Archipel: une couche au dessus de libvirt

Emmanuel Blindauer
IUT R.Schuman, Université de Strasbourg
72 route du Rhin, ILLKIRCH

Résumé

Archipel est un logiciel de gestion d'hyperviseurs et de machines virtuelles qui, contrairement à d'autres solutions, n'implique pas de remise en question de l'architecture existante. En parallèle, libvirt s'impose dans le monde du logiciel libre comme une librairie d'interfaçage avec les machines hôtes, gommant certaines contraintes propriétaires.

C'est ce support qui a été choisi par le projet Archipel, qui se place résolument dans une optique de réutilisation maximale des existants, afin d'apporter une solution stable. Par exemple, les protocoles de dialogues entre entités utilisent XMPP (EXtensible Messaging and Presence Protocol), le client web utilise le framework Cappuccino, l'implémentation de XMPP la librairie Strophe, et la partie VNC, noVNC.

Archipel permet, via son système de permissions, de donner accès à des utilisateurs authentifiés, uniquement à certaines ressources de gestion : reboot, accès à la console VNC, modification des paramètres réseaux, redéfinitions de paramètres...

Toutes les entités (utilisateurs, machines virtuelles, hyperviseurs) sont identifiées par des comptes au niveau XMPP. C'est un agent sur chaque hyperviseur qui expose sa présence, ainsi que pour chaque machine virtuelle.
La gestion de la liste de contacts au niveau du serveur XMPP, permet d'avoir une sécurité supplémentaire entre les entités. L'utilisation conjointe de la technologie PubSub et des événements de libvirt permet d'avoir une application asynchrone, donc résistant mieux aux montées en charge. Le client de supervision est une application en HTML5, le dialogue avec les composants se fait également via XMPP.

Le déploiement d'Archipel est possible sur une infrastructure libvirt en production, sans interruption de service.

Cet article est le résultat d'une mise en œuvre à des fins pédagogiques d'une plate-forme de virtualisation pour une quarantaine d'étudiants, puis de la mise en place sur une demi- douzaine de serveurs de production.

Mots clefs

XMPP, virtualisation, libvirt, HTML5

1 Introduction

La virtualisation est devenu un point majeur de l'évolution des infrastructures systèmes. La montée en puissance de celle-ci et l'accroissement des services proposés aux utilisateurs a décuplé l'utilisation de ces systèmes. L'isolation entre les applications permet de pouvoir éviter des effets de bord de modification du système, et de déléguer plus facilement la gestion des applications, etc.

La facilité d'utilisation de la virtualisation conduit rapidement à l'explosion du nombre de machines virtuelles à gérer. Il faut alors se lancer dans une rationalisation et industrialisation de la gestion des machines.

Sur les plate-formes Linux, la popularisation de la virtualisation s'est faite avec KVM. D'autres solutions de virtualisation, para-virtualisation et associées telles que Xen, OpenVZ ont également profité d'une nouvelle visibilité avec le développement de la virtualisation.

Les solutions de gestion d'hyperviseurs sont devenues nécessaires. Elles permettent de gérer l'ensemble des serveurs hôtes, les serveurs virtuels qui y sont créés, d'effectuer des migrations entre hyperviseurs. La gestion des architectures réseaux et des architectures de stockages y est généralement aussi intégrée, ainsi que la pré-provision des serveurs virtuels qui permet de disposer de serveurs prêt à l'emploi dans des délais très courts.

Ces solutions ne sont généralement pas flexibles sur les aspects du stockage ou de l'architecture réseau, et pour les services informatiques ayant déjà une certaine infrastructure en place, la mise en œuvre de ces solutions peut vite devenir une charge complexe et lourde

2 Libvirt

Suite à la venue de *qemu* et *kvm* dans le logiciel libre, une API ouverte a vu le jour : *libvirt*, associée à des outils tels que : *virsh*, *virt-manager*. Le premier est un outil en ligne de commande permettant d'effectuer un très grand nombre d'opérations sur l'hyperviseur, le second étant doté d'une interface graphique permettant de faire des opérations d'une façon conviviale.

Des contributions ont permis de prendre en charge d'autres hyperviseurs que kvm : actuellement, un peu moins d'une dizaine d'hyperviseurs sont (plus ou moins) supportés, comme par exemple VMware ESX, Xen, OpenVZ ou Microsoft Hyper-V. Cela permet d’élargir le champ d'application de *libvirt* et d'exposer une API standard malgré des hyperviseurs différents.

