STACK LABS

Pascal Gillet <pascal.gillet@stack-labs.com>

[hippi-spark-k8s] Feedback - Suivi de mission

Pascal Gillet <pascal.gillet@stack-labs.com> 23 novembre 2020 à 17:38
À : Florence BRARD <FBRARD@sii.fr>, David Desviel <david.desviel@stack-labs.com>, Xavier Villalba <xavier.villalba@stack-labs.com>, Pascal ROUANET <PROUANET@sii.fr>, "QUINCHARD, Eric" <eric.quinchard@airbus.com>, "DELBENDE, Marc" <marc.delbende@airbus.com>, "ROUGET, Matthieu" <matthieu.rouget@airbus.com>

Bonjour à Tous,

Voici les travaux réalisés la semaine dernière:

État d'avancement: **Warning**

- **Spark Web UI:**
Le but est d'exposer la Spark Web UI à l'extérieur du cluster k8s. Ceci peut être fait au travers d'un service de type `NodePort` et/ou d'une `Ingress`.
La solution `NodePort` est simple à mettre en œuvre mais nécessite que les nœuds du cluster soient accessibles avec leur adresse IP externe.
La solution `Ingress` (point d'entrée unique) est préférée dans un contexte de production, et dans l'éventualité d'ouverture de la plate-forme à des clients extérieurs.
La solution `Ingress` est néanmoins plus difficile à configurer (nécessite de la conf au niveau Spark et au niveau load balancer) pour contourner un problème de redirect HTTP et révèle un bug de l'UI: la page des executors reste blanche (voir https://github.com/jupyterhub/jupyter-server-proxy/issues/57#issuecomment-699163115)
La solution `Ingress` a pour l'instant été testée avec un *controller* Nginx.

Ce besoin, bien que nécessaire, a été omis par Airbus et par Stack Labs lors de la définition de mission. J'ai travaillé une bonne partie de la semaine passée sur ce point en particulier, et ça a décalé d'autant le travail d'intégration dans la librairie spark-client. Pour cette raison, je mets le statut à **Warning**.

- **Ownership/dependent objects & garbage collection:**
Le rôle du garbage collector Kubernetes est de supprimer certains objets qui avaient autrefois un propriétaire, mais qui n'en ont plus.
Le but est de s'assurer que le garbage collector supprime bien les ressources devenues inutiles lorsqu'on *kill* une application Spark.
Il est important de bien libérer les ressources du cluster k8s lorsqu'on va exécuter des dizaines/centaines d'applications Spark en parallèle.

Pour cela, certains objets Kubernetes peuvent être déclarés propriétaires d'autres objets. Les objets "possédés" sont appelés dépendants de l'objet propriétaire. Chaque objet dépendant a un champ `metadata.ownerReferences` qui pointe vers l'objet propriétaire. **Quand on supprime un objet propriétaire, tous les objets dépendants sont supprimés en cascade (par défaut).**

Le Spark Operator définit automatiquement la valeur de `ownerReference`, et l'objet propriétaire est la ressource custom `SparkApplication`.

Côté spark-submit, l'objet propriétaire est le pod `driver` et les pods *executors* fixent aussi automatiquement le champ *ownerReference*. Pour les autres ressources de type `ConfigMap`, `Service` et `Ingress`, il a fallu le définir "manuellement". Ceci est géré de manière impérative dans le code, et non pas déclarative dans les fichiers de définition YAML (on doit en effet récupérer l'`uid` auto-généré du pod `driver` nouvellement créé et l'injecter dans les objets dépendants, ceci ne peut se faire qu'au runtime dans le code).

Pour les applications normalement terminées, les pods executors sont `terminated` et sont nettoyés, mais le pod `driver` conserve les logs et reste à l'état "completed" dans l'API Kubernetes "*jusqu'à ce qu'il soit finalement garbage collecté ou nettoyé manuellement*".
(Notez que dans l'état `completed`, le pod `driver` n'utilise aucune ressource de CPU ou de mémoire.)
Je ne vois rien dans la doc qui précise comment les pods `driver` sont supprimés au final. À terme, on peut imaginer un simple `CronJob` k8s qui s'exécuterait périodiquement pour les supprimer automatiquement.

- **ConfigMap "dynamique":**
Comme les objets sont maintenant automatiquement *garbage collectés*, il n'y a plus aucune raison de faire de l'économie de ressources et on peut maintenant gérer une `ConfigMap` par job Spark, avec une configuration spécifique pour ce job. On peut donc réintroduire l'info de priorité et d'autres paramètres qui seront définis au runtime.

- **Conventions de nommage: labels & selectors**
Le nommage des objets K8s au spark-submit a été revu de manière à être consistant avec le nommage Spark Operator.
On peut ainsi gérer et requêter les objets de la même manière qu'ils aient été créés via Spark Operator ou via spark-submit.

Suite à la réunion de ce matin:
[Éric ou Marc] Est-ce que tu peux me confirmer que vous utilisez bien Traefik comme Ingress controller, auquel cas je l'utiliserai / configurerai de mon côté?

TODO de mon côté:
- Paramétrer la `node affinity` dans les templates YAML, et séparément pour les pods `driver` et `executors`.
- Paramétrer spark.app.name (le problème se posera de toute façon au moment de l'intégration)
- Paramétrer le nom de l'application racine ${APP_ROOT_NAME}
- Déplacer au maximum la conf du job Spark dans la `ConfigMap`

Cordialement,
--
STACK LABS
Pascal GILLET | Big Data & Cloud Architect
stack-labs.com
21 Boulevard de la Marquette, 31000 Toulouse
France · Canada