# BOM

## Создание репозитория на git

<https://github.com/smilyk/atsarat-briut-microservices>

1. Клонируем и открываем этот репозиторий в интелледж

smilykq@MacBook-Pro ~ % cd Desktop

smilykq@MacBook-Pro Desktop % cd AtsaratBriut

smilykq@MacBook-Pro AtsaratBriut % ls

AtcaratBriut\_plan

AtcaratBriut\_plan.docx

children-service

email-service

gymnast-server

scheduler-service

school-service

tsofim-service

user-service

~$caratBriut\_plan.docx

~$er-service-documentation.docx

~$щая документация.docx

Общая документация

Общая документация.docx

smilykq@MacBook-Pro AtsaratBriut % mkdir atsarat

smilykq@MacBook-Pro AtsaratBriut % cd atsarat

smilykq@MacBook-Pro atsarat % git clone https://github.com/smilyk/atsarat-briut-microservices.git

Клонирование в «atsarat-briut-microservices»…

warning: Похоже, что вы клонировали пустой репозиторий.

smilykq@MacBook-Pro atsarat %

Изображение выглядит как текст

Автоматически созданное описание

<https://github.com/smilyk/atsarat-briut-microservices.git>

1. Открываем в интелледже ->New project from maven
2. pom.xml, .gitignore, readme
3. удалить src
4. добавляем модули
   1. Изображение выглядит как текст

      Автоматически созданное описание
5. git push

## Общий pom –

1. Все одинаковые зависимости убираются из pom сервисов и добавляется в общий родительский pom
2. Каждый сервис получает свой порт и свое имя

# ACTUATOR

<!-- ACTUATOR-->  
<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-actuator</artifactId>  
</dependency>

#actuator  
spring.jackson.serialization.indent\_output=true – нормальный внешний вид  
management.endpoints.web.base-path=/monitor – изменили ссылку. Теперь вместо …/actuator -> …/monitor

GET http://localhost:8082/monitor  
management.endpoints.web.exposure.include=\* - включили все конечные точки  
management.endpoint.shutdown.enabled=true – подключение точки shutdown  
POST http://localhost:8082/monitor/shutdown  
info.application.name= user-service actuator  
info.application.description=project "Atsarat-Briut"  
info.application.version=0.0.1

Если все ОК, то при старте мы увидим:

Exposing 14 endpoint(s) beneath base path '/monitor'

# RABBITMQ очередь

1. Открываем раббит в докере

|  |  |
| --- | --- |
| docker run -d --hostname rabbitmq --name rabbitmq -p 5672:5672 -p 15672:15672 -p 15674:15674 -p 25672:25672 -p 61613:61613 -v rabbitmq\_data:/var/lib/rabbitmq -e RABBITMQ\_DEFAULT\_USER=user -e RABBITMQ\_DEFAULT\_PASS=password rabbitmq:3.6.14-management | Создаем и запускаем контейнер  логин:user  пароль:password |

1. В общий pom добавляем зависимость:

<!-- rabbitMQ-->  
 <dependency>  
 <groupId> org.springframework.boot </groupId>  
 <artifactId> spring-boot-starter-amqp </artifactId>  
 </dependency>  
<!--end of rabbitMQ-->

## Настраиваем сервис Sheduler.

1. Сервис будет создавать 3 очереди –

* schoolQueue
* gymnastQueue
* tsofimQueue

1. application.properties
2. #rabbitMQ  
   spring.rabbitmq.host = localhost  
   spring.rabbitmq.port = 5672  
   spring.rabbitmq.username = user  
   spring.rabbitmq.password = password  
     
   tsofim.key=ts  
   school.key=sc  
   gymnsat.key=gym  
     
   tsofim.queue=TSOFIM  
   school.queue=SCHOOL  
   gymnsat.queue=GYMNAST  
     
   rabbitmq.exchange=atBriut
3. пишем конфигурацию для producer (class RabbitProducerConfig)

@Bean этот бин создает exchange в рабите ()то есть перераспределение – как бы канал, на который передаются все сообщения  
DirectExchange exchange() {  
 String name = exchange;  
 return new DirectExchange(name, true, false);  
}

После того, как создан exchange необходимо создать бин, который будет отправлять в конкретную очередь с конкретным ключом конкретное сообщение

