Gestion de ressources multi-nuages dirigé par des modèles

Fawaz Paraiso, Yahya Al-Dhuraibi, Stéphanie Challita, Philippe Merle

Compas'2016 : Parallélisme / Architecture / Système
Lorient, France, du 5 au 8 juillet 2016

Gestion de ressources multi-nuages dirigé par des modèles

Fawaz Paraiso Yahya Al-dhuraibi Stéphanie Challita Philippe Merle

Université de Lille & Inria Lille - Europe du Nord,
CRISTAL UMR CNRS 9189
prénom.nom@inria.fr

Résumé
L'informatique multi-nuages s'est imposée comme un paradigme de choix pour acquérir des ressources provenant de différents fournisseurs de nuage et obtenir le meilleur de ces derniers pour exécuter des applications. L'informatique multi-nuages consiste en l'utilisation de multiples environnements de nuages indépendants qui ne nécessitent pas d'accord préalable entre les fournisseurs de nuage ou un tiers. Cependant la gestion des ressources dans un environnements multi-nuages reste une tâche difficile. Plusieurs bibliothèques multi-nuages comprenant Apache Libcloud, Apache jclouds, δ-cloud, Daseincloud, fog et pkgcloud existent sur le marché de nuage. Mais la gestion des ressources multi-nuages par ces bibliothèques se fait toujours en programmant dans un langage spécifique et reste de bas niveau. Par conséquent, nous avons besoin d'une solution permettant aux développeurs de gérer de manière simple les ressources multi-nuages. Dans cet article nous fournissons une approche dirigée par les modèles pour gérer à un niveau d'abstraction plus haut les ressources provenant de plusieurs nuages. Nous illustrons notre proposition en fournissant un prototype du Designer multi-nuages pour gérer des ressources provenant de plusieurs nuages.

Mots-clés : Informatique multi-muages, Ressource, Gestion de ressources, Modèle, Docker.

1. Introduction

De nos jours, l'informatique multi-nuages [7] ou multi-Cloud Computing [8] est en train de devenir une réalité. D'une part, il y a pléthore de fournisseurs de nuage publics comme Amazon EC2, Google Compute Engine, Microsoft Azure pour ne nommer que quelques-uns et des piles logicielles de nuages comme VMware ESX, OpenStack, CloudStack, OpenNebula, Eucalyptus permettant de construire un nuage privé ou public. Chaque plateforme de nuage fournit sa propre API de gestion des ressources de nuage ou Cloud Resource Management API (CRM-API) permettant aux administrateurs d'ajouter et de retirer, de dimensionner et de redimensionner des ressources de leur nuage. Toutefois, ces CRM-API sont de très bas niveau, souvent présentés sous forme d'une API REST seulement utilisable par des experts. Leurs hétérogénéités limitent l'interopérabilité entre les nuages. En outre, une plateforme de nuage est souvent équipée d'une interface graphique intégrée au-dessus du CRM-API. Cette interface graphique est seulement appropriée aux tâches de gestion manuelle. D'autre part, l'utilisation de plusieurs nuages simultanés permet d'optimiser les coûts de fonctionnement pour faire face aux pics de demande de service, réagir face aux échecs, etc. Pour faire face à l'hétérogénéité du

CRM-API, plusieurs bibliothèques multi-nuages existent comme Apache Libcloud [1], Apache jclouds [2], δ-cloud [3], Daseincloud [4], fog [5] et pkgcloud [6] sur le marché de nuage. Mais la gestion des ressources multi-nuages par ces bibliothèques se fait toujours en programmant dans un langage et reste de bas niveau. Ainsi, la gestion de ressources multi-nuages à un niveau d'abstraction plus élevé est requis.
Dans cet article, nous proposons une approche dirigée par les modèles pour gérer des ressources dans des environnements multi-nuages. Notre approche vise à aider les utilisateurs à gérer les ressources de plusieurs nuages.
Le reste de l'article est organisé comme suit. Dans la Section 2, nous décrivons notre approche. Ensuite, dans la Section 3 nous présentons le résultat d'expériences de gestion d'une application dans un environnement multi-nuages. Puis, dans la Section 4, nous discutons des travaux connexes. Enfin, la Section 5 conclut l'article avec de perspectives futures.

2. Gestion de ressource multi-nuages

Dans cette section, nous présentons notre approche. Nous commençons par donner un aperçu de l'architecture de la solution. Ensuite, nous présentons le Designer multi-nuages. Enfin, nous décrivons comment les ressources multi-nuages sont modélisées.

