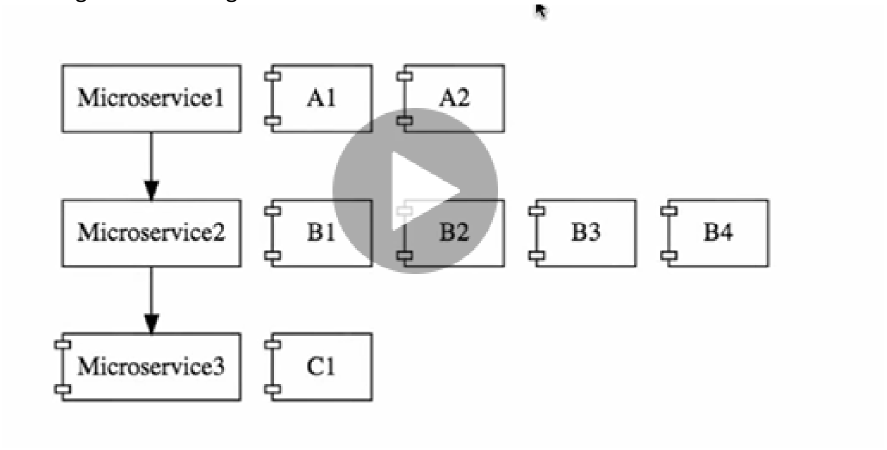
Saturday, December 26, 2020 2:55 AM

Microservice

1.Bounded Context

Earlier we were building one monolthic app now 10-20 microservicce
How to identify what to do in each of microservices
What to do or not?

2.Configuration Management



5 microservice with different number of instances

3 Dynamic scale up and scale down

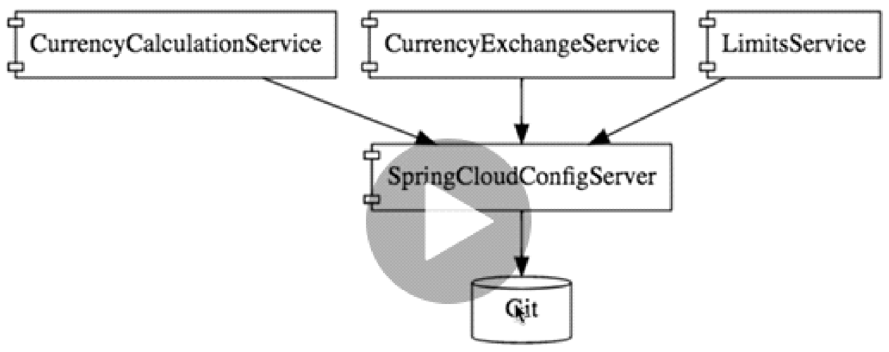
Load different instance at different time
At particular time I want two instance and after that I don't. Bring down when we don't want

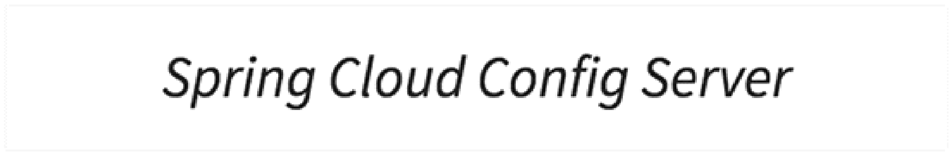
4. Visibility

How to identify where is bug. We want centralized and monitor tha logs and server which is up and down

