Docker

Alexandre Ancel

Cemosis - Université de Strasbourg

26 Mars 2015

Plan

Outline

Introduction

- Parcours
  - Licence/Master d'informatique, 2003-2008, Université de Strasbourg
  - Doctorat en informatique, 2008-2011, Université de Strasbourg
    "Eclairage haute performance pour visualisation volumique"
  - Post-Doctorat, 2012-2013, Inria, Grenoble
    "Petaflow: Visualisation scientifique entre Grenoble, France et Osaka, Japon"
  - Mots-clés: visualisation scientifique, calcul haute performance
- Actuellement:
  - Ingénieur de recherche contractuel, 2013-2016, Université de Strasbourg
    - Travail au sein de Cemosis (`http://www.cemosis.fr`)
      Structure de promotion de la modélisation et de la simulation mathématique pour l'Université et les entreprises locales
      (Similaire à la Maison de la Simulation, Saclay ou MaiMoSiNE, Grenoble)
    - En collaboration avec C. Prudhomme et V. Huber

Outline

Travail actuel et utilisation de docker

- Feel++: librairie pour la résolution d'équations à dérivées partielles
  - Open source et activement dévelopée (`https://github.com/feelpp/feelpp`)
  - Suivre les évolutions rapides dans le calcul scientifique (C++-11, Solveurs, Visualisation ...)
    $\Rightarrow$ Requiert l'utilisation de librairies/compilateurs récents, Pas forcément dans les système de packages des Linux HPC (Scientific Linux, Red Hat, CentOS, ...)
- Actuellement, aide à l'administration d'une machine de calcul à l'Irma
  - AMD Opteron, 4 * 16 coeurs
  - 512 Go de RAM
  - 50 To de stockage
  - Debian stable
- Solution actuelle: Système stable pour l'administration
- Utilisation de chroots pour proposer des versions plus récentes
- Utilisation de Travis CI (Intégration continue, Ubuntu 12.04)
- Docker = solution intéressante pour les contraintes de Feel++

Qu'est ce que Docker ?

- Deux composants: Docker Engine et Docker Hub
- Projet open-source pour le déploiement d'applications dans des conteneurs (`http://github.com/docker/docker`)
- Implémenté en langage Go (Langage compilé, Google, 2009)
- Qu'est ce qu'un conteneur ?
  - Système de virtualisation, utilisant l'isolation de resources comme principe
    - *cgroups* : Limite et isole l'utilisation des resources (CPU, IO, ...) par un ensemble de processus;
    - *namespaces* : Cloisonne un ensemble de processus, ils ne peuvent plus voir les processus dans d'autres groupes.

    $\Rightarrow$ Son propre espace de processus et sa propre interface réseau (contrairement aux chroots)
  - Pas de modification du noyau, car partagé avec l'hote
  - Installation de paquets avec root
- Multi-plateforme:
  - Linux: support natif par le kernel (Version >= 3.8+, cf. Issue #407)
  - Windows & Mac OS X: boot2docker (VM légère, 25 Mb)
  - Windows Server: support natif de l'API mi-2015 pour le noyau Windows
  - ARM (e.g. Raspberry pi, téléphones portables) : support non complet

Utilisation ? Utilisateurs ?

- Cas d'utilisation:
  - Application Web / SQL
  - Big Data
  - Séparation de la production et du test
  - Intégration continue
  - ...
- Utilisateurs:
  - Ebay, Spotify, Baidu, Yelp, ...

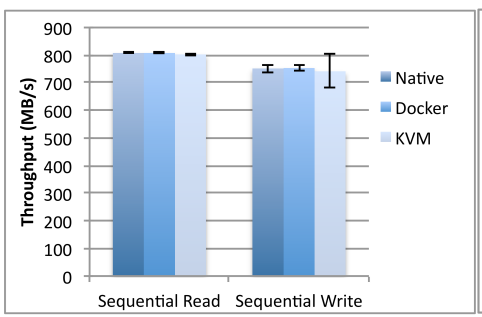
Interet de Docker ?

