Documentation de la Phase 2 du Projet de Réseau Social en Microservices

Introduction

Cette documentation décrit la deuxième phase du projet de réseau social, qui se concentre sur l'optimisation de l'architecture en microservices. Cette phase vise à intégrer des technologies spécifiques pour renforcer l'application et garantir une performance optimale

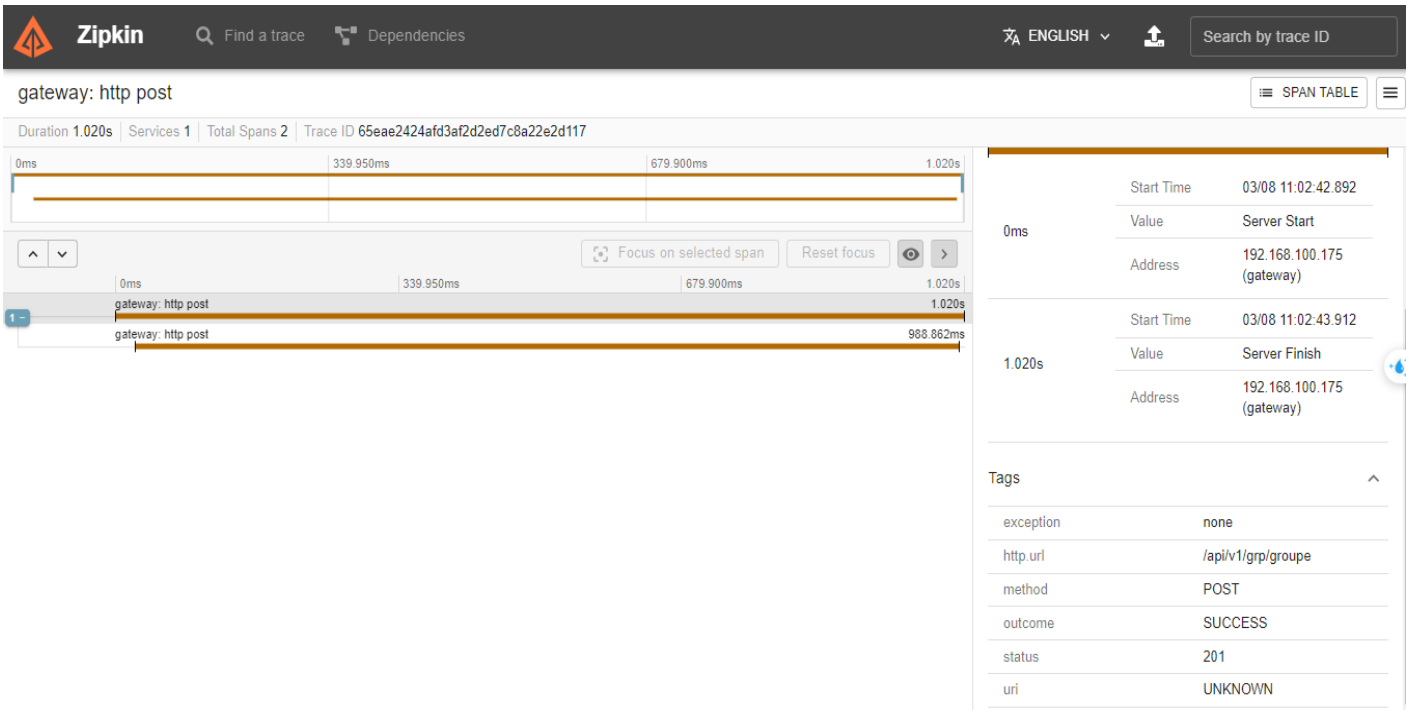
Objectifs

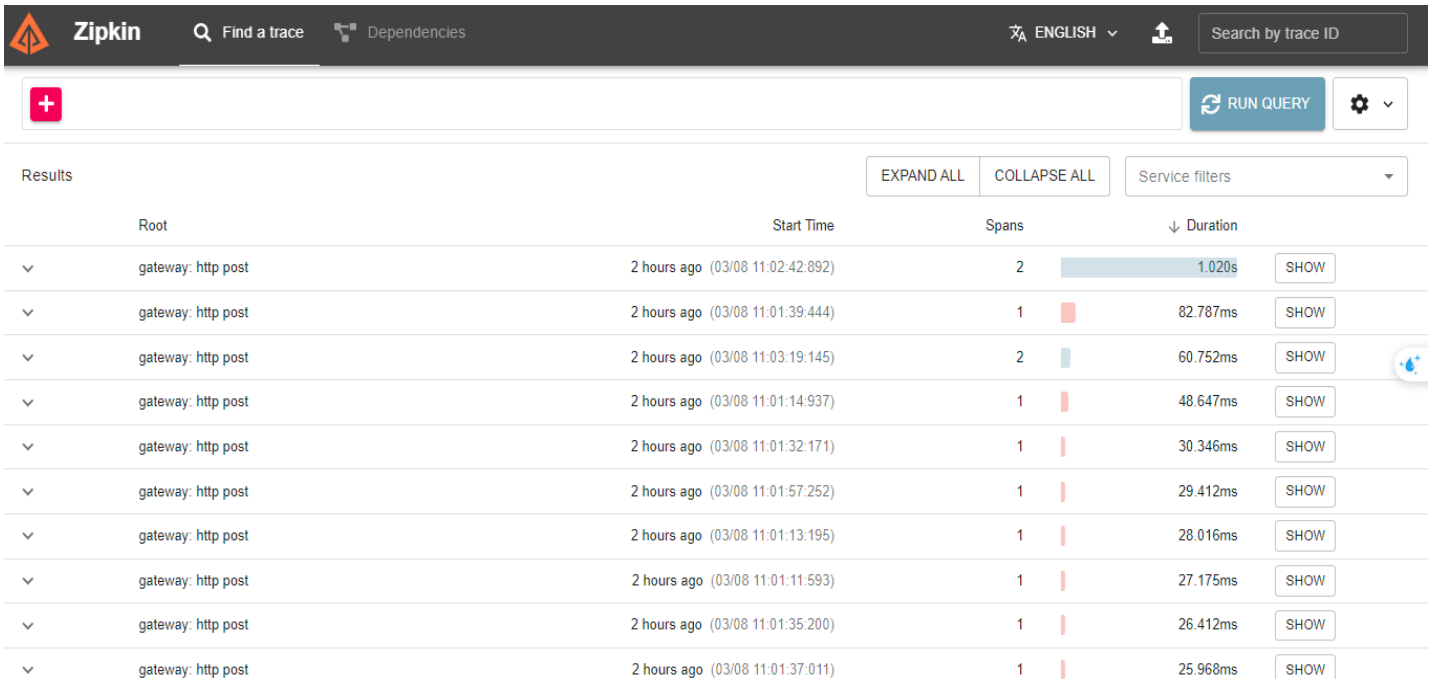
- Optimisation de l'architecture en microservices : Améliorer la performance, la résilience et la scalabilité de l'application.
- Intégration de technologies avancées : Utiliser Zipkin, Spring Cloud Gateway, Resilience4j, Spring Cloud Load Balancer et Apache Kafka pour renforcer l'application.
- Amélioration de l'expérience utilisateur et de l'efficacité opérationnelle : Assurer une application robuste, flexible et hautement performante.

Technologies Intégrées

Zipkin

- Objectif : Traçage des requêtes pour surveiller les performances des microservices.
- Fonctionnalités : Identification des goulots d'étranglement, optimisation du débit global.


Zipkin
Find a trace
Dependencies
ENGLISH
Search by trace ID
gateway: http post
SPAN TABLE
Duration 1.020s | Services 1 | Total Spans 2 | Trace ID 65eae2424afd3af2d2ed7c8a22e2d117
0ms
339.950ms
679.900ms
1.020s
Focus on selected span
Reset focus
gateway: http post 1.020s
gateway: http post 988.862ms
0ms
Start Time 03/08 11:02:42.892
Value Server Start
Address 192.168.100.175 (gateway)
1.020s
Start Time 03/08 11:02:43.912
Value Server Finish
Address 192.168.100.175 (gateway)
Tags
exception none
http.url /api/v1/grp/groupe
method POST
outcome SUCCESS
status 201
uri UNKNOWN



Spring Cloud Gateway

- Objectif : Gestion du trafic entre les microservices.
- Fonctionnalités : Distribution équilibrée des charges, sécurité renforcée grâce à l'authentification et l'autorisation centralisées.

