



Confluent Schema Registry, un premier pas vers la gouvernance des données



© Ippon Technologies

Confluent Schema Registry, un premier pas vers la gouvernance des données

👤 Mohamed HILIA (/author/mohamed/) 📅 18 Nov 2019

🗨 0 Commentaires (/2019/11/18/confluent-schema-registry-un-premier-pas-vers-la-gouvernance-des-donnees/#disqus_thread)

Dans cet article, nous présentons le Schema Registry de Confluent, un composant clé dans une plateforme data. Nous présentons son rôle et ses nombreux avantages. L'article donne des recommandations accompagnées d'exemples permettant d'explorer les vertus de ce composant ainsi que de maîtriser ses fonctionnalités.

Context

Le composant Confluent Schema Registry est un composant open source initialement développé par Confluent (<https://www.confluent.io/>). Le Schema Registry permet la gestion centralisée de l'ensemble des schémas d'une plateforme de données et donc une meilleure gouvernance des données. Ces schémas sont utilisés dans les processus de sérialisation et désérialisation des messages par les clients de la plateforme.

Le Schema Registry représente un référentiel partagé de schémas qui permet aux applications d'interagir de manière flexible les unes avec les autres. Il permet :

- d'enregistrer des schémas,
- de les consulter,
- de les faire évoluer en gardant l'historique,
- le versioning des schémas,
- la vérification de compatibilité entre les différentes versions.

Dans plusieurs contextes d'intégration des données, de collecte des open data (<https://opendata.paris.fr/pages/home/>), ou encore l'ingestion des données en flux provenant de l'internet des objets, les données sont en perpétuelle évolution et leur stockage (e.g. BigQuery (<https://cloud.google.com/bigquery/>)), Hadoop (<https://www.geeksforgeeks.org/hadoop-ecosystem/>)) nécessite une définition de schémas bien documentée, validée et gérée. Avoir des schémas bien définis

aidera à mieux intégrer des données et services dans des architectures microservices ainsi qu'une meilleure exploration et exploitation des données à des fins d'analytics (e.g. RGPD). Grâce à la validation des schémas, la qualité des données sera assurée.

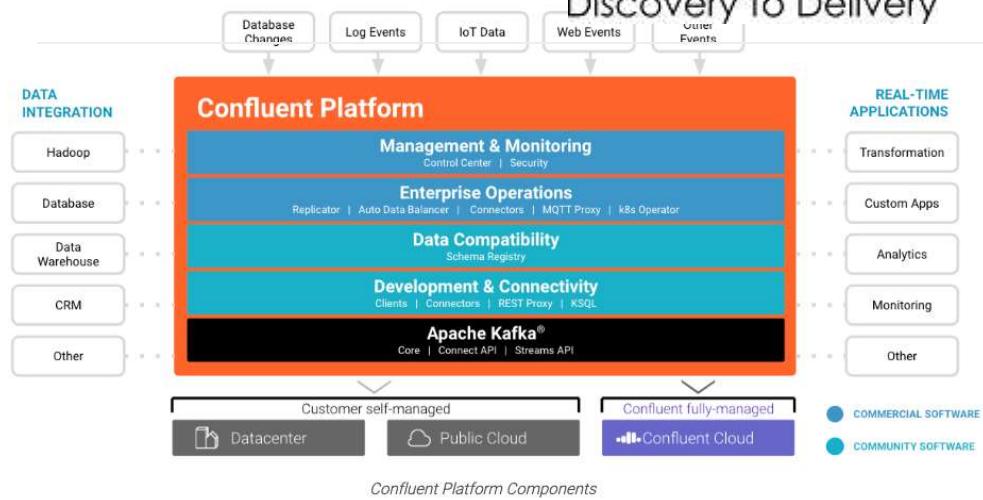


Fig. 1 : Source (<https://docs.confluent.io/current/platform.html>)

Rôle d'un Schema Registry

Les schémas permettent de garder une cohérence de la structure des messages échangés. En effet, il ne faut pas se focaliser sur le premier cas d'usage. L'utilisation des données augmentera avec le temps et d'autres équipes peuvent venir à la chasse de ces données pour de nouveaux cas d'utilisation. Les besoins fonctionnels et techniques peuvent vous amener à changer la structure de vos messages, ce qui impactera gravement les consommateurs si on n'a pas de schéma de données.

Prenons l'exemple de l'une des plateformes de nos clients, où toute publication de message devrait être contextualisée via un ensemble d'attributs : localisation, timestamp, et bien d'autres. Dans cette situation un objet métier contexte a été requis pour la publication de tous les messages dans la plateforme. Devant cette contrainte, la formalisation d'une structure de message est primordiale. Cette structure nous a permis le contrôle de conformité des messages. Par conséquent, la mise en place d'un Schema Registry permet une meilleure gouvernance des données dans nos pipeline Data. Il introduit aussi une dimension d'efficacité opérationnelle en fournissant des schémas réutilisables, en définissant les relations entre les schémas et en permettant aux fournisseurs de données et aux consommateurs d'évoluer à des rythmes différents.