Libvirt propose essentiellement une vue limitée à un hyperviseur à la fois (bien que l'on puisse faire des accès à distance), sans gestion de droits fine. C'est un outil pratique, dans un environnement restreint, et où les administrateurs se font totalement confiance.

3 Archipel

3.1 Introduction

Les outils libres qui permettent de gérer un parc d'hyperviseurs, de leurs machines virtuelles associés, sont rares. Certains sont restés à l'état de version préliminaire et sont morts à ce stade (*oVirt*). D'autres, comme *Proxmox, conVirt, OpenNebula* ont des fonctionnalités très intéressantes, mais imposent leur propre architecture de réseau ou de stockage, pas forcément adaptée à un existant ou n'utilisent pas *libvirt*. D'autres ont une approche très (trop?) orienté Cloud (*CloudStack*).

Archipel repose totalement sur *libvirt* comme API de gestion des machines virtuelles. De fait, l'outil s'impose d'être de plus haut niveau. Pour la gestion des utilisateurs, on peut directement utiliser un serveur XMPP existant pour authentifier les personnes.

3.2 Architecture

L'utilisation d'un serveur XMPP peut sembler étrange. L'application devra envoyer et recevoir des informations avec les hyperviseurs et les machines virtuelles. Plutôt que de réécrire un n-ième protocole, de l'implémenter, de devoir le débugger et le sécuriser, l'équipe d'Archipel avec XMPP a fait le choix de la réutilisation. Ce protocole est défini par des RFC, et possède de nombreuses implémentations.

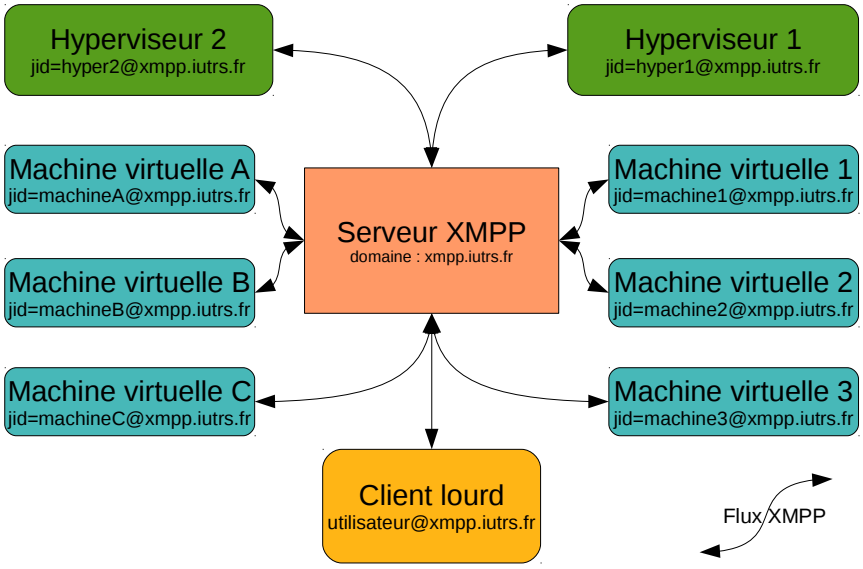

Illustration 1: Flux de messages entre les différentes entités

Il suffit donc d'avoir un client adapté pour les hyperviseurs et pour les machines virtuelles. Les machines virtuelles, à un instant donné étant localisées sur un serveur, c'est au niveau de l'hyperviseur que reposera la création des *n* instances de clients pour chacune des *n* machines virtuelles. Au niveau de l'application de l'utilisateur, il y a également un client XMPP pour que les messages soient transmis aux entités.

En terme logiciel, un agent est installé sur chaque hyperviseur. Cet agent est scindé en deux parties :

- une incluant un client XMPP chargé de se connecter au serveur XMPP pour lui signaler sa présence.
- l'autre, plus complexe, est chargée de créer autant d'agents qu'il y a de machines virtuelles définies, chacun de ces agents intègre également un client XMPP.

