Vers une plate-forme communautaire d'hébergement à base de micro-services collaboratifs

Bruno Stévant, Jean-Louis Pazat, Alberto Blanc

Compas'2017 : Parallélisme/ Architecture / Système
Sophia Antipolis, France, du 27 au 30 juin 2017

Vers une plate-forme communautaire d'hébergement à base de micro-services collaboratifs

Bruno Stévant, Jean-Louis Pazat, Alberto Blanc

IRISA, UMR CNRS 6074,
263 Avenue Général Leclerc, 35000 Rennes
{prénom}.{nom}@irisa.fr

Résumé
Cet article propose une solution pour héberger des services à destination d'une communauté sur des équipements partagés par ses membres. Ces équipements possèdent des ressources de calcul, de stockage et de communication hétérogènes ainsi qu'une disponibilité non garantie. Afin de s'adapter à ces contraintes, nous proposons une approche qui s'appuie sur les micro-services virtualisés dans des conteneurs logiciels. Cette solution a été mise en œuvre sur des équipements réels distribués chez des utilisateurs afin de valider la pertinence de l'approche.

Mots-clés : Micro-service, Conteneur logiciel, Plateforme distribuée, Disponibilité, QoS

1. Introduction

Les plates-formes de réseaux sociaux sont aujourd'hui utilisés par un grand nombre d'internautes. Ils permettent aux membres de communautés de tailles diverses d'interagir entre eux. Si l'un des principaux avantages de ces plates-formes est leur capacité à passer à l'échelle à l'aide de grands *datacenters*, le facteur essentiel pour leur utilisation généralisée est le fait qu'elles soient gratuites pour les utilisateurs, à condition qu'ils soient prêts à y créer un compte utilisateur. Or ce faisant, les utilisateurs renoncent au contrôle de leurs données, permettant ainsi à ces plate-formes de se financer. Cela soulève aujourd'hui certaintes préoccupations concernant la préservation de la vie privée des utilisateurs et la dépendance de la communauté vis à vis d'une plate-forme spécifique.
Dans le même temps, les équipements en possession des utilisateurs (ordinateurs personnels, Network attached Storage (NAS), Raspberry Pi, ...) ont acquis des capacités de calcul et de stockage non négligeables et sont de plus maintenant connectés à des réseaux haut-débit (4G, Fibre). La pratique de l'auto-hébergement de services personnels comme le stockage de données sur un disque de type NAS ou la messagerie est de plus en plus encouragée afin de préserver la confidentialité de ses données. Des projets comme CozyCloud [1] ou OwnMailbox [2] visent à rendre cette pratique accessible à tous.
Par extension, il peut être envisagé d'utiliser ces équipements domestiques pour héberger des services à destination d'une communauté et offrir ainsi une solution pour s'affranchir des plate-formes des réseaux sociaux. Chaque membre de la communauté peut dans cet objectif partager sur la base du volontariat une partie des capacités de ses équipements domestiques. En

1. https://cozy.io/
2. https://www.own-mailbox.com/

agregeant ces contributions, il peut être possible de construire une plate-forme d'hébergement capable d'accueillir les services de la communauté. Cette communauté s'élargissant, le besoin en terme de services et de ressources d'hébergement va augmenter, mais dans le même temps, de nouveaux membres pourront contribuer à ajouter des ressources à cette plate-forme.

2. Problématiques liées aux ressources domestiques

Les ressources domestiques présentent un certain nombre de limitations qui rendent difficiles la réalisation d'un service d'hébergement proche du niveau de celui d'un *datacenter*.
Les équipements mis à disposition par les membres de la communauté ont ainsi des capacités de calcul et de stockage relativement faibles par rapport aux serveurs d'un *datacenter*. Ces ressources sont de plus disponibles en quantités différentes chez chacun d'eux, en fonction de la capacité des équipements, ainsi que de ce que l'utilisateur souhaitera partager avec la communauté. Un premier utilisateur pourra partager les ressources d'un ordinateur personnel, un second celles d'un équipement type Raspberry Pi. De même pour l'espace de stockage, la capacité partagée par les utilisateurs sera différente en fonction de l'espace disponible chez chacun d'eux.
La capacité d'accéder et d'utiliser ces ressources domestiques par les autres utilisateurs sera aussi différente en fonction des caractéristiques du réseau connectant ces ressources à Internet. Certaines ressources seront accessibles via des réseaux très haut débit comme de la fibre optique offrant des débits proches de 100Mbits symétriques. D'autres ressources seront connectées par un accès type ADSL avec un débit asymétrique de quelques Mbits. Il peut être noté que la Qualité de Service (QoS) pour accéder à une même ressource pourra être perçue de manière différente par les utilisateurs de cette ressource, en fonction de sa proximité dans le réseau. Une ressource sera accédée avec une bonne QoS par un utilisateur si celle-ci est locale ou dans le réseau du même opérateur, alors qu'elle aura une moins bonne QoS pour un utilisateur distant.
La disponibilité des ressources est plus difficile à garantir dans un contexte domestique par rapport à l'environnement d'un *datacenter*. Alors qu'une infrastructure d'hébergement comme Amazon EC2 offre une disponibilité proche 99,99% [7], les équipements domestiques sont sujets à des pannes matérielle ou des coupures électriques locales. De plus certains utilisateurs, dans un soucis d'économie d'énergie, peuvent choisir d'éteindre des équipements la nuit par exemple. Cette disponibilité est aussi liée à celle des réseaux connectant ces équipement à Internet, qui peuvent aussi être sujets aux pannes.