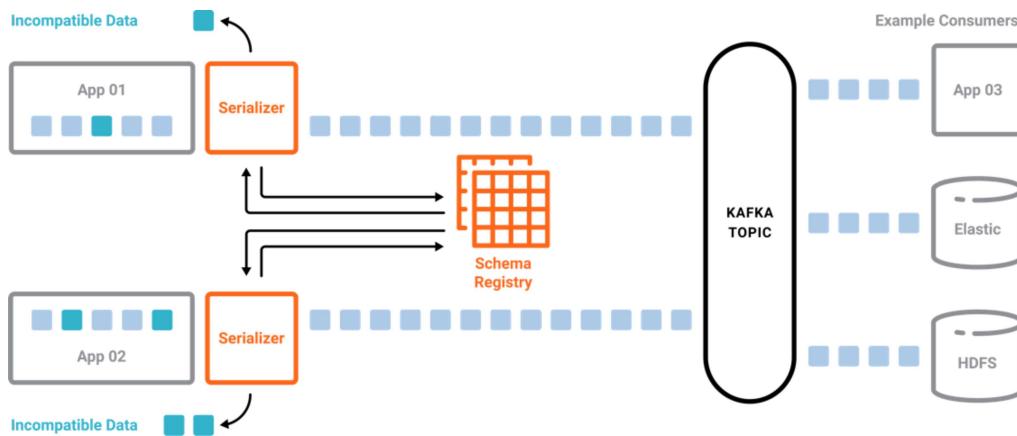


Fig. 2 : Srialisation de messages et Schema Registry

On peut se poser la question du rôle d'un Schema Registry dans une organisation. La première motivation est de ne pas voir les données mal formatées prendre leur chemin vers les moyens de stockage ou vers nos topics (Fig.2). Nous avons eu l'expérience avec la mise en œuvre d'une plateforme de distribution et de communication entre des points de productions et de consommations géo-répartis sur trois régions, à savoir, Europe, US et APAC. Avoir des schémas par région n'est sûrement pas la solution la plus optimale, sachant que les données peuvent être consommées et analysées à un niveau fédéré. Le découplage entre les consommateurs et les producteurs représente aussi une motivation pour mettre un Schema Registry central. Ce dernier permettra la gouvernance de l'ensemble des schémas utilisés au niveau global et assurera la compatibilité et la vérification des évolutions des schémas. Le Schema Registry a aussi le rôle de limiter les accès en écriture via des politiques de contrôle d'accès basées sur ACL.

(https://docs.confluent.io/current/confluent-security-plugins/schema-registry/authorization/sracl_authorizer.html) ou RBAC (<https://registry/security/rbac-schema-registry.html>). Le Schema Registry composants Confluent comme Kafka Connect (<https://docs.confluent.io/current/connect.html>), et bien d'autres outils comme Apache



Apache AVRO



Lien : <http://avro.apache.org/> (<http://avro.apache.org/>)

Apache Avro (<http://avro.apache.org/>) est un système de sérialisation de données. Il fournit des structures de données riches, un format de données binaires et un format de fichier conteneur pour stocker les données persistantes. Avro permet de stocker les données et leurs schémas au même endroit. Par conséquent, Avro ne nécessite pas de génération de code et s'intègre facilement avec des langages dynamiques comme JavaScript, Python, Ruby, C, C#, C++, Java, Scala. Avro est très utilisé dans l'écosystème Hadoop et représente la première recommandation pour Kafka par Confluent.

Apache Avro se compare souvent à Thrift (<https://thrift.apache.org/>), Protocol Buffers (<https://developers.google.com/protocol-buffers/>), JSON (<https://www.json.org/>), etc. Il a besoin d'encoder moins d'informations car il stocke les noms et les types dans le schéma, ce qui réduit la duplication et la taille de sérialisation. Un autre avantage clé d'Avro est son support des schémas évolutifs qui assure les contrôles de compatibilité et permet de faire évoluer vos données dans le temps. Avro est largement supporté par plusieurs plateformes Big Data comme Hadoop, Hortonworks ou d'ingestion et traitement de données comme Apache Nifi, StreamSet, BigQuery, etc. Dans le contexte de Kafka, Avro est utilisé pour la sérialisation des clés et valeurs des messages publiés et consommés des topics.

Avro utilise JSON pour la définition des schémas, et sérialise les données dans un format binaire. La définition des schémas est composée de types primitifs (**null**, **boolean**, **int**, **long**, **float**, **double**, **bytes**, **string**) ou complexes (**record**, **enum**, **array**, **map**, **union**, **fixed**).

Dans la Fig. 3, une exemple du schéma pour les données consommateurs.

Subject Name *
Consumer
This field is required

Schema:

```

1  {
2      "namespace": "org.ippon.schemaregistry.avro",
3      "type": "record",
4      "name": "Consumer",
5      "doc": "This is a sample Avro schema for a Consumer",
6      "fields": [
7          {"name": "customer_id", "type": "string"},
8          {"name": "civility", "type": { "type": "enum", "name": "Civility", "symbols": ["Miss", "Mr", "Ms"]}},
9          {"name": "last_name", "type": "string"},
10         {"name": "first_name", "type": "string"},
11         {"name": "city", "type": "string"},
12         {"name": "country", "type": "string"},
13         {"name": "email", "type": "string"},|
14         {"name": "phone_number", "type": "string"}
15
16
17     ]
18 }
```

Fig. 3 : Exemple d'un schéma en AVRO

Schema Registry & Kafka

Un message dans Kafka se compose principalement d'une clé, d'une valeur et optionnellement d'un *header*. Avro est utilisé pour la sérialisation des clés et valeurs des messages envoyés dans les topics Kafka. Par ailleurs, dans le Schema Registry, les schémas sont définis et évoluent dans le cadre d'un espace de nom défini par un **Subject**. Le nom du subject dépend de la stratégie configurée par *subject*, qui par défaut sera dérivée du nom du topic. Rappelons que le nom du topic peut être complètement indépendant du nom du schéma.