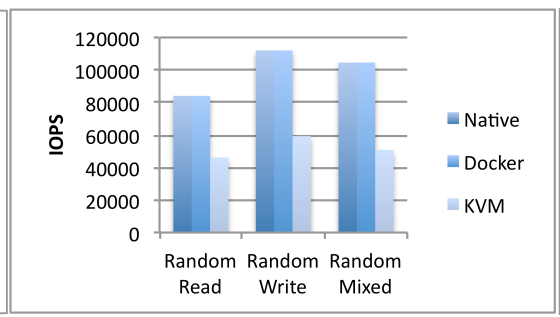
- Une fois une image créée pour le déploiement d'une application, celle-ci peut etre déployée n'importe où Docker est installé !
  - Applications avec dépendances spécifiques (versions de librairies ...)
  - Installation sur un autre système: Compatibilité ?
    Ex: Laptop en Ubuntu, Serveur en CentOS
    Des librairies manquent ... On recompile ?
- Si une application fonctionne localement, elle fonctionne avec un comportement identique sur le serveur (modulo les changements de version de librairies linux dans les images, mais possibilité d'utiliser directement une image créée)
- Séparer la production des tests, cloisonnés dans des conteneurs différents

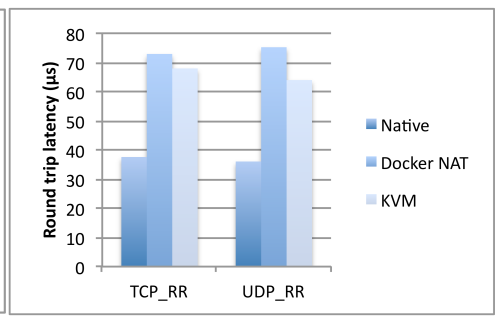
Performance

- Performance:
  - "An Updated Performance Comparison of Virtual Machines and Linux Containers", Felter et al., IBM, 07/14
    - PXZ : Utilitaire de compression parallèle (lossless, LZMA)
    - Linpack : Solver de systèmes denses d'équations linéaires (LU fact.)
    - RandomAccess : Accès et modifications aléaloires en mémoire
    - Stream : Mesure de la bande passante mémoire

Workload		Native	Docker	KVM-untuned	KVM-tuned
PXZ (MB/s)		76.2 [±0.93]	73.5 (-4%) [±0.64]	59.2 (-22%) [±1.88]	62.2 (-18%) [±1.33]
Linpack (GFLOPS)		290.8 [±1.13]	290.9 (-0%) [±0.98]	241.3 (-17%) [±1.18]	284.2 (-2%) [±1.45]
RandomAccess (GUPS)		0.0126 [±0.00029]	0.0124 (-2%) [±0.00044]	0.0125 (-1%) [±0.00032]	Tuned run not warranted
Stream (GB/s)	Add	45.8 [±0.21]	45.6 (-0%) [±0.55]	45.0 (-2%) [±0.19]	
	Copy	41.3 [±0.06]	41.2 (-0%) [±0.08]	40.1 (-3%) [±0.21]	
	Scale	41.2 [±0.08]	41.2 (-0%) [±0.06]	40.0 (-3%) [±0.15]	
	Triad	45.6 [±0.12]	45.6 (-0%) [±0.49]	45.0 (-1%) [±0.20]	







- Docker est équivalent ou meilleur par rapport à KVM

Performance

- Pourquoi peu d'overhead ?
  - Les processus sont isolés, mais sont lancés sur l'hote
    Performance CPU = native
  - L'hote garde la trace de la mémoire données aux conteneurs
    (désactivable pour la performance)
  - Overhead pour la partie réseau
- Par rapport aux VMs classiques ?
  - Pas besoin d'un environnement complet Guest OS
    $\Rightarrow$ Les conteneurs sont légers, une VMs prend typiquement 5 à 10 Go
  - Démarrer VM = démarrer un kernel + des services,
    Avec Docker, juste l'application souhaitée
  - Si nouvelle release, on ne veut pas réuploader 5 à 10 Go
- Cout peu important: Qu'est ce que cela permet ?
  - Tester une application 100 fois: chaque test a besoin d'une DB de base
    $\Rightarrow$ Avec Docker, on démarre 100 tests
  - Les conteneurs rendent le testing plus facile

Docker Hub: Images

- Image = instantané d'un système
- Image = tarball + metadata (information sur les ports "exposés", les services démarrés ...)
- Que se passe-t-il quand on fait des modifications sur une image ?
  - Les images Docker sont en lecture seule
  - Un service prennant en charge l'"union mount" (UnionFS, Aufs ...): Permet de créer un système de fichier cohérent à partir de couches de filesystems séparés en les superposant.
- Sauvegarder une image sous forme d'archive:
  - `docker save imagename > image.tar`
  - `docker load imagename < image.tar`

Demo

- Exemple de switch de système d'exploitation:
  - `docker run -ti ubuntu /bin/bash`
  - `docker run -ti redhat /bin/bash`
  - `docker run -ti fedora /bin/bash`
  - `docker run -ti ubuntu:12.04 /bin/bash`

