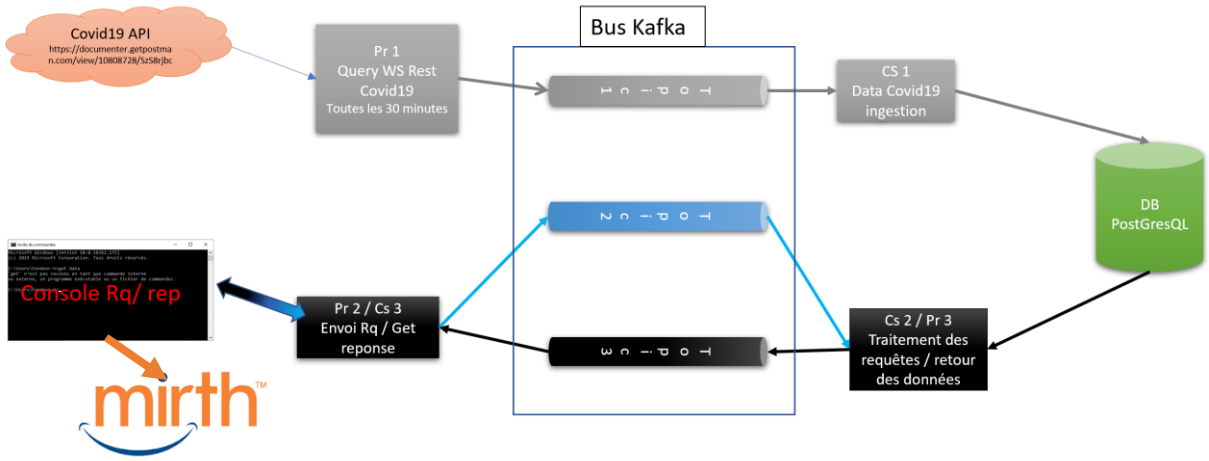
TP Intergiciel
Utilisation de Kafka

Objectif

Ecrire une application en mode console texte ou web avec le langage Java en utilisant les API Kafka-Client et/ou le Framework SpringBoot qui permettra de lancer des requêtes « Custom » au travers d'un producteur Kafka « Pr2 » vers le Topic Kafka « Topic2 », ce message de requête sera interprété par le custom consommateur « Cs2 » connecté à ce « Topic2 » et permettra de requêter la base de données PGSQL selon le besoin exprimé par la requête (interprétation de la commande en requête SQL).
Le retour des informations se fera par le custom producteur « Pr3 » vers le Topic Kafka « Topic3 », le consommateur « Cs3 » consommera les messages pour le présenter en affichage sur la console.

La base de données sera alimentée par les données provenant du site « COVID19 API » toutes les 30 minutes, un custom producteur « Pr1 » va réaliser cette requête API vers le site « COVID19 API » et pousser le résultat vers le topic Kafka « Topic1 », un consommateur custom « Cs1 » va consommer le message et ingérer les données dans la base PostgreSQL dédié.

Synoptique des échanges



Descriptions des données sources

Le site COVID 19 API (https://covid19api.com/), donne accès gratuitement aux données sur COVID19 via une API simple sous licence Creative Commons Attribution 4.0 International. Les données proviennent de Johns Hopkins CSSE.

https://github.com/CSSEGISandData/COVID-19 , a cette adresse vous trouverez toutes les origines des sources de données et plus d'infos sur les éléments.

Une documentation de ces API se trouvent à cette adresse :
https://documenter.getpostman.com/view/10808728/SzS8rjbc

L'API essentielle qui va nous intéresser dans notre projet est la suivante :
https://api.covid19api.com/summary

Qui donne un résumé des nouveaux cas et du nombre total de cas par pays mis à jour quotidiennement.

Le format de retour des données est de type JSON et comporte le schéma ci-dessous :

```
{
	"Message": String,
	"Global": {
		"NewConfirmed": number,
		"TotalConfirmed": number,
		"NewDeaths": number,
		"TotalDeaths": number,
		"NewRecovered": number,
		"TotalRecovered": number
	},
	"Countries": [
		{
			"Country": String,
			"CountryCode": "AF",
			"Slug": "afghanistan",
			"NewConfirmed": number,
			"TotalConfirmed": number,
			"NewDeaths": number,
			"TotalDeaths": number,
			"NewRecovered": number,
			"TotalRecovered": number,
			"Date": DateTimeISO8601,
			"Premium": Object
		},
		...
	],
	"Date": DateTimeISO8601
}
```

Exemple simplifié de réponse :