La stratégie de nommage des *Subjects* est configurable et peut être de deux types. Le premier, permet aux messages du même topic de se conformer au même schéma, et n'impose pas cette contrainte et autorise qu'un même topic pu

IPPON I type
(<http://blog.ippon.fr>)

Les clients peuvent donc définir la stratégie de nommage du *Subject* et les paramètres de configuration suivants :

```
key.subject.name.strategy
value.subject.name.strategy
```

La stratégie par défaut est TopicNameStrategy, configurable par position de cette clé (io.confluent.kafka.serializers.subject.TopicNameStrategy).

```
value.subject.name.strategy=io.confluent.kafka.serializers.subject.TopicNameStrategy
```

Cette stratégie nomme le schéma en fonction du nom du topic et exige implicitement que tous les messages du même topic soient conformes au même schéma. Une comparaison de ces différentes stratégies est disponible sur le site de documentation (<https://docs.confluent.io/current/schema-registry/serializer-formatter.html>).

Il faut sans doute préciser certaines limitations dans l'utilisation de ces conventions. Elles sont configurables toutes les trois seulement avec les clients Java. Les autres clients utilisent seulement la stratégie par défaut, à savoir TopicNameStrategy. KSQL utilise exclusivement la stratégie par défaut, et ne supporte pas plusieurs schémas dans un même topic. Pour plus de détails sur d'autres limitations, veuillez vous rendre sur la documentation (<https://docs.confluent.io/current/schema-registry/serializer-formatter.html>).

La Fig. 4 présente le composant Schema Registry et les interactions avec les autres composants d'une plateforme Kafka.

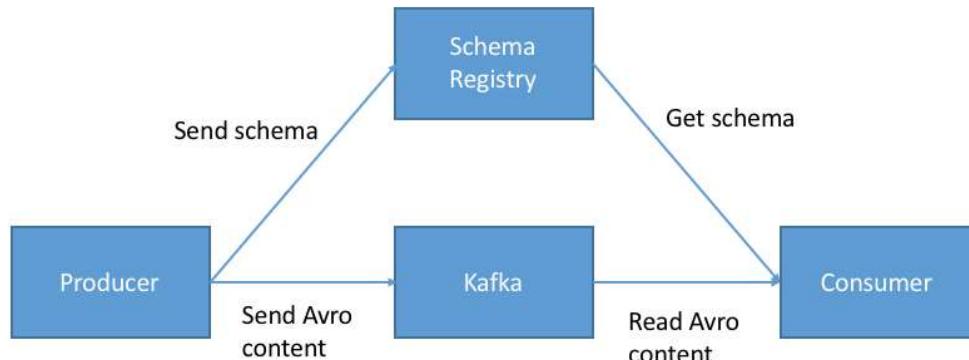


Fig. 4 : Schema Registry, Kafka Cluster, Producer & Consumer

Prenons l'exemple suivant pour l'envoi des données en flux sur un topic **consumers**. Le producteur de messages produit des données avec le schéma **Consumer** présenté ci-dessus au topic **consumers**. Si le producteur sérialise la valeur du message en Avro, le Schema Registry aura un sujet appelé **consumers-value**, dans ce sujet on trouvera au moins le schéma **Consumer** avec un identifiant unique global ID attribué par le Schema Registry. Si le producteur sérialise la clé, un autre sujet sera dans la Schema Registry nommé **consumers-key**. Dans ce périmètre, les schémas seront validés suite aux différents changements, mises à jour, évolutions, etc.

Il est recommandé que les instances du Schema Registry soient séparées des *Brokers* Kafka. Nous décrivons dans la suite le fonctionnement du Schema Registry avec les producteurs et consommateurs Kafka. Lors de l'envoi d'un message par le producteur, le sérialiseur s'assurera que le schéma est enregistré, obtiendra son ID ou enregistrera de manière automatique une nouvelle version du schéma pour vous. Ceci est même recommandé de désactiver via la propriété auto.register.schemas à false.

Enfin, le Schema Registry est une application Java. Il a été validé sur certaines des versions les plus récentes de Java, dont Java 11, mais pas sur toutes les versions comme Java 9 ou 10.

Du côté Producteur, enregistrement d'un schéma

- Le producteur envoie le schéma **Consumer** au Schema Registry.
- Le Schema Registry enregistre le schéma dans le périmètre. L'attribution d'ID de schéma se produit toujours dans le nœud primaire et les ID de schéma augmentent toujours de façon monotone.

- Le producteur reçoit l'ID du Schema Registry pour le mettre localement dans un cache pour futures utilisations, et donc, le Schema Registry n'est interrogé qu'une seule fois.