Docker Hub

- Stockage d'images sur Docker Hub:
  - Images publiques: Accès gratuit
  - Images privés: Stockage d'images privées (payant)
  - Builds automatiques, lien avec github/bitbucket
    (Générer des images automatiquement dès qu'un push sur un repository est fait (trigger))
- Commandes utiles:
  - `docker login`
  - `docker search imagename`
  - `docker pull imagename`
  - `docker push imagename`
  - `docker images`

    (–all pour les images intermédiaires)

Docker Engine

- Conteneur = version active d'une image
- Commandes de base:
  - `docker ps -a`
  - `docker run -ti ubuntu:14.10 /bin/bash`
    - -t -i: Permet d'accéder au conteneur de manière interactive avec un terminal
    - -d: mode détaché (background). Possibilité de s'attacher avec attach.
    - –name <nom>: donner un nom au conteneur
  - `docker logs <container>`

    Récupère la sortie de l'exécution du conteneur
  - `docker top <container>`

    Lister les process en cours dans un conteneur
  - Une fois, un conteneur lancé:

    `docker stop`

    `docker start`

    `docker attach`

    si interactif

Docker Engine

- Commiter une image (enregistrer les modifications faites à un conteneur):
  - `docker diff`
    Inspecte les différences par rapport à la couche de base
  - `docker commit <container> <image[:tag]>`
  - `docker tag image image:tag`
    tagge une image
- Faire le ménage (après docker stop):
  - `docker rmi imagename`
    Supprimer une image
  - `docker rm ‘docker ps -aq‘`
    Supprimer tous les conteneurs lancés

Communiquer avec l'extérieur

- Réseau: exemple: le port 80 pour un serveur web
  - `docker run -p 80 nginx /usr/bin/nginx -g "daemon off;"`
  - `docker run -p 49000:80 nginx /usr/bin/nginx -g "daemon off;"`
  - `docker run -P nginx /usr/bin/nginx -g "daemon off;"`
- Données:
  - `docker run ... -v /var/log/nginx ...`
  - `docker run ... --volumes-from <container> ...`
  - `docker run ... -v /data/code:/opt/code ...`

Dockerfile

- Un fichier texte décrivant la construction d'une image Docker (Makefile)
- Permet l'automatisation de la création des images
- Example: création d'un environnement de développement MPI

```
FROM ubuntu:14.10
RUN apt-get update -y
RUN apt-get install -y openssh-server gcc
RUN apt-get install -y openmpi-bin libopenmpi-dev
CMD [ "/bin/bash" ]
```

- `docker build -t imagename .`
- Autre exemple: test d'un fichier Dockerfile lançant une image centos avec un serveur node.js

Docker engine

- REST API:
  - Docker est un daemon et tout est présenté par l'API REST.
    les commandes "docker *" (e.g. docker run) envoient une requete au daemon avec cet API
  - L'API peut être ouverte sur un socket TCP
    $\Rightarrow$ accès local, mais permet aussi de controler un conteneur à distance
    - Construire des images et les lancer
- Orchestration: Fig et Docker Compose

Outline

Conclusion

- Docker:
  - Un environement complet pour le déploiement d'applications dans des conteneurs:
    - Docker Engine
    - Docker Hub
  - avec Docker, on peut:
    - Mettre un(des) logiciel(s) dans un conteneur
    - Executez ces conteneurs n'importe où avec un comportement maitrisé
    - Ecrire/Générer des recettes d'automatisation de construction des conteneurs
  - Si cela fonctionne localement, cela fonctionne n'importe où !
  - Peu importe les versions de librairies, de distros et des dépendances
  - Peut être vu comme une alternative sur les systèmes sans systèmes de packages (Windows, MacOS X (brew ...))
- Un peu plus compliqué:
  - Gestion/automatisation des utilisateurs
  - Export X11

Fin de la présentation

- Merci de votre attention !
- Sources et liens utiles:
  - Tutorial interactif en 10 min: `https://www.docker.com/tryit/`
  - Documentation de Docker: `http://docs.docker.com/`
  - Presentation de J. Petazzoni: `https://www.youtube.com/watch?v=1z77HDXeIEw`
  - "An Updated Performance Comparison of Virtual Machines and Linux Containers", Felter et al., 07/14 `http://domino.research.ibm.com/library/cyberdig.nsf/papers/0929052195DD819C85257D2300681E7B/$File/rc25482.pdf` Associated content: `https://github.com/thewmf/kvm-docker-comparison`
  - `http://www.slideshare.net/BodenRussell/kvm-and-docker-lxc-benchmarking-with-openstack`