new Blog(perso);

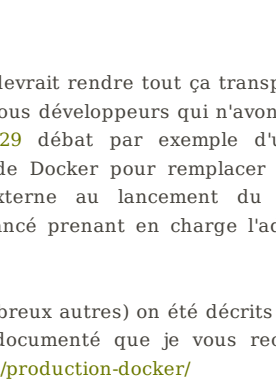
Yet another Java blog, comme on dit

03 août 2015

Docker Zombies

Un intérêt principal pour moi à maintenir l'image Docker de Jenkins c'est que j'apprend plein de choses, entre autre sur la partie Système/Linux pour laquelle j'ai une bonne marge de progression :)

Récemment, on m'a rapporté des problèmes de zombies avec l'image Jenkins, que j'ai dans un premier temps rejeté en demandant de montrer que ce problème est propre à l'utilisation de Docker et non un problème Jenkins général - je reçois en effet pas mal de bug report de ce style qui n'ont rien à voir avec Docker.



Sauf que pour une fois il y avait bien un loup.

Quel est le problème ?

Jenkins lance pas mal de process externes à la JVM, par exemple pour faire le checkout ou tout simplement pour exécuter les outils de build. Lorsque ce process-fils se termine, il passe dans l'état *defunct* (zombie) et le noyau le faire savoir à son parent via un signal SIG_CHLD, parent qui va faire un wait pour acquitter la fin de vie de son fils et permettre au process d'être éliminé complètement.

Suite a un arrêt brutal d'un process (KILL), les process fils se retrouvent orphelins et l'OS n'a plus de parent à notifier pour prendre en charge le nettoyage, aussi il les propose pour l'adoption à *init*, le process d'id 1. Ce mécanisme est un peu le garbage collector du système. Dans le cas de Jenkins, on peut par exemple considérer le plantage d'un build qui laisse tout un graphe de processus fils dans la nature.

init est prévu pour ça et fait donc tout ce qu'il faut, mais … mais nous sommes dans un conteneur Docker, et via le mécanismes des namespaces le process d'ID 1 n'est pas un *init*, mais le shell script qui sert de point d'entrée à l'image. Et bien sur bash n'est pas du tout conçu pour adopter des process zombie et les enterrer proprement.

La solution ?

Ce problème de terminaison correcte des processus est expliqué sur cet article et montre qu'il faut faire pas mal de chose pour bien traiter les signaux. Le problème évidement c'est qu'on est pas du tout habitués à devoir gérer ce genre de choses vu qu'on a en temps normal un OS et son *init* pour s'en charger.

Pour l'image Docker Jenkins j'ai utilisé tini, un petit bout de programme qui fait juste ce qu'il faut comme un *init* normal dans le contexte d'un conteneur : passer les signaux, adopter et terminer les processus orphelins. Ca ajoute quelques Mb à l'image Docker (mais bon, avec Debian + JDK + Jenkins on est plus vraiment à ça près) et ça règle le soucis.

Quelque soit votre appli Docker je vous encourage donc à adopter ce petit bout de code qui vous évitera des surprises.

La vraie solution ?

Idéalement Docker devrait rendre tout ça transparent pour qu'on ai pas à s'en soucier nous développeurs qui n'avons pas lu le guide du noyau Linux. #11529 débat par exemple d'une implémentation globale au niveau de Docker pour remplacer la simple exécution d'une command externe au lancement du conteneur par un mécanisme plus avancé prenant en charge l'adoption des process orphelins.

Ce point (et de nombreux autres) on été décrits dans un article très populaire et bien documenté que je vous recommande vivement : http://sirupsen.com/production-docker/

Si cet article ne vous aidera pas à convaincre vos Ops de passer à Docker, en tout cas il pose de vraies bonnes questions et vous

About Me

NICOLAS DE LOOF

Senior Support Engineer at CloudBees and Jenkins Community member

AFFICHER MON PROFIL COMPLET

Blog Archive

pousse donc à réfléchir à la façon dont vous devrez penser votre infra ou vos images Docker. Bonne lecture.

0 COMMENTAIRES:

Enregistrer un commentaire

LIENS VERS CET ARTICLE

Créer un lien

Accueil Article plus ancien

Inscription à : Publier les commentaires (Atom)