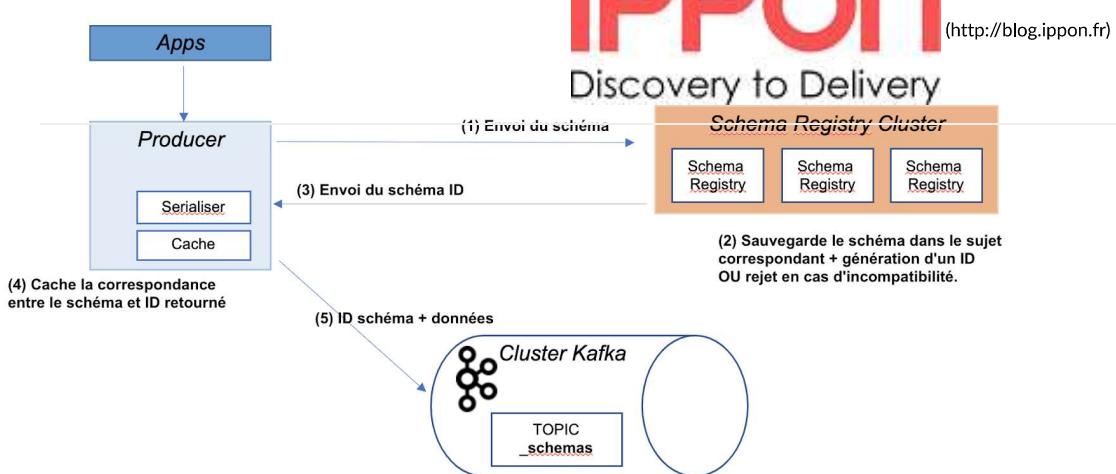


Fig. 5 : Interaction Schema Registry et producteurs

Les propriétés suivantes sont à rajouter du côté du producteur :

```
props.put("key.serializer", "io.confluent.kafka.serializers.KafkaAvroSerializer")
props.put("value.serializer", "io.confluent.kafka.serializers.KafkaAvroSerializer")
```

Du côté consommateur, lecture d'un schéma

- Le consommateur lit les données et l'ID de leur schéma.
- Il envoie une demande du schéma identifié par ID au Schema Registry.
- Le Schema Registry récupère le schéma correspondant à l'identifiant ID et renvoie le schéma au consommateur.
- Le consommateur reçoit le schéma du Schema Registry pour le mettre localement dans un cache pour futures utilisations, et donc, le schéma registry n'est interrogé qu'une seule fois.

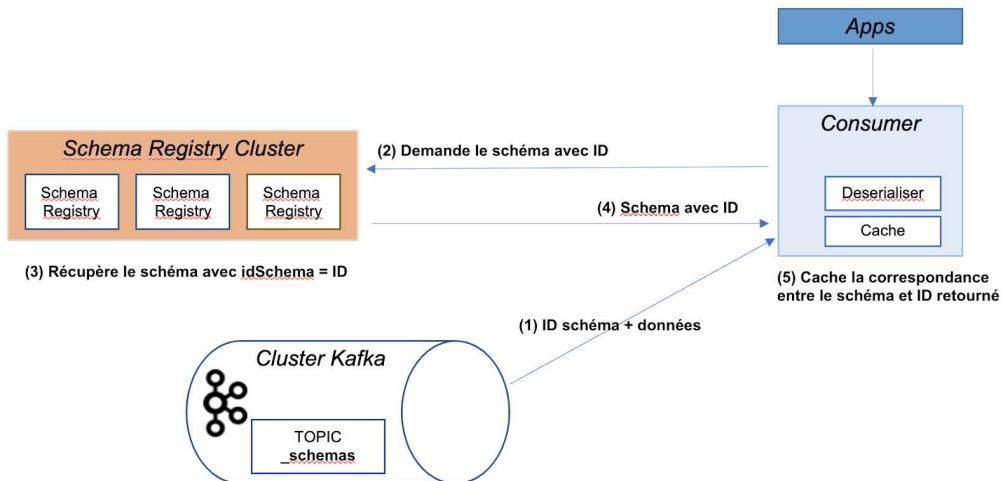


Fig. 6 : Interaction Schema Registry et consommateurs

Gouvernance des données

La gouvernance des données (https://en.wikipedia.org/wiki/Data_governance) correspond aux processus et à l'organisation mis en place permettant d'encadrer la collecte de données et leur utilisation. L'un des processus de gouvernance consiste en la vérification de l'exactitude des données en entrée de votre plateforme de données pour ensuite appliquer des politiques centralisées au sein de votre organisation. Le Schema Registry est un composant indispensable pour arriver à une gouvernance centralisée des structures de vos données. Le contrôle d'accès à ces schémas fait aussi partie des processus à mettre en place. Pour cela, le Schema Registry met à disposition les ACLs et RBAC pour la restriction des accès uniquement aux personnes autorisées. Pour une meilleure gouvernance, pensez aussi à désactiver l'enregistrement automatique des schémas. Les évolutions et la définition des relations entre schémas sont effectuées de manière centralisée et incrémentale avec le test de compatibilité et l'historisation des modifications. Les options de compatibilité configurées par *subject* sont les suivantes :



