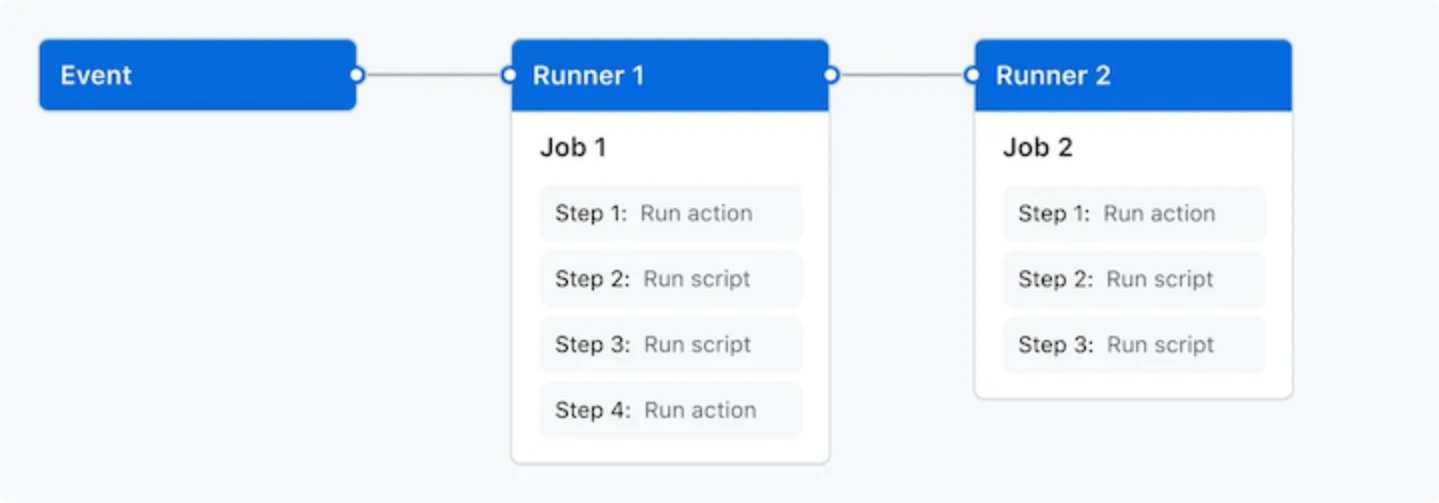
GitHub Actions

GitHub jest platformą CI/CD pozwalającą na automatyzację procesów budowania, testowania oraz uruchamiania aplikacji.

Konfiguracja oparta jest o przepływy pracy (ang. workflows) opisujące kolejne składowe procesu. Przepływy mogą zostać uruchomione na żądanie lub też na podstawie konkretnych zdarzeń (ang. events).

Dodatkowo platforma ta dostarcza środowiska uruchomieniowe, które odpowiedzialne są za wykonywanie działań, takich jak budowanie aplikacji. Umożliwia też dostarczanie własnych środowisk uruchomieniowych.

GitHub Actions: przepływ



Źródło: https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions

GitHub Actions: komponenty

- **workflow** - przypływ pracy opisujący automatyzację procesu CI/CD
- **event** - zdarzenie na repozytorium powodujące uruchomienie przepływu
- **job** - jednostka pracy składająca się ze zbioru kroków (ang. steps), każdy z kroków opisuje akcja lub skrypt, działają na tym samym runnerze
- **action** - aplikacja platformy GitHub Actions pozwalająca na definiowanie skomplikowanych kroków, które później mogą by używane w innych procesach, możliwe jest definiowanie własnych akcji
- **runner** - serwer używany do wykonywania jednostki pracy, wbudowane serwery obsługują Ubuntu Linux, Microsoft Windows oraz macOS

GitHub Actions: przykład

Konfiguracja oparta jest o pliki w formacie `YAML` umieszczane w katalogu `.github/workflows/`.