2.1. Vue d'ensemble de l'architecture

Comme cela est décrit dans la Figure 1, l'architecture présentée est divisée en cinq couches : *Designer multi-nuages*, *Modélisation multi-nuages*, *Modèle à l'exécution*, *multi-nuages* et *Virtualisation & Conteneurisation*. Une brève description de ces couches (de haut en bas) est fournie ci-dessous :

- **Designer multi-nuages** : cette couche fournit un outil graphique pour la conception, le développement et la gestion des ressources provenant de plusieurs nuages. Cet outil interagit avec la couche **Modélisation multi-nuages**.
- **Modélisation multi-nuages** : c'est le modèle défini qui permet aux utilisateurs de représenter différents types de ressources aux niveaux des hyperviseurs, nuages publics et Conteneurs.
- **Modèle à l'exécution** : il est basé sur la couche de **Modélisation multi-nuages**, le modèle à l'exécution fournit la capacité d'exécution de ces modèles en ligne.
- **Multi-nuages** : cette couche représente les environnements de nuage cibles où les ressources sont approvisionnées et gérées.
- **Virtualisation et Conteneurisation** : cette couche repose sur les solutions de virtualisation à base d'hyperviseur ou de conteneur qui sont utilisés par des fournisseurs d'infrastructure de nuage. Comme vous l'avez constaté dans la Figure 1, les deux solutions de virtualisation (machine et conteneur) peuvent être utilisées de manière combinées ou disjointes selon les besoins.

2.2. Designer multi-nuages

L'outil *Designer multi-nuages* dans la Figure 1 offre une approche structurée et une interface graphique intuitive aux utilisateurs pour la conception, le déploiement et la gestion des ressources

1. https://libcloud.apache.org/
2. https://jclouds.apache.org/
3. https://deltacloud.apache.org/
4. http://dasein-cloud.sourceforge.net/
5. http://fog.io/
6. https://github.com/pkgcloud/pkgcloud

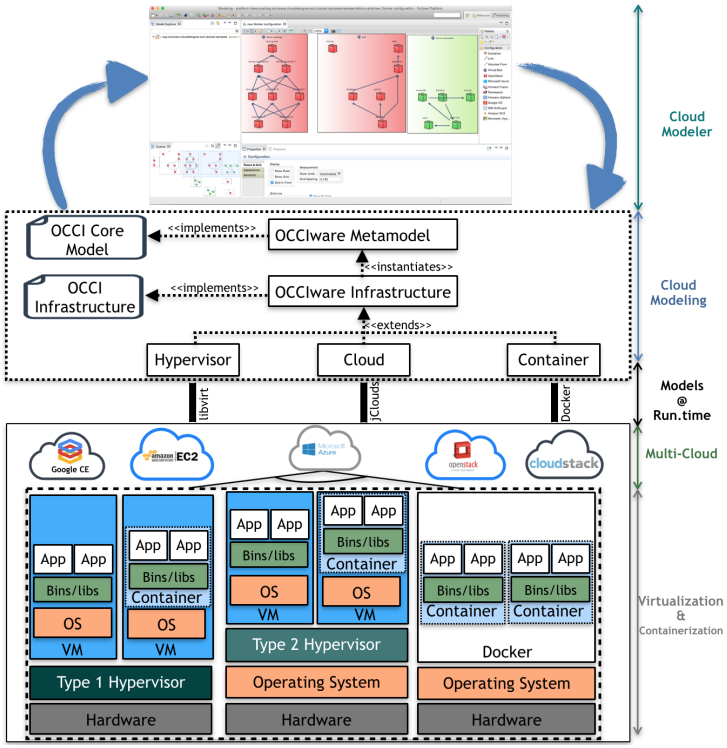

FIGURE 1 – Architecture pour gérer les ressources multi-nuages.

à travers plusieurs nuages. Cet outil gère les ressources multi-nuages à un niveau d'abstraction élevé. Il fournit un moyen explicite permettant de gérer les ressources. En utilisant notre Designer multi-nuages, les utilisateurs peuvent concevoir et gérer les ressources provenant de plusieurs nuages. Ces ressources peuvent être fournies par les nuages publics, des hyperviseurs et les conteneurs. Le *Designer multi-nuages* représente tous les concepts de ressources multi-nuages définis dans le modèle sous-jacent. Ainsi, l'utilisateur peut concevoir une machine virtuelle et un conteneur d'application en interagissant avec le modèle cible.

2.3. Modélisation multi-nuages

Notre métamodèle pour Open Cloud Computing Interface (OCCI) nommé OCCIware métamodèle [9] est basé sur l'intergiciel Eclipse Modeling (EMF) [10]. Dans le projet OCCIware[7], nous avons fait le choix d'utiliser des modèles comme couche d'abstraction de haut niveau pour la gestion des ressources multi-nuages.