BACKWARD	(défaut) les consommateurs utilisent les données écrites par les producteurs enregistrés noté S. Les messages seront donc compatibles avec la dernière version courante du schéma S, ou celle de la version d'avant S-2.
NONE	les contrôles de compatibilité des schémas sont désactivés
BACKWARD_TRANSITIVE	les consommateurs utilisant le nouveau schéma peuvent lire les données écrites par les producteurs utilisant tous les schémas précédemment enregistrés.
FORWARD	les consommateurs utilisant le dernier schéma enregistré S-1 peuvent lire les données écrites par les producteurs utilisant le nouveau schéma S.
FORWARD_TRANSITIVE	les consommateurs utilisant tous les schémas précédemment enregistrés peuvent lire les données écrites par les producteurs utilisant le nouveau schéma. Les données produites par un schéma S peuvent être consommées par les consommateurs avec le schéma S-2, S-1, ou S.
FULL	le nouveau schéma est compatible en avant (FORWARD) et en arrière (BACKWARD) avec le dernier schéma enregistré.
FULL_TRANSITIVE	le nouveau schéma est compatible avec tous les schémas précédemment enregistrés.

On peut se demander comment on peut faire évoluer son schéma et garantir la politique d'évolution sous-jacente. Merci à Confluent de nous avoir récapitulé dans le tableau ci-dessous les opérations possibles :

Type de compatibilité	Changement autorisés	Vérifier par rapport à quel schéma	Mettre à jour en premier
BACKWARD	- Supprimer des attributs - Ajouter des attributs optionnels	Dernière version	Consommateurs
BACKWARD_TRANSITIVE	- Supprimer des attributs - Ajouter des attributs optionnels	Toutes les versions d'avant	Consommateurs
FORWARD	- Ajouter des attributs - Supprimer des attributs optionnels	Dernière version	Producteurs
FORWARD_TRANSITIVE	- Ajouter des attributs - Supprimer des attributs optionnels	Toutes les versions d'avant	Producteurs
FULL	- Modifier des attributs optionnels	Dernière version	Libre
FULL_TRANSITIVE	- Modifier des attributs optionnels	Toutes les versions d'avant	Libre

Pour une meilleure gouvernance, il est fortement recommandé qu'une équipe indépendante s'occupe de la gestion du cycle de vie des schémas en terme d'enregistrement, de mise à jour, d'évolution et de suppression. Une autre recommandation est de mettre en place la vérification des compatibilités entre les différentes versions via des processus d'intégration continue et de déploiement continu.

Récemment, sur la version Confluent 5.4-preview, l'éditeur a annoncé que la validation des messages par rapport à ces schémas sera prévue au niveau des *Brokers Kafka*. Une erreur sera renvoyée par le *broker Kafka* si le message ne respecte pas le schéma fourni pour la clé et pour la valeur à une limitation près,

l'erreur renvoyée par Confluent Server lorsque la validation de schéma est activée est trop générale.

Définition du schéma avec Avro



Afin d'avoir rapidement un premier environnement de développement de Confluent conteneurisé. Les images sont disponibles sur Docker Hub (<https://hub.docker.com/u/confluentinc/>) pour chacun des composants de la plateforme Confluent. Une version *all-in-one* packagée et déployable via docker-compose (<https://docs.docker.com/compose/>). Un environnement de développement est rapidement provisionné via le script suivant :

```
git clone https://github.com/confluentinc/examples (https://github.com/confluentinc/examples)
cd examples
git checkout 5.3.1-post
cd cp-all-in-one/
docker-compose up -d --build
```

La liste des composants installés sont les suivants via la commande :

```
docker-compose ps
```

Name	Command	State	Ports
broker	/etc/confluent/docker/run	Up	0.0.0.0:29092->29092/tcp, 0.0.0.0:9092->9092/tcp
connect	/etc/confluent/docker/run	Up (healthy)	0.0.0.0:8083->8083/tcp, 9092/tcp
control-center	/etc/confluent/docker/run	Up	0.0.0.0:9021->9021/tcp
ksql-cli	/bin/sh	Up	
ksql-daggen	bash -c echo Waiting for K ...	Up	
ksql-server	/etc/confluent/docker/run	Up	0.0.0.0:8088->8088/tcp
rest-proxy	/etc/confluent/docker/run	Up	0.0.0.0:8082->8082/tcp
schema-registry	/etc/confluent/docker/run	Up	0.0.0.0:8081->8081/tcp
zookeeper	/etc/confluent/docker/run	Up	0.0.0.0:2181->2181/tcp, 2888/tcp, 3888/tcp

Ensuite, nous aurons l'interface graphique du Control Center est accessible sur <http://localhost:9021/> (<http://localhost:9021/>).

The screenshot shows the Confluent Control Center interface. On the left, there's a sidebar with a blue header containing the word 'CO' and 'Cluster 1'. The main area has a dark header with the 'confluent' logo. Below it, the title 'Home' is displayed. A callout box highlights 'controlcenter.cluster'. To the right, there are two counts: '1 Healthy clusters' and '0 Unhealthy clusters'. Below these, there's a search bar labeled 'Search cluster name' and a toggle switch labeled 'Hide healthy clusters'. The main content area is titled 'controlcenter.cluster' and shows its status as 'Running'. It includes sections for 'Overview' (listing Brokers, Partitions, Topics, Production, and Consumption metrics) and 'Connected services' (listing KSQL clusters and Connect clusters). The entire interface has a modern, clean design with a white background and blue accents.