3. Travaux existants

De précédents travaux ont proposé d'agréger des ressources domestiques de calcul et de stockage pour offrir une solution capable d'héberger des services. Cloud@Home [4] définit une architecture pour la mise en œuvre d'une infrastructure à la demande (Infrastructure-as-a-Service (IaaS)) comme abstraction de ces ressources distribuées. Cette architecture réalise notamment la gestion des ressources à travers des *brokers* dédiés pour le calcul ou le stockage. Des travaux ont repris cette architecture pour définir le concept de Peer-to-Peer (P2P) Cloud [1][6] tout en montrant l'importance d'un certain degré d'autonomie de chacun des noeuds participants afin d'assurer le passage à l'échelle et la robustesse aux pannes. D'autres travaux ont étudié les *Community Clouds* [5], notamment les différents moyens pour inciter les participants à contribuer à l'infrastructure. Ces travaux se focalisent principalement sur la réalisation d'une infrastructure d'hébergement à partir de ressources distribuées. La gestion de ces ressources permet de prendre en compte les différentes capacités de calcul et de stockage des équipements partici-

pants.
Les récents travaux autour du *Fog Computing* [9] proposent également d'utiliser des ressources domestiques ou situées dans les réseaux des opérateurs afin de délocaliser des tâches depuis les *datacenters* vers les utilisateurs. Le premier objectif est de pouvoir traiter au plus tôt des données générées par des capteurs pour éviter un goulot d'étranglement dans le réseau. Le second objectif est de localiser les services au plus près des utilisateurs afin d'offrir une plus grande réactivité. Cette approche montre l'importance, dans le cadre de l'utilisation de ressources domestiques, de considérer cette localisation du service par rapport aux utilisateurs afin d'améliorer la qualité perçue du service.
Ces solutions proposent un service de niveau IaaS avec une granularité se situant au niveau de la Machine Virtuelle (VM). L'utilisateur d'un tel service décrit les besoins de la VM qu'il souhaite héberger en termes de ressources. Les approches proposées prennent en compte la dispersion des ressources entre différentes localisations pour permettre le placement de cette VM là où les ressources demandées sont disponibles. Or dans notre cas, l'ensemble des ressources nécessaires à une VM pour un service complet ne seront pas forcément disponibles au même endroit. De plus la description des besoins d'une VM ne permet pas de s'assurer de la performance effective du service hébergé. Une approche différente doit donc être considérée pour notre système d'hébergement communautaire.

4. Vers une approche orientée service

L'émergence de nouvelles technologies système et logiciel amènent aujourd'hui à reconsidérer les architectures décrites précédemment. Les solutions de conteneurs logiciel telles que Docker [3] permettent, au même titre qu'une VM, de déployer un service sur un équipement avec un minimum de prérequis et d'assurer un isolement suffisant des autres services déployés sur le même équipement. Les conteneurs logiciels ont cependant un surcoût moins important en ressources processeur et mémoire. Il est possible de virtualiser des services sous formes de conteneurs logiciels sur des architectures matérielles trop contraintes pour une VM. Cette solution permet donc de déployer des services sur un plus grand nombre d'équipements domestiques.
Les micro-services [8] sont une nouvelle approche des architectures logicielles orientées service. La conception à base de micro-services d'une application s'oppose à une conception monolithique où toutes les fonctions de l'application sont situées dans le même logiciel. Au contraire, un micro-service intègre une fonction unique de l'application qui peut être déployée indépendamment et réutilisée dans plusieurs applications. Chaque microservice doit spécifier une interface (*API*) bien définie et utilisée par d'autres microservices. L'application est alors constituée par une composition de plusieurs micro-services indépendants et non-spécialisés à l'application.
Cette utilisation conjointe des conteneurs logiciels et des micro-services apporte comme premier avantage dans notre contexte d'utilisation de mieux s'adapter à l'hétérogenéité des ressources. Par exemple, l'application d'un service de partage de photos peut utiliser deux microservices, un pour le stockage des photos et l'autre pour la génération de vignettes. Chacun de ces composants sollicite un type de ressource différent (stockage et CPU). Il est donc possible de les déployer séparement sur différents équipements participants selon les ressources qu'ils mettent à disposition. Le micro-service stockage pourra être déployé par exemple sur un NAS tandis que le micro-service de génération de vignettes sera déployé sur un Raspberry Pi.