Resilience4j

- Objectif : Amélioration de la résilience des microservices.
- Fonctionnalités : Circuit breaker, rétentive, fallback pour tolérance aux pannes et récupération gracieuse.

```
{
  "status": "UP",
  "components": {
    "circuitBreakers": {
      "status": "UP",
      "details": {
        "likeservice": {
          "status": "UP",
          "details": {
            "failureRate": "-1.0%",
            "failureRateThreshold": "50.0%",
            "slowCallRate": "-1.0%",
            "slowCallRateThreshold": "100.0%",
            "bufferedCalls": 0,
            "slowCalls": 0,
            "slowFailedCalls": 0,
            "failedCalls": 0,
            "notPermittedCalls": 0,
            "state": "CLOSED"
          }
        }
      }
    }
```

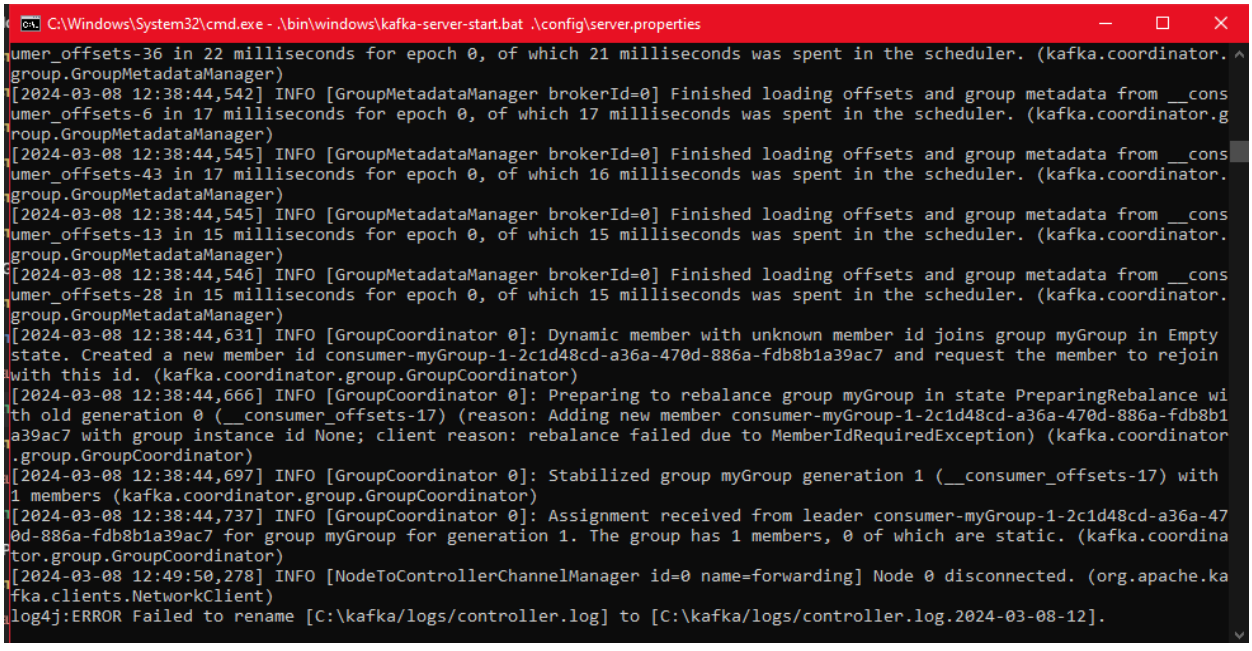
```
{
  "status": "UP",
  "components": {
    "circuitBreakers": {
      "status": "UP",
      "details": {
        "userserice": {
          "status": "UP",
          "details": {
            "failureRate": "-1.0%",
            "failureRateThreshold": "50.0%",
            "slowCallRate": "-1.0%",
            "slowCallRateThreshold": "100.0%",
            "bufferedCalls": 0,
            "slowCalls": 0,
            "slowFailedCalls": 0,
            "failedCalls": 0,
            "notPermittedCalls": 0,
            "state": "CLOSED"
          }
        }
      }
    }
```

