STACK LABS

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

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

Pascal Gillet <pascal.gillet@stack-labs.com> 9 novembre 2020 à 14:25
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: **Nominal**

- Gestion des priorités dans le Spark Operator:
Nous avons vu la semaine dernière que l'info de priorité se trouve uniquement au niveau de la ressource `SparkApplication`. Cela garantit l'ordre du scheduling pendant la mise en file d'attente des applications Spark mais ne permet pas la préemption des jobs, car le driver et les executors se retrouvent tous avec la même priorité (pas de priorité en fait (0)).
J'ai créé une MR dans le projet GoogleCloudPlatform/spark-on-k8s-operator pour pallier à ce problème:
https://github.com/GoogleCloudPlatform/spark-on-k8s-operator/pull/1054
À voir si elle sera acceptée en l'état.

- Comment remonter l'info de préemption d'un job?
Il est possible de débugger et introspecter des applications Kubernetes avec `kubectl get events`. Notamment, on peut voir quand un pod a été préempté (voir ex).
Cette solution n'est pas idéale car il faut surveiller (watch) en permanence les events, être capable de distinguer les derniers events depuis la dernière requête, etc.

Une meilleure solution consisterait à déclencher du code via des hooks sur le cycle de vie des containers. Notamment, le hook `PreStop` est appelé immédiatement avant qu'un conteneur se termine en cas de préemption.

- **Scénarios souhaités en cas de préemption:**
[EQU] Idéalement, on souhaite re-scheduler automatiquement un job qui a été préempté. Ceci pose le problème d'une gestion trop automatisée des jobs qui finirait par s'accaparer les ressources du cluster (voir les quotas de ressources par classe de priorité)...
/! La gestion de la préemption n'est pas prioritaire et sera envisagée s'il reste du temps sur le projet.

- Premiers pas avec le client Python de Kubernetes:
- Création d'un compte de service avec les seuls droits nécessaires pour créer des applications Spark depuis le client Python (principe de moindre privilège). À ne pas confondre avec le compte de service nécessaire au Spark driver pour créer les pods executors.
- Récupération des infos relatives à ce compte de service dans un fichier `kubeconfig`
- Authentification en Python avec `config.load_kube_config("path/to/kubeconfig-file")`
- Submission d'un job Spark (spark-submit en mode client en argument de l'image Docker). Les `spec` des ressources k8s nécessaires sont sous forme de templates YAML instanciés au runtime.
- Listing des pods dans le namespace "spark-jobs"
- Suppression (kill) des pods et services d'une application Spark

Marc, Éric, je vous ai ajoutés au repo Gitlab en tant que "reporter". Vous pouvez faire vos remarques directement au niveau du code ou de la doc. Vous pouvez aussi créer des *issues*.
Normalement, vous avez dû recevoir une invitation. Dites-moi si vous avez des problèmes d'accès.

Travaux prévus cette semaine, à minima:
- Submission d'un job Spark en Python avec le Spark Operator
- Obtenir le statut du pod driver
- Obtenir les logs du pod driver

Cordialement,
--
STACK LABS
Pascal GILLET | Big Data & Cloud Architect

stack-labs.com
21 Boulevard de la Marquette, 31000 Toulouse
France · Canada