Konkretny plik w z góry opisanym formacie definiuje kolejny przepływ.

```yaml
name: Who triggered on push
run-name: Run by ${{ github.actor }}
on: [push]
jobs:
  show-files:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - run: ls -al
```

CI/CD: przyklad-github-actions

Przykład konfiguracji CI za pomocą GitHub Actions dla projektu Spring Boot budowanego narzędziem Maven.

Kod źródłowy przykładu pod adresem:
https://github.com/MaciejGowin/przyklad-github-actions

CD/CD: zadanie

Zaimplementuj uproszczoną wersję procesu CI/CD z użyciem GitHub Actions dla swojego projektu Java. Proces ten powinien pobrać najnowszą wersję kodu aplikacji napisanej w Javie, zbudować ją oraz uruchomić testy.

Bazy NoSQL

Nadszedł czas rozpoczęcia kolejnego projektu. Z wymagań wynika, że musimy być przygotowani na obsługę dużego ruchu do naszej aplikacji oraz przechowywanie dużej ilości danych.

Bazy danych SQL, które poznaliśmy do tej pory, oferują możliwości skalowania w celu obsługi większego ruchu i ilości danych, jednak są w tym aspekcie dosyć ograniczone i wymagają dodatkowych czynności w celu jej zapewnienia.

Dla takich zastosowań powstały właśnie bazy NoSQL. Dostarczają one znacznie lepsze możliwości skalowania bez podejmowania dodatkowych działań.

Ponadto w wielu przypadkach znacznie ułatwiają modelowanie danych przechowywanych w bazie danych.

Typy baz NoSQL

Najpopularniejsze typy baz NoSQL to:

- **Klucz-wartość** - klucz jest znany, ale wartości nie. Sprawdzają się w przechowywaniu nieustrukturyzowanych danych.
- **Dokumentowe** - rozszerzenie baz klucz-wartość, oferują możliwość zagnieżdżania par klucz-wartość i używania ich w zapytaniach.
- **Rodziny kolumn** - dane są przechowywane w postaci rodziny kolumn. Pozwala to na szybkie przeszukiwanie dużej ilości danych, ale wymaga to dobrze przemyślanego schematu bazy.
- **Grafowe** - bazy reprezentowane w postaci grafu. Węzły grafu reprezentują dane, a krawędzie relacje między nimi.

NoSQL: bazy klucz wartość i dokumentowe

```
{
  "id": "1",
  "nazwa": "Hotel WSB",
  "adres": {
    "ulica": "Sportowa 12",
    "miasto": "Warszawa",
    "kraj": "Polska"
  },
  "pokoje": [
    {
      "numer": 1,
      "liczbaPokoi": 2
    },
    {
      "numer": 2,
      "liczbaPokoi": 3
    }
  ]
}
```

NoSQL: rodziny kolumn

Klucz składa się z wielu posortowanych kolumn

Przykład:

Klucz = Firma/Linia Autobusowa/Czas/Numer rejestracyjny

Klucz	Lokalizacja
MTA/M86-SBS/2020-01-01T13:01:00/NYCT_5824	(40.781212,-73.961942)
MTA/M86-SBS/2020-01-01T13:02:00/NYCT_5840	(40.780664,-73.958357)
MTA/M86-SBS/2020-01-01T13:03:00/NYCT_5867	(40.780281,-73.946890)

