Programowanie aplikacji w Java

Maciej Gowin

Zjazd 10 - dzień 1

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 - 1

- **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 - 2

- **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 - 1

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.

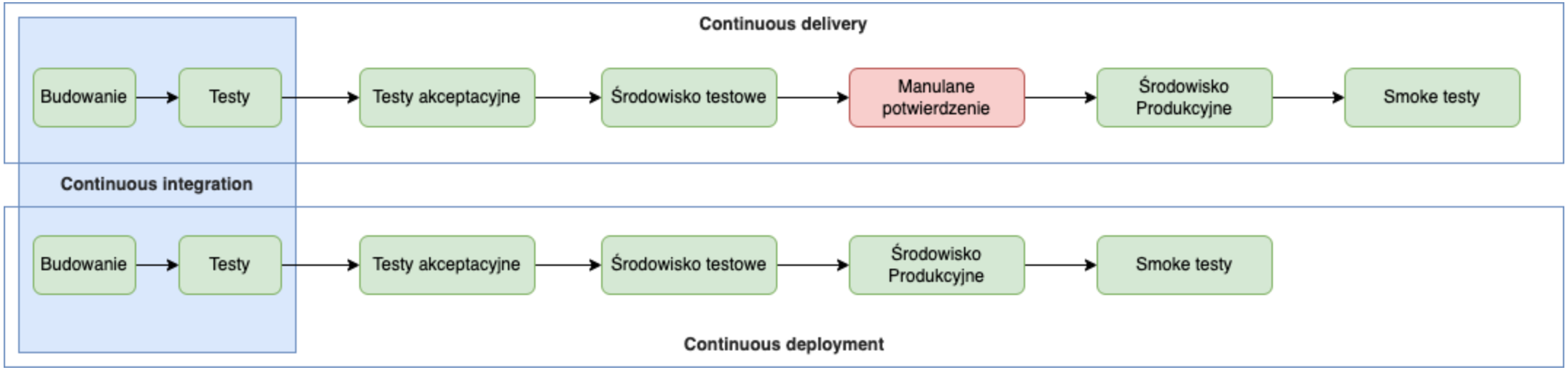
Aby zaoszczędzić nasz czas, chcemy zautomatyzować process wdrażania kolejnych wersji aplikacji. W tym celu zamierzamy
stworzyć process CI/CD.

CI/CD -2

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.

CI/CD - Continuous Integration vs Continuous Delivery vs Continuous Deployment

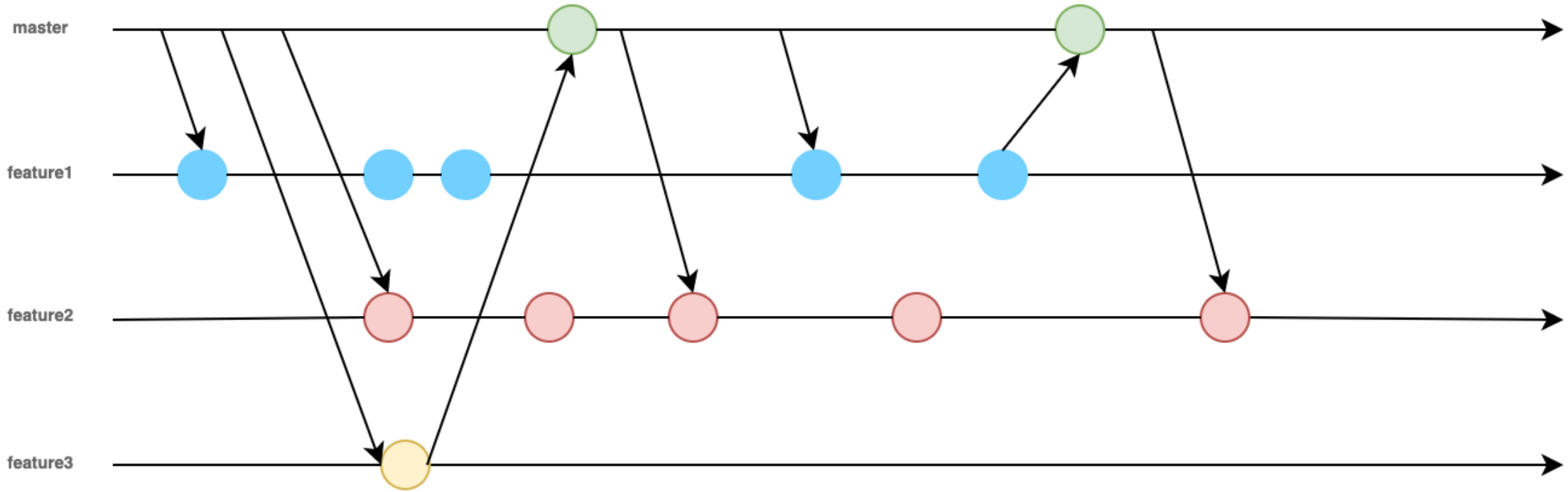


Strategie integracji kodu

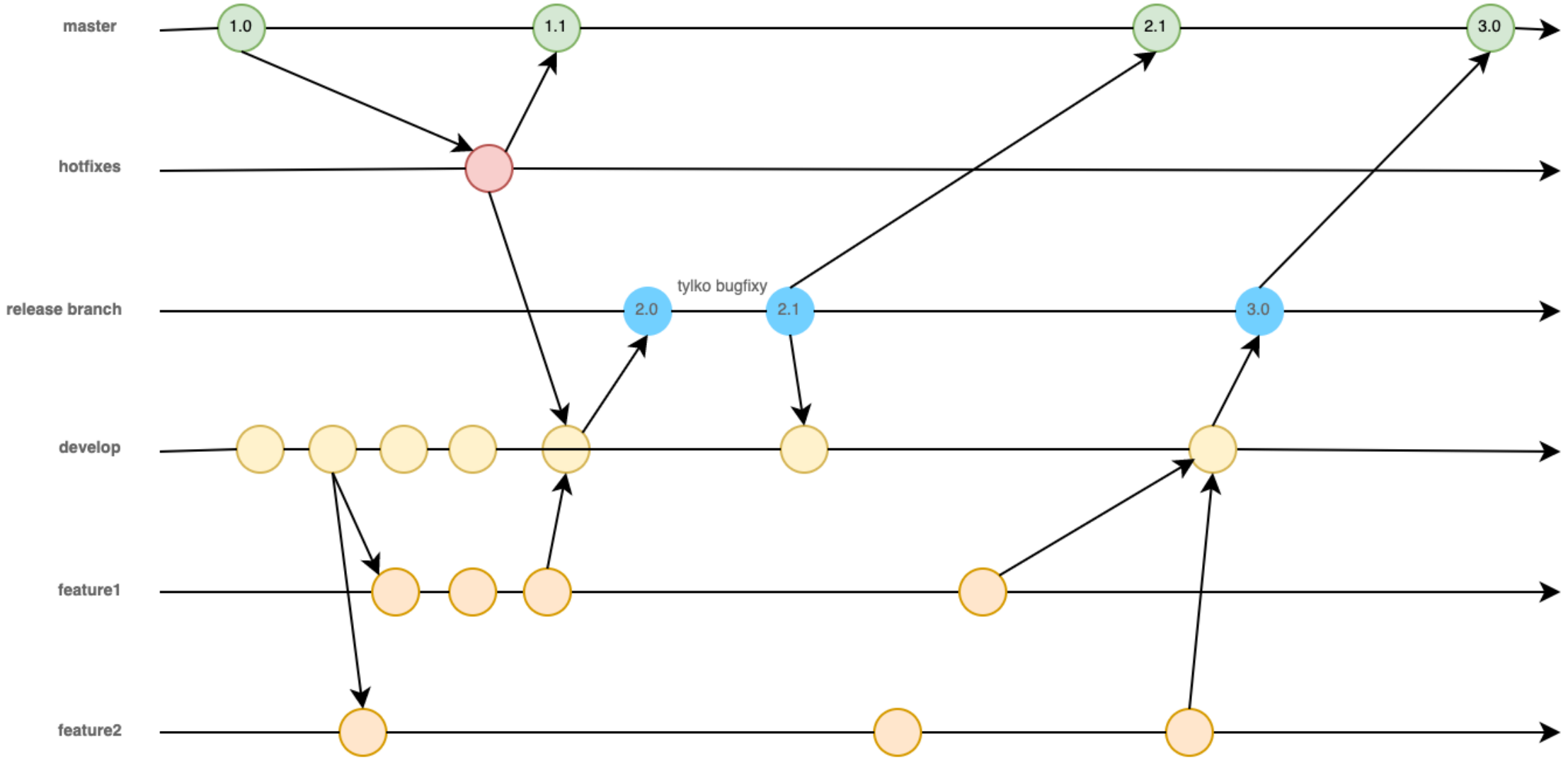
Najpopularniejsze strategie integracji kodu to:

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

Trunk-based development



Git-flow

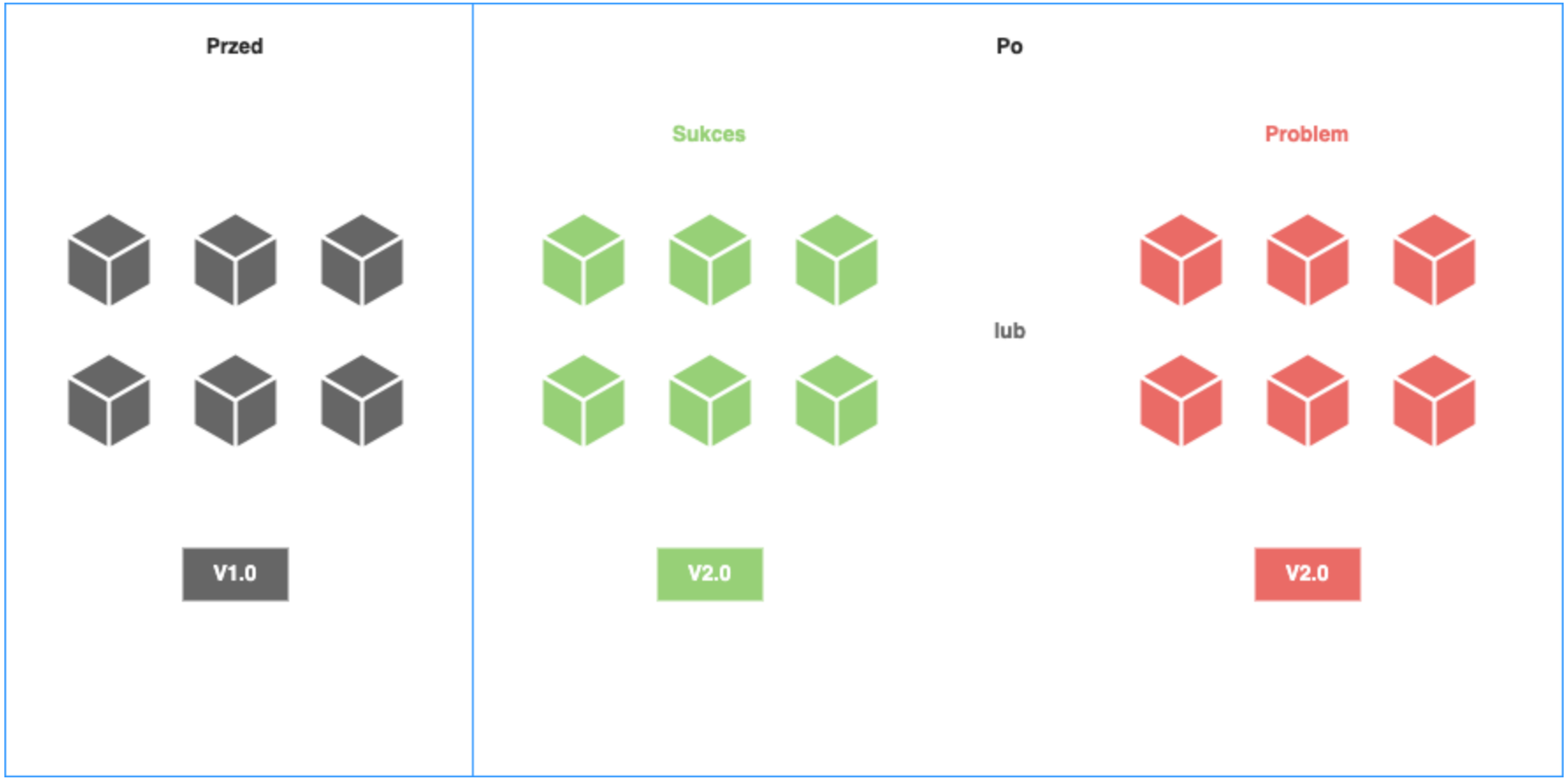

master
1.0
1.1
2.1
3.0
hotfixes
release branch
tylko bugfixy
2.0
2.1
3.0
develop
feature1
feature2