Du coté du client utilisateur, l'interface intègre un client XMPP. Toutes les communications bénéficient donc des fonctionnalités du serveur XMPP, par exemple le *roster* (liste des contacts autorisés) qui donne une certaine sécurité dans les dialogues entre entités. On « ajoute » donc un contact dans son *roster*, qui peut être soit une machine virtuelle ou un hyperviseur. Lorsque dans l'interface client, on demande un démarrage de la VM (« Virtual Machine », en français machine virtuelle), un *stanza* (un message XMPP) est envoyé vers le serveur XMPP. Celui-ci vérifie que l'émetteur est dans le *roster* du destinataire, et lui transmettra alors le message s'il est autorisé. L'agent, coté machine virtuelle, va recevoir ce message et l'interpréter, donc lancer via l'API *libvirt* le signal *start* .

Les agents, qu'il s'agisse de ceux liés à un hyperviseur ou à une machine virtuelle, exploitent la librairie *libvirt* via le *binding* python. En particulier, c'est la partie *events* de *libvirt* qui est utilisée. De cette manière, même si une interaction parallèle est effectuée sur les hyperviseurs (par exemple, l'utilisation de *virsh* ou de *virt-manager* pour démarrer une entité, ajouter un nouveau réseau, ou créer une nouvelle VM par un script), les agents seront informés et pourront remonter le nouvel état via un message XMPP.
Ce choix laisse la porte ouverte à une intégration fine dans un système existant : si vous aviez vos

propres scripts *virsh* pour créer des machines virtuelles, l'adoption d'Archipel ne vous force nullement à renoncer à votre travail déjà en place !

3.3 Haute disponibilité et montée en charge

Dans les solutions de gestions d'hyperviseurs, un critère souvent retenu est la Haute Disponibilité (HA). Mais tout le monde ne met pas les mêmes éléments derrière ces mots. En terme de HA au niveau des machines virtuelles, Archipel se repose complètement et uniquement sur *libvirt*. Au niveau d'Archipel lui même, la haute disponibilité concerne l'échange des messages entre les entités, donc la haute disponibilité du serveur XMPP lui même. Une utilisation d'une solution cluster de serveurs XMPP répond donc à la demande. Ce mode cluster permet aussi de se rassurer au niveau de la montée en charge du serveur car la multiplication des VM et des hyperviseurs va augmenter le nombre de *stanza* échangés.

Au niveau des messages échangés, la notification de toutes les entités n'est pas nécessaire, ni voulue, pour des raisons de charge de messages. Une solution consisterait à stocker du coté de l'agent, les entités et les messages qu'elles doivent recevoir, mais il faudrait alors maintenir des états. L'architecture a recours à un mécanisme très intéressant du coté du serveur XMPP. Il s'agit de la partie *PubSub* (*PUBlish-SUBscribe*) : cela fonctionne comme les groupes de multicast au niveau réseau IP. Les entités qui souhaitent diffuser des informations à plusieurs entités publient via une entrée dans le service *PubSub* du serveur, et les entités qui souhaitent recevoir les messages pour cette entrée s'y abonnent. Ainsi, c'est encore une fois le serveur XMPP qui va faire le travail de diffuser aux entités abonnées les messages. Cela donne un flux optimal de messages, et simplifie la programmation des agents.

3.4 Sécurité

Dans une telle plate-forme, la gestion de la sécurité est importante. L'utilisation de XMPP en tant que protocole permet déjà d'être rassuré sur l'intégrité des messages échangés : XMPP est issue d'un processus de validation et de standardisation via les RFC et est implémenté depuis de nombreuses années.

La sécurité qui permet de savoir quel utilisateur peut dialoguer avec une entité repose d'abord sur le *roster* de celle-ci, car comme le roster est stocké coté serveur XMPP, c'est sans doute la meilleure solution qui puisse être retenue. Il est donc possible de définir clairement quel utilisateur peut dialoguer avec une machine virtuelle ou un hyperviseur donnés, et également de savoir qui a accès à une entité donnée.

Archipel va encore plus loin dans la gestion de la sécurité. Comme ce sont des *stanza* qui définissent les demandes entre les entités, et que cela se traduit par des appels à l'API de *libvirt*, un filtrage coté agent est en place, il permet d'autoriser ou non certaines actions. Ainsi, actuellement, 110 rôles possibles sont définis et il est possible d'attribuer à un utilisateur, l'accès ou non à chacun de ces rôles. Par exemple, on peut donner à un webmestre, le droit d'accéder à la « console VNC » de sa machine virtuelle, ainsi que l'action *start* et *stop*. Ou à un développeur, le droit de prendre des instantanés d'une machine virtuelle, et de les restaurer.