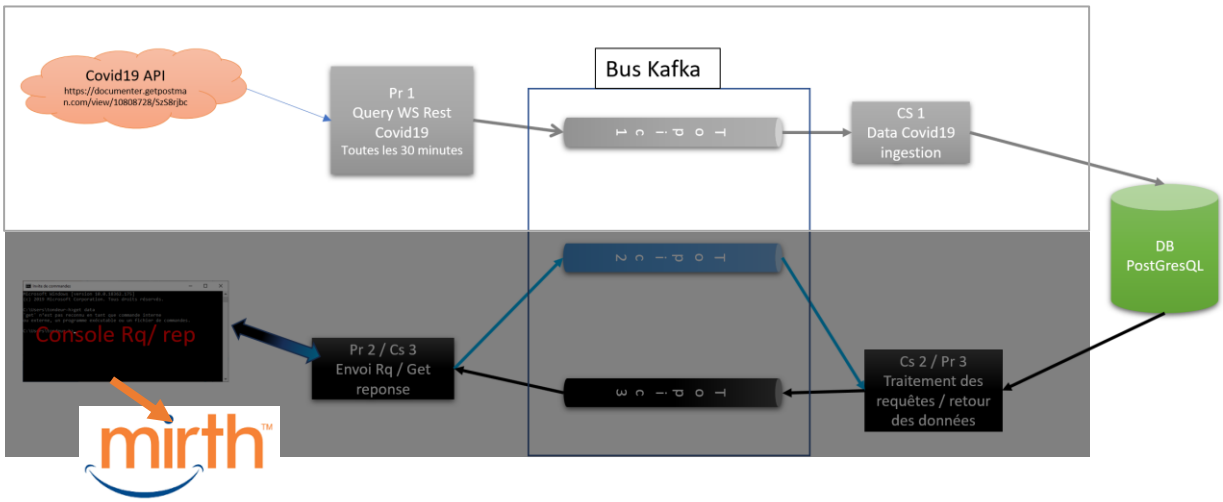
```
{
	"Message": "",
	"Global": {
		"NewConfirmed": 468460,
		"TotalConfirmed": 41688829,
		"NewDeaths": 5884,
		"TotalDeaths": 1137115,
		"NewRecovered": 226280,
		"TotalRecovered": 28339727
	},
	"Countries": [
		{
			"Country": "Afghanistan",
			"CountryCode": "AF",
			"Slug": "afghanistan",
			"NewConfirmed": 116,
			"TotalConfirmed": 40626,
			"NewDeaths": 4,
			"TotalDeaths": 1505,
			"NewRecovered": 7,
			"TotalRecovered": 33831,
			"Date": "2020-10-23T08:20:32Z",
			"Premium": {}
		}
	],
	"Date": "2020-10-23T08:20:32Z"
}
```

Dans cette réponse on ne conservera pas le champ « Message », ni le champ « Countries.Premium » qui n'ont pas d'intérêt dans notre développement.

On pourra translater ces données en deux tables :

Covid19.Global(NewConfirmed:Integer, TotalConfirmed:Integer, NewDeaths:Integer, TotalDeaths:Integer, NewRecovered:Integer, TotalRecovered :Integer, Datemaj :TimeStamp)
Covid19.Countries(Country:varchar(200),CountryCode:varchar(6),Slug:vzrchar(200),NewConfirmed:Integer,TotalConfirmed:Integer, NewDeaths: Integer, TotalDeaths: Integer,NewRecovered: Integer,TotalRecovered: Integer,Datemaj: TimeStamp)

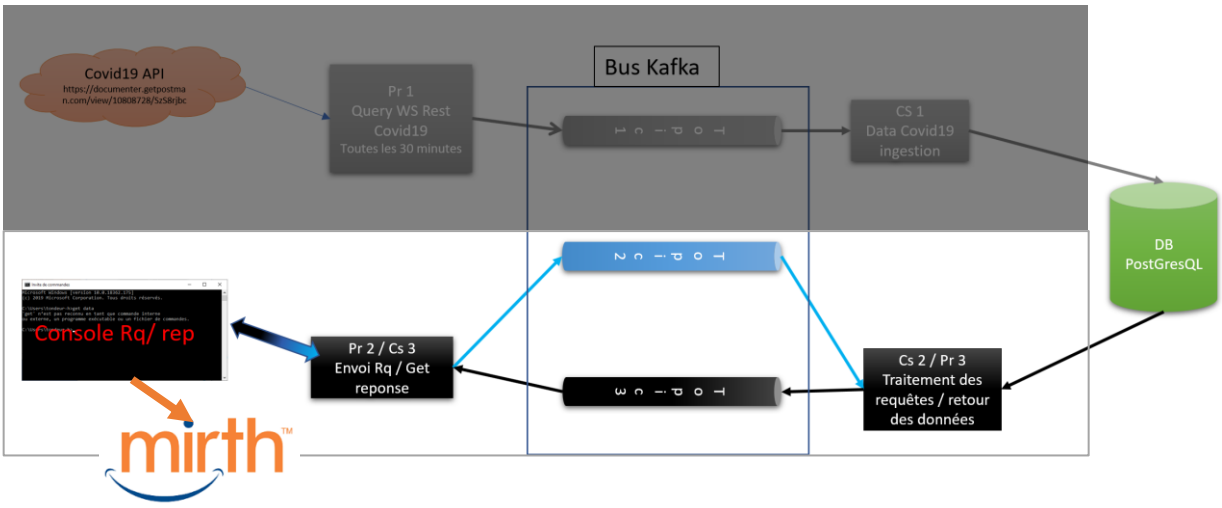
Description des échanges et programmes à développer