Strategie dostarczania kolejnych wersji aplikacji

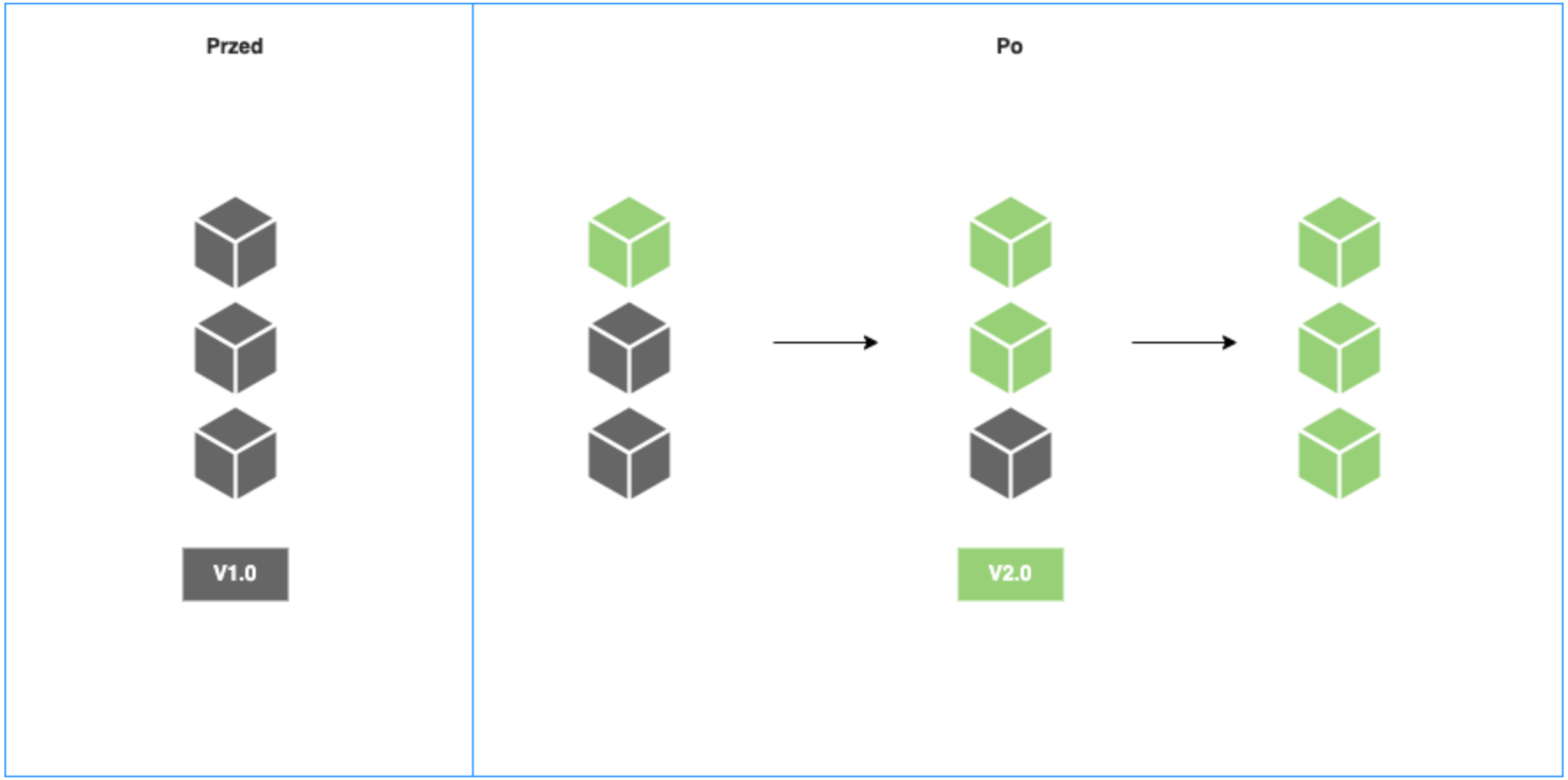
Najczęściej używanymi strategiami dostarczania nowych wersji aplikacji do użytkownika są:

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

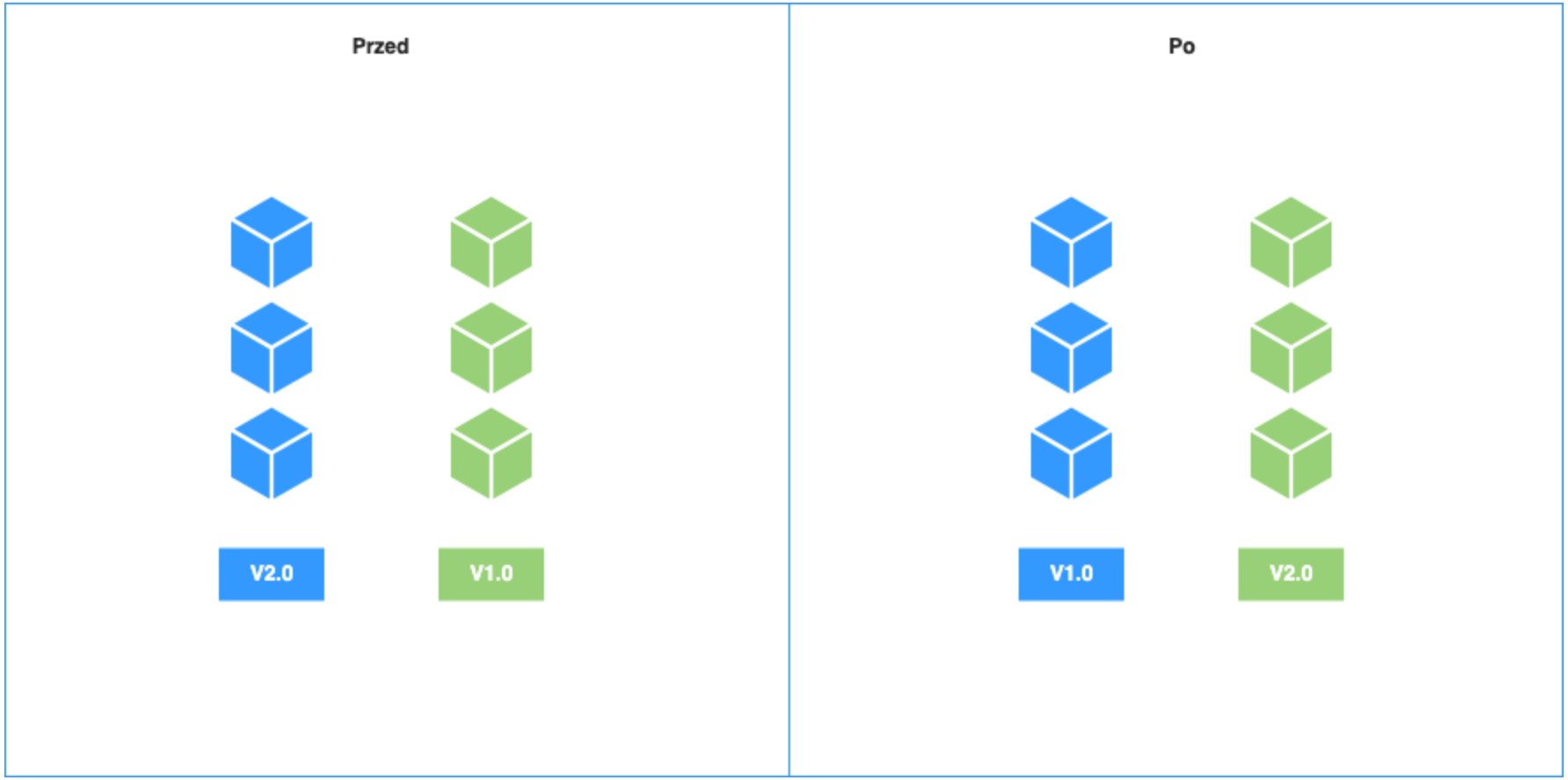
Basic deployment



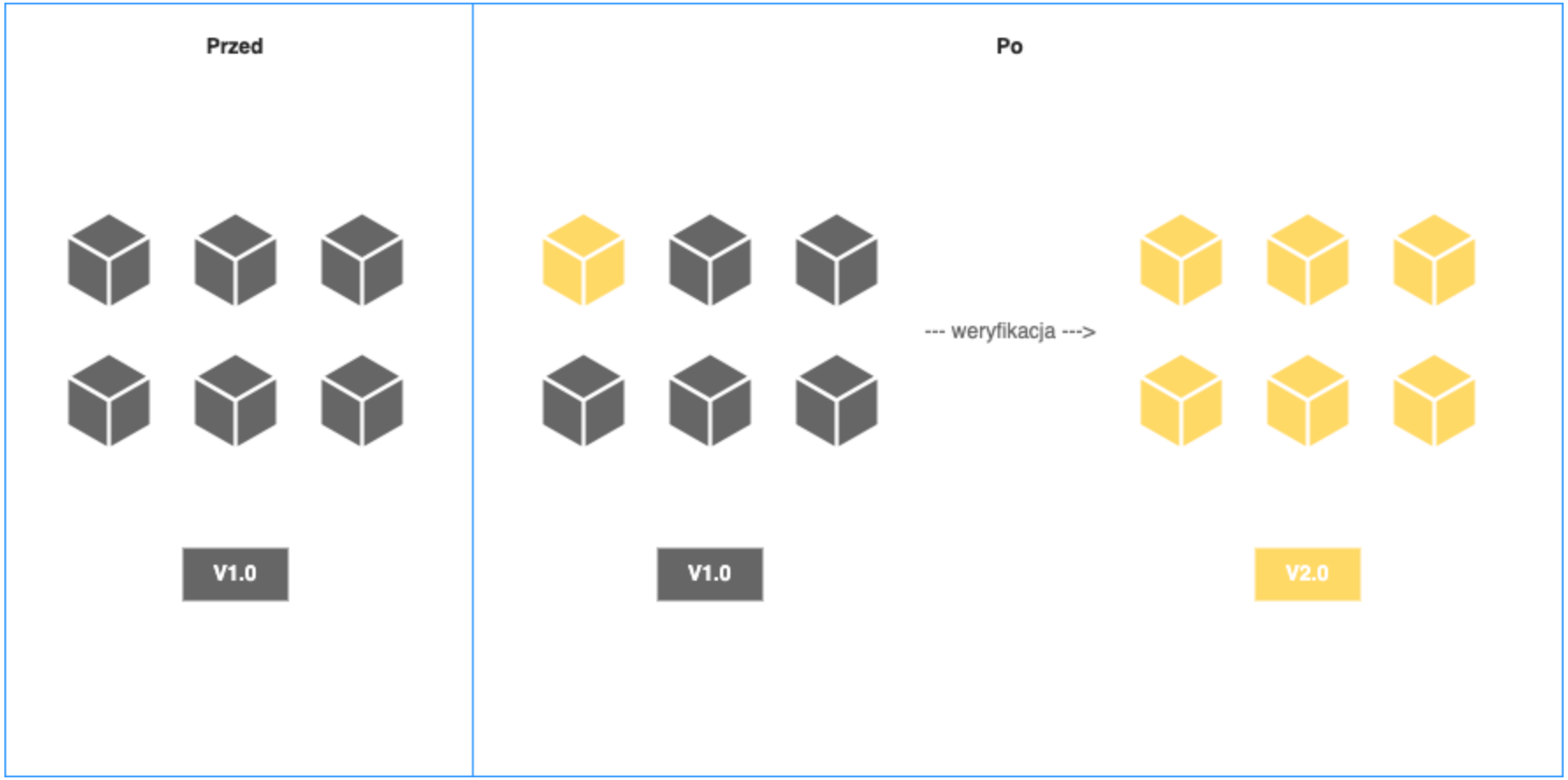
Rolling update



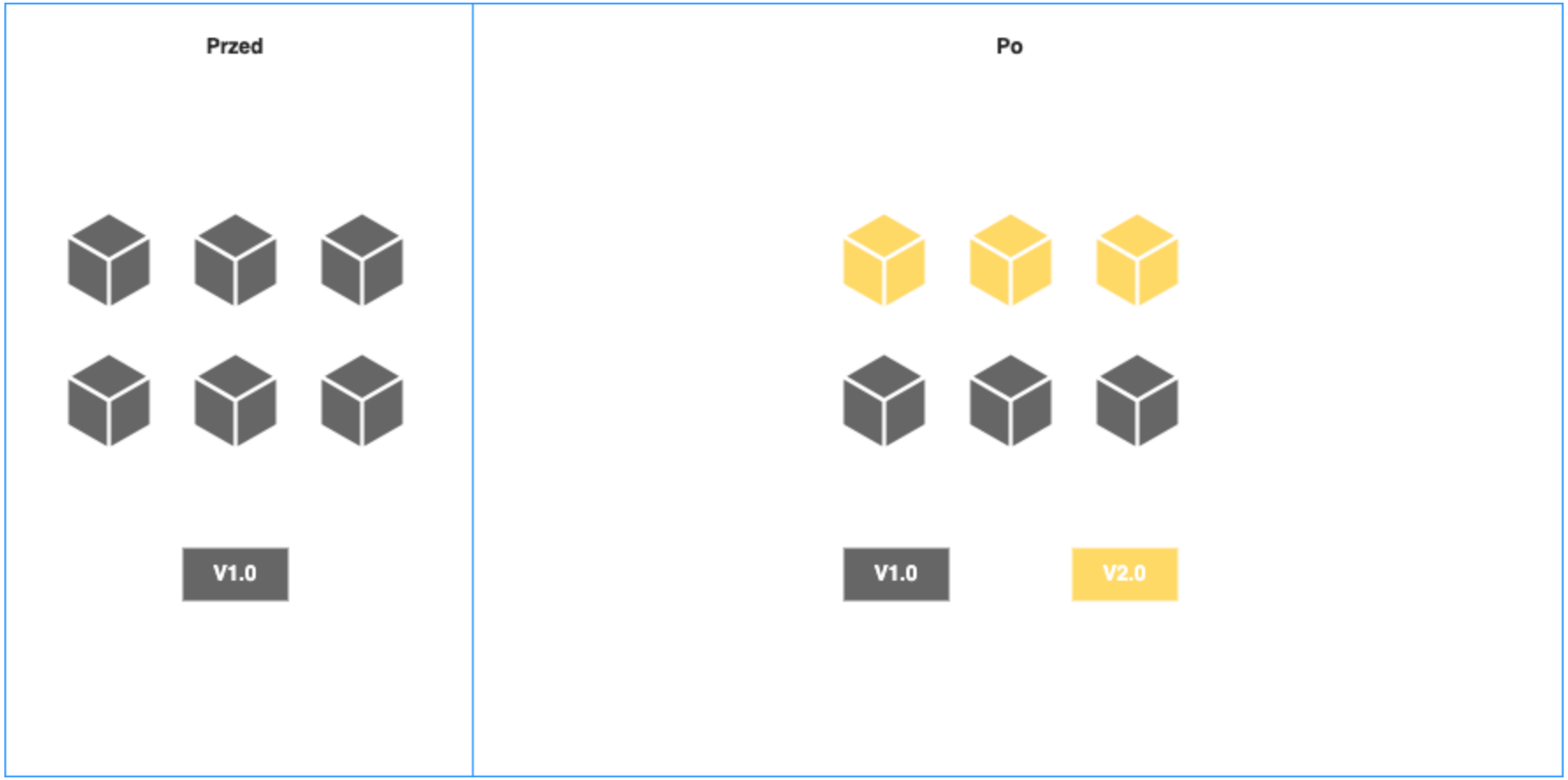
Blue-Green deployment



Canary deployment



A/B Testing



Narzędzia

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

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

Zadanie

Zaimplementuj uproszczoną wersję procesu CI/CD z użyciem Gitlab CI/CD. 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 bardzo 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.