Le Control Center, qui fait partie de l'offre entreprise de confluent, est un outil graphique de supervision des clusters. Il permet aussi de créer et configurer des topics et de schémas est d'utiliser l'API REST exposée par le Schema Registry.



Nous avons défini un schéma via le Schema Registry UI pour l'**Consumer** topic. Il comporte plus d'une trentaine de champs, et des champs imbriqués. Dans la suite, pour des raisons de simplicité, nous ne présentons qu'une sous partie de notre modélisation.

```
{
  "type": "record",
  "name": "Consumer",
  "namespace": "fr.ippon.schemaregistry.avro",
  "fields": [
    { "name": "consumer_id" , "type": "string"},
    { "name": "last_name" , "type": "string"},
    { "name": "first_name", "type": "string" },
    { "name": "city" , "type": "string"},
    {"name": "civility",
      "type": {
        "type": "enum",
        "name": "Civility",
        "symbols": ["MISS", "MR", "MS"]
      }
    },
    {"name": "country", "type": "string" },
    {"name": "phoneNumber", "type": [ "string", "null" ]},
    {"name": "email", "type": [ "string", "null" ]}
  ]
}
```

Subject : value-consumers

Pour plus d'information sur la modélisation des schémas, veuillez vous rendre sur ce site (<https://avro.apache.org/docs/1.8.1/spec.html#schemas>).

Enregistrement d'un schéma via REST APIs

```
curl -X POST -H "Content-Type: application/vnd.schemaregistry.v1+json" --data \
'{"schema": "{\"type\":\"record\", \"name\":\"Consumer\", \"namespace\":\"org.ippon.schemaregistry.avro\", \"fields\":[{\"name\":\"consumer_id\", \"type\":\"string\"}, {\"name\":\"last_name\", \"type\":\"string\"}, {\"name\":\"first_name\", \"type\":\"string\"}, {\"name\":\"city\", \"type\":\"string\"}, {\"name\":\"civility\", \"type\":{\"type\":\"enum\", \"name\":\"Civility\", \"symbols\":[\"MISS\", \"MR\", \"MS\"]}}, {\"name\":\"country\", \"type\":\"string\"}, {\"name\":\"phoneNumber\", \"type\": [ \"string\", \"null\" ]}, {\"name\":\"email\", \"type\": [ \"string\", \"null\" ]}}"}' \
http://localhost:8081/subjects/[TOPIC.NAME]-value/versions
```

Résultat :

```
{"id":1}
```

Enregistrement du schéma pour la valeur du schéma

Voici le résultat dans le Control Center, avec un TOPIC.NAME = test.ippon.topic

The screenshot shows the Confluent Control Center interface. On the left, there's a sidebar with 'Topics' selected. In the main area, under 'ALL TOPICS > test.ippon.topic', the 'Schema' tab is active. It shows a JSON schema definition for the 'Value' type:

```
{
  "type": "record",
  "name": "Consumer",
  "namespace": "org.ippon.schemaregistry.avro",
  "fields": [
    ...
  ]
}
```

Below the schema, there are buttons for 'Edit schema', 'Version history', and 'Download'. A note at the bottom right says 'Version 1'.

Vous pouvez donc produire l'événement suivant via vos clients Java, Scala ou via les scripts du Kafka.

```
{
    "customer_id": "abc-1234",
    "civility": "MISS",
    "last_name": "Z",
    "first_name": "X",
    "city": "Paris",
    "country": "FR",
    "email": "*****@ippon.tech",
    "phone_number": "+336xxxxxxxx"
}
```



Production des enregistrements

Maintenant que nous avons un environnement de développement, nous présentons le processus d'enregistrement d'un schéma et le code permettant de publier des messages conformes à ce schéma.

Pour la publication de nouveaux événements conformes à un schéma, deux possibilités :

- La première consiste à générer le code d'une classe *Builder* de l'événement et construire l'objet par la suite. La génération est automatique via Maven. Cette approche nécessite une compilation à chaque modification du schéma.
- La seconde consiste à utiliser un objet générique "GenericRecord" (<https://avro.apache.org/docs/current/api/java/index.html>) pour construire les messages à envoyer.