Comme le montre la Figure 1, notre approche dirigée par le modèle est basée sur un métamodèle précis [6] d'OCCI. OCCI est la spécification d'Open Grid Forum (OGF) qui définit une interface ouverte pour la gestion de tout type de ressources en nuage au niveau (IaaS, PaaS et SaaS). Ce métamodèle est encodé avec EMF. Les ressources d'infrastructure de nuage, connu comme le cpu, la mémoire, le réseau et le stockage sont représentées par notre modèle d'**Infrastructure**. Le modèle d'**Infrastructure** est étendu pour gérer des aspects spécifiques aux hyperviseurs (modèle **Hyperviseur**), aux plateformes de nuage (modèle **Cloud**) et aux conteneurs (modèle **Docker**). Ensemble, ces trois modèles (**Hyperviseur**, **Cloud** et **Docker**) ont permis de modéliser les ressources de multiples nuages simultanément et de façon transparente.

7. http://www.occiware.org/

2.4. Modèle à l'exécution

L'exécution des modèles repose sur ce que nous appelons les connecteurs. Les connecteurs assurent la connexion entre les modèles et le réel (système en cours d'exécution). Fondamentalement, les connecteurs prennent en entrée un modèle source et projettent ce dernier dans le système en cours d'exécution. De manière réciproque, les connecteurs surveillent, introspectent les systèmes en cours d'exécution afin de transformer le réel en modèle. Comme on peut le voir sur la Figure 1, nous avons trois connecteurs : *Hypervisor*, *Cloud* et *Docker*. Le connecteur *Cloud* est spécifique au modèle Cloud, il effectue les interactions entre le modèle et les nuages publics. Le connecteur *Hypervisor* est spécifique au modèle hypervisor quant au connecteur *Docker*, il est spécifique au modèle Docker. Evidemment, les connecteurs *Hypervisor* et *Docker* gèrent les interactions entre les modèles *Hypervisor* et *Docker* avec les hyperviseurs et les conteneurs respectivement.

Les connecteurs sont basés sur les principes suivants :

— **Transformation** : les connecteurs fournissent des techniques de transformation expressives de modèles basés sur des patrons de conception, ce qui facilitent la spécification des traductions entre les modèles et les systèmes en cours d'exécution.

— **Introspection** : pour introspecter le système en cours d'exécution, les connecteurs emploient les techniques de l'ingénierie dirigées par les modèles ou Model-Driven Engineering (MDE), qui gèrent l'introspection et l'analyse du système à un haut niveau d'abstraction du modèle. En utilisant des techniques MDE, différents modèles décrivant certaines contraintes sont dérivées et maintenues à l'exécution.

— **Synchronisation** : les connecteurs assurent la synchronisation incrémentale entre un système en cours d'exécution et les modèles. Afin de détecter des modifications dans le modèle, les connecteurs utilisent un mécanisme de notification qui signale un événement lorsque l'élément du modèle source est modifié. Pour synchroniser les changements du modèle avec le système en cours d'exécution, les connecteurs vérifient si les éléments du modèle restent cohérents en naviguant de manière efficace entre le modèle source et sa correspondance dans le système en cours d'exécution. Lorsqu'il n'y a pas de cohérence, les ajustements appropriés sont effectués par les connecteurs.

3. Expérimentation

3.1. Implémentation

Pour nos expériences, nous avons implémenté [8] un prototype des Designers *Hypervisor*, *Cloud* et *Docker* comme un système de plugin Eclipse respectivement pour les modèles : Hypervisor, Cloud et Docker. Notre Designer multi-nuages peut être téléchargé à cette adresse [9]. Dans ce travail, nous illustrons uniquement le Designer Docker. Une capture d'écran de notre Designer Docker est illustrée sur la Figure 2. Le cadre (a) de la Figure 2 montre l'explorateur de modèle Eclipse utilisé pour naviguer à travers des projets Docker, contenant un modèle Docker. Le cadre (b) de la Figure 2 montre une perspective ou une vision globale des conteneurs modélisés. Evidemment, cette perspective peut être ajustée pour fournir une vue plus optimale de l'environnement de conception. Le cadre (c) montre la zone de conception qui fournit des outils permettant de créer une représentation graphique du modèle Docker. Comme le montre le cadre (c), les éléments de modèle sont coloriés en vert ou en rouge. La couleur verte de la machine ou des conteneurs montre l'état **started** ou **démarré** des conteneurs et de leur machine hôte. La couleur rouge indique l'état **stopped** ou **arrêté** des conteneurs et de leur machine hôte.