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
    }
  ]
}
```

Rodziny kolumn - 1

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)

Rodziny kolumn - 2

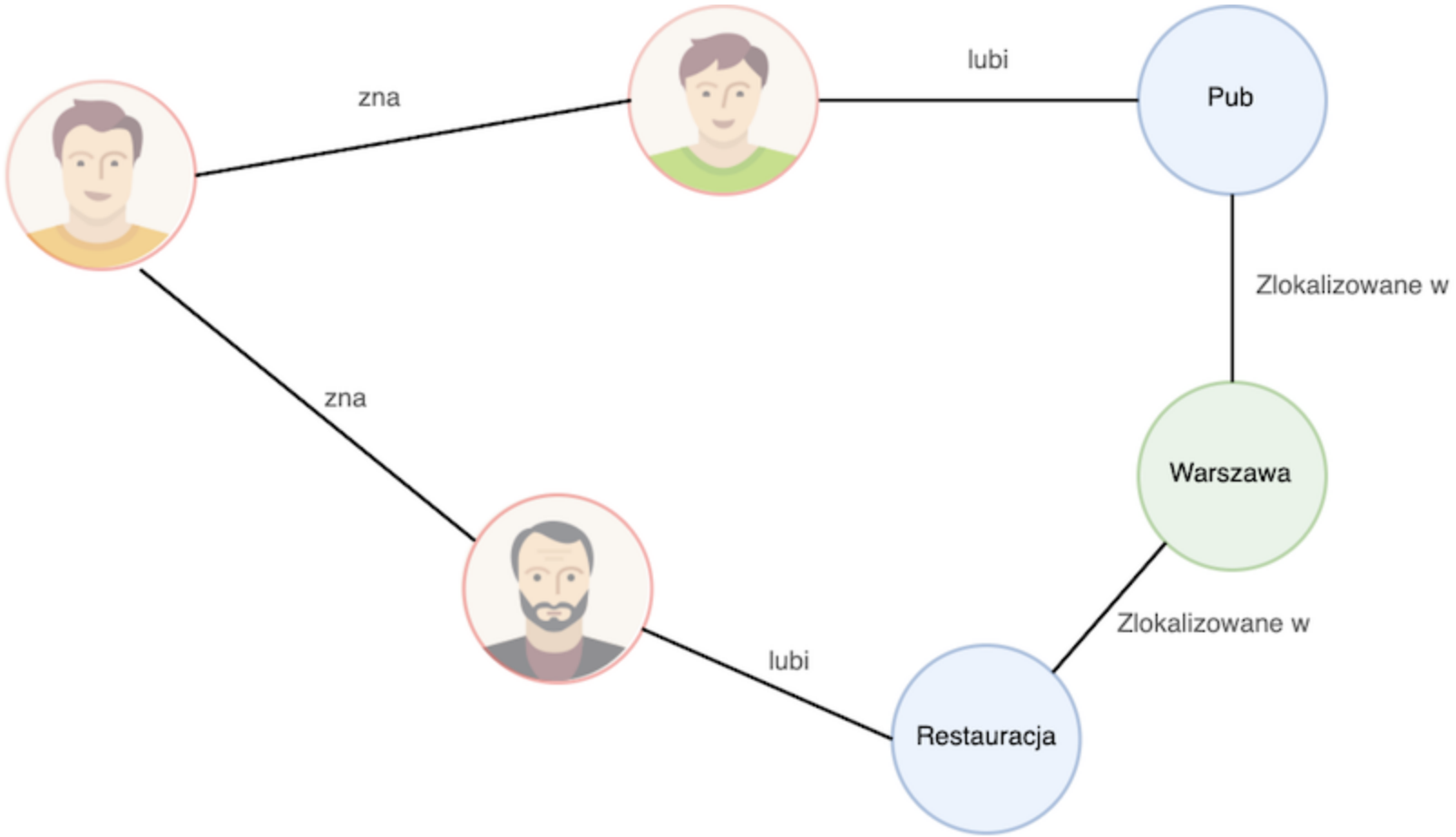
Przykład wydajnego zapytania:

- Lokalizacje konkretnego busa w danym zakresie czasu

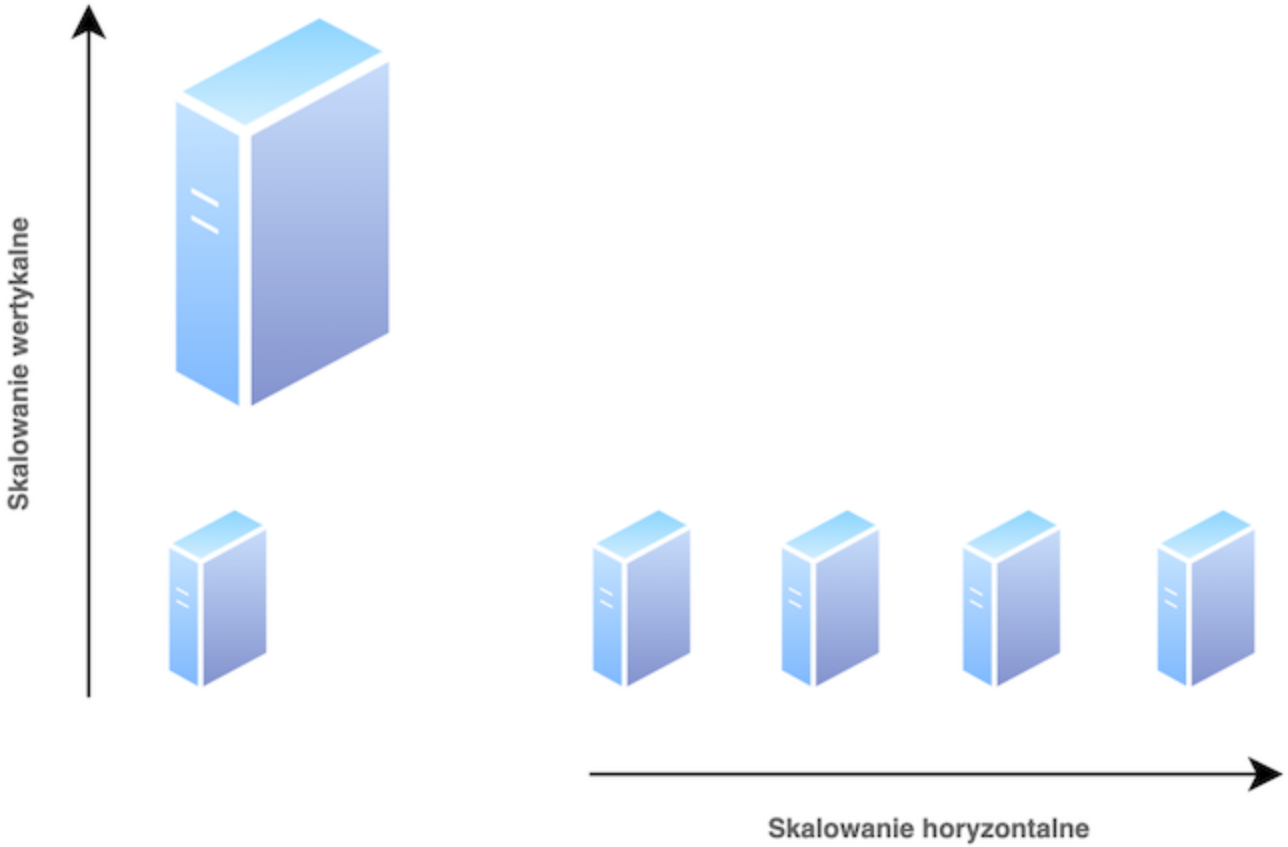
Przykład niewydajnego zapytania:

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

Bazy grafowe



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



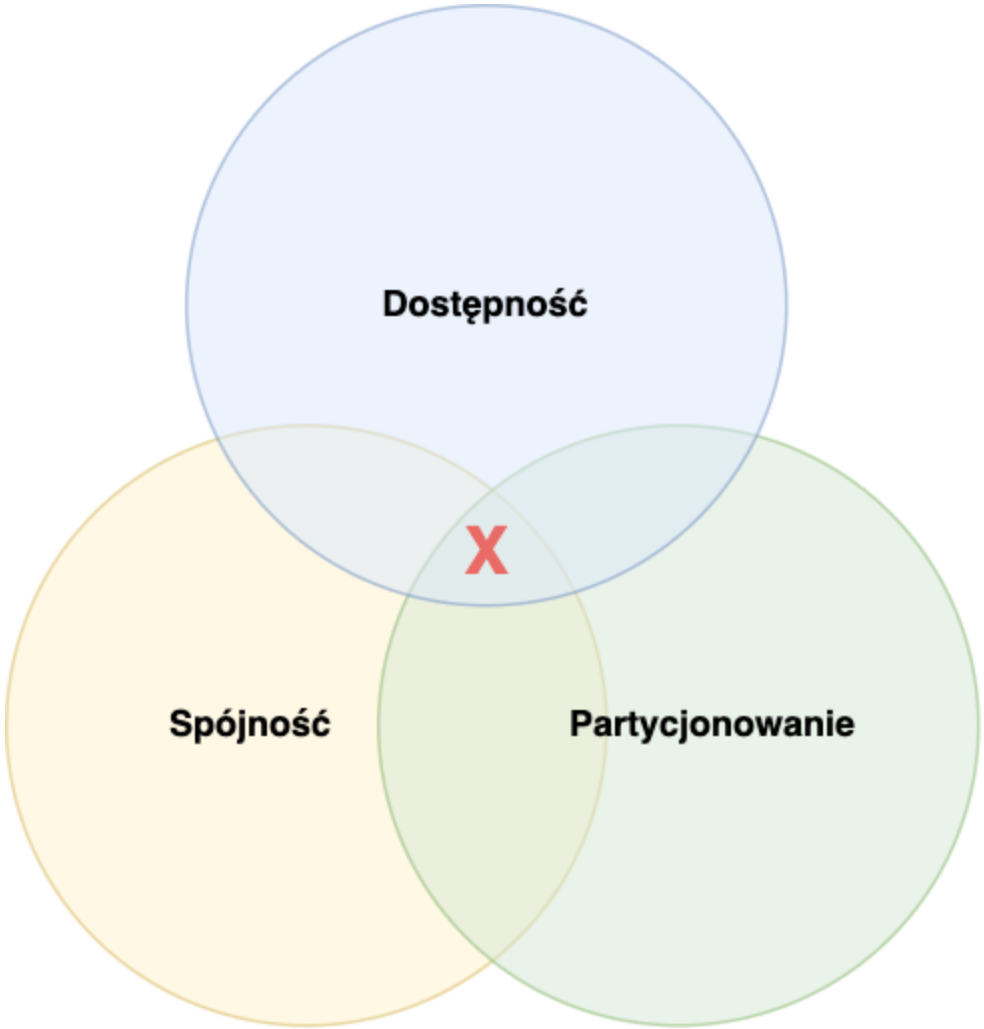
ACID - przypomnienie

- (A)tomicity - Niepodzielność
- (C)onsistency - Spójność
- (I)solation - Izolacja
- (D)urability - Trwałość

Twierdzenie CAP - 1

- (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.

Twierdzenie CAP - 2



Zadanie

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

Testowanie aplikacji

- **Ręczne**
  - Łatwe w przypadku małych aplikacji
  - Przy częstych zmianach kodu staje się uciążliwe
  - Bardzo wolne
  - Łatwo popełnić błąd
- **Automatyczne**
  - Wymaga stworzenia kodu testującego kod
  - Pomaga w programowaniu
  - Szybkie (w uruchomieniu / otrzymaniu wyników)
  - Rzetelne

Rodzaje testowania

- **Testowanie statyczne**
  Analiza napisanego kodu poprzez **code review** albo użycie aplikacji do statycznej analizy kodu, np. **Sonar**
- **Testowanie dynamiczne**
