

Spring Cloud. Микросервисная архитектура.









Евгений Манько

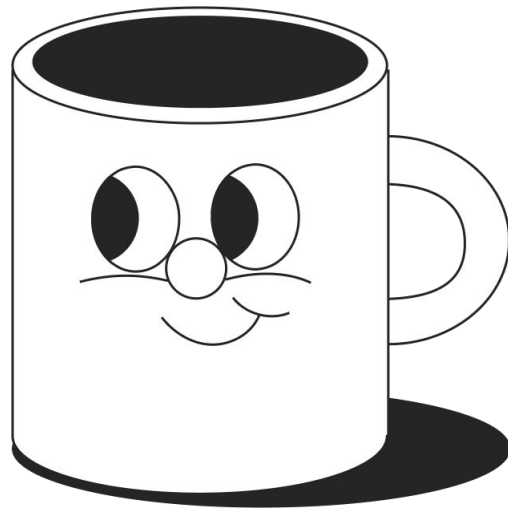
Java-разработчик, создатель данного курса

- ✨ Разрабатывал бэкенд для Яндекс, Тинькофф, МТС;
- ✨ Победитель грантового конкурса от «Росмолодежь»;
- ✨ Руководил IT-Департаментом «Студенты Москвы».



Что будет на уроке сегодня

-  Spring Cloud
-  Компоненты Spring Cloud
-  Eureka
-  Zulu
-  Spring Cloud Config
-  Hystrix





Популярность микросервисов

1. Быстрый time to market
2. Опытные команды
3. Масштабирование при минимальных затратах
4. Разделение ответственности
5. Нововведения и эксперименты



Spring Cloud

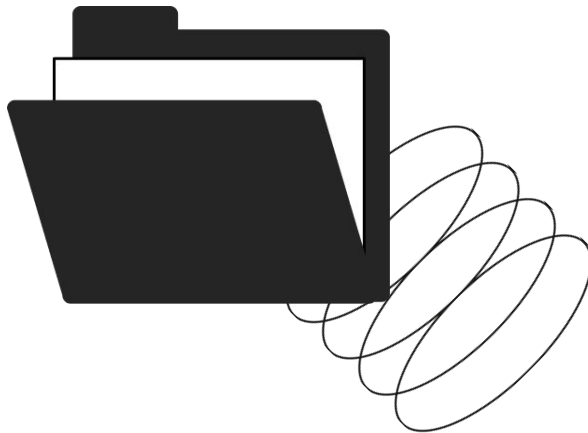
1. Упрощение разработки
2. Бесшовное взаимодействие
3. Отказоустойчивость





Компоненты Spring Cloud

1. Spring Cloud Config
2. Spring Cloud Netflix
3. Spring Cloud Gateway
4. Spring Cloud Bus





Eureka

Eureka — это система обнаружения сервисов,
которую можно представить как «телефонную книгу»
для ваших микросервисов.





Eureka



Зачем?

1. Динамичность
2. Балансировка нагрузки
3. Отказоустойчивость



Компоненты

1. Eureka Server
2. Eureka Client



Настройка Eureka Server

1

```
1 <dependency>
2   <groupId>org.springframework.cloud</groupId>
3   <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
4 </dependency>
```

2

```
1 @SpringBootApplication
2 @EnableEurekaServer
3 public class EurekaServerApplication {
4     public static void main(String[] args) {
5         SpringApplication.run(EurekaServerApplication.class, args);
6     }
7 }
```

3

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



Регистрация микросервиса как Eureka Client Интеграционные тесты

1

```
1 <dependency>
2   <groupId>org.springframework.cloud</groupId>
3   <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
4 </dependency>
```

2

```
1 @SpringBootApplication
2 @EnableEurekaClient
3 public class MyMicroserviceApplication {
4     public static void main(String[] args) {
5         SpringApplication.run(MyMicroserviceApplication.class, args);
6     }
7 }
```

3

```
1 eureka.client.service-url.defaultZone=http://localhost:8761/eureka/
```



Zuul

Zuul — это API Gateway, что в переводе можно назвать «шлюзом для API».

В чем польза?

