TRANSITION
TECHNOLOGIES
**MANAGED**
**SERVICES**

GROUP
TRANSITION
TECHNOLOGIES

# MIKROSERWISY CZ. 2

KRZYSZTOF ŁOPUCKI • 2018

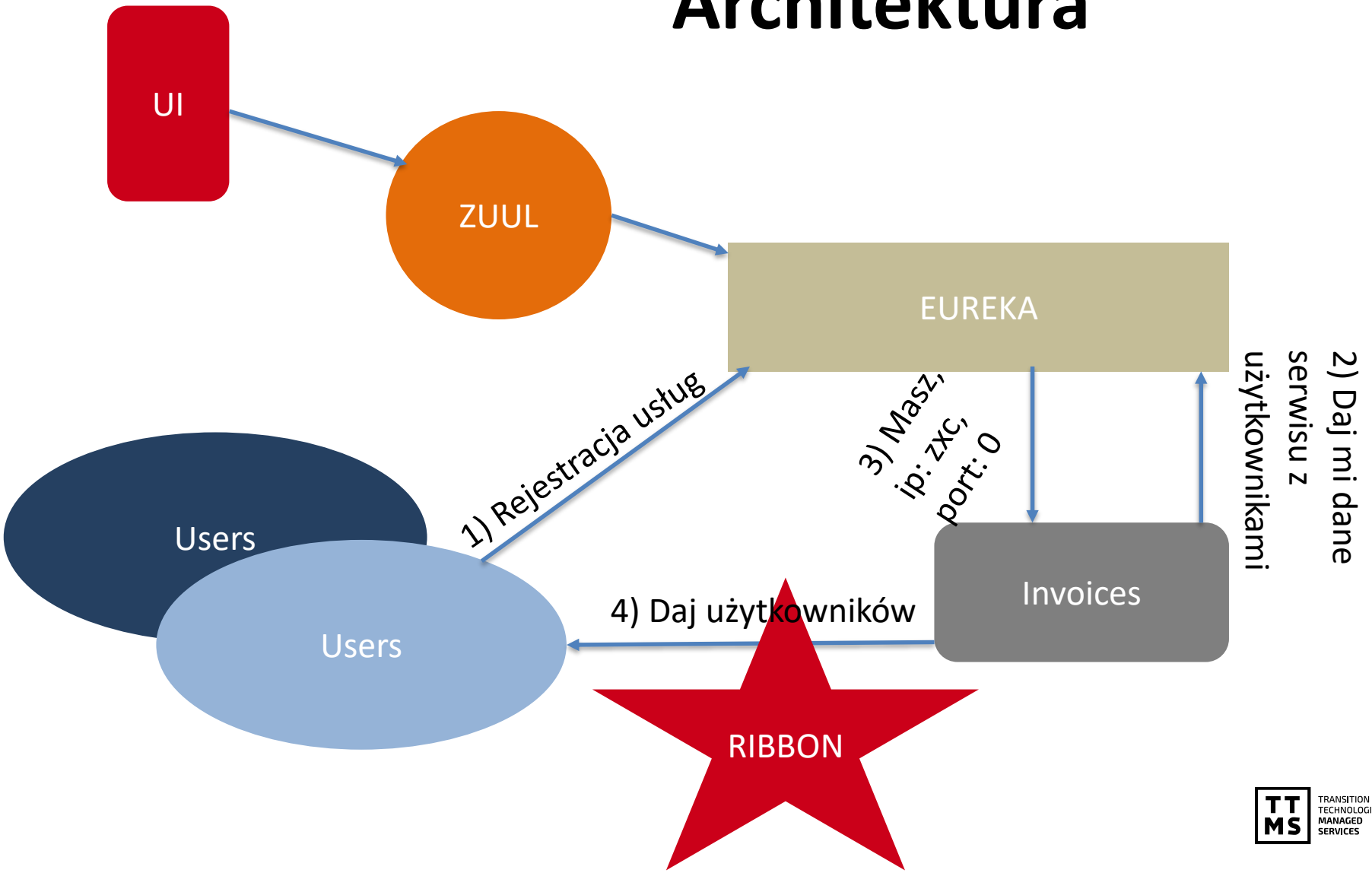
# Wyzwania

- Komunikacja
- Skalowalność
- Konfiguracja
- Obsługa awarii
- Znajdowanie błędów
- API

# Pomocne narzędzia

- Komunikacja - **Eureka**
- Skalowalność - R**ibbon**
- Konfiguracja - **Spring Cloud Config**
- Obsługa awarii - **Hysterix**
- Znajdowanie błędów - **Zipkin**
- API - **Zuul**

# Architektura



# Eureka

- Adresy i porty mogą się zmieniać
- Skalowane aplikacje mogą mieć różne porty

- Eureka implementuje wzorzec Discovery

# Eureka - serwer

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
    <version>2.0.2.RELEASE</version>
</dependency>