On va développer un premier Producteur Kafka (noté « Pr1 » dans le schéma) qui va permettre toutes les 30 minutes de requêter cette API et de récupérer les données de réponses au format JSON.
Ce fichier sera transmis au bus Kafka qui assurera le transport vers notre base de données PostgreSQL.

L'ingestion de ces données pourra se faire de deux manières différentes par le consommateur « Cs1 »,

- Soit par le Parse du fichier JSON et intégration de ces données dans une table d'une base de données relationnelle classique selon le schéma décrit plus haut (fin de page 2).
- Soit par l'intégration du fichier JSON dans une table PostgreSQL supportant le format JSONB et les requêtes de type JSON Path (https://www.postgresql.org/docs/12/datatype-json.html & https://www.postgresql.org/docs/12/functions-json.html) (pour les plus téméraires).



La console Texte ou Web « Cs » permettra de lancer des commandes simples (voir liste ci-dessous), ces commandes seront envoyées vers le bus Kafka par le producteur « Pr3 » sous forme de messages simple sur le Topic Kafka « Topic2 », un consommateur « Cs2 » permettra de consommer ces messages de type commandes, et va transformer ces demandes en requêtes SQL ou JSON Path qui seront exécuté sur la base de données PostgreSQL.

Le résultat de la demande sera envoyé sous forme d'un message sur le Bus Kafka via le producteur custom « Pr3 » vers le Topic Kafka « Topic3 » (ici le format n'a pas d'importance et reste à votre entière inventivité).

Le custom consommateur « Cs3 » va consommer ces messages sur le Topic Kafka « Topic3 » et afficher en retour le contenu du message sur la console Texte ou Web vers laquelle est partie la demande.

Liste des commandes de la console

Get_global_values (retourne les valeurs globales clés Global du fichier json)
Get_country_values v_pays (retourne les valeurs du pays demandé ou v_pays est une chaine de caractère du pays demandé)
Get_confirmed_avg (retourne une moyenne des cas confirmés sum(pays)/nb(pays))
Get_deaths_avg (retourne une moyenne des Décès sum(pays)/nb(pays))
Get_countries_deaths_percent (retourne le pourcentage de Décès par rapport aux cas confirmés)
Un fonction export qui permet d'exporter les données de la base de données en XML (penser à xmlencoder) dans un fichier ou via une socket tcp/ip, données qui pourront être consommées par un EAI Mirth Connect et mis à disposition sous forme de socket par cet EAI mirth connect…

Help (affiche la liste des commandes et une explication comme ci-dessus)

Vous pouvez proposer d'autres commandes possibles …

Livraisons attendues

Date limite de remise le : Vendredi 1 avril 2022

- Les commandes de création des Topics (Topic1, Topic2, Topic3)
- La commande SQL CREATE de la base et table d'ingestion des données.
- Le code source du custom Producteur (Pr1)
- Le code source du custom Consommateur (Cs1)
- Le code source du custom Consommateur et Producteur (Cs2, Pr3)
- Le code source du custom consommateur, Producteur et console (Pr2, Cs3, Cs)

Uniquement les codes sources, soit via un fichier zip sur les boites mails herve.tondeur@uphf.fr et adresse de sécurité sur tondeur.herve@yahoo.fr

Soit de préférence, par un lien vers un Git public qui sera transmis par mail aux adresses ci-dessus avec les noms des développeurs dans le mark down README qui accompagnera obligatoirement ce projet.

Il est possible et conseillé de travailler a plusieurs sur ce TP, dans le cas d'un travail collaboratif, me fournir l'ensemble des noms des collaborateurs dans le message du mail et sur le readme en mark down.