@Bean  
public AmqpTemplate rabbitTemplate(ConnectionFactory connectionFactory) {  
 final RabbitTemplate rabbitTemplate = new RabbitTemplate(connectionFactory);  
 rabbitTemplate.setMessageConverter(jsonMessageConverter());  
 return rabbitTemplate;  
}

Для того, что бы сообщение передавалось в формате Json а не требовало String необходим конвертер

@Bean  
public MessageConverter jsonMessageConverter() {  
 return new Jackson2JsonMessageConverter();  
}

Все эти три функции хранятся в классе RabbitProducerConfig

1. пишем сервис

Класс RabbitServiceImpl

Для того, что бы передавать сообщения в конкретную очередь с конкретным ключом – необходимо сначала создать эту очередь и привязать к ранее созданному exchange конкретную очередь с ее ключом

Метод createQuene();

Создаем новую ConnectionFactory для того, что бы переопределить ее. Нам не нужна фабрика по умолчанию, нам нужна та, которая сделает то что мы ей скажем.

Устанавливаем пароль и имя пользователя (так как у нас не стандартный набор guest/guest)

ConnectionFactory factory = new ConnectionFactory();  
factory.setPassword(rabbitPassword);  
factory.setUsername(rabbitUserName);

Создаем соединение и канал

try (Connection connection = factory.newConnection();  
 Channel channel = connection.createChannel()) {

Создаем три очереди

channel.queueDeclare(tsofimQueue, false, false, false, null);  
channel.queueDeclare(schoolQueue, false, false, false, null);  
channel.queueDeclare(gymnsatQueue, false, false, false, null);

Теперь нужно привязать очереди к exchange для этого – нужно определить exchange к которому мы будем «байндить»

channel.exchangeDeclare(exchange, *type*, true);

type в данном случае – это direct (то есть сообщение будут передаваться по ключу в свою очередь. Тут может быть любой, необходимый тип из четырех возможных)

И наконец создаем привязки. К каждой очереди по ее ключу

channel.queueBind(tsofimQueue, exchange, tsofimRoutingkey);  
channel.queueBind(schoolQueue, exchange, schoolRoutingkey);  
channel.queueBind(gymnsatQueue, exchange, gymnastRoutingkey);

## Настраиваем сервис GYMNAST.

У сервиса будет 2 задачи.

* Первая – он будет слушать и принимать сообщения от сервиса Scheduler, которые будут ему отправлены (только свои сообщения)
* Вторая – он будет создавать сообщения и отправлять их сервису email.

### Consumer на прием сообщения от Scheduler

1. Application.properties
2. #rabbitMQ  
   spring.rabbitmq.host = localhost  
   spring.rabbitmq.port = 5672  
   spring.rabbitmq.username = user  
   spring.rabbitmq.password = password  
     
     
   gymnsat.key=gym  
   gymnsat.queue=GYMNAST  
   rabbitmq.exchange=atBriut  
     
   email.key=email  
   email.queue=EMAIL  
   email.exchange=Email

!!! Ключ, очередь и exchange должны быть такими же как у scheduler service, который отправляет сообщения.

1. пишем конфигурацию для consumer (class RabbitConsumerConfig)

создаем очередь для получения сообщений (что бы не падать, если вдруг очередь почему то удалена из Rabbit)

@Bean  
 public void createGymnastQueue() {  
 ConnectionFactory factory = new ConnectionFactory();  
 factory.setPassword(rabbitPassword);  
 factory.setUsername(rabbitUserName);  
 try (Connection connection = factory.newConnection();  
 Channel channel = connection.createChannel()) {  
 channel.queueDeclare(gymnsatQueue, false, false, false, null);  
 channel.exchangeDeclare(exchange, *type*, true);  
 channel.queueBind(gymnsatQueue, exchange, gymnastRoutingkey);  
 } catch (TimeoutException | IOException e) {  
 e.printStackTrace();  
 }  
}

Создаем очередь для отправки письма

@Bean  
public void createEmailQueue() {  
 ConnectionFactory factory = new ConnectionFactory();  
 factory.setPassword(rabbitPassword);  
 factory.setUsername(rabbitUserName);  
 try (Connection connection = factory.newConnection();  
 Channel channel = connection.createChannel()) {  
 channel.queueDeclare(emailQueue, false, false, false, null);  
 channel.exchangeDeclare(emailExchange, *type*, true);  
 channel.queueBind(emailQueue, emailExchange, emailRoutingkey);  
 } catch (TimeoutException | IOException e) {  
 e.printStackTrace();  
 }  
}