8. https://github.com/occiware/ecore
9. http://www.obeo.fr/download/occiware/

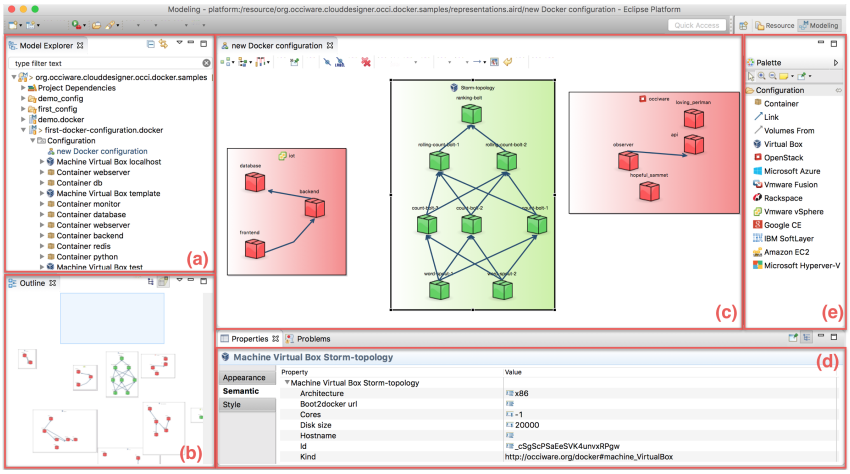

FIGURE 2 – Designer multi-nuages Docker.

Le cadre (d) de la Figure 2 montre les éditeurs de propriétés Eclipse qui permettent de visualiser et de modifier des attributs d'un élément modélisé. Les connecteurs *Hypervisor*, *Cloud* et *Docker* sont mis en œuvre en utilisant Libvirt [3], jclouds Apache [1] et Docker [5] respectivement.

3.2. Evaluation de performances

Pour évaluer notre modèle Docker dans un environnement de nuage, nous avons exécuté une application distribuée composée de huit conteneurs. Cette application distribuée effectue le traitement des événements complexes ou Complex Event Processing permettant d'analyser de grands volumes de données en continu. Pour évaluer la performance réelle de notre modèle Docker, toutes nos expériences ont été effectuées en utilisant un ordinateur de type Macbook Pro muni de processeur Intel Core à i7 à 2,2 GHz, d'une mémoire de 16 Go à 1600 MHz, d'un système d'exploitation OS X version 10.11.2 et le logiciel Oracle Java 1.7 pour exécuter notre Designer Docker. Les déploiements des conteneurs sont effectués chez Scalair [10] qui est un fournisseur de nuage.

Pour déterminer la surcharge introduite par notre modèle Docker, nous avons évalué deux scénari où notre application conteneurisée subit des actions de *création*, *démarrage* et *arrêt*. Nous avons encapsulé notre application de deux manières : i) nativement en utilisant Docker et ii) en utilisant Docker en plus de notre modèle. Les scénari ont été exécutés cent fois sur chacune des deux implémentations.

Dans les tableaux 1, 2 et 3, nous présentons les résultats de la durée moyenne pour la création, le démarrage et l'arrêt des conteneurs pour chaque mise en œuvre, c'est à dire i) et ii), ainsi que la moyenne de coût additionnel introduit par le modèle Docker.

Le coût supplémentaire introduit par notre modèle lors de la création des conteneurs est de **1.11%**. Cette phase de création consiste à télécharger une seule fois l'image des conteneurs et créer ces derniers. Ensuite, lors du démarrage des conteneurs, le coût supplémentaire introduit par notre modèle est de **2.12%**. Le coût supplémentaire introduit par notre modèle lors de l'arrêt des conteneurs est de **2.25%**. La petite fluctuation de temps constatée entre l'action de création et les autres actions (démarrage et arrêt) des conteneurs est dûe à la manipulation des éléments du modèle lors du démarrage et de l'arrêt des conteneurs.

10. `http://www.scalair.fr`

Cette expérience montre qu'il y a une surcharge introduite en ajoutant le modèle Docker. Le coût est négligeable comparé aux avantages offerts par notre approche.

TABLE 1 – Temps et coût supplémentaire introduit lors de la création du conteneur.

Action de création	Moy. temps de création	Coût du modèle Docker
Docker	168.509 sec	-
Docker avec Modèle	170.382 sec	1.11%

TABLE 2 – Temps et coût supplémentaire introduit lors du démarrage du conteneur.

