TTMS
TRANSITION TECHNOLOGIES MANAGED SERVICES

GROUP TRANSITION TECHNOLOGIES

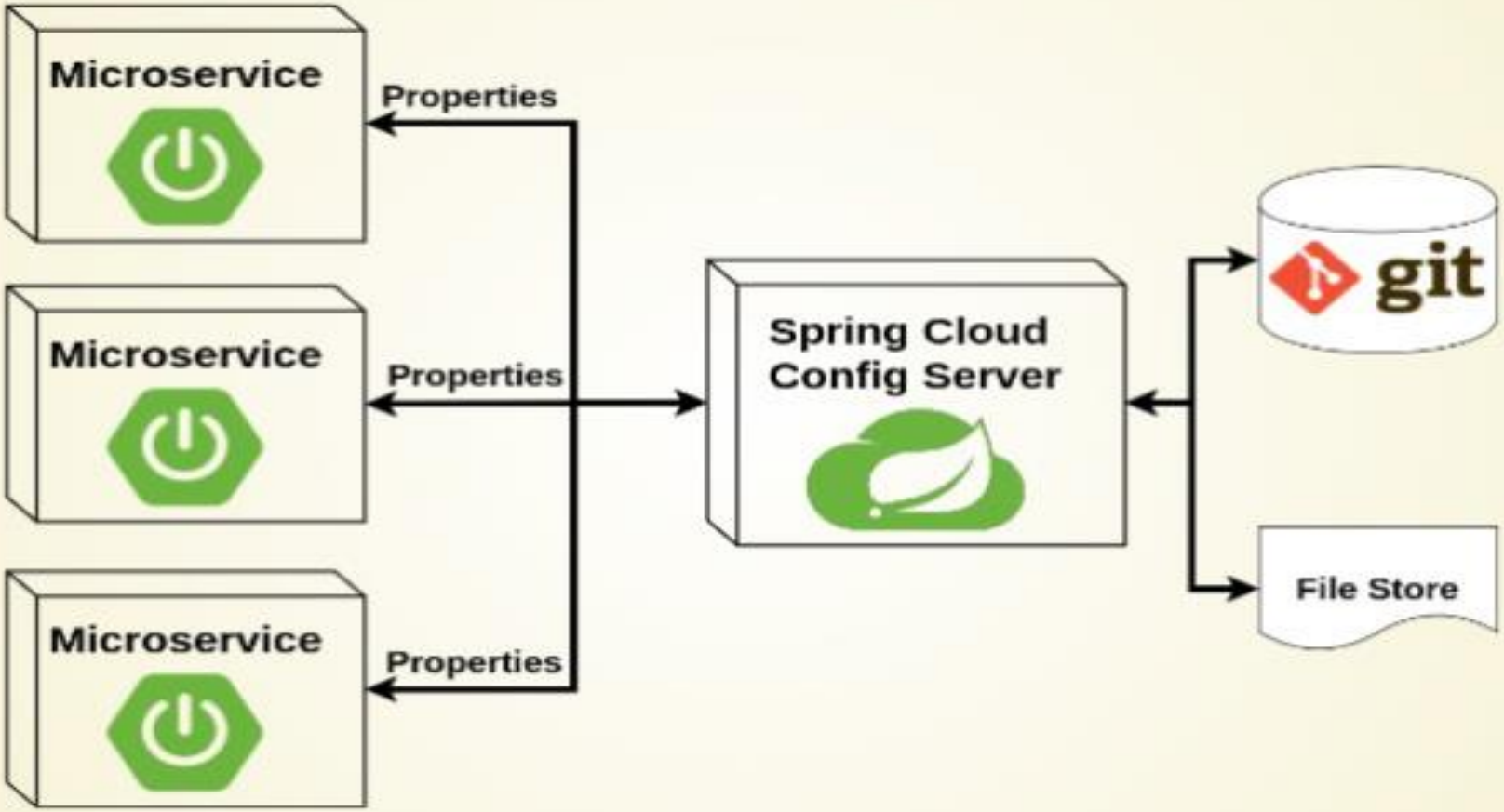
MIKROSERWISY CZ. 3

KRZYSZTOF ŁOPUCKI • 2018

Spring Cloud Config

- Propertiesy przechowywane w jednym miejscu
- Łatwa zmiana danych
- Dzilenie konfiguracji pomiedzy wiele mikroserwisów
- Zapobiegają niechcianym commitom z hasłami
- Automatyczne odświeżanie części kontekstu aplikacji

Spring Cloud Config



Zależności

- Zależność spring-cloud-config-serwer w celu uruchomienia serwera konfiguracji
- Serwer wystawia odpowiednie interfejsy restowe więc umożliwijmy mu to dodając zależność spring-boot-starter-web
- Do klasy z metodą main dodajemy adnotację **@EnableConfigServer**

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-config-server</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

Konfiguracja

– Do serwera dodajemy application.properties i odpowiednie wartości:

```
spring.application.name=configuration
server.port=8888
spring.cloud.config.server.git.uri=file:///${user.home}/config-cloud
spring.cloud.config.server.git.clone-on-start=true
```

