

# ALOM - TP 11 - EDA !

## Table of Contents

1. Présentation et objectifs .....	1
2. Pré-requis .....	2
3. <b>mailing-api</b> .....	2
3.1. Démarrage d'une instance locale de pulsar .....	2
3.2. Configuration Spring Boot .....	3
3.3. Ajout d'un listener .....	3
4. <b>trainer-api</b> .....	5
4.1. Configuration .....	5
4.2. Ajout d'un producer .....	5
5. <b>battle-api</b> .....	5
6. Configuration pour Clever Cloud .....	6

## 1. Présentation et objectifs

Le but est de continuer le développement de notre architecture "à la microservice".

Nous allons utiliser Apache Pulsar, pour implémenter un début d'architecture EDA entre les micro-services game et mailing.

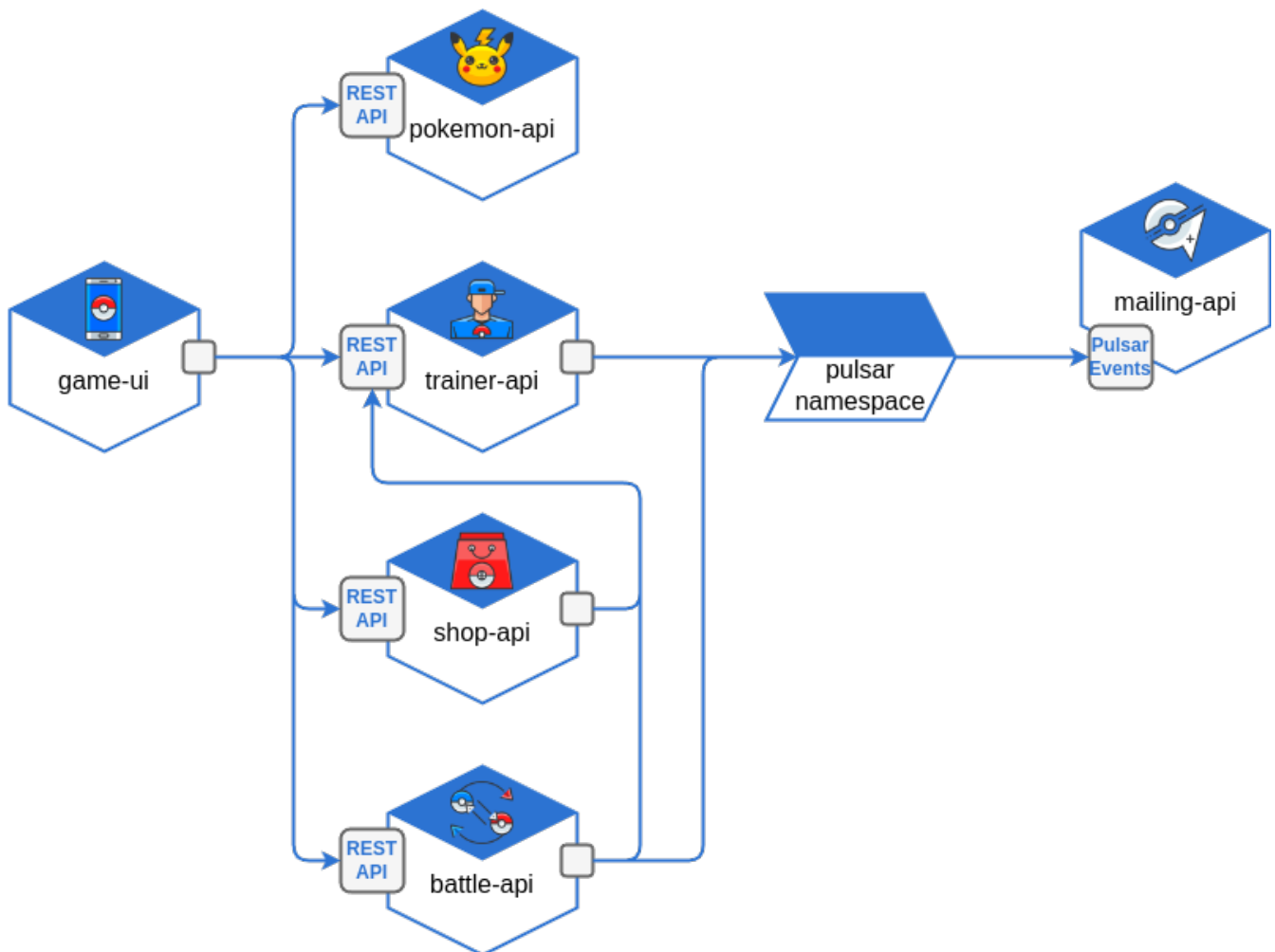


Figure 1. Notre architecture !

## 2. Pré-requis

En pré-requis à ce TP, il faut avoir implémenté la partie [6. Envoi d'emails](#).

## 3. mailing-api

### 3.1. Démarrage d'une instance locale de pulsar

Apache Pulsar est disponible sous la forme d'une image Docker : [apachepulsar/pulsar](#).

Démarrez une instance de pulsar sur votre poste avec la commande suivante :

```
docker container run -d \  
  --name pulsar \  
  -p 6650:6650 \  
  -p 6080:8080 \  
  apachepulsar/pulsar:4.0.1 \  
  bin/pulsar standalone
```

## 3.2. Configuration Spring Boot

Dans votre projet `mailing`, importez la dépendance `spring-boot-starter-pulsar` :

*pom.xml*

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-pulsar</artifactId>
</dependency>
```