NoSQL: rodziny kolumn

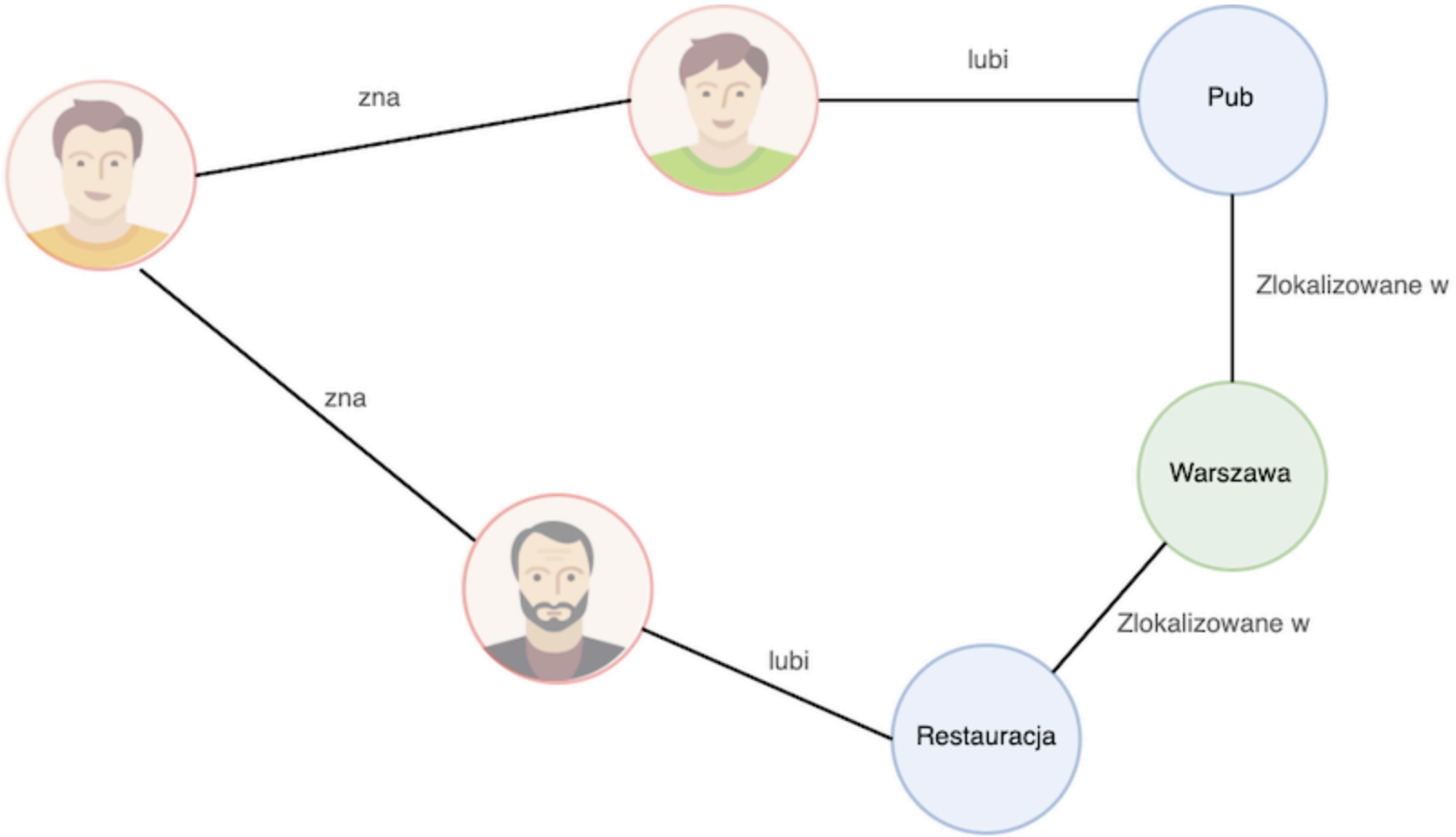
Przykład wydajnego zapytania:

- Lokalizacje konkretnego busa w danym zakresie czasu

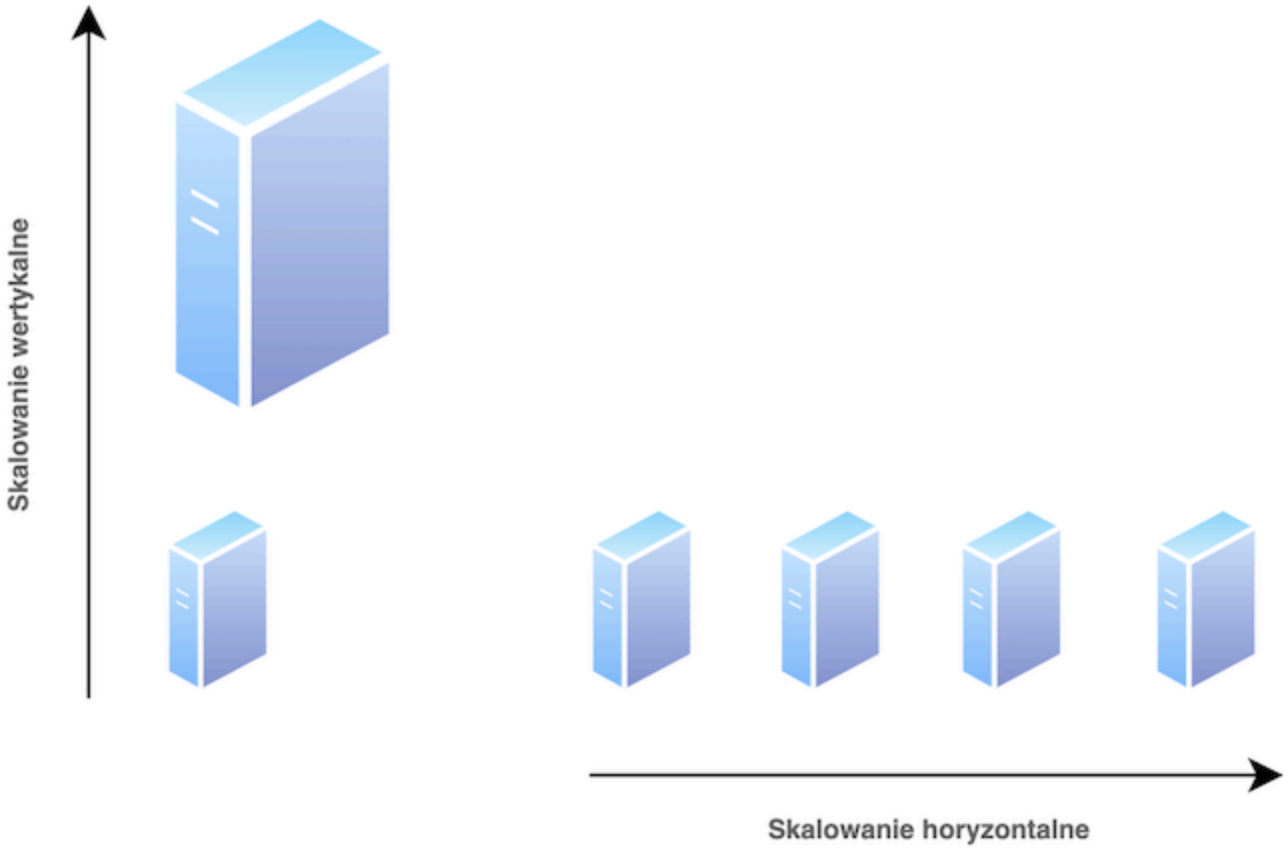
Przykład niewydajnego zapytania:

- Nazwy busów znajdujących się w prostokącie pomiędzy P1(40, -73), P2(41, -74)

Bazy grafowe



Skalowanie wertykalne (w górę), a skalowanie horyzontalne (wszerz)