  Testy uruchamiane na działającej aplikacji. Sprawdzają, czy program działa tak jak się tego spodziewamy

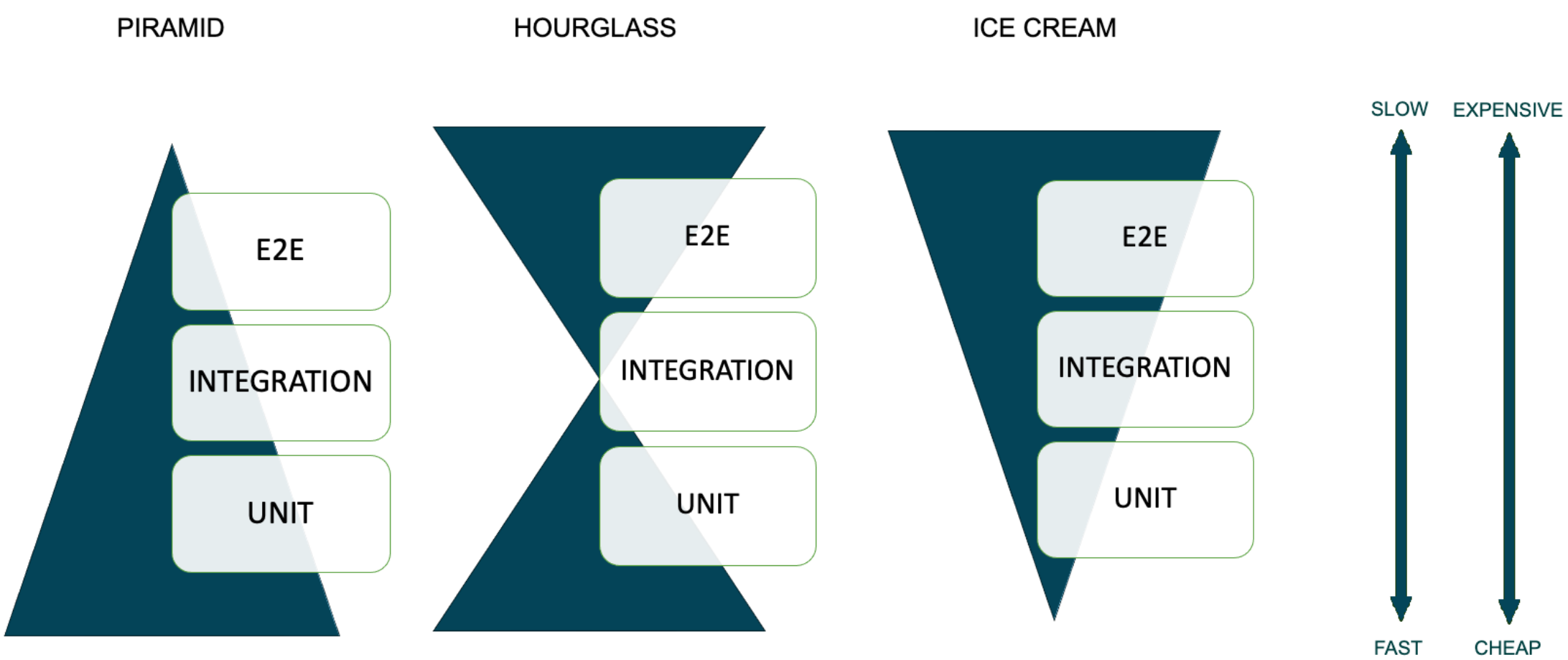
Rodzaje testowania

- **Testy funkcjonalne** (ang. black-box testing)
  Tester wie, jak program ma się zachować, nie zna szczegółów implementacji.
- **Testy strukturalne** (ang. while-box testing)
  Testy skupiające się na wewnętrznej pracy pojedynczego modułu.

Poziomy testowania

- **Testy jednostkowe**
  Jest to najniższy poziom testów. Ich zadaniem jest sprawdzenie poszczególnych funkcjonalności aplikacji. Testowane są zwykle małe fragmenty kodu, po czym wynik porównywany jest z wartością oczekiwaną. Najczęściej pisane są przez programistę w trakcie tworzenia implementacji.
- **Testy integracyjne**
  Testy sprawdzające działanie poszczególnych interface'ów aplikacji i ich wzajemne oddziaływanie.
- **Testy end-to-end**
  Testy sprawdzające działanie aplikacji jako całości od początku do końca (stąd nazwa end-to-end). Mają na celu znalezienie błędów wpływających na użytkownika.

Poziomy testowania



Junit 5

Junit - framework służący do pisania testów w języku Java. Wspomaga nas w pisaniu oraz uruchamianiu testów
jednostkowych.

Biblioteka dostarcza:

- adnotacje wspomagające pisanie testów
- mechanizmy do uruchamiania testów
- mechanizmy do grupowania testów
- raportowanie

Junit 5