Action de démarrage	Moy. temps de démarrage	Coût du modèle Docker
Docker	5.033 sec	-
Docker avec Modèle	5.04 sec	2.12%

TABLE 3 – Temps et coût supplémentaire introduit lors de l'arrêt du conteneur.

Action d'arrêt	Moy. temps d'arrêt	Coût du modèle Docker
Docker	84.12 sec	-
Docker avec Modèle	86.01 sec	2.25%

4. Etat de l'art

Les bibliothèques multi-nuages comme Apache Libcloud, Apache jclouds, δ-cloud, Dasein-cloud, fog et pkgcloud sont utilisées dans un langage de programmation, ce qui rend la refonte de la gestion des ressources difficile et sujette aux erreurs.
Les auteurs de l'article [4] proposent une plateforme de gestion multi-nuages qui fait la corrélation entre les utilisateurs et les fournisseurs de nuage en proposant des services de nuage unifiés qui utilisent le SOA. Cependant, ils atteignent la gestion unifiée de plusieurs nuages au niveau du code qui ne sont pas flexibles pour répondre aux besoins personnalisés.
Les auteurs de l'article [2] proposent un intergiciel basé sur un modèle appelé CLOUDMF pour gérer plusieurs nuages. Leurs travaux sont à un haut niveau d'abstraction mais ils manquent de support de connexion réseau InterCloud.

5. Conclusion

Nous avons proposé une nouvelle approche dirigée par les modèles pour la gestion des ressources de plusieurs nuages. A travers la construction et la transformation du modèle, les ressources provenant de plusieurs nuages peuvent être gérées de manière unifiées et personnalisées. Notre approche englobe un métamodèle précis OCCI et des modèles spécifiques pour la gestion des hyperviseurs, des plateformes de nuages et les conteneurs. Dans notre approche, nous garantissons également la synchronisation entre le modèle personnalisé et les environnements multi-nuages cibles. De plus, l'environnement des modèles à l'exécution facilite le raisonnement sur l'adaptation dynamique des systèmes en cours d'exécution [2] en fournissant une représentation abstraite du système causalement relié au système en cours d'exécution.
En ce qui concerne les travaux futurs, nous prévoyons d'utiliser les systèmes autonomes comme support pour la gestion des ressources multi-nuages en réutilisant le modèle à l'exécution.

Bibliographie

1. Apache jclouds. – The Java Multi-Cloud Toolkit. – Website `https://jclouds.apache.org/`, February 2016.
2. Ferry (N.), Chauvel (F.), Rossini (A.), Morin (B.) et Solberg (A.). – Managing multi-cloud systems with cloudmf. – In *Proceedings of the Second Nordic Symposium on Cloud Computing &#38 ; Internet Technologies*, *NordiCloud '13*, NordiCloud '13, pp. 38–45, New York, NY, USA, 2013. ACM.
3. Hat (R.). – libvirt : The virtualization api. _____, *http ://libvirt.org*, 2012.
4. Liu (T.), Katsuno (Y.), Sun (K.), Li (Y.), Kushida (T.), Chen (Y.) et Itakura (M.). – Multi cloud management for unified cloud services across cloud sites. – In *Cloud Computing and Intelligence Systems (CCIS), 2011 IEEE International Conference on*, pp. 164–169, Sept 2011.
5. Merkel (D.). – Docker : Lightweight linux containers for consistent development and deployment. *Linux J.*, vol. 2014, n239, mars 2014.
6. Merle (P.), Barais (O.), Parpaillon (J.), Plouzeau (N.) et Tata (S.). – A Precise Metamodel for Open Cloud Computing Interface. – In *Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on*, pp. 852–859. IEEE, 2015.
7. Paraiso (F.). – *socloud : distributed Multi-Cloud Platform for deploying, executing and managing distributed applications*. – Theses, Université des Sciences et Technologie de Lille - Lille I, juin 2014.
8. Paraiso (F.), Haderer (N.), Merle (P.), Rouvoy (R.) et Seinturier (L.). – A Federated Multi-Cloud PaaS Infrastructure. – In *5th IEEE International Conference on Cloud Computing*, pp. 392 – 399, hawaii, United States, juin 2012.
9. Parpaillon, Jean and Merle, Philippe and Barais, Olivier and Dutoo, Marc and Paraiso, Fawaz. – OCCIware-A Formal and Tooled Framework for Managing Everything as a Service.
10. Steinberg (D.), Budinsky (F.), Merks (E.) et Paternostro (M.). – *EMF : eclipse modeling framework*. – Pearson Education, 2008.