La partie d'envoi de messages est présentée ci-dessous, l'ensemble du projet est accessible sur github (<https://github.com/mhilia/schema-registry-poc/>).

```
KafkaProducer<String, Consumer> producer = new KafkaProducer<>(props);
Random r = new Random();
for(int i=0; i< NUM_RECORD ; i++){
    String consumerId = "eu#" + r.nextInt( bound: 5 ) + ( System.currentTimeMillis() - Long.MAX_VALUE );
    System.out.println(consumerId);

    final Consumer consumer = Consumer.newBuilder()
        .setConsumerId(consumerId)
        .setCivility(Civility.MISS)
        .setLastName("Z")
        .setFirstName("X")
        .setCountry("FR")
        .setCity("Paris")
        .setEmail("*****@ippon.tech")
        .setPhoneNumber("+336xxxxxxxx")
        .build();

    final ProducerRecord<String, Consumer> record = new ProducerRecord<>(TOPIC, consumer.getConsumerId().toString(), consumer);
    Future<RecordMetadata> future = producer.send(record);
    log.info("Message : TOPIC : %s, OFFSET : %s, PARTITION : %s", future.get().topic(), future.get().offset(), future.get().partition());
    producer.flush();
}
```

Consommation des enregistrements

Pour la partie consommation, rien n'est plus facile. Voici le code correspondant à la boucle de consommation de messages avec un envoi automatique des offsets.

```
KafkaConsumer<String, Consumer> kafkaConsumer = new KafkaConsumer<String, Consumer>(props);
kafkaConsumer.subscribe(Arrays.asList(TOPIC));

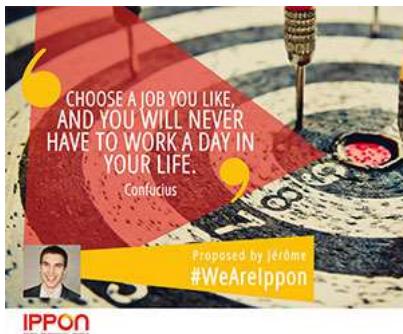
while (true) {
    ConsumerRecords<String, Consumer> records = kafkaConsumer.poll(Duration.ofMillis(1000));
    log.info("Number of Records : " + records.count());

    if (records.count() > 0)
        for (ConsumerRecord<String, Consumer> record : records) {
            log.info("Offset = %d, Key = %s, Value = %s", record.offset(), record.key(), record.value());
        }
}
```

L'ensemble du code du consommateur est en libre service sur github (<https://github.com/mhilia/schema-registry-poc/>).

Conclusion

Le Schema Registry est une brique essentielle dans nos architectures de données. Il représente une réponse optimale aux questions de gouvernance, d'évolution, d'organisation et de flexibilité. Les autres avantages de la mise en place d'un Schema Registry sont sa scalabilité et sa haute disponibilité pour répondre à des variations de charge.

POST RÉCENTS[Cucumber et RestTemplate](#)[Aug 31, 2020](#)[\(http://blog.ippon.fr/2020/08/31/cucumber-et-resttemplate/\)](http://blog.ippon.fr/2020/08/31/cucumber-et-resttemplate/)[REX, prise en main de Selenium WebDriver](#)[Aug 20, 2020](#)[\(http://blog.ippon.fr/2020/08/20/rex-sur-la-prise-en-main-de-selenium-webdriver/\)](http://blog.ippon.fr/2020/08/20/rex-sur-la-prise-en-main-de-selenium-webdriver/)[Construire la CI d'un monorepo: les parent-child pipelines de Gitlab-ci](#)[Jul 31, 2020](#)[\(http://blog.ippon.fr/2020/07/31/construire-la-ci-dun-monorepo-les-parent-child-pipelines-de-gitlab-ci/\)](http://blog.ippon.fr/2020/07/31/construire-la-ci-dun-monorepo-les-parent-child-pipelines-de-gitlab-ci/)**WE ARE IPPON**[\(http://www.ippon.fr/nous-rejoindre/\)](http://www.ippon.fr/nous-rejoindre/)**Partagez cet article:**

[**f**](#) (<https://www.facebook.com/sharer/sharer.php?u=http://blog.ippon.fr/2019/11/18/confluent-schema-registry-un-premier-pas-vers-la-gouvernance-des-donnees/>)

[**twitter**](#) (<https://twitter.com/share?text=Confluent%20Schema%20Registry%20un%20premier%20pas%20vers%20la%20gouvernance%20des%20donn%C3%A9es&url=http://blog.ippon.fr/2019/11/18/confluent-schema-registry-un-premier-pas-vers-la-gouvernance-des-donnees/>)

[**g+**](#) (<https://plus.google.com/share?url=http://blog.ippon.fr/2019/11/18/confluent-schema-registry-un-premier-pas-vers-la-gouvernance-des-donnees/>)

[**digg**](#) (<http://www.digg.com/submit?url=http://blog.ippon.fr/2019/11/18/confluent-schema-registry-un-premier-pas-vers-la-gouvernance-des-donnees/>)

[**reddit**](#) (<http://reddit.com/submit?url=http://blog.ippon.fr/2019/11/18/confluent-schema-registry-un-premier-pas-vers-la-gouvernance-des-donnees/&title=Confluent%20Schema%20Registry%20un%20premier%20pas%20vers%20la%20gouvernance%20des%20donn%C3%A9es>)

[**linkedin**](#) (<http://www.linkedin.com/shareArticle?mini=true&url=http://blog.ippon.fr/2019/11/18/confluent-schema-registry-un-premier-pas-vers-la-gouvernance-des-donnees/>)



[**stumbleupon**](#) (<http://www.stumbleupon.com/submit?url=http://blog.ippon.fr/2019/11/18/confluent-schema-registry-un-premier-pas-vers-la-gouvernance-des-donnees/&title=Confluent%20Schema%20Registry%20un%20premier%20pas%20vers%20la%20gouvernance%20des%20donn%C3%A9es>)

[\(/author/mohamed/\)](#)[Mohamed HILIA \(/author/mohamed/\)](#)

Dans ce billet nous ne nous sommes pas attardés sur l'évolution des schémas et la compatibilité ascendante.