3. `https://docker.io`

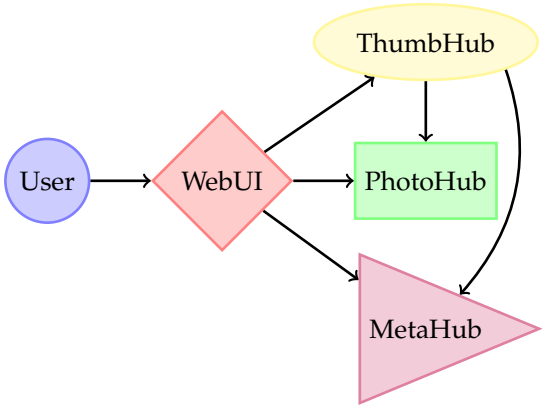

FIGURE 1 – Architecture de l'application de partage de photos.

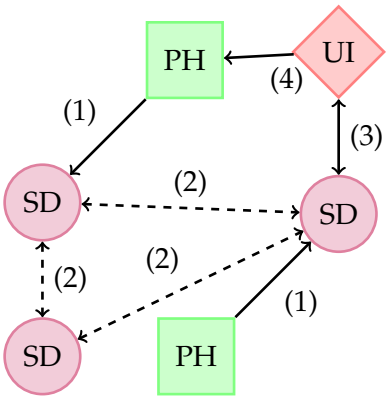

FIGURE 2 – Enregistrement et découverte d'instances.

De plus, la capacité d'un micro-service à pouvoir être instancié plusieurs fois permet d'assurer une meilleure disponibilité du service. Dans notre contexte d'utilisation, il peut être envisagé que des instances d'un même micro-service soient déployées sur des équipements d'utilisateurs différents. Le service communautaire sera ainsi toujours disponible même en cas de panne d'un des équipements ou d'une défaillance de la connectivité chez un utilisateur. Par contre, cette séparation de l'application en plusieurs services indépendants, s'exécutant sur des équipements distants introduit un coût de communication non négligeable pour chaque requête d'un service à un autre, là où une application monolithique ne fait que des appels à des fonctions locales.

5. Architecture proposée pour le support des micro-services

Comme décrit précédemment, la conception logicielle à base de micro-services invite à définir des services indépendants à partir de chaque fonction de l'application. Afin de réaliser un exemple de service de partage de photos, nous avons défini 4 micro-services :

- *WebUI* : Ce micro-service fournit une galerie de photos présentant des vignettes et les méta-données.
- *PhotoHub* : Ce micro-service stocke les fichiers des photos et les récupére à partir d'un identifiant.
- *ThumbHub* : Ce micro-service génère une vignette pour une photo à partir de son identifiant.
- *MetaHub* : Ce micro-service stocke les méta-données des photos, ainsi que le catalogue de photos présentes dans la galerie.

Ces micro-services, bien que déployés et s'éxecutant de manière indépendante, peuvent également utiliser d'autre micro-services pour remplir leur fonction. Ainsi *WebUI* joue le rôle de composant *front-end* et fait appel à chacun des autres composants constituant le *back-end* de l'application. Le micro-service *ThumbHub* utilise *PhotoHub* pour récupérer le fichier de la photo à réduire ainsi que des informations de *MetaHub* pour effectuer la tâche. Les interactions entre les différents micro-services de l'application sont présentées dans la figure 1.