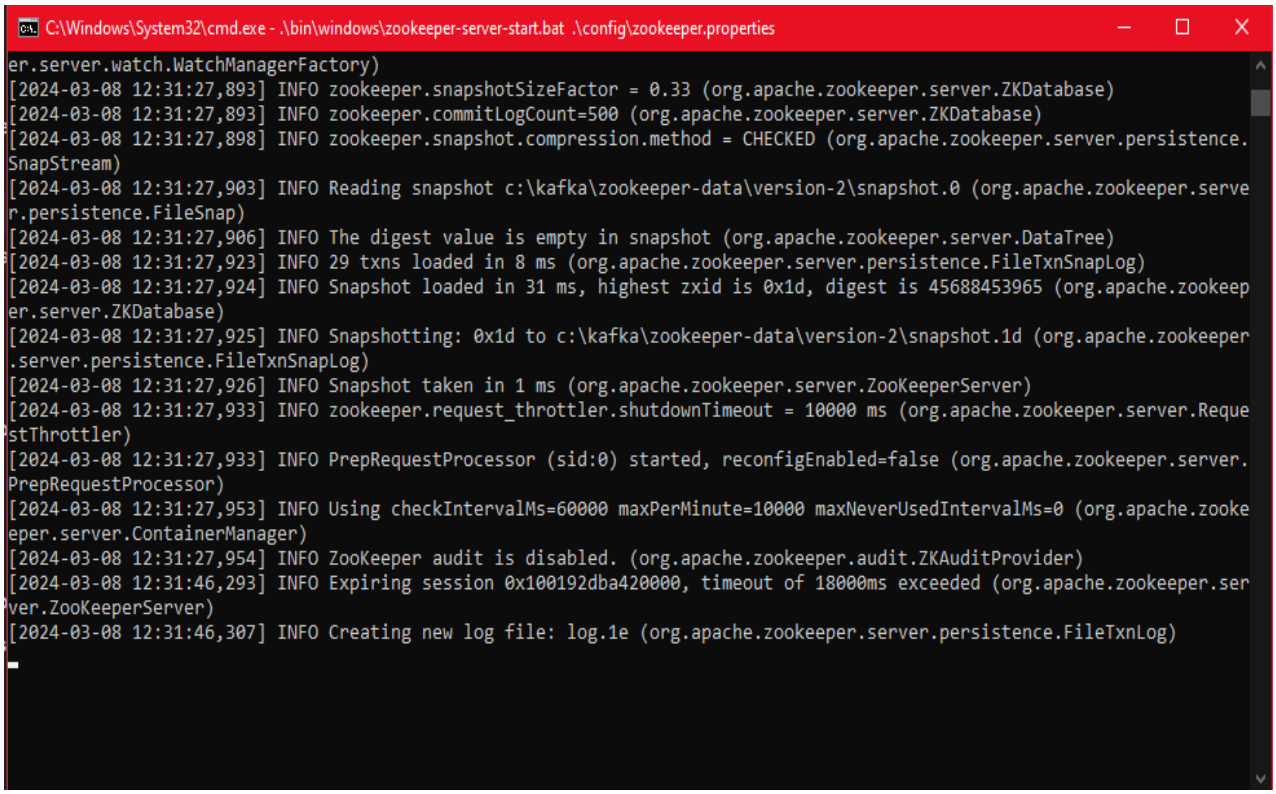
Spring Cloud Load Balancer

- Objectif : Optimisation de la répartition de charge.
- Fonctionnalités : Utilisation efficace des ressources, scalabilité horizontale.

Apache Kafka

- Objectif : Gestion de la messagerie entre microservices.
- Fonctionnalités : Communication asynchrone, orchestration efficace des événements.






```java
@Configuration
public class KafkaTopicConfig {

    @Value("quickstart-events")
    private String topicName;

    // spring bean for kafka topic
    @Bean
    public NewTopic topic(){
        return TopicBuilder.name(topicName)
                .build();
    }
}
```

```java
@Service
public class postProducer {

    private static final org.slf4j.Logger LOGGER = LoggerFactory.getLogger(postProducer.class);

    @Autowired
    private KafkaTemplate<String, PostDto> kafkaTemplate;

    public void sendMessage(PostDto event){
        LOGGER.info(String.format("Post event => %s", event.toString()));

        Message<PostDto> message = MessageBuilder
                .withPayload(event)
                .setHeader(KafkaHeaders.TOPIC, headerValue: "quickstart-events")
                .build();
        kafkaTemplate.send(message);
    }
}
```

```java
public cl

    @Autowired
    private IGroupeService groupeService;

    @KafkaListener(topics = "quickstart-events", groupId = "myGroup", containerFactory = "kafkaListenerContainerFactory")
    public void consume(PostDto postDto) {
        GroupeDto group = groupeService.getGroupeById(postDto.getGroupeId());
        groupeService.deleteGroupe(postDto.getGroupeId(),group.getAdminId());
        System.out.println("Consumed message: upp" + postDto);
    }
}
```

Conclusion

L'intégration de ces technologies avancées est cruciale pour consolider l'architecture en microservices et garantir une application de réseau social moderne, robuste, flexible et hautement performante. Cette phase du projet vise à atteindre de nouveaux sommets en termes d'efficacité opérationnelle et d'expérience utilisateur.