Configurez les propriétés suivantes dans un fichier `application-local.properties`, pour utiliser votre instance locale :

*application-local.properties*

```
spring.pulsar.client.service-url=pulsar://localhost:6650
spring.pulsar.admin.service-url=http://localhost:6080
```

## 3.3. Ajout d'un listener

Créez une classe ou un record qui représentera votre événement `trainer:new`. Cette classe contient l'adresse mail du trainer, ainsi que son nom.

Annotez cette classe avec l'annotation `@PulsarMessage(schemaType = SchemaType.JSON)` pour indiquer à Spring Boot que cet objet devra être sérialisé en JSON.

Implémentez une classe `NewTrainerEventPulsarAdapter`. L'annotation `@PulsarListener` permet de marquer une méthode Java comme devant être appelée lorsqu'un message est disponible dans un topic Pulsar. Ajoutez une méthode à cette classe, qui porte l'annotation, et reçoit en paramètre votre classe d'évènement :

```
@PulsarListener(topics = "trainer:new", subscriptionType = SubscriptionType.Shared)
void receiveEvent(NewTrainerEvent event) {
    // TODO
}
```

Cette méthode écoute sur un topic `<votre-nom>:trainer:new`.



On utilise un nommage de topic `trainer:new` avec votre nom en préfixe, pour éviter les conflits entre les étudiants !

Le paramètre `SubscriptionType.Shared` permet de partager la connexion au topic entre plusieurs instances de votre micro-service !

Cette méthode pourra appeler la méthode de service existant par ailleurs.

Le démarrage de votre application devrait afficher les logs de connexion au topic pulsar :