5 Pack of cards
If one microservice fail then all will fail

Spring cloud

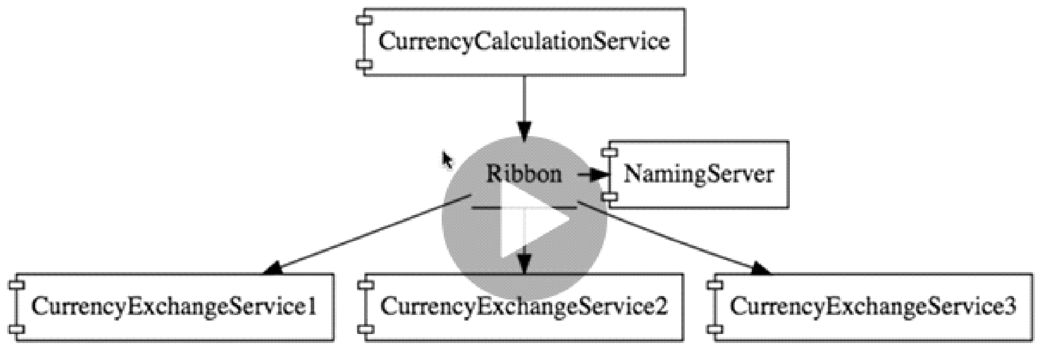


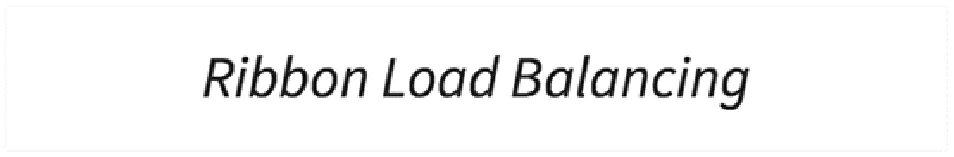


Spring Cloud Config Server

We can store all the configuration different server and instance to one place…And easy to maintain

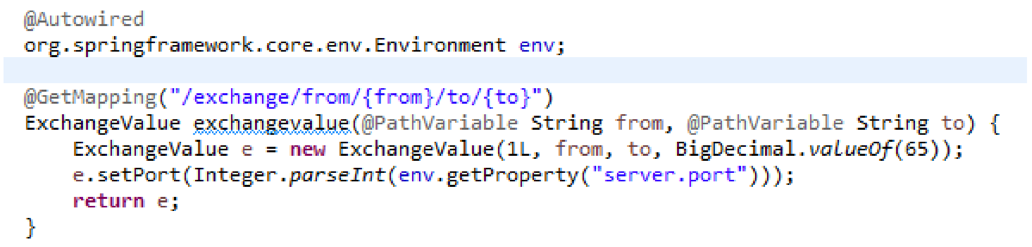
Dynamic scale up and down



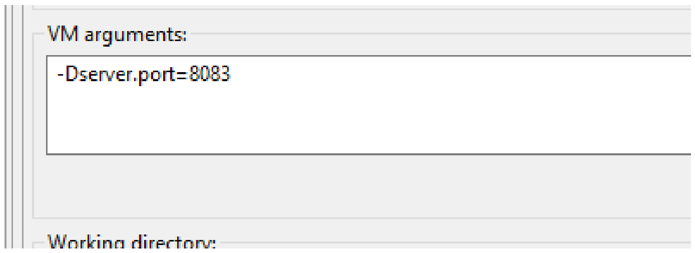


Ribbon Load Balancing

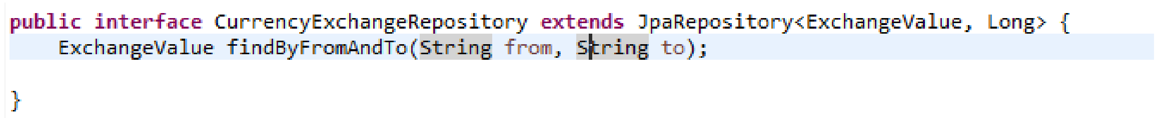
Setting property from application properties



```
@Autowired
org.springframework.core.env.Environment env;

@GetMapping("/exchange/from/{from}/to/{to}")
ExchangeValue exchangevalue(@PathVariable String from, @PathVariable String to) {
    ExchangeValue e = new ExchangeValue(1L, from, to, BigDecimal.valueOf(65));
    e.setPort(Integer.parseInt(env.getProperty("server.port")));
    return e;
}
```



```
-Dserver.port=8083
```



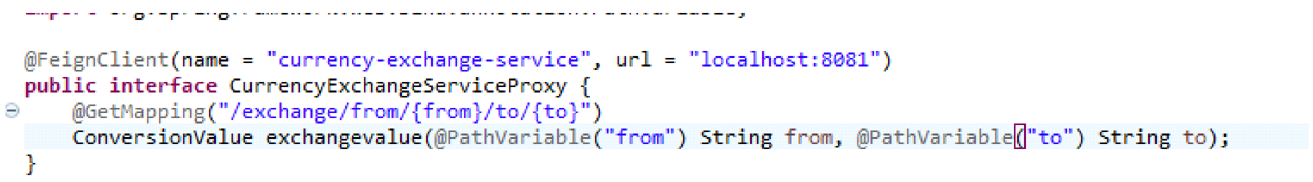
```
public interface CurrencyExchangeRepository extends JpaRepository<ExchangeValue, Long> {
    ExchangeValue findByFromAndTo(String from, String to);

}
```

