Cloud et virtualisation

Rapport de projet :

Déploiement d'une solution cloud d'hébergement d'image

Thomas Dumond & Ethan Huret

**Git :** <https://github.com/EthanAndreas/CloudAppDeployment>

Table des matières

[Table des matières 2](#_Toc135476267)

[I. Déploiement de l'application 3](#_Toc135476268)

[I.1 Application à déployer 3](#_Toc135476269)

[I.2 Infrastructure 3](#_Toc135476270)

[I.3 Déploiement de l'architecture 4](#_Toc135476271)

[II. Déploiement d’une nouvelle version de l’application 5](#_Toc135476272)

[II.1 Création de nouvelles images Docker 5](#_Toc135476273)

[II.2 Lancement avec Nomad 6](#_Toc135476274)

[III. Maintenance planifiée d’un nœud 6](#_Toc135476275)

[IV. Scalabilité horizontale 6](#_Toc135476276)

[V. Impacts en cas de panne 7](#_Toc135476277)

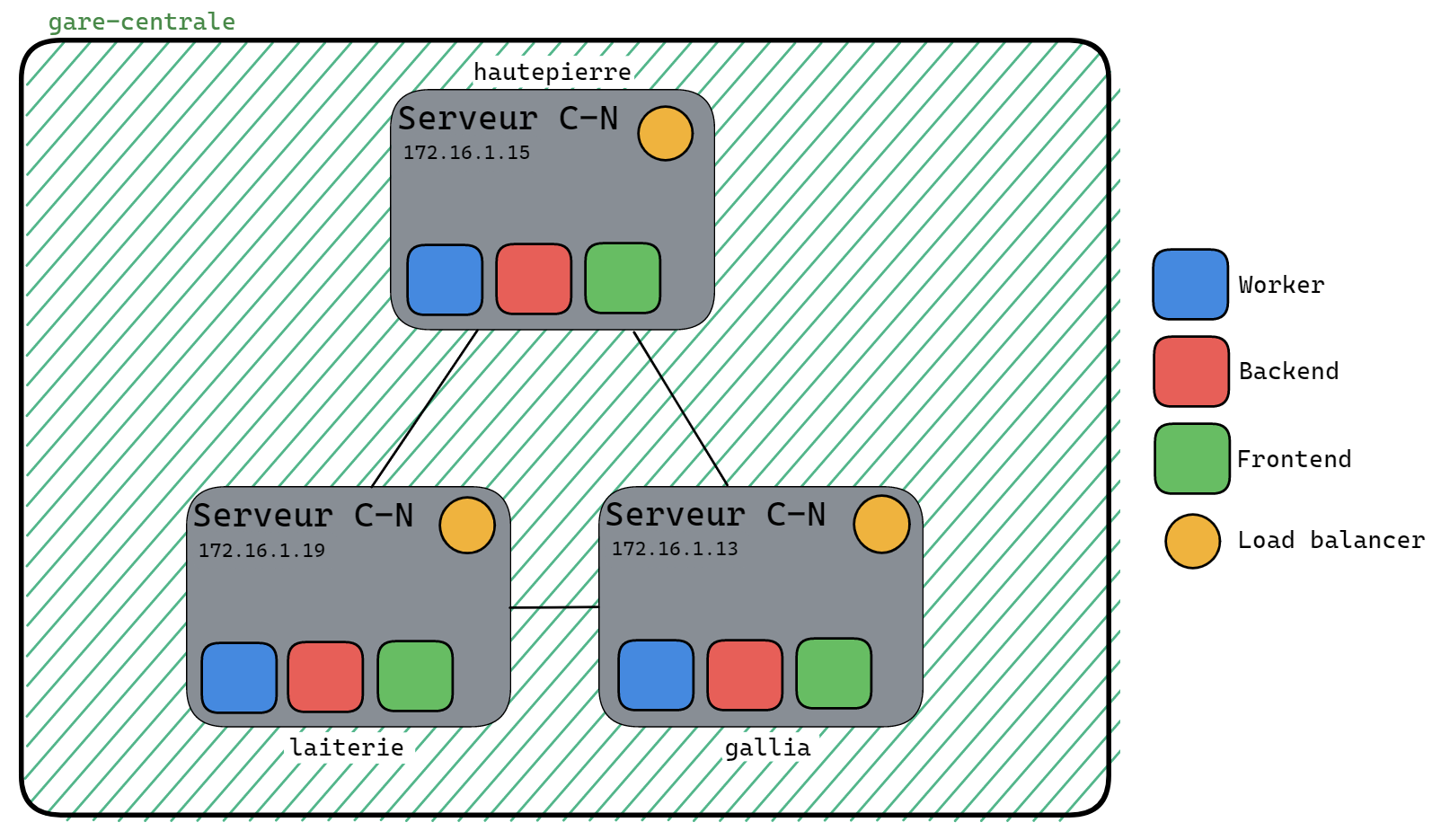
[VI. Limites 7](#_Toc135476278)

1. Déploiement de l'application

I.1 Application à déployer

L'application à déployer correspond à une solution d'hébergement et de redimensionnement d'image en ligne. Elle contient 3 parties : frontend, backend et worker. La partie frontend correspond à l'interface web, où l'utilisateur upload l'image qu'il veut stocker. La partie backend récupère les images uploadées et les stocke sur un service de stockage de type S3. Et enfin, la partie worker permet de redimensionner les images.

I.2 Infrastructure

 Voici un exemple d'architecture à 3 nœuds permettant de déployer l'application. Ces nœuds correspondent à des Virtual Machine, qui possèdent une adresse IP (*192.168.70.*) pour accéder à la machine en SSH et une adresse IP (*172.16.1.*) pour communiquer entre elles via des tunnels VXLAN. Toutes ces VM sont configurées en tant que serveur Consul-Nomad dont une sera désigné automatiquement comme leader. Le déploiement des différentes parties de l'application (frontend, backend, worker) se fait via des conteneurs Docker. Ces conteneurs sont ensuite déployés dans un job Nomad, et sont répertoriés et regroupés en tant que service (pour chaque partie de l'application) dans Consul.

Une image contenant texte, capture d’écran, diagramme

Description générée automatiquementDe plus, une adresse IP flottante est configurée afin de recevoir le flux http de l'application sur une des VM et de le distribuer aux différents conteneurs de la partie frontend via un load balancer HAProxy. S'ajoute à cela, un proxy qui permet de rediriger le trafique http de l'URI *https://gare-centrale.100do.se/* vers l'IP flottante *172.16.3.4*. Ainsi, l'interface web de l'application se trouve à cette [URI](https://gare-centrale.100do.se/).

I.3 Déploiement de l'architecture

Pour déployer cette architecture, nous avons utilisé Ansible afin de définir les différentes configurations des VM et de déployer l'application selon l'architecture précédente. Pour cela un playbook est fourni dans le [Git](https://github.com/EthanAndreas/CloudAppDeployment). Le playbook est structuré de la manière suivante :

Une image contenant texte, capture d’écran, Police, conception

Description générée automatiquement

Où chaque élément a pour objectif :

- playbook.yaml lance les différents rôles et leurs tâches.

- reset.yaml stoppe les services Consul, Nomad, KeepAlived et Docker.

- inventory.yaml définit les différents nœuds de la configuration.

- les rôles définissent les différentes suites de tâches à exécuter :

* Consul déploie les configurations consul.hcl de chaque VM et démarre le service.
* Nomad déploie les configuration nomad.hcl de chaque VM, démarre le service et lance les jobs définis dans job.hcl, où eux-mêmes lancent les conteneurs Docker du frontend, du worker/backend et le load balancer HAProxy.
* Docker lance le service et ajoute Nomad au groupe Docker.
* KeepAlived[[1]](#footnote-2) déploie les configurations keepalived.conf de chaque VM et démarre le service.

Chaque rôle possède un répertoire var qui permet de définir les caractéristiques de l'architecture, notamment pour Nomad, où var/main.yml permet de mettre les liens des répertoires Git de chaque image Docker de l'application.

Une image contenant texte, capture d’écran, carte de visite, Police

Description générée automatiquement Pour déployer ansible et lancer tous les rôles, il faut exécuter à la racine du [Git](https://github.com/EthanAndreas/CloudAppDeployment) :

Une image contenant texte, carte de visite, capture d’écran, Police

Description générée automatiquement Pour déployer seulement un rôle, il faut exécuter :

Une image contenant texte, carte de visite, capture d’écran, Police

Description générée automatiquement Pour stopper tous les services, il faut exécuter :

II. Déploiement d’une nouvelle version de l’application

II.1 Création de nouvelles images Docker

Lors de la mise à jour de l’application (ie: de l’api ou du frontend), il faut commencer par rebuild les images de Docker. Comme cela nécessite un registry Github, et que par conséquent, une connexion via des identifiants est nécessaire, nous ne l’avons pas ajouté dans le playbook d'Ansible, l'utilisateur devra donc placer ses images sur un Git, et entrera les URI des répertoires contenant les images dans les variables du rôle Nomad.

Voici les commandes à réaliser pour construire les images :

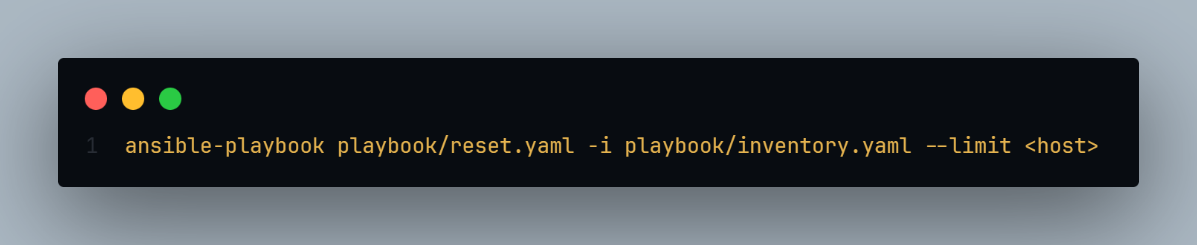
* cd web
* docker build -t frontend .
* docker tag frontend <registry>/frontend:<tag>
* docker push <registry>/frontend:<tag>
* cd api
* docker build -t worker .
* docker tag worker <registry>/worker:<tag>
* docker push <registry>/worker:<tag>

II.2 Lancement avec Nomad

Pour déployer la nouvelle version mise à jour de l'application, il faut modifier les URI des variables de Nomad, afin qu'il prenne les nouvelles images dans le job, et ensuite, il faut lancer Ansible seulement pour le rôle Nomad.

III. Maintenance planifiée d’un nœud

 A la suite d’une mise à jour d’un nœud, il faut relancer Ansible pour tous les rôles ou seulement le rôle pour lequel le service est mis à jour, pour ce nœud en particulier :

 Il est aussi possible de stopper tous les services d'un nœud en particulier si l'on souhaite effectuer des modifications :

IV. Scalabilité horizontale

Une image contenant texte, capture d’écran, affichage, logiciel

Description générée automatiquement

Pour ajouter un serveur, il suffit d'ajouter un nœud dans inventory.yaml. En reprenant l'exemple de la partie **Déploiement de l'application**, 3 nœuds sont déjà configurés.

Ainsi, pour ajouter un nœud, il faut procéder de la même manière c’est-à-dire :

il faut ajouter l'adresse IP de l'interface VXLAN du nœud et lui donner une priorité vis-à-vis de l'attribution de l'adresse IP flottante via KeepAlived.

Pour supprimer un serveur, la procédure est à la même il faut supprimer l'host correspondant, et modifier le nombre de serveur Consul et Nomad dans leur fichier vars/main.yml respectifs.

V. Impacts en cas de panne

Dans le cas où un nœud venait à tomber en panne, cela ne poserait pas de problème. En effet, Consul mettrait à jour son catalogue, ainsi les instances nomad et nomad-client reliés à consul et dépendant de ce nœud seraient supprimés. Ensuite, Nomad mettrait à jour sa liste de serveur-client où le nœud passerait en mode Down. Tous les jobs tournants sur le nœud en question seraient récupérés par d'autres nœuds. Enfin, étant donné que le nœud est en panne il n'enverra plus de requête VRRP et donc un autre nœud se désignerait auprès de KeepAlived afin de récupérer l'adresse IP flottante.

VI. Limites

Une des premières limites de l'implémentation actuelle est que HAProxy ne redirige pas le flux de l'adresse IP flottante vers le frontend. En effet, la liaison doit se faire sur la configuration d'HAProxy où il faut indiquer au load balancer de relier le flux vers les services frontend de Consul. Une autre limite de l'implémentation est que le frontend ne peut pas communiquer avec le backend, car le backend ne répond pas aux requêtes du frontend sur son port. Enfin, une dernière limite concerne l'implémentation des jobs dans Nomad. En effet, cette implémentation définit un seul job pour les différents conteneurs de l'application, or, afin d'ajouter de la redondance, il faudrait ajouter un job par conteneur.

1. Nous avons rencontré des problèmes avec ce service qui se retrouvait avec des configurations du service erronés. Pour cela, nous avons réécris la configuration du service. Pour l'utiliser il faut décommenter le code associé dans les tâches de KeepAlived. [↑](#footnote-ref-2)