```
o.a.p.c.impl.ConsumerStatsRecorderImpl : Starting Pulsar consumer status recorder
with config: {"topicNames":["persistent://public/default/new-
trainer"],"topicsPattern":null,"subscriptionName":"org.springframework.Pulsar.PulsarLi
stenerEndpointContainer#0","subscriptionType":"Shared","subscriptionProperties":null,"
subscriptionMode":"Durable","receiverQueueSize":1000,"acknowledgementsGroupTimeMicros"
:100000,"maxAcknowledgmentGroupSize":1000,"negativeAckRedeliveryDelayMicros":60000000,
"maxTotalReceiverQueueSizeAcrossPartitions":50000,"consumerName":null,"ackTimeoutMilli
s":0,"tickDurationMillis":1000,"priorityLevel":0,"maxPendingChunkedMessage":10,"autoAc
kOldestChunkedMessageOnQueueFull":false,"expireTimeOfIncompleteChunkedMessageMillis":6
0000,"cryptoFailureAction":"FAIL","properties":{},"readCompacted":false,"subscriptionI
nitialPosition":"Latest","patternAutoDiscoveryPeriod":60,"regexSubscriptionMode":"Pers
istentOnly","deadLetterPolicy":null,"retryEnable":false,"autoUpdatePartitions":true,"a
utoUpdatePartitionsIntervalSeconds":60,"replicateSubscriptionState":false,"resetInclud
eHead":false,"batchIndexAckEnabled":false,"ackReceiptEnabled":false,"poolMessages":fal
se,"startPaused":false,"autoScaledReceiverQueueSizeEnabled":false,"topicConfigurations
":[],"maxPendingChunkedMessage":10}
o.a.p.c.impl.ConsumerStatsRecorderImpl : Pulsar client config:
{"serviceUrl":"pulsar://localhost:6650","authPluginClassName":null,"authParams":null,"
authParamMap":null,"operationTimeoutMs":30000,"lookupTimeoutMs":30000,"statsIntervalSe
conds":60,"numIoThreads":22,"numListenerThreads":22,"connectionsPerBroker":1,"connecti
onMaxIdleSeconds":25,"useTcpNoDelay":true,"useTls":false,"tlsKeyFilePath":null,"tlsCer
tificateFilePath":null,"tlsTrustCertsFilePath":null,"tlsAllowInsecureConnection":false
,"tlsHostnameVerificationEnable":false,"concurrentLookupRequest":5000,"maxLookupReques
t":50000,"maxLookupRedirects":20,"maxNumberOfRejectedRequestPerConnection":50,"keepAli
veIntervalSeconds":30,"connectionTimeoutMs":10000,"requestTimeoutMs":60000,"readTimeou
tMs":60000,"autoCertRefreshSeconds":300,"initialBackoffIntervalNanos":1000000000,"maxBa
ckoffIntervalNanos":600000000000,"enableBusyWait":false,"listenerName":null,"useKeyStor
eTls":false,"sslProvider":null,"tlsKeyStoreType":"JKS","tlsKeyStorePath":null,"tlsKeyS
torePassword":null,"tlsTrustStoreType":"JKS","tlsTrustStorePath":null,"tlsTrustStorePa
ssword":null,"tlsCiphers":[],"tlsProtocols":[],"memoryLimitBytes":67108864,"proxyServi
ceUrl":null,"proxyProtocol":null,"enableTransaction":false,"dnsLookupBindAddress":null
,"dnsLookupBindPort":0,"dnsServerAddresses":[],"socks5ProxyAddress":null,"socks5ProxyU
sername":null,"socks5ProxyPassword":null,"description":null,"openTelemetry":null}
o.a.pulsar.client.impl.ConsumerImpl :
[persistent://public/default/trainer:new][org.springframework.Pulsar.PulsarListenerEnd
pointContainer#0] Subscribing to topic on cnx [id: 0x58864231, L:/127.0.0.1:38618 -
R:localhost/127.0.0.1:6650], consumerId 0
o.a.pulsar.client.impl.ConsumerImpl :
[persistent://public/default/trainer:new][org.springframework.Pulsar.PulsarListenerEnd
pointContainer#0] Subscribed to topic on localhost/127.0.0.1:6650 -- consumer: 0
```

Implémentez toutes les méthodes disponibles sur le mailing service avec des listener Pulsar :

- envoi d'un email de bienvenue (reçoit le nom d'un Trainer en paramètre, ainsi que son email)
- envoi d'un email de fin de combat (reçoit les noms des 2 adversaires, leurs emails, et l'information de qui a gagné le combat)

## 4. trainer-api

### 4.1. Configuration

Ajoutez la dépendance `spring-boot-starter-pulsar` à votre projet `trainer-ui`.

Configurez les propriétés suivantes dans un fichier `application-local.properties`, pour utiliser votre instance locale :

*application-local.properties*