Afin d'être déployés en plusieurs instances capable de rendre un service équivalent, certains micro-services de cette application nécessitent une fonction de partage et de réplication des

données entre les instances des données du service. Le micro-services *WebUI* ou *ThumbHub* ne stocke pas de données propres à leurs services. Ils utilisent les données stockées par les micro-services *PhotoHub* et *MetaHub*. Ces informations doivent être disponibles pour toutes les instances du même micro-service. Une fonction de partage et de réplication des données entre les différentes instances d'un même micro-service est donc nécessaire. Il en existe plusieurs solutions comme les bases de données distribuées ou les systèmes de fichiers répartis, chacune plus ou moins adaptées au contexte d'une infrastructure communautaire [3]. Chaque solution maintient un certain niveau de cohérence des données entre les instances et de disponibilité permanente de ces informations au niveau de chaque instance, en conformité avec le théorème CAP[2]. Le type de solution de partage de donnée choisie par un micro-service impacte donc le niveau de cohérence entre ses instances.
Bien que développés dans différents langages, les micro-services exposent une interface bien définie et connue de chaque composant souhaitant les utiliser. Dans l'exemple de l'application de partage de photos, certains services ont été développés en Python, d'autres en NodeJS. Ils offrent chacun une interface de *Web services RESTful* permettant d'accéder à leurs fonctionnalités. Cette interface doit de plus être asynchrone en fonction des types de requêtes. Si une requête à un micro-service nécessite pour sa réalisation de faire appel à d'autres micro-services, le résultat ne sera pas disponible immédiatement. Une interface asynchrone permet d'indiquer au client la progression du résultat.
Pour spécifier la destination de cette requête, un micro-service a besoin de faire appel à une fonction de découverte de services pour localiser les instances des autres micro-services dont il a besoin. Pour être découverte, chaque instance d'un micro-service s'enregistre au préalable auprès de cette fonction. Afin que d'assurer la disponibilité de la fonction de découverte de services, elle peut être distribuée sur les équipements comme les autres micro-services. Le mécanisme de partage des données évoqué précédemment permet ensuite aux enregistrements d'être consultés quelle que soit l'instance de la fonction de découverte de service.
La figure 2 présente les interactions entre la fonction de découverte de service et les instances de micro-services. En (1), deux instances du service *PhotoHub* s'enregistrent auprès de la fonction de découverte de service *SD*. Cette fonction propage en (2) ces enregistrements par la fonction de partage des données. Une instance du micro-service *WebUI* peut ensuite demander en (3) à découvrir les instances du micro-service *PhotoHub*. La fonction de découverte lui fournira une liste d'instances à partir de laquelle le micro-service choisira la destination de sa requête (4).

6. Déploiement et tests

Nous souhaitons évaluer cette architecture orientée micro-service dans un cas réel de déploiement de ressources domestiques distribuées. Plusieurs utilisateurs ont accepté d'héberger chez eux un boitier de type mini-PC connecté à leur réseau domestique. Ces boitiers APU2, assemblés par PC-Engine [4], offrent des capacités de calcul et de stockage proches de celles que pourra offrir un équipement connecté moyen dans les années futures : un processeur 4 coeurs 1GHz et 4Go de RAM. Le disque de stockage de 32Go peut être utilisé pour le stockage local des photos. Ces boitiers sont configurés avec un système Linux standard avec les outils de gestion des conteneurs logiciel Docker. A partir de chaque boitier il est possible d'instancier un micro-service depuis un dépôt privé contenant les images des services de l'application de partage de photos.
Ces boitiers sont joignables directement les uns des autres à travers la connectivité offerte par

4. https://www.pcengines.ch/apu.htm

Src \ Dst	node1	node2	node 3	node 4
node1 (Free-ADSL)	-	0.26	0.27	0.26
node2 (Free-ADSL)	0.66	-	0.7	0.71
node3 (Free-FO)	3.6	1.9	-	76
node4 (Orange-FO)	4.3	2	17	-

TABLE 1 – Mesure de la bande passante (Mbits)

	node2	node 3	node 4
node1	69	58	50
node2	-	63	57
node3	-	-	22

TABLE 2 – Mesure des Round Trip Time (RTT) symétriques (ms)

le réseau domestique. Nous utilisons l'adressage IPv6 afin que les services déployés sur les nœuds soient globalement accessibles. Les tableaux 1 et 2 présentent les caractéristiques de cette interconnexion entre les nœuds de notre système. Ces mesures permettent de remarquer que ces valeurs sont assez hétérogènes d'un nœud à un autre.

Afin d'assurer la synchronisation des données, nous avons utilisé plusieurs solutions selon le type de donnée considéré. Pour les données de type fichier, nous avons utilisé IPFS [5], un système de fichier réparti avec de bonnes performances et capable de passer à l'échelle [3]. Ainsi les photos sauvergardées par le micro-service *PhotoHub* sont stockées dans un volume IPFS. La clé de hashage que retourne IPFS est utilisée par la suite comme identifiant pour retrouver la photo correspondante depuis n'importe quelle instance de *PhotoHub*. Cette solution fonctionne comme un cache P2P distribué où chaque fichier est récupéré à la demande depuis un *peer* possèdant une copie puis stocké localement. Il est donc possible qu'une photo qui n'ait jamais été demandée (et donc jamais répliquée) disparaisse. Il pourra être nécessaire, dans un environnement où les équipements ont une faible disponibilité, d'utiliser un système de fichier réparti assurant des copies redondantes des fichiers.