<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-parent</artifactId>
            <version>Finchley.SR2</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

# Eureka - serwer

```
@SpringBootApplication
@EnableEurekaServer
public class EurekaServerApplication {
    public static void main(String[] args) {
        SpringApplication.run(EurekaServerApplication.class, args);
    }
}
```

```
# application.properties

server.port=8761
eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false
```

# Mikroserwisy cz. 2

http://localhost:8761

# Eureka - klient

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-starter</artifactId>
    <version>2.0.2.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
    <version>2.0.1.RELEASE</version>
</dependency>
```

```
public interface GreetingController {
    @RequestMapping("/greeting")
    String greeting();
}
```

# Eureka - klient

```
@SpringBootApplication
@RestController
public class EurekaClientApplication implements GreetingController {

    @Autowired
    @Lazy
    private EurekaClient eurekaClient;

    @Value("${spring.application.name}")
    private String appName;

    public static void main(String[] args) {
        SpringApplication.run(EurekaClientApplication.class, args);
    }

    @Override
    public String greeting() {
        return String.format("Hello from '%s'!",
          eurekaClient.getApplication(appName).getName());
    }
}
```

# Eureka - klient

```
spring.application.name=users

server.port=8081

eureka.client.service-
url.defaultZone:${EUREKA_URI:http://localhost:8761/eureka}
eureka.instance.prefer-ip-address=true
```

# Eureka - konsument

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-starter</artifactId>
    <version>2.0.2.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
    <version>2.0.1.RELEASE</version>
</dependency>
```

# Eureka - konsument

```
spring.application.name=konsumer

server.port=8082

eureka.client.service-
url.defaultZone:${EUREKA_URI:http://localhost:8761/eureka}
eureka.instance.prefer-ip-address=true
```

# Eureka - konsument

```
@Configuration
public class RestTemplateConfig {
    @Bean
    public RestTemplate restTemplate() {
        return new RestTemplate();
    }
}
```

# Eureka - konsument

```
@SpringBootApplication
@EnableDiscoveryClient
@RestController
public class UserConsumer {

    @Autowired
    private EurekaClient eurekaClient;

    public static void main(String[] args) {
        SpringApplication.run(UserConsumer.class, args);
    }

    @Autowired
    private RestTemplate restTemplate;

    @GetMapping("/dajGlos")
    public String dajGlos() {
        Application application
                = eurekaClient.getApplication("users");
        List<InstanceInfo> instances = application.getInstances();
        InstanceInfo instanceInfo = instances.iterator().next();
        String hostname = instanceInfo.getHostName();
        int port = instanceInfo.getPort();
        return restTemplate.getForObject(
           "http://"+hostname+":"+port+"/greeting", String.class) + " Roxx!";
    }
}
```

# Testy jednostkowe

- Testy wykonywane dla jednostki – JEDNA METODA
- Uruchamiane często
- Oszczędzają czas (jeeeej !! Oglądajmy śmieszne koty… szkoda że już nie bawią…)
- Usprawniają działanie systemu
- Dają pewną gwarancję niezawodności systemu
- Nienawidzi ich biznes

# Testy integracyjne

- Testy uruchamiane rzadko, najczęściej przed przekazania kodu do review lub w nocy
- Testują całą ścieżkę złożoną z wielu modułów
- Trwają bardzo długo

# JUNIT

- Standardowe narzędzie do testowania
- Pozwala pisać testy jednostkowe i integracyjne

# Assert

- Warunki na poprawność danych

# Przykład

```
    public int sum(int x, int y) {
        return x + y;
    }
```

```
@Test
public void shouldAddTwoDigits() {
    int a = 2, b = 4;
    int result = new UserProvider().sum(a,b);
    Assert.isTrue(result == 6, "Condition not correct");
}
```

# Ćwiczenie

Napisz 4 mikroserwisy komunikujące się ze sobą. Muszą one być odporne na zmianę adresów i portów serwerów.

W co najmniej jednym mikroserwisie napisz kilka prostych testów sprawdzających poprawne wyniki metody.

Twoim zadaniem jest stworzenie systemu do zarządzania kadrami.

Każdy pracownik powinien móc wziąć urlop na określoną ilość dni.

Pracownicy posiadają stanowisko i przypisane do nich biurko więc system powinien informować przy którym biurku siedzą.

Pracownicy podpisują zlecenia co w danym miesiącu robili.

TRANSITION
TECHNOLOGIES
MANAGED
SERVICES

GROUP
TRANSITION
TECHNOLOGIES

# DZIĘKUJĘ

**Krzysztof Łopucki**

**Krzysztof.Lopucki@ttms.pl**