```
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-engine</artifactId>
    <version>5.8.1</version>
    <scope>test</scope>
</dependency>
```

Junit 5 - adnotacje

Podstawowe adnotacje dostarczone przez bibliotekę:

- `@Test` - oznaczenie metody testującej
- `@BeforeAll` – oznaczenie metody uruchamianej przed wszystkimi metodami testującymi.
- `@AfterAll` – oznaczenie metody uruchamianej po wszystkich metodach testujących.
- `@BeforeEach` – oznaczenie metody uruchamianej przed każdym testem.
- `@AfterEach` – oznaczenie metody uruchamianej po każdym teście.
- `@Disabled` – oznaczona metoda tą adnotacją nie zostanie wywołana.

Wyżej wymienione adnotacje różnią się nieco między poprzednią wersją biblioteki (junit4) a obecną (junit5)

Junit 5 - przykład testu

```
import org.junit.jupiter.api.Test;
import org.springframework.util.StringUtils;
import static org.junit.jupiter.api.Assertions.assertEquals;

public class ExampleTest {

    @Test
    void shouldCapitalizeWord() {
        // given
        String lowercase = "wsb";
        // when
        String result = StringUtils.capitalize(lowercase);
        // then
        assertEquals("Wsb", result);
    }
}
```

Junit 5 - podstawowe zasady

1. Każda metoda testująca opatrzona jest adnotacją `@Test`
2. Metoda testująca nic nie zwraca (void)
3. Klasa testowa powinna nazywać się tak jak klasa, dla której pisany jest test z suffixem `Test`
4. Klasa testowa może zawierać wiele testów (dotyczących jednej bądź wielu metod dostępnych w klasie głównej)

Test jednostkowy - podział

given

Część ustawiająca wartości na potrzeby danego testu.
Inicjalizacja obiektów przekazywanych jako argumenty metody, która jest poddana testowi.

when

Wywołanie metody poddawanej testowi.
Najczęściej w tej części znajduje się jedna linijka kodu - wywołanie metody

then

Asercje, sprawdzenie, czy wartości oczekiwane odpowiadają tym otrzymanym podczas wykonywania testu

Asercje

- Asercje są to warunki, których spełnienie jest wymagane do zaliczenia testu.
- Porównują wartość otrzymaną w wyniku wykonania kawałka kodu z wartością oczekiwaną
- Pojedynczy test zawiera **co najmniej** jedną asercję.
- Niepowodzenie którejkolwiek z asercji powoduje przerwanie testu z wynikiem negatywnym.

Przykłady:

```
assertEquals(expected, actual);          // org.junit.jupiter.api.Assertions
assertThat(actual).isEqualTo(expected); // org.assertj.core.api.Assertions
```

Testy + Maven