Ce point sera détaillé dans un futur article. Dans cette vidéo (<https://www.youtube.com/watch?v=4ngG34TteoU&feature=youtu.be>) je présente rapidement comment ajouter un attribut "age" à un schéma avec un attribut "age".



Références

- Blog post: Schemas, Contracts, and Compatibility (https://www.confluent.io/blog/schemas-contracts-compatibility?_ga=2.164627766.822023958.156675058-51442238.1563800618&_gac=1.251815675.1566565111.CjwKCAjwnf7qBRAteiwAseBO_JfY2_uJdTUNaxfMnEjkdOHbRLX)
- Blog post: 17 Ways to Mess Up Self-Managed Schema Registry (<https://www.confluent.io/blog/17-ways-to-mess-up-self-managed-schema-registry>)
- Blog post: Yes, Virginia, You Really Do Need a Schema Registry (<https://www.confluent.io/blog/schema-registry-kafka-stream-processing-yes-virginia-you-really-need-one/>)
- Blog post: How I Learned to Stop Worrying and Love the Schema (<https://www.confluent.io/blog/how-i-learned-to-stop-worrying-and-love-the-schema-part-1/>)
- <https://mapr.com/docs/61/Kafka/KafkaSchemaRegistry/KafkaSchemaRegistry.html> (<https://mapr.com/docs/61/Kafka/KafkaSchemaRegistry/KafkaSchemaRegistry.html>)
- Evolution de schémas <https://www.youtube.com/watch?v=4ngG34TteoU&feature=youtu.be> (<https://www.youtube.com/watch?v=4ngG34TteoU&feature=youtu.be>)
- Code source : <https://github.com/mhilia/schema-registry-poc> (<https://github.com/mhilia/schema-registry-poc>)

Vous avez trouvé cette publication utile? Cliquer sur

LIVRE BLANC

Explore Big Data.

Ce livre blanc prend "un peu" de hauteur sur les technologies Big Data afin d'aider les décideurs à faire les bons choix techniques et à y voir plus clair sur les opportunités offertes par les différentes technologies.

[JE TÉLÉCHARGE \(HTTPS://FR.IPPON.TECH/EXPLORE-BIG-DATA/\)](https://fr.ippon.tech/explore-big-data/)

VIDÉO MÉTIER

Architecte Data @Ippon c'est ... (avec Arnaud Col)



REX CLIENTS

Ippon & ENGIE Digital : Accélération du produit eCare



Ippon

Ippon est un cabinet de conseil en technologies, créé en 2002 par un si
ambition de devenir leader sur les solutions Digitales, Cloud et BigData



(<http://blog.ippon.fr>)

Ippon accompagne les entreprises dans le développement et la transformation de leur système d'information avec des applications performantes et des solutions robustes.

Ippon propose une offre de services à 360° pour répondre à l'ensemble des besoins en innovation technologique : Conseil, Design, Développement, Hébergement et Formation.

Nous avons réalisé, en 2019, un chiffre d'affaires de 42 M€. Nous sommes aujourd'hui un groupe international riche de plus de 400 consultants répartis en France, aux USA, en Australie et en Russie.

📍 FRANCE 🌐 WEBSITE ([HTTP://WWW.IPPON.FR](http://WWW.IPPON.FR))

LinkedIn ([HTTPS://WWW.LINKEDIN.COM/COMPANY/IPPON-TECHNOLOGIES](https://WWW.LINKEDIN.COM/COMPANY/IPPON-TECHNOLOGIES))

Post précédent

Vault - Une présentation - Partie 2/3

(/2019/11/26/vault-une-presentation-partie-2-3/)

Poste suivant

Jenkins Pipelines et Shared Librairies

(/2019/11/18/untitled-3/)

ÉGALEMENT SUR BLOG IPPON TECHNOLOGIES

Trois ans en compagnie de ... il y a 2 ans • 1 commentaire Dans mon précédent article ...	Scrub : Un atelier agile pour fabriquer ... il y a 2 ans • 2 commentaires Scrub est un atelier agile pour fabriquer des cubes. Il permet de découvrir ...	AWS re:Invent 2018 - Jour 1 il y a 2 ans • 1 commentaire Annonces AWS re:Invent 2018 - Jour 1	Couvert multi-m ... il y a 2 ans Cet article comment couverture
--	--	--	--

0 Commentaires

Blog Ippon Technologies

🔒 Règles de confidentialité de Disqus

S'identifier ▾

♥ Recommander

Tweet

Partager

Les meilleurs ▾



(<http://www.ippon.fr>)

Discovery to Delivery

Ippon est un cabinet de conseil qui accélère les projets innovants des ses clients de la page blanche au Cloud.

Nos équipes dans le monde accompagnent les organisations dans la transformation d'idées innovantes en solutions logicielles de haute qualité avec un focus particulier sur le Time To Market.

© 2020 IPPON (<http://blog.ippon.fr>). Tous les droits sont réservés.