Dans un soucis de séparer les utilisateurs des entités machines virtuelles et hyperviseurs, on peut aller encore plus loin (et s'intégrer encore mieux dans le système d'information). Les serveurs XMPP sachant dialoguer ensemble, via une autorisation explicite au niveau du serveur pour autoriser le *S2S (Server to Server)*, il est possible d'utiliser un autre serveur XMPP existant pour les utilisateurs (comme cela, ils peuvent utiliser par exemple une authentification liée à leur ENT). Les messages transitent alors par les deux serveurs successivement avant d'arriver aux entités coté Archipel.

En terme de sécurité, là encore, Archipel se base sur l'existant, et ajoute de la finesse dans la délégations de droits.

3.5 Fonctionnalités

Archipel étant encore jeune, les fonctionnalités les plus évidentes ont été mises en place. Le système de module permet d'ajouter aisément de nouvelles fonctions. Tout ce qui est nécessaire à un travail quotidien existe.

Actuellement, la plupart des fonctions disponibles via l'interface *virt-manager* sont déjà en place : définition d'une nouvelle VM, manipulations du réseau et du stockage, accès à la console VNC, gestions des snapshots, etc... Les opérations de migration sont également prises en charge. Les nouvelles possibilités sont celles qu'on attend d'un logiciel de management d'hyperviseurs : reporting sur l'état de hyperviseur, création de nouvelles machines à partir de flux RSS (*VMCast*), planifications de taches, gestions des droits des différents utilisateurs, création d'une machine avec détection automatique du serveur le moins chargé, etc... Les développeurs sont pleins d'idées, il même est prévu, par exemple, un module de facturation !

La mise en place d'Archipel est très simple : il faut installer un serveur XMPP avec une configuration correcte, puis installer sur chaque agent, un agent écrit en python. Cet agent, une fois correctement configuré, va se cloner pour s'exécuter pour toutes les machines virtuelles, et va joindre le serveur XMPP.
A partir de ce moment là, Archipel est exploitable. Les machines virtuelles déjà en place bénéficient des fonctionnalités d'Archipel (sauf au niveau de la gestion disque et migration). On ne peut pas faire beaucoup plus simple.

Et pour couronner le tout, l'application client n'est pas un client lourd standard. Tout le client est en fait une application *HTML5*, en *Javascript*. C'est ce qui déconcerte généralement les premiers utilisateurs : il n'y a pas de serveur web où l'application s'exécute. Il faut juste un serveur web pour stocker les fichiers (dans certains cas et avec le navigateur chrome uniquement, il est même possible d'exécuter l'application depuis les fichiers stockés en local). Une fois chargé, c'est uniquement le Javascript qui fait tourner tout le client. Le framework utilisé est Cappuccino, qui donne un aspect très MacOsX. Le client XMPP y est également intégré. Donc quand vous utilisez votre client, vous dialoguez directement vers le serveur jabber !

4 Conclusion

Archipel s'intègre bien dans le courant UNIX : Keep It Simple, Stupid. D'autres solutions préfèrent construire des monstres d'infrastructure. Se baser sur *libvirt* permet de s'assurer d'avoir la main via *virsh* ou *virt-manager*, en cas de soucis de client, d'agent, ou même si le serveur XMPP pose problème, cela permet de dormir tranquille.

Cependant Archipel a encore quelques faiblesses. A force d'exploiter la robustesse de XMPP, peu de serveurs implémentent tout ce qui est nécessaire. Ejabberd est pour l'instant le seul serveur XMPP qui est recommandé. Archipel a également mis en évidence un certain nombre de bugs dans *libvirt*. En particulier la gestion de Xen et de Vmware dans *libvirt* reste problématique, et rend inopérant Archipel (la balle est dans le camp des développeurs de *libvirt*). Enfin Archipel propose déjà un certain nombre de fonctionnalités qui le rend largement utilisable, et il reste de la place pour l'innovation.

La mise en place n'affecte pas l'existant, elle est relativement aisée à effectuer, ce qui permet de faire une migration en douceur.

Enfin, Archipel répond a un besoin récurrent mais simple : pouvoir donner un accès restreint à certaines machines pour certains utilisateurs. La délégation de droits est simple à effectuer. De plus, avec une interface conviviale et sans client lourd, en HTML5, il n'y a pas de problème pour donner un accès à des non informaticiens.