WHEN WE WANT TO FIND BY FROM AND TO String then jpa repo query

Response mapping from other localhost

```
Map<String, String> urivariables = new HashMap<String, String>();
        urivariables.put("from", from);
        urivariables.put("to", to);
        ResponseEntity<ConversionValue> responseEntity =  restTemplate().getForEntity(
                "http://localhost:8081/exchange/from/{from}/to/{to}", ConversionValue.class,
                urivariables);
        ConversionValue body = responseEntity.getBody();
```

Rest template alternative feign

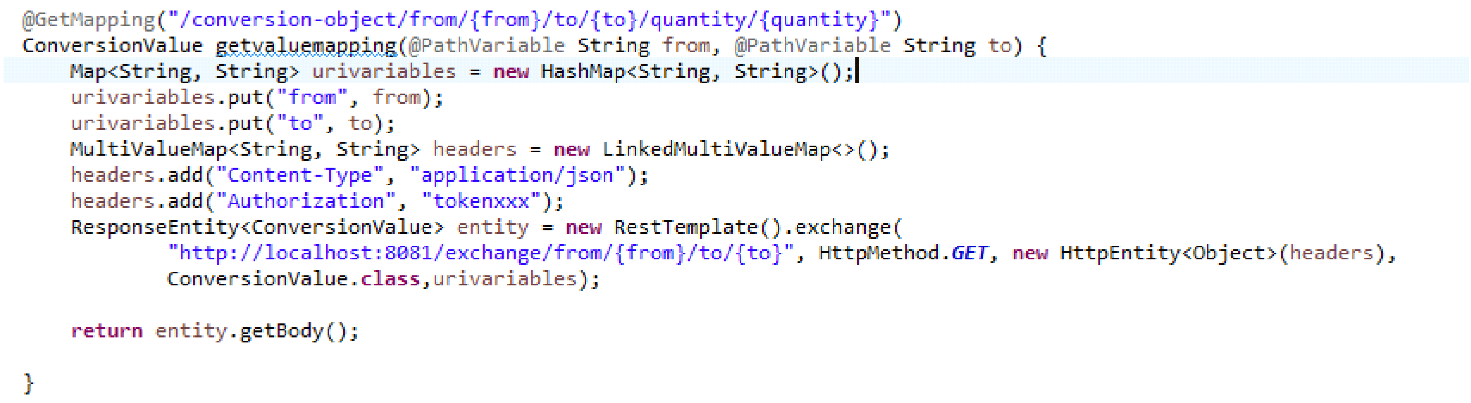


```
@FeignClient(name = "currency-exchange-service", url = "localhost:8081")
public interface CurrencyExchangeServiceProxy {
    @GetMapping("/exchange/from/{from}/to/{to}")
    ConversionValue exchangevalue(@PathVariable("from") String from, @PathVariable("to") String to);
}
```

Post for object
Post for entity
Exchange method of Spring RestTemplate - Part 1 || Calling REST API using RestTemplate