```
spring.pulsar.client.service-url=pulsar://localhost:6650
spring.pulsar.admin.service-url=http://localhost:6080
```

### 4.2. Ajout d'un producer

Spring propose un `PulsarTemplate` pour poster des messages dans un topic. Dans une classe de `game-ui`, nommée `UserEventsPulsarAdapter`, utilisez le `PulsarTemplate` (reçu en injection de dépendance) pour envoyer les événements à destination du `mailing-api` dans le topic consacré `<votre-nom>:trainer:new`.

À titre d'exemple, voici un usage du `PulsarTemplate` pour envoyer un message sur un topic :

```
pulsarTemplate.send(
    "trainer:new",
    new NewTrainerEvent("Ash", "ash@gitlab-classrooms.org")
);
```



N'hésitez pas à copier/coller votre classe correspondant à l'évènement attendu depuis `mailing-api`.



Isolez toutes ces classes dans un package consacré. Vous pouvez aussi exposer une interface `UserEventsPort` dans votre couche "domaine" si elle existe, pour reprendre les principes de l'architecture hexagonale.

## 5. battle-api

Dans `battle-api`, implémentez l'envoi d'un message lors de la fin d'un combat, à destination du `mailing-api` dans le topic consacré `<votre-nom>:battle:end`, de la même manière que cela a été fait entre `game-ui` et `mailing-api`.

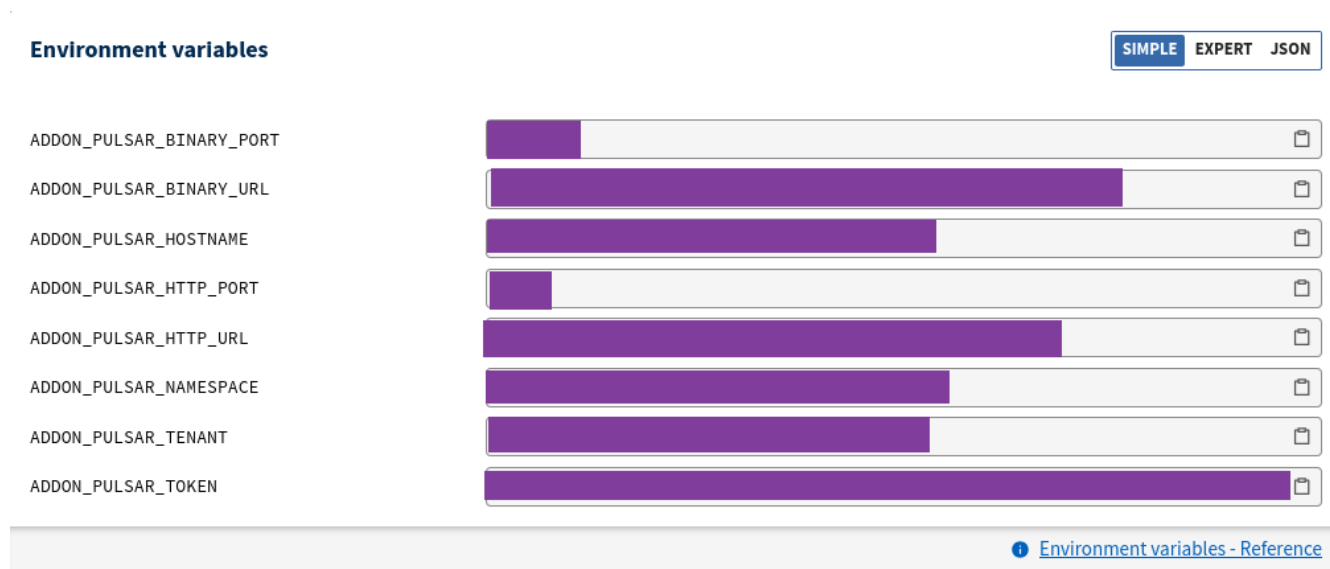
## 6. Configuration pour Clever Cloud

La configuration de l'utilisation de Pulsar sur Clever Cloud nécessite quelques ajustements :

- l'instance de Pulsar est partagée entre tous les étudiants
- la connexion à Pulsar nécessite une authentification
- les topics peuvent être créés dans un tenant/namespace unique

Toutes les informations nécessaires à la connexion à Pulsar sont disponibles à 2 endroits :

- dans la console [Clever Cloud](#).
- dans un secret sur [Vault](#).



The screenshot displays the 'Environment variables' section of the Clever Cloud console. At the top right, there are three tabs: 'SIMPLE' (selected), 'EXPERT', and 'JSON'. Below the tabs, a list of environment variables is shown, each with a name, a value represented by a purple bar, and a copy icon. The variables are:

Variable Name	Value (represented by bar)	Action
ADDON_PULSAR_BINARY_PORT	[Redacted]	Copy
ADDON_PULSAR_BINARY_URL	[Redacted]	Copy
ADDON_PULSAR_HOSTNAME	[Redacted]	Copy
ADDON_PULSAR_HTTP_PORT	[Redacted]	Copy
ADDON_PULSAR_HTTP_URL	[Redacted]	Copy
ADDON_PULSAR_NAMESPACE	[Redacted]	Copy
ADDON_PULSAR_TENANT	[Redacted]	Copy
ADDON_PULSAR_TOKEN	[Redacted]	Copy

At the bottom right of the console, there is a link: [Environment variables - Reference](#).

Figure 2. Les secrets Pulsar dans Clever Cloud

























Overview	Secret	Metadata	Paths
<input type="checkbox"/> JSON			
Key	Value		
ADDON_PULSAR_BINARY_PORT			 ■■■■■■■■■■
ADDON_PULSAR_BINARY_URL			 ■■■■■■■■■■
ADDON_PULSAR_HOSTNAME			 ■■■■■■■■■■
ADDON_PULSAR_HTTP_PORT			 ■■■■■■■■■■
ADDON_PULSAR_HTTP_URL			 ■■■■■■■■■■
ADDON_PULSAR_NAMESPACE			 ■■■■■■■■■■
ADDON_PULSAR_TENANT			 ■■■■■■■■■■
ADDON_PULSAR_TOKEN			 ■■■■■■■■■■

Figure 3. Les secrets Pulsar dans Vault

Configurez dans vos projets `mailing-api`, `game-ui` et `battle-api` l'utilisation de l'instance Pulsar de Clever Cloud :

`application-clever.properties`

```
spring.pulsar.client.service-url= ①
spring.pulsar.client.authentication.plugin-class-name
=org.apache.pulsar.client.impl.auth.AuthenticationToken
spring.pulsar.client.authentication.param.token= ②

spring.pulsar.admin.service-url= ③
spring.pulsar.admin.authentication.plugin-class-name
=org.apache.pulsar.client.impl.auth.AuthenticationToken
```

```
spring.pulsar.admin.authentication.param.token= ②
```

```
spring.pulsar.defaults.topic.tenant= ④
```

```
spring.pulsar.defaults.topic.namespace= ⑤
```

1. Utilisez la valeur de `ADDON_PULSAR_BINARY_URL`
2. Utilisez la valeur de `ADDON_PULSAR_TOKEN`
3. Utilisez la valeur de `ADDON_PULSAR_HTTP_URL`
4. Utilisez la valeur de `ADDON_PULSAR_TENANT`
5. Utilisez la valeur de `ADDON_PULSAR_NAMESPACE`