1. Маршрутизация
2. Фильтрация
3. Защита
4. Отказоустойчивость



Zuul

В основе работы Zuul лежат фильтры.
Есть несколько типов фильтров:

1. Пред-фильтры (Pre Filters)
2. Фильтры маршрутизации (Route Filters)
3. Пост-фильтры (Post Filters)
4. Фильтры ошибок (Error Filters)



Настройка Zuul как API Gateway

1

```
1 <dependency>
2   <groupId>org.springframework.cloud</groupId>
3   <artifactId>spring-cloud-starter-netflix-zuul</artifactId>
4 </dependency>
5 <dependency>
6   <groupId>org.springframework.cloud</groupId>
7   <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
8 </dependency>
```

2

```
1 @SpringBootApplication
2 @EnableZuulProxy
3 public class ZuulGatewayApplication {
4     public static void main(String[] args) {
5         SpringApplication.run(ZuulGatewayApplication.class, args);
6     }
7 }
```

3

```
1 eureka.client.service-url.defaultZone=http://localhost:8761/eureka/
2
3 zuul.routes.myservice.url=http://localhost:8080|
```



Применение фильтров в Zuul

```
1 @Component
2 public class SimplePreFilter extends ZuulFilter {
3
4     private static final Logger log = LoggerFactory.getLogger(SimplePreFilter.class);
5
6     @Override
7     public String filterType() {
8         return "pre";
9     }
10
11     @Override
12     public int filterOrder() {
13         return 1;
14     }
15
16     @Override
17     public boolean shouldFilter() {
18         return true;
19     }
20
21     @Override
22     public Object run() {
23         RequestContext ctx = RequestContext.getCurrentContext();
24         HttpServletRequest request = ctx.getRequest();
25
26         log.info(String.format("%s request to %s", request.getMethod(), request.getRequestURL().toString()));
27
28         return null;
29     }
30 }
```



Spring Cloud Config



Как это работает?

1. Централизованное хранилище
2. Динамическое обновление
3. Безопасность



Почему так ценно?

1. Упрощение управления
2. Быстрое внедрение изменений
3. Масштабируемость и единообразие



Настройка Config Server

1

```
1 <dependency>
2   <groupId>org.springframework.cloud</groupId>
3   <artifactId>spring-cloud-config-server</artifactId>
4 </dependency>
```

2

```
1 @SpringBootApplication
2 @EnableConfigServer
3 public class ConfigServerApplication {
4     public static void main(String[] args) {
5         SpringApplication.run(ConfigServerApplication.class, args);
6     }
7 }
```

3

```
1 spring.cloud.config.server.git.uri=URL_ВАШЕГО_РЕПОЗИТОРИЯ
```




Использование Config Client

1

```
1 <dependency>
2     <groupId>org.springframework.cloud</groupId>
3     <artifactId>spring-cloud-starter-config</artifactId>
4 </dependency>
```

2

```
1 spring.cloud.config.uri=http://АДРЕС_ВАШЕГО_CONFIG_SERVER
```



Обновление конфигурации в реальном времени

```
1 <dependency>
2     <groupId>org.springframework.cloud</groupId>
3     <artifactId>spring-cloud-starter-bus-amqp</artifactId>
4 </dependency>
```



Hystrix

Hystrix — это библиотека из мира Spring Cloud, которая предоставляет «обертку» вокруг ваших внешних вызовов.

Функции:

1. Обнаружение ошибок
2. Обход ошибок
3. Ограничение потоков
4. Резервные стратегии
5. Мониторинг и метрики



Подключение Hystrix

```
1 <dependency>
2     <groupId>org.springframework.cloud</groupId>
3     <artifactId>spring-cloud-starter-netflix-hystrix</artifactId>
4 </dependency>
```



Активация Hystrix

```
1 @SpringBootApplication
2 @EnableCircuitBreaker
3 public class Application {
4     public static void main(String[] args) {
5         SpringApplication.run(Application.class, args);
6     }
7 }
```



Защита метода с помощью Hystrix

```
1 public String fetchData() {  
2     // ... вызов внешнего сервиса  
3     return "Data from external service";  
4 }
```

```
1 @HystrixCommand(fallbackMethod = "defaultData")  
2 public String fetchData() {  
3     // ... вызов внешнего сервиса  
4     return "Data from external service";  
5 }  
6  
7 public String defaultData() {  
8     return "Default data";  
9 }
```



Настройка Hystrix

Hystrix предоставляет множество настроек для тонкой настройки.
Например, в application.properties:

```
1 hystrix.command.default.execution.isolation.thread.timeoutInMilliseconds=2000|
```



Мониторинг с Hystrix Dashboard

1

```
1 <dependency>
2   <groupId>org.springframework.cloud</groupId>
3   <artifactId>spring-cloud-starter-netflix-hystrix-dashboard</artifactId>
4 </dependency>|
```

2

```
1 @SpringBootApplication
2 @EnableHystrixDashboard
3 public class DashboardApplication {
4     public static void main(String[] args) {
5         SpringApplication.run(DashboardApplication.class, args);
6     }
7 }|
```




Спасибо за внимание