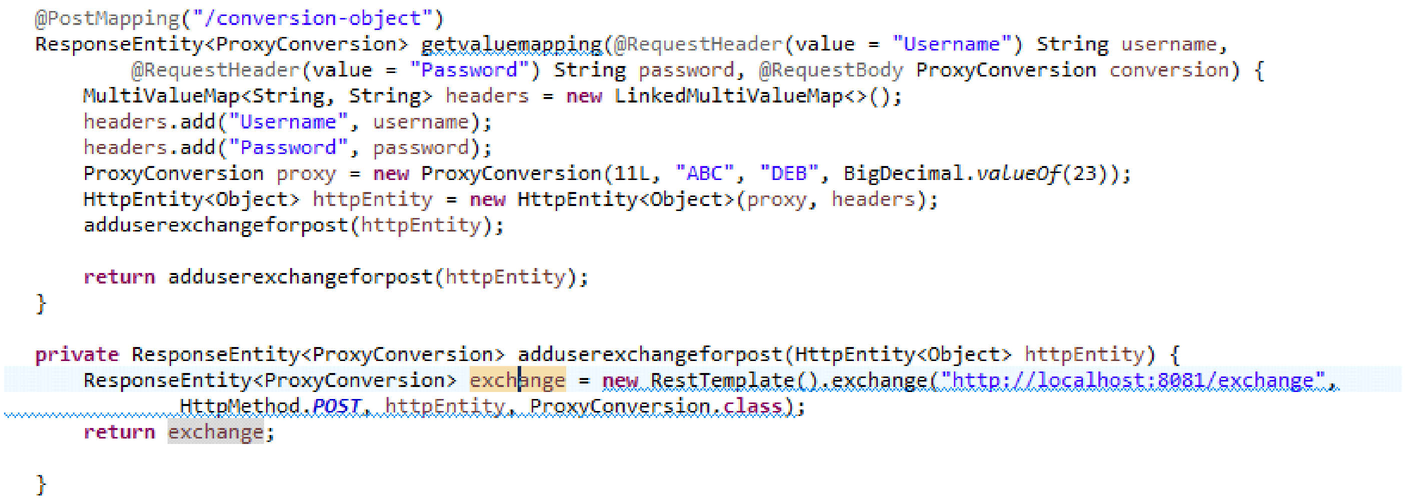
@Get from other api

```
@GetMapping("/conversion-object/from/{from}/to/{to}/quantity/{quantity}")
ConversionValue getvaluemapping(@PathVariable String from, @PathVariable String to) {
    Map<String, String> urivariables = new HashMap<String, String>();
    urivariables.put("from", from);
    urivariables.put("to", to);
    MultiValueMap<String, String> headers = new LinkedMultiValueMap<>();
    headers.add("Content-Type", "application/json");
    headers.add("Authorization", "tokenxxx");
    ResponseEntity<ConversionValue> entity = new RestTemplate().exchange(
            "http://localhost:8081/exchange/from/{from}/to/{to}", HttpMethod.GET, new
            HttpEntity<Object>(headers),
                    ConversionValue.class,urivariables);

        return entity.getBody();

}
```

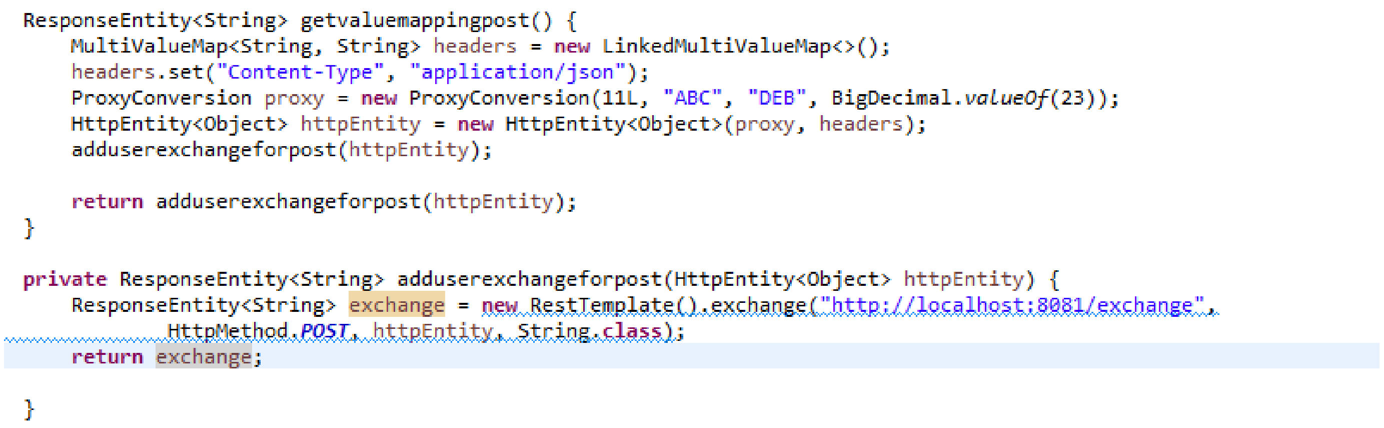


```
@GetMapping("/conversion-object/from/{from}/to/{to}/quantity/{quantity}")
ConversionValue getvaluemapping(@PathVariable String from, @PathVariable String to) {
    Map<String, String> urivariables = new HashMap<String, String>();
    urivariables.put("from", from);
    urivariables.put("to", to);
    MultiValueMap<String, String> headers = new LinkedMultiValueMap<>();
    headers.add("Content-Type", "application/json");
    headers.add("Authorization", "tokenxxx");
    ResponseEntity<ConversionValue> entity = new RestTemplate().exchange(
            "http://localhost:8081/exchange/from/{from}/to/{to}", HttpMethod.GET, new HttpEntity<Object>(headers),
            ConversionValue.class,urivariables);

    return entity.getBody();

}
```