Pour des données moins volumineuses, nous utilisons une base de donnée distribuée type clé-valeur. Nous avons choisi la solution Consul [6], largement utilisée dans le monde des micro-services. Elle assure la cohérence de la base de donnée par un algorithme de consensus distribué. Consul intègre de plus les structures de données pour l'enregistrement de service, ainsi qu'une interface type Domain Name System (DNS). Cependant Consul reste une solution orientée *datacenter* et ne prend pas en compte une QoS différente pour chaque instance de service. Or cette information permettrait au client de sélectionner l'instance du service avec la meilleure QoS et ainsi minimiser l'impact de la distribution des micro-services sur la performance globale du service. La base de données distribuée sert aussi de support au fonctionnement des micro-services. Ainsi le service *MetaHub* stocke les méta-données des photos sous la forme d'enregistrement clé-valeur dans cette base de données.

7. Conclusion

Dans cet article, nous avons considéré l'usage de ressources domestiques pour héberger des services communautaires afin de s'affranchir des plate-formes des réseaux sociaux. Nous avons fait le constat des limitations imposées par ce type de ressources et proposé une approche orienté micro-services afin de s'adapter au mieux à leur hétérogénéité et à leur faible disponibilité. Nous avons ensuite mis en œuvre cette approche par le développement d'une application de partage photo et son déploiement sur des équipements réels présents chez plusieurs utilisateurs.

5. https://ipfs.io/
6. https://consul.io

Cette expérimentation a montré certaines limitations des solutions de synchronisation des données et de découverte de services, nécessaires au fonctionnement des micro-services. Ces solutions conçues dans le contexte d'un *datacenter* doivent être adaptées au contexte des ressources domestiques. Nous souhaitons de plus intégrer dans cette étude la variabilité de la disponibilité des ressources dans le temps. Nous envisageons à terme que notre système sera capable de détecter des changements dans les ressources disponibles et d'y adapter le déploiement des micro-services.

Bibliographie

1. Babaoglu (O.), Marzolla (M.) et Tamburini (M.). – Design and Implementation of a P2P Cloud System. *In : Proceedings of the 27th Annual ACM Symposium on Applied Computing*. pp. 412–417. – New York, NY, USA, 2012.
2. Brewer (E. A.). – Towards Robust Distributed Systems (Abstract). *In : Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing*. pp. 7–. – New York, NY, USA, 2000.
3. Confais (B.), Lèbre (A.) et Parrein (B.). – Quel système de stockage pour les architectures Fog ? *In : Compas'2016*. – Lorient, France, juillet 2016.
4. Cunsolo (V. D.), Distefano (S.), Puliafito (A.) et Scarpa (M.). – Volunteer Computing and Desktop Cloud : The Cloud@Home Paradigm. *In : 2009 Eighth IEEE International Symposium on Network Computing and Applications*, pp. 134–139.
5. Khan (A. M.), Selimi (M.) et Freitag (F.). – Towards Distributed Architecture for Collaborative Cloud Services in Community Networks. *In : 6th International Conference on Intelligent Networking and Collaborative Systems (INCoS'14). Salerno, Italy : IEEE*.
6. Mayer (P.), Klarl (A.), Hennicker (R.), Puviani (M.), Tiezzi (F.), Pugliese (R.), Keznikl (J.) et Bure (T.). – The autonomic cloud : A vision of voluntary, peer-2-peer cloud computing. *In : Self-Adaptation and Self-Organizing Systems Workshops (SASOW), 2013 IEEE 7th International Conference On*. pp. 89–94. – IEEE.
7. Nabi (M.), Toeroe (M.) et Khendek (F.). – Availability in the cloud : State of the art. *Journal of Network and Computer Applications*, vol. 60, janvier 2016, pp. 54–67.
8. Newman (S.). – *Building Microservices : Designing Fine-Grained Systems*. – "O'Reilly Media, Inc.", février 2015. ISBN 978-1-4919-5033-3.
9. Vaquero (L. M.) et Rodero-Merino (L.). – Finding your way in the fog : Towards a comprehensive definition of fog computing. *ACM SIGCOMM Computer Communication Review*, vol. 44, n5, 2014, pp. 27–32.