Создаем слушателя

@Override  
@RabbitListener(queues = "${gymnsat.queue}")  
public void receivedMessage(RabbitDto incomingMessage) {  
 String uuidChild = incomingMessage.getUuidChild();  
 gymnastCrawlerService.sendFormToGymnast(uuidChild);  
 *LOGGER*.info(LoggerMessages.*GET\_ATSARAT\_BRIUT* + LoggerMessages.*CHILD* + LoggerMessages.*WITH\_UUID* + uuidChild);  
 System.*out*.println("Recieved Message From RabbitMQ: " + incomingMessage.getUuidChild());  
}

### PRODUCER на отправку сообщения в EMAIL

@Override  
public void sendToEmailService(EmailDto emailDto) {  
 rabbitTemplate.convertAndSend(emailExchange, emailRoutingkey, emailDto);  
}

По такому же принципу пишем остальные сервисы. Tsofim и Schools – на прием и отправку, Email – на прием сообщений

# Eureka service

1. в общий pom добавляем зависимость для спринг бута

<!--for spring cloud-->  
<dependencyManagement>  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-dependencies</artifactId>  
 <version>${spring-cloud.version}</version>  
 <type>pom</type>  
 <scope>import</scope>  
 </dependency>  
 </dependencies>  
</dependencyManagement>  
<!--end for spring cloud-->

и зависимость для клиента эврики (так как все сервисы будут клиентами эврики)

<!-- eureka-client-->  
<dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>  
</dependency>  
<!-- end of eureka-client-->

1. создаем модуль eureka-service
2. в нем в pom добавляем зависимость для сервера

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

1. в  application.properties

server.port=8761  
  
#  
##eureka  
eureka.client.fetch-registry=false   
eureka.client.register-with-eureka=false – запрещаем эврике регистрировать себя как клиента (нам это не нужно)  
eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka – по этой ссылке все остальные будут конекится к эврике

1. main

@EnableEurekaServer

# Eureka client

Все сервисы – в application.properties добавляем

#eureka  
eureka.client.serviceUrl.defaultZone=${EUREKA\_URL:http://localhost:8761/eureka}  
eureka.instance.instance-id=${spring.application.name}:${spring.application.instance\_id:${random.value}}

Так как раньше мы уже добавили в pom зависимость для клиента эврики, нужно просто в main ->

@EnableEurekaClient

# Gateway - service

## Построение

1. Новый модуль
2. Pom

<dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-netflix-zuul</artifactId>  
</dependency>

1. Application-properties

server.port=8011  
spring.application.name=zuul-service  
  
##eureka  
eureka.client.serviceUrl.defaultZone=${EUREKA\_URL:http://localhost:8761/eureka}  
eureka.instance.instance-id=${spring.application.name}:${spring.application.instance\_id:${random.value}}

1. Main

@EnableEurekaClient  
@EnableZuulProxy

## Проверка

1. Start eureka-service
2. Start zuul-service
3. Start user-service
4. Postman

localhost:8011/user-service/monitor

1. Результат – вывод актуатора user-service

Изображение выглядит как текст

Автоматически созданное описание

# Config-server - local

## Server

1. Создаем модуль, в котором backend файловая система

<dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-config-server</artifactId>  
</dependency>

1. Application.properties

spring.application.name=config-server  
server.port=8012  
spring.profiles.active=native //нужен для того, что бы переопределить и заставить сервисы брать данные не из git, а с рабочего стола  
spring.cloud.config.server.native.search-locations=file://${user.home}/Desktop/AtsaratBriut/atsarat-briut-microservices/config-server-repo

1. Main

@SpringBootApplication  
@EnableConfigServer

1. Start Eureka-service
2. Start Config-service
3. Должны увидеть в Eureka что конфиг сервер подключился
4. Создаем в папке config-server-repo файл application.properties в этот файл нужно перенести все свойства, которые касаются ВСЕХ сервисов

Например – эврика и актуатор (они для всех сервисов одинаковые)

1. Создаем в папке config-server-repo файл user-service.properties в этот файл нужно перенести все свойства из application.properties из user-service

!!! Название файла в config-server-repo должно совпадать с spring.application.name в user-service!!!