Post Mapping



```
 @PostMapping("/conversion-object")
 ResponseEntity<ProxyConversion> getvaluemapping(@RequestHeader(value = "Username") String username,
         @RequestHeader(value = "Password") String password, @RequestBody ProxyConversion conversion) {
     MultiValueMap<String, String> headers = new LinkedMultiValueMap<>();
     headers.add("Username", username);
     headers.add("Password", password);
     ProxyConversion proxy = new ProxyConversion(11L, "ABC", "DEB", BigDecimal.valueOf(23));
     HttpEntity<Object> httpEntity = new HttpEntity<Object>(proxy, headers);
     adduserexchangeforpost(httpEntity);

     return adduserexchangeforpost(httpEntity);
 }

 private ResponseEntity<ProxyConversion> adduserexchangeforpost(HttpEntity<Object> httpEntity) {
     ResponseEntity<ProxyConversion> exchange = new RestTemplate().exchange("http://localhost:8081/exchange",
             HttpMethod.POST, httpEntity, ProxyConversion.class);
     return exchange;

 }
```



```
ResponseEntity<String> getvaluemappingpost() {
    MultiValueMap<String, String> headers = new LinkedMultiValueMap<>();
    headers.set("Content-Type", "application/json");
    ProxyConversion proxy = new ProxyConversion(11L, "ABC", "DEB", BigDecimal.valueOf(23));
    HttpEntity<Object> httpEntity = new HttpEntity<Object>(proxy, headers);
    adduserexchangeforpost(httpEntity);

    return adduserexchangeforpost(httpEntity);
}

private ResponseEntity<String> adduserexchangeforpost(HttpEntity<Object> httpEntity) {
    ResponseEntity<String> exchange = new RestTemplate().exchange("http://localhost:8081/exchange",
            HttpMethod.POST, httpEntity, String.class);
    return exchange;

}
```

Pagebale concept

```java
@Service
public class UserServiceImpl implements UserService {

    @Autowired
    private UserRepository userRepository;

    @Override
    public List<UserDto> getUsers(int page, int limit) {
        List<UserDto> returnValue = new ArrayList<>();

        Pageable pageableRequest = PageRequest.of(page, limit);
        Page<UserEntity> users = userRepository.findAll(pageableRequest);
        List<UserEntity> userEntities = users.getContent();

        for (UserEntity userEntity : userEntities) {
            UserDto userDto = new UserDto();
            BeanUtils.copyProperties(userEntity, userDto);
            returnValue.add(userDto);
        }

        return returnValue;
    }

}
```

https://www.javainuse.com/spring/SpringBootUsingPagination

```java
  @RequestParam(value = "page", defaultValue = "0") int page
```

ıd the limit request parameter our method above has the **limit** method argument:

```java
  @RequestParam(value = "limit", defaultValue = "30") int limit
```