– Nazywamy naszą aplikację, chociaż w przypadku serwera wpis ten jest nadmiarowy
– Standardowy port na jakim serwer powinien być uruchomiony to **8888**, mimo to serwer zadziała również na innym porcie
– Lokalizacja repozytorium git, w naszym przypadku niech to będzie ścieżka do katalogu z konfiguracją.
– Uruchamiamy opcję clonowania konfiguracji podczas uruchamiania serwera

Konfiguracje dla innich aplikacji

- Utwórzmy katalog z konfiguracją inicjując repozytorium git.
- Dodajemy do katalogu plik**. UWAGA!!!** Nazwa pliku ma być taka sama jak nazwa aplikacji klienta.
- Możemy do nazwy pliku dodać konfiguracje do konkretnego profilu.
- Np.: **config-client.dev.properties** – nasza aplikacja pobierająca ustawienia będzie się nazywała config-client, jest to konfiguracja na środowisko deweloperskie.
- Wklejamy zawartość pliku np.: **user.permissions=ALL**

Konfiguracja

- **UWAGA!!!** Podczas tworzenia aplikacji nazywamy ją tak jak nazwaliśmy plik. W naszym przypadku będzie to **config-client.**
- Dodajemy zależności maven.
  - Najważniejsza zależność to dodanie samej sależności konfiguracyjnej
  - Dostaniemy się do aplikacji przez api restowe więc dodajemy odpowiednią zależność
  - Potrzebujemy jeszcze aktuatora, który udostępni nam kilka endpointów, dzięki którym będziemy mogli wykonywać pewne czynności na aplikacji bez konieczności ingerencji w serwer.

Konfiguracja

```
<dependencies>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-config-client</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-actuator</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
</dependencies>
```

Konfiguracja

- Konfiguracja klienta przebiegnie nieco inaczej. Stworzony inny plik: **bootstrap.properties**, który zostanie załadwoany we wcześniejszym etapie uruchamiania aplikacji i dodamy propertiesy.

```
spring.application.name=config-client
spring.profiles.active=dev
spring.cloud.config.uri=http://localhost:8888
```

- **!!!Nazwa aplikacji znów musi być taka sama!!!**
- Aktywny profil. Oczywiście jeśli mamy konfiguracje z profilami.
- Adres serwera z konfiguracjami.

Konfiguracja

- Ostatnia rzecz to dodanie do aplikacji **application.properties.**
- Tam dodamy jeden wpis informujący aplikację o włączeniu dodatkowej konfiguracji np z aktuatorów.

```
management.endpoints.web.exposure.include=*
```

Konfiguracja

- Klasę z metodą main adnotujemy `@EnableAutoConfiguration`
- Pobieramy wartość z propertiesów:

```
@Value("${user.permissions}")
private String perm;
```

- Piszemy metodę pobierającą uprawnienie:

```
@GetMapping("/permission")
public String whoami() {
    return String.format("User permission %s", perm);
}
```

Uruchamiamy aplikacje

- Po uruchomieniu serwera sprawdzamy konfigurację wpisując w przeglądarce adres składający się z adresu serwera wraz z portem, nazwy aplikacji oraz profilu:

- http://localhost:8888/config-client/dev

- Tutaj już widać, że serwer złapał naszą konfigurację.

Uruchamiamy aplikacje

- Po uruchomieniu klienta w logach dostrzegamy informację o pobranych propertiesach:

  Fetching config from server at : http://localhost:8888

– Wywołujemy stworzony przez nas endpoint i naszym oczom w przeglądarce ukazują uprawnienia : **ALL**

Zmiana propertiestów

- Zmieniamy zawartość ustawień w client-config.dev.properties
- W normalnych warunkach commitujemy i wypychamy zawartość pliku.
- Nasze zmiany w bezpośrednio wskazanym pliku serwer konfiguracji sam zauważy.
- Pozostaje restart aplikacji klienta korzystającej z propertiesu

……

- Coś tu nie tak ….

@RefreshScope

- Dodajemy adnotację **@RefreshScope** do naszego kontrolera
- Restartujemy aplikację, żeby aplikacja była zbudowana z dodatkową adnotacją.
- Zmieniamy propertiesy.
- Uruchamiamy rządanie odświeżające zmienne:
  http://localhost:8080/refresh
  - Sprawdzamy czy zmienione zostały wartości ☺

Łączenie

- Konfiguracje możemy łączyć z innymi modułami spring cloud.
- Np.: możemy napisać aplikacje gdzie będziemy podmieniać połączenie z bazą danych.
- Albo przechowywać konfigurację nie w pliku tylko w bazie danych.

Ćwiczenie

- Napisz trzy aplikacje:

1. Aplikacja dostarczająca konfigurację (udostępnia dozwolone domeny, definiuje role)

2. Aplikacja udostępniająca zdefiniowaną rolę użytkownika (role pobrane z serwera konfiguracji)

3. Aplikacja pobierająca role użytkownika ale tylko w przypadku kiedy requestuje o nią użytkownik z konkretnej domeny (pobrane z serwera konfiguracji)

- Adres serwera 3 pobierającego role powinien być zdefiniowany w serwerze konfiguracji.
- Przetestuj aplikację zmieniając port aplikacji z rolami, a następnie odświeżając konfigurację na serwerze.

TRANSITION
TECHNOLOGIES
MANAGED
SERVICES

GROUP
TRANSITION
TECHNOLOGIES

DZIĘKUJĘ

Krzysztof Łopucki

Krzysztof.Lopucki@ttms.pl