STACK LABS

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

Doc Spark K8S

Pascal Gillet <pascal.gillet@stack-labs.com> 4 décembre 2020 à 11:34
À : "DELBENDE, Marc" <marc.delbende@airbus.com>
Cc : "QUINCHARD, Eric" <eric.quinchard@airbus.com>

Bonjour Marc,

Mes réponses dans le texte.

Pascal

Le jeu. 3 déc. 2020 à 17:33, DELBENDE, Marc <marc.delbende@airbus.com> a écrit :

> Airbus Amber
>
> Pascal,
>
> Suite à une première relecture des fichiers « *.md » livrés, j'ai quelques questions / remarques.
>
> Il est possible que certains de ces points soient déjà validés par les développements déjà réalisés …
>
> Merci pour tes réponses,
>
> MDE
>
> **docs/pod-priority-preemption.md:**
>
> *« There is no need to specify a `priorityClassName` on executor pods. Only driver pods must be prioritized. If a driver*
>
> *pod is preempted in favor of a driver with an higher priority, it will be terminated, and so its executors. This is all*
>
> *the more true that drivers and executors run on two types of nodes (node affinities)."*
>
> C'est vrai pour les priorités préemptives mais pas pour les autres : elles doivent être propagées aux exécuteurs (pour que des jobs « urgent » arrivés après soient schédulés avant les « routine » par exemple).

Cette doc compilait mes premières réflexions, puis nous avons débattu de ce sujet avec Éric. Depuis, la priorité est bien propagée aux executors en natif. Côté Spark Operator, j'ai fait une PR pour propager la priorité de la `SparkApplication` aux pods driver et executors.

J'ai corrigé la doc.

> **k8s_python_client_examples/README.md:**
>
> *"But we **DO NOT** want to rely on the default `kubeconfig` file. Instead, we're going to generate one especially for*
>
> *the service account created above, with the help of the script [`kubeconfig-gen.sh`](./kubeconfig-gen.sh)."*
>
> *"Here, we're going to configure the Python client in the most programmatic way possible.*

> *First, we need to fetch the credentials to access the Kubernetes cluster. We'll store these in python environmental*
>
> *variables.*
>
> ***```bash***
>
> *export APISERVER=$(kubectl config view --minify -o jsonpath='{.clusters[0].cluster.server}')*
>
> *…"*
>
> Là, quelque chose m'échappe : la commande « *kubectl config view …* » nécessite la présence du fichier kubeconfig dont on tente de se passer …
>
> Par ailleurs, ce fichier « kubeconfig » sera utilisé pour tous les autres outils et accès à Kubernetes, alors pourquoi le contourner ?

> Le script `kubeconfig-gen.sh` utilise effectivement le fichier `kubeconfig` par défaut (celui spécifié dans la variable d'env `KUBECONFIG` ou à défaut, dans `~/.kube/config`).

> Ce fichier `kubeconfig` est le tien, en tant qu'utilisateur de la commande `kubectl`. Concrétement, tu as le droit de faire un peu près tout dans le cluster K8s, et ce dans tous les namespaces.

> Le but du script est de générer un autre fichier `kubeconfig` qui configure l'accès au cluster pour le compte de service `python-client-sa`, avec les seuls droits nécessaires pour le spark_client dans le seul namespace `spark-jobs` ("principe de moindre privilège").

> **k8s_python_client_examples/garbage-collection.md :**
>
> *"When an application completes normally, the executor pods terminate and are cleaned up, but the driver pod persists*
>
> *logs and remains in "completed" state in the Kubernetes API "until it's eventually garbage collected or cleaned up*
>
> *manually".*
>
> *Note that in the completed state, the driver pod does not use any compute or memory resources.*
>
> *There is nothing in the doc that specifies how driver pods (and `SparkApplications`) are ultimately deleted*
>
> *. Ultimately, we can think of a simple Kubernetes [`CronJob`](https://kubernetes.io/docs/concepts/workloads/controllers/cron-jobs/)*
>
> *that would run periodically to delete them automatically."*
>
> Dans le cas des "sparkapplication", le tag « .spec.timeToLiveSeconds » sert à définir la durée de rétention des jobs terminés.

Bien vu, je l'ai ajouté dans le YAML. J'ai mis une journée (86400s).

> Dans le cas du submit, ne faudrait-il pas plutôt déployer des « jobs » plutôt que des « pods » dans les templates ?
>
> Le positionnement du tag « .spec.ttlSecondsAfterFinished » permettrait alors de régler ce problème …
>
> Alors j'ai essayé de convertir le Pod driver en Job. C'est direct puisqu'un Job est juste une ressource "chapeau" qui contient un template de Pod.

Le Job lance effectivement un Pod driver qui va créer les Pods executors...Mais le Pod driver crashe au démarrage avec une exception pas très claire liée au `SparkContext`. Kubernetes ajoute automatiquement un suffixe unique au nom du driver pod. Or, on a besoin du nom du pod driver pour le transmettre aux executors avec la

propriété spark.kubernetes.driver.pod.name . On pourrait retrouver le nom complet par introspection avant de le transmettre, mais c'est un petit peu compliqué. Je suppose que le problème vient de là...Je laisse tomber pour l'instant, mais je réessaierai s'il me reste du temps.

Pour info, il y a des demandes en cours pour supporter le TTL dans les pods: *"TTL controller only handles Jobs for now, and may be expanded to handle other resources that will finish execution, such as Pods and custom resources."*

Aussi, le Spark Operator fonctionne exclusivement avec des pods et non pas des jobs. Côté natif, la doc Spark K8s n'évoque jamais cette possibilité de passer par un job...Sujet à creuser!

[Texte des messages précédents masqué]

--

STACK LABS

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