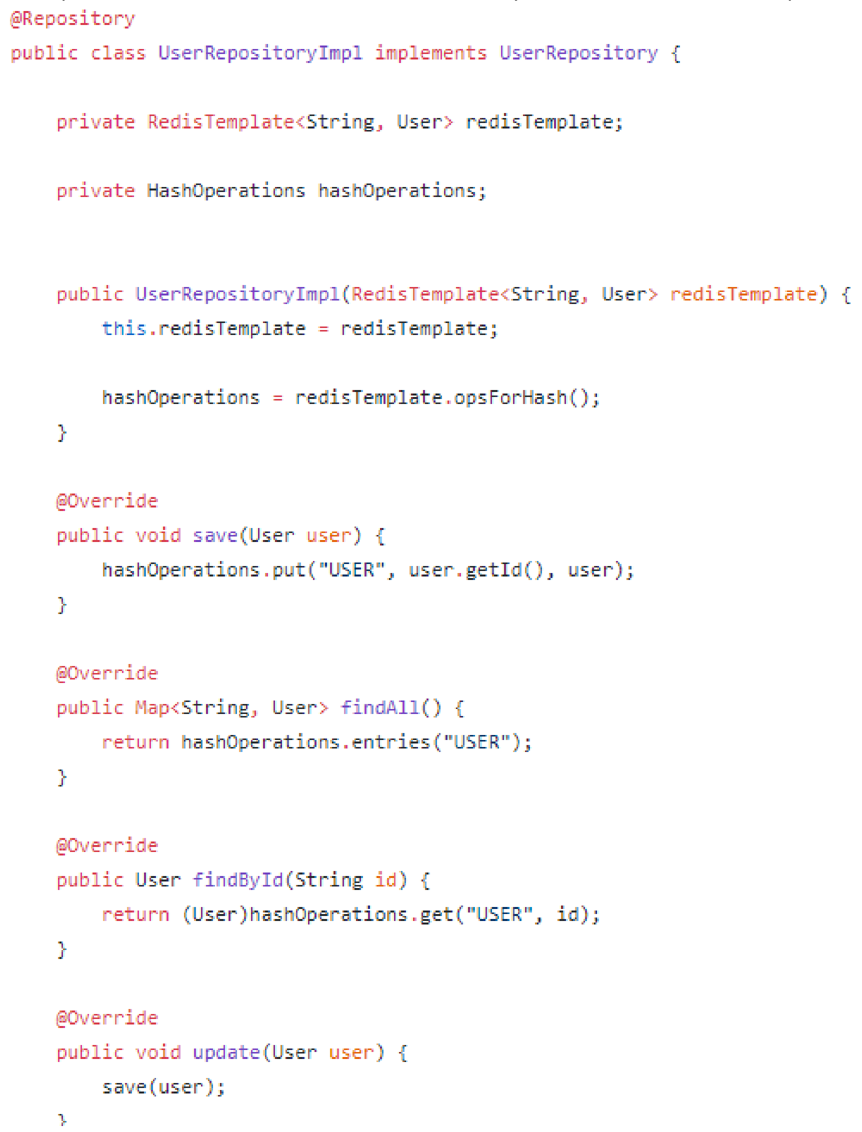
21
Apart from these mentioned differences in framework, one major difference is `@RequestParam` will always expect a value to bind. Hence, if value is not passed, it will give error. This is not the case in `@QueryParam`
Query param if we have already the value then we search using query param

From <https://stackoverflow.com/questions/26709560/what-is-the-difference-b-w-requestparam-and-queryparam-anotation>

Redis server

```java
@SpringBootApplication
public class CurrencyConversionApplication {

    @Bean
    JedisConnectionFactory factory() {
        return new JedisConnectionFactory();
    }

    @Bean
    RedisTemplate<String, User> getredis() {
        RedisTemplate<String, User> redis = new RedisTemplate<String, User>();
        redis.setConnectionFactory(factory());
        return redis;
    }

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

}
```

Hashoperation we cannot use reddis server directly we need to use via hashoperation



```
@Repository
public class UserRepositoryImpl implements UserRepository {

    private RedisTemplate<String, User> redisTemplate;

    private HashOperations hashOperations;


    public UserRepositoryImpl(RedisTemplate<String, User> redisTemplate) {
        this.redisTemplate = redisTemplate;

        hashOperations = redisTemplate.opsForHash();
    }

    @Override
    public void save(User user) {
        hashOperations.put("USER", user.getId(), user);
    }

    @Override
    public Map<String, User> findAll() {
        return hashOperations.entries("USER");
    }

    @Override
    public User findById(String id) {
        return (User)hashOperations.get("USER", id);
    }

    @Override
    public void update(User user) {
        save(user);
    }
```