## Client

1. Pom –

<dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-config</artifactId>  
</dependency>

1. В сервисе User-service создаем файл bootstrap.properties

spring.cloud.config.uri=http://localhost:8012  
spring.cloud.config.name=user-service

## Проверка

1. Запуск эврика
2. Запуск конфиг

Должны увидеть

main] com.config.ConfigServiceApplication : The following profiles are active: native

1. Запуск user-service

Должны увидеть

Fetching config from server at : <http://localhost:8012>

и

2020-10-18 16:37:35.131 INFO 8707 --- [ main] b.c.PropertySourceBootstrapConfiguration : Located property source: [BootstrapPropertySource {name='bootstrapProperties-file:///Users/smilykq/Desktop/AtsaratBriut/atsarat-briut-microservices/config-server-repo/user-service.properties'}]

То же самое нужно сделать со всеми сервисами.

После этого, одинаковые свойства выносятся в application.properties

# Config-server – git

Конфигурация для гита – в конфиг сервисе:

#git  
#spring.cloud.config.server.git.uri=https://github.com/smilyk/home-accounting-2-repo  
#spring.cloud.config.server.git.username=smilyk  
#spring.cloud.config.server.git.password=Liza200811!  
#spring.cloud.config.server.git.clone-on-start=true

# !!! так как git изменил настройки и теперь в новых проектах нет ветки мастер – необходимо всем сервисам сказать, что они теперь ищут данные с label =main

spring.cloud.config.label=main

# Monitoring config

1. Config-server –

<dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-bus-amqp</artifactId>  
</dependency> - нужен для того, что бы отслеживать изменения и перегружать только то, что необходимо  
<dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-config-monitor</artifactId>  
</dependency> - нужен для того, что бы отслеживать все события и перегружать все сервисы, используя refresh config-server только. Остальные сервисы перегрузят данные сами

1. @Bean  
   GithubPropertyPathNotificationExtractor GithubPropertyPathNotificationExtractor(){  
    return new GithubPropertyPathNotificationExtractor();  
   } – нужен для того, что бы конфиг сервер отслеживал изенения, которые происходят на гите в репозитории свойств
2. Client-server

<!-- spring cloud bus-->  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-bus-amqp</artifactId>  
 </dependency>  
<!-- end of spring cloud bus-->

1. Над тем классом, который нужно будет обновлять – ставим аннотацию

@RefreshScope

### Проверка monitor (т е перезагрузка только конфиг сервера)

* 1. Вызываем

http://localhost:8090/users/v1/ping

* 1. Получаем

Users-Service working 2020-10-21 secret word is: "MyTokenSecret"

* 1. На гите меняем значение tokenSecret на «111»
  2. Вызываем

<http://localhost:8012/monitor>

!!!!!! это актуатор КОНФИГ\_СЕРВЕРА

это POST

в хедерс прописываем

X-Github-Event:push

и в теле метода передаем

|  |
| --- |
| {  "commits":[  {  "modified":["user-service.properties"]  }  ]  }  Где user-service.properties – название того списка свойств, которое мы отслеживаем(то есть которое изменили) |

* 1. В консоли user service видим

Keys refreshed [config.client.version, tokenSecret] – что говорит о том, что tokenSecretдолжен быть перегружен.

* 1. Вызываем

http://localhost:8090/users/v1/ping

* 1. Получаем

Users-Service working 2020-10-21 secret word is: "111" – то есть свойство изменилось!!!

### Проверка monitor (т е перезагрузка только конфиг сервера)

1. Вызываем

http://localhost:8090/users/v1/ping

1. Получаем

Users-Service working 2020-10-21 secret word is: "111"

1. На гите меняем значение tokenSecret на «222»
2. Вызываем

<http://localhost:8090/actuator/bus-refresh>

POST

это актуатор USER\_SERVICE

1. Получаем 204 – No Content
2. В консоли user service видим

Keys refreshed [config.client.version, tokenSecret] – что говорит о том, что tokenSecretдолжен быть перегружен.

1. Вызываем

http://localhost:8090/users/v1/ping

1. Получаем

Users-Service working 2020-10-21 secret word is: "222"

# Hystrix

Пример – хистрикс между user-service и email-service.

user servise шлет запрос email-service что бы отправить письмо

## User-client

1. Pom

<!-- hystrix-->  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-netflix-hystrix</artifactId>  
 </dependency>  
<!-- end of hystrix-->