Broker - Les logs d'utilisation de l'API V2 du module broker-module-livedata

Sommaire

Chargement des broks initiaux par un regenerator (créateur d'objets des modules de broker) et vérifier que c'est bien la même configuration charger entre les regenerators / Scheduler / Arbiter

Les logs suivants permettent de suivre le chargement de la configuration de supervision entre l'arbiter les schedulers jusqu'aux interfaces : webui / livestatus / livedata

Il existe 2 types d'identifiants de configuration (représentation de la configuration)

- **configuration_uuid**: uuid de configuration totale générée par l'Arbiter
- **configuration_part_id**: id de la partie de configuration géré par un Scheduler

Quand un module de broker avec un regenerator charge une nouvelle configuration :

[2020-05-15 16:29:49] INFO : [WebUI3] [CONFIGURATION] [NEW] [REGENERATOR] configuration part retrieved: [configuration_part_id=**configuration_part_id**, scheduler=**scheduler_name** configuration_uuid=**configuration_uuid**, arbiter=**arbiter_name** date=**creation_date**]

- **configuration_part_id:** id de la partie de configuration gérée par le Scheduler (unique par Scheduler)
- **scheduler_name:** nom du Scheduler qui gère cette partie de la configuration
- **configuration_uuid:** uuid crée lors du démarrage de l'Arbiter qui correspond donc à l'id de la configuration gérée par l'Arbiter
- **creation_date:** date du démarrage de l'Arbiter
- **arbiter_name:** nom de l'Arbiter qui a créé cette configuration

Exemple Log Broker - module WebUI3 chargement de la nouvelle configuration

```
[YYYY-MM-DD HH:MM:SS] INFO   : [WebUI3] [ CONFIGURATION ] [ NEW ] [ REGENERATOR ] configuration part
retrieved : [ configuration_part_id=8, scheduler=scheduler-master
configuration_uuid=fe5982b29bfb48cdadb35523799f7cec, arbiter=arbiter-master1 date=15-05-2020 16:13:40 ]
```

Quand un module de broker avec un regenerator rejette une configuration :

Dans le cas où la configuration d'un Scheduler est déjà gérée par un regenerator (cas qui arrive si par exemple un module crash) on redemande les broks initiaux. Tous les modules vont recevoir la nouvelle configuration, mais ceux qui la gère déjà ne vont pas la recharger et vont loguer :

```
[YYYY-MM-DD HH:MM:SS] INFO : [WebUI3] [ CONFIGURATION ] [ NEW ] [ REGENERATOR ] No need to reload the
configuration part because I already handle it [ configuration_part_id=configuration_part_id,
scheduler=scheduler_name configuration_uuid=configuration_uuid, arbiter=arbiter_name date=creation_date ]
```

- **configuration_part_id:** id de la partie de configuration gérée par le Scheduler (unique par Scheduler)
- **scheduler_name:** nom du Scheduler qui gère cette partie de la configuration
- **configuration_uuid:** uuid crée lors du démarrage de l'Arbiter qui correspond donc à l'id de la configuration gérée par l'Arbiter
- **creation_date:** date du démarrage de l'Arbiter
- **arbiter_name:** nom de l'Arbiter qui a créé cette configuration

Exemple Log Broker - module WebUI3 chargement de la nouvelle configuration

```
[YYYY-MM-DD HH:MM:SS] WARNING: [WebUI3] [ CONFIGURATION ] [ NEW ] [ REGENERATOR ] No need to reload the
configuration part because I already handle it [ configuration_part_id=8, scheduler=scheduler-master
configuration_uuid=fe5982b29bfb48cdadb35523799f7cec, arbiter=arbiter-master1 date=15-05-2020 16:13:40 ]
```

Temps de locks trop long entre la consommation des Broks et les requêtes des utilisateurs

Actuellement on ne sait pas consommer les broks, et répondre aux utilisateurs en même temps. On a donc une concurrence entre deux parties:

- Récupération et consommation des broks depuis le broker, et mise à jour des hôtes/checks/clusters (et tous les autres objets) depuis les informations des broks
- Réponses aux requêtes des utilisateurs (parcours des hôtes, checks, clusters, ...)

Un des principal risque est une famine d'un des deux groupes d'actions:

- si on ne fait qu'avaler des broks et ne jamais répondre aux utilisateurs, ceci va poser problème
- symétriquement, si on ne fait que répondre aux utilisateurs, et jamais avaler des broks, on va avoir des informations périmées, voir on ne finira jamais de consommer de nouvelles configurations

Le gestionnaire de lock essaie de partager au mieux le temps d'exécution entre les deux groupes, en cas de forte charge, des logs vont remonter les lenteurs observées

Quand on a trop de requêtes de lectures, et qu'elles ne rendent pas la main pendant plus de 30s aux broks, on aura un log suivant (Brok BLOQUE par les requêtes):

```
ERROR: [ ITEMS ACCESS ORDONNANCER ] [ LONG LOCK ] Broks management are waiting (1 thread) since 30s (> log error limit=30s) because HTTP resquests (20 threads) has the LOCK
```

Quand on a trop de consommation de Broks, et que les requêtes sont bloquées (Requêtes utilisateurs BLOQUÉES par les Broks)

```
ERROR: [ ITEMS ACCESS ORDONNANCER ] [ LONG LOCK ] HTTP resquests are waiting (5 threads) since 30s (> log error limit=30s) because Broks management (1 thread) has the LOCK
```

Quand les requêtes en lecture mettent trop de temps à rendre la main au consommateur de Broks et que d'autres requêtes en lecture attendent de pouvoir s'exécuter depuis trop longtemps :

```
ERROR: [ ITEMS ACCESS ORDONNANCER ] [ LONG LOCK ] Still have 9 running tasks ongoing (HTTP resquests). => ( 1 ) Broks management and then ( 11 ) HTTP resquests are waiting since 30s (>= log error limit:30s)
```

Quand la consommation de Broks met trop de temps à rendre la main pour la gestion de requêtes en lecture, et que d'autres consommateurs attendent de s'exécuter depuis trop longtemps (cas théorique, n'est pas supposé survenir en fonctionnement normal) :

```
ERROR: [ ITEMS ACCESS ORDONNANCER ] [ LONG LOCK ] Still have 1 running tasks ongoing (Broks management). => ( 12 ) HTTP requests then ( 1 ) Broks management are waiting since 30s (>= log error limit:30s)
```