



# POC Incrément 2

---

David THIBAU – 2024

david.thibau@gmail.com



# Rappels des objectifs

---

Prendre en main les architectures micro-services :

- Techniquement : Spring Cloud
- Méthodologie de dev :
  - Workflow Git
  - Indépendance des développements et évolutions
  - Pipeline CI augmentant la qualité et la transparence
- Intégration des services de support comme Kafka et Keycloak

Valider la solution JMX



# Bilan de l'incrément 1

---

Décomposition métier : OK

Outil de collaboration Gitlab (MR, Issues, ..) :  
Moyennement utilisé

Gros problème : Dépendances entre services posent  
des problèmes pour l'environnement de dev.

Couverture des tests trop faible

Quelques problèmes qualité

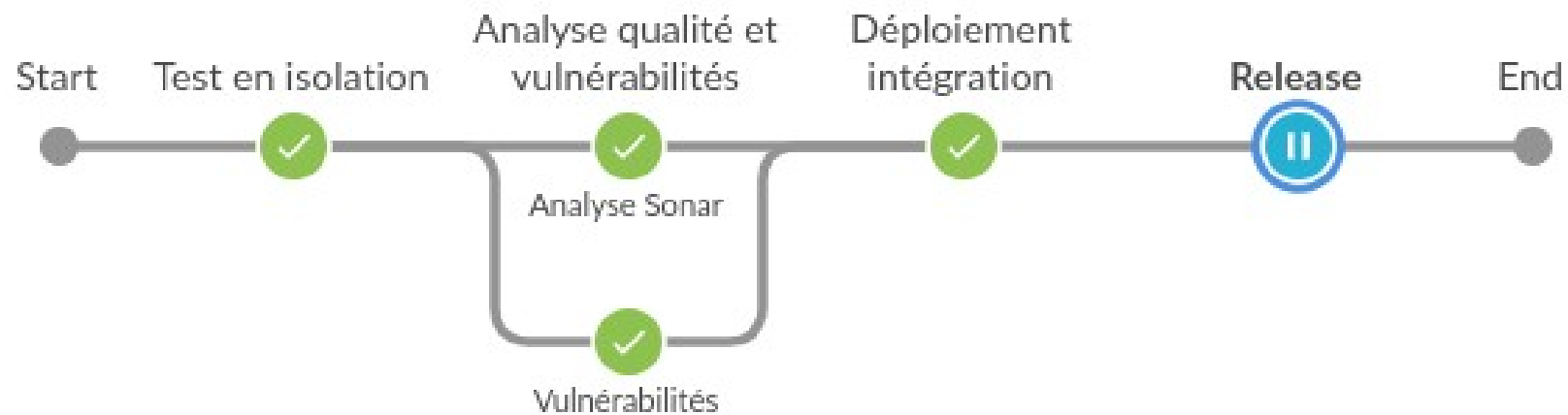
Points techniques restés en suspens : Jmix, Keycloak

Projets peu documentés



# Pipeline CI Incrément 2

---





# Tests en isolation

---

Tests sans les autres services.

- Les services de support sont présents (Config, Eureka, Kafka, Postgres, Keycloak, ...)

=> Mocker les appels REST

=> Envoyer des messages Kafka



# Mock des appels REST

---

org.springframework.cloud :spring-cloud-starter-contract-stub-runner (scope test)

```
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.MOCK,  
    properties = {"spring.cloud.discovery.enabled=false","eureka.client.enabled=false"} )  
@AutoConfigureWireMock(port = 0) // Configuration automatique de WireMock  
@EnableAutoConfiguration(exclude= EurekaClientAutoConfiguration.class)  
public class BanqueControllerIsolationTest extends BanqueControllerIntegrationTest {
```

```
    @TestConfiguration  
    static class TestConfig {  
        @Value("${wiremock.server.port}")  
        String wiremockPort;  
        @Bean  
        @Primary  
        public RestTemplate testRestTemplate(RestTemplateBuilder builder) {  
            return builder.rootUri("http://localhost:"+wiremockPort).build();  
        }  
    }  
}
```



# Mock

---

```
@BeforeEach
```

```
void stubMock() throws JsonProcessingException {
```

```
    stubFor(WireMock.get(urlEqualTo("/api/banques"))
        .willReturn(aResponse()
            .withBody(objectMapper.writeValueAsString(banquesDto))
            .withHeader("Content-Type", "application/json")
            .withFixedDelay(10)));
```

```
    stubFor(WireMock.get(urlEqualTo("/api/operations/journee/1"))
        .willReturn(aResponse()
            .withBody(objectMapper.writeValueAsString(operations))
            .withHeader("Content-Type", "application/json")
            .withFixedDelay(10)));
```

```
}
```



# Analyse qualité

---

## Conditions on Overall Code

Metric	Operator	Value
Coverage	is less than	70.0%
Duplicated Lines (%)	is greater than	3.0%
Reliability Rating	is worse than	A
Security Hotspots Reviewed	is less than	100%
Security Rating	is worse than	A
Unit Test Success (%)	is less than	100%





# Documentation

---

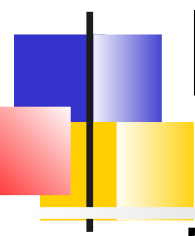
Utiliser Reame.MD du dépôt.

Diagramme de classe des entités

Interactions avec dépendances :

- Appel Rest avec DTOs
- Format des messages

Java doc sur les méthodes publiques de service



# Environnement d'intégration

---

## Network *ci*

- Jenkins : 8082 admin/admin
- SonarQube : 9000 admin/admin123

## Network *poc-integration*

- Config : 8888
- Eureka : 1111
- Kafka : 9092, Redpanda 8079
- Keycloak : 8180
- Reglement-postgresql : 5462
- PgAdmin : 90
- Smtip : 2525



# Déploiement

---

Construction d'image

Push vers DockerHub (dthibau)

Démarrage dans le réseau poc-integration,  
chaque micro-service à un port  
particulier.

```
sh ""docker run -e SERVER_PORT=8080 -p 10001:8080 \  
    -e SPRING_CLOUD_CONFIG_URI=http://cloud-config-server:8888 \  
    -e EUREKA_CLIENT_SERVICEURL_DEFAULTZONE=http://eureka:1111 \  
    -network poc-integration -d \  
    --name poc-compensation dthibau/poc-compensation:${version}""
```



# Revue de code

---

Trop de chaînes en dur, nécessite une externalisation en particulier les endpoints des services

Attention à ne pas sauvegarder dans sa base des entités détenues par d'autres services (banque, pays, ..)

MapStruct Lombok pas possible de les faire fonctionner ensemble ?

Pas de gestion d'erreurs dans les adapters Kafka, en particulier erreurs de sérialisation

Trouver un moyen pour plus de liberté côté sérialisation/désérialisation Kafka (StringSerialiser ?)

Interaction requête/réponse possibilité d'utiliser un *ReplyKafkaTemplate*

Du code mort



# Sécurité service

---

Tous les services sont référencées dans Keycloak

Les services métier ont un client-id et client-secret permettant d'obtenir des jetons.

Le jeton contient le scope *service*

Ce scope peut être utilisé pour protéger les APIs

Le projet reglement a validé la faisabilité



# Sécurité Gateway-sica

---

Le projet Gateway-sica identifie un utilisateur avec OpenIdConnect.

Pour l'instant 3 utilisateurs sont dans le realm, leurs mot de passe sont secret :

- agent/AGENT
- manager/MANAGER
- banque/BANQUE

Questions à régler, ou sont mis les ACLs en fonction des rôles utilisateur ?  
gateway ET/OU service métier