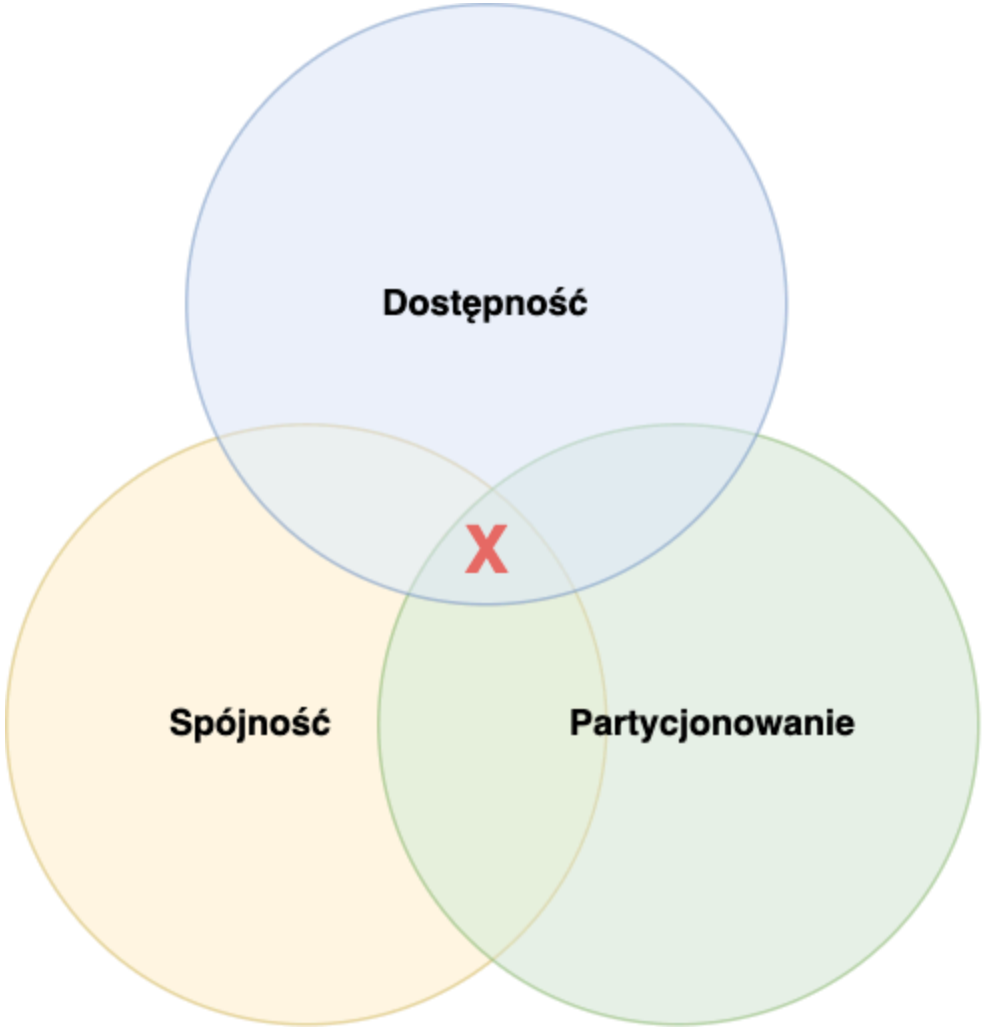
ACID: przypomnienie

- **[A] tomicity** - Niepodzielność
- **[C] onsistency** - Spójność
- **[I] solation** - Izolacja
- **[D] urability** - Trwałość

NoSQL: Twierdzenie CAP

- **[C] onsistency** - każda część rozproszonego systemu zwraca dane w najnowszej wersji.
- **[A] vailability** - system rozproszony zwróci dane nawet pomimo problemu z jego częścią, jednak nie ma gwarancji, że będą one w najnowszej wersji.
- **[P] artition-tolerance** - system rozproszony działa poprawnie pomimo problemów z jego częścią.

NoSQL: Twierdzenie CAP



Zadanie: zadanie-mlab

Naszym zadaniem jest utworzenie bazy danych NoSQL MongoDB na platformie https://mlab.com załadowanie do niej danych oraz napisanie kilku prostych zapytań.

Wszegóły zadania w opisie `README.md`