```
src
  |- main
  |  |- java
  |  |  |- pl.wsb
  |  |  |  |- ExampleClass
  |  |- resources
  |- test
  |  |- java
  |  |  |- pl.wsb
  |  |  |  |- ExampleClassTest
  |  |- resources
```

- Wszystkie testy należy umieszczać w katalogu `src/test/java`
- Uruchomienie testów - `mvn test`

Maven

Wykonanie komendy `mvn test` spowoduje uruchomienie testów programu.

Przykładowy rezultat:

```
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.106 s - in pl.wsb.tests.TestsApplicationTests
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  3.045 s
[INFO] Finished at: 2022-05-14T09:57:35+02:00
[INFO] ------------------------------------------------------------------------
```

Cykl Maven dla testów

Lifecycle Reference

W lifecycle Maven (default) istnieją 2 fazy, które umożliwiają uruchomienie testów.

- test
- integration-test

Istnieje możliwość pogrupowania testów (separacji testów unitowych oraz integracyjnych), np.

- poprzez użycie innych suffixów: **Test** oraz **IntegrationTest**
- poprzez oznaczenie poszczególnych testów poprzez adnotację `@Tag`

Cykl Maven dla testów

Wybór grupowania testów należy skonfigurować w opisie plugina w `pom.xml` np.

`@Tag(junit5)`

```
<build>
    <plugins>
        <plugin>
            <artifactId>maven-surefire-plugin</artifactId>
            <configuration>
                <properties>
                    <includeTags>junit5</includeTags>
                </properties>
            </configuration>
        </plugin>
    </plugins>
</build>
```

Asercje - dostępne opcje

- wbudowane - asercje dostępne w bibliotece JUnit

```
assertEquals(expected, actual);
assertArrayEquals(expected, actual);
assertNull(object);
```

- Hamcrest

```
assertThat(array, hasItemInArray("text"));
assertThat("text", isOneOf(array));
assertThat(5, greaterThanOrEqualTo(5));
```

Asercje - dostępne opcje

- Truth

```
assertThat(text).contains("wsb");
assertThat(projectsByTeam()).valuesForKey("field1").containsExactly("w","s","b");
```

- AssertJ

```
assertThat(stringVariable).isEqualTo("Frodo");
assertThat(array).hasSize(9).contains(a1, a2).doesNotContain(a3);
assertThatThrownBy(() -> { throw new Exception("boom!"); }).hasMessage("boom!");
```

AssertJ

Bogata biblioteka dostarczająca zbiór asercji.

Świetnie współpracuje z JUnit.

Umożliwia pisanie bardzo czytelnych asercji, co ułatwia pracę i przyspiesza detekcję błędów.

```
import static org.assertj.core.api.Assertions.*;
```

AssertJ - zależność Maven

```
<dependency>
    <groupId>org.assertj</groupId>
    <artifactId>assertj-core</artifactId>
    <version>3.22.0</version>
    <scope>test</scope>
</dependency>
```

AssertJ - podstawowe asercje

Podstawowa forma:

```
assertThat(referenceOrValue).<ASERCJA>
```

W przypadku potrzeby przetestowania przypadku wystąpienia wyjątku forma ma nieco inny wygląd:

```
assertThatThrownBy(() -> {} }
```

AssertJ - assertThat

`assertThat()`

- obiekty
  - `isEqualTo() / isNotEqualTo()` - porównanie dwóch obiektów
  - `isEqualToComparingFieldByFieldRecursively()` - porównanie pól dwóch obiektów tego samego typu
- typ Boolean
  - `isTrue() / isFalse()` - sprawdza, czy wartość typu boolean jest równa true albo false
- kolekcje
  - `hasSize()` - sprawdza wielkość kolekcji
  - `isEmpty() / isNotEmpty()` - sprawdza, czy kolekcja jest pusta / nie jest pusta
  - `containsAll() / containsExactly()` - sprawdza, czy kolekcja zawiera podane wartości

AssertJ - assertThat

Powyższe przykłady to jedynie ułamek możliwości tej biblioteki.
Dla krótszego zapisu w sytuacji, w której chcemy sprawdzić kilka rzeczy, dla sprawdzanego obiektu można połączyć kilka metod, używając *method chaining*, np.:

```
assertThat(myList)
          .isNotNull()
          .isNotEmpty()
          .hasSize(2)
          .containsExactly(1,2);
```

Junit 5 - testy parametryzowane

```
public class NumberUtils {

  public static boolean isOdd(int number) {
    return number % 2 != 0;
  }

}
```

```
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.ValueSource;
import static org.junit.jupiter.api.Assertions.assertFalse;
import static org.junit.jupiter.api.Assertions.assertTrue;

public class NumberUtilsParametrizedTest {

    @ParameterizedTest
    @ValueSource(ints = {1, 3, 5, -3, 15})
    void shouldReturnTrueForOddNumbers(int number) {
        assertTrue(NumberUtils.isOdd(number));
    }
}
```

Junit 5 - testy parametryzowane

```
import java.math.BigDecimal;
import java.util.Map;

public class CurrencyConversion {

  private static Map<String, BigDecimal> CURRENCY_TO_PLN_RATIO = Map.of(
          "USD", BigDecimal.valueOf(4.5123),
          "EUR", BigDecimal.valueOf(4.1989)
  );

  public static BigDecimal convert(BigDecimal priceInPLN, String currencyCode) {
    return priceInPLN.multiply(CURRENCY_TO_PLN_RATIO.get(currencyCode));
  }
}
```

Junit 5 - testy parametryzowane

```
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.Arguments;
import org.junit.jupiter.params.provider.MethodSource;
import java.math.BigDecimal;
import java.util.stream.Stream;
import static org.assertj.core.api.Assertions.assertThat;

public class CurrencyConversionParametrizedTest {

    private static Stream<Arguments> shouldCalculatePriceInGivenCurrency() {
        return Stream.of(
            Arguments.of(BigDecimal.valueOf(10), "USD", BigDecimal.valueOf(45.123)),
            Arguments.of(BigDecimal.valueOf(10.34), "EUR", BigDecimal.valueOf(43.416626)));
    }

    @ParameterizedTest
    @MethodSource
    void shouldCalculatePriceInGivenCurrency(
          BigDecimal priceInPLN, String currencyCode, BigDecimal expected) {

        BigDecimal actual = CurrencyConversion.convert(priceInPLN, currencyCode);
        assertThat(actual.doubleValue()).isEqualTo(expected.doubleValue());
    }
}
```

Zadanie

1. Popraw funkcję `CurrencyConversion.convert` tak, aby zaokrąglała w dół, do 2 miejsc po przecinku w następujący sposób: 1.6 -> 1, 1.5 -> 1, 1.1 -> 1, 1.0 -> 1
2. Dodaj więcej przypadków testowych. Pamiętaj o scenariuszach negatywnych - niepoprawne dane, błędy

--

Testowanie - ciekawe linki

- A Guide to JUnit 5
- Intellij IDEA - uruchamianie testów
- Running a Single Test or Method With Maven
- AssertJ docs
- AssertJ - wprowadzenie