CLOUDSYS – IAAS

Déploiement sur le Cloud

Scripts de déploiement
https://github.com/samuel-fringeli/cloudsys-iaas-deployment/tree/main

Application
https://github.com/samuel-fringeli/samf-appstore

Technologies
Loïc Guibert : AWS Amazon
Marc Imbert : GCE
Rémy Vuagniaux : Azure
Léonard Noth : SwitchEngine
Samuel Fringeli : Exoscale

Context

The first practical work asked during the CloudSys course consists of a three-tier application developed with a IaaS architecture. This present document describes its architecture, links and environments.

Application Presentation

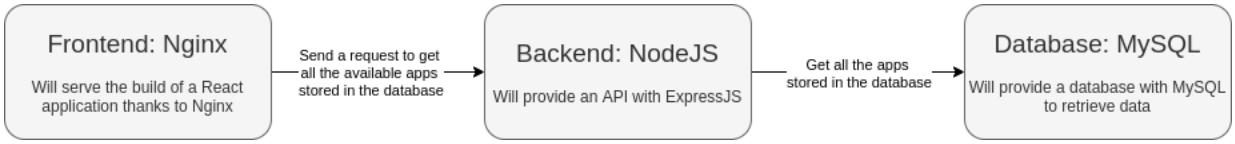
Our application, “samf-appstore”, is a landing page designed to install custom iOS applications on Apple mobile devices. Some applications need special capabilities only provided by an Apple Developer Accounts, linked to the unique identifier of mobile devices.

The landing page is reserved to users that have their UDID linked to the Apple Developer Account of Samuel Fringeli. If it is the case, they can install offered mobile applications with an “in-house deployment”, which avoid using the official App Store to do this.

This method is useful to distribute apps that are in development to a restricted number of users.

Application Architecture

Our application is composed of three modules, defined by their purpose.



All mobile applications proposed to users are stored in a **MySQL database**, which is relational. This environment has been chosen because of its large community and adoption through the world.

The **API (backend)** has been built using a NodeJS server, which is a JavaScript runtime. Its event-driven design coupled with the Express framework, used to create HTTP(S) routes, led to its choice.

Finally, the **landing page (frontend)** has been conceived using the React library. It is served by a Nginx webserver, which is really suited for such web applications. This last tier’s technologies were chosen according to the developer skills.3

Application Links

The **frontend** uses two routes of the API, using HTTP GET requests: */api/apps*, to get a list of all mobile applications, and /api/[appId]/manifest.plist, used to generate a *.plist* file to install the application on an iOS device. Therefore, it needs to know either the domain name or the IP address of the backend.

For the **backend**, a connection to the database must be established at its start, using a SQL driver. This connection is bidirectional but based on a client-server paradigm. The connection uses the TPC/IP protocol. A domain name or an IP address pointing to the database tier must be specified.

The last tier, the **database**, do not need any network resource.

Those two IP addresses must be known by the first two tiers, which means that we must specify them in their configuration. Their values could be given to the tiers after the creation of all virtual instances, by storing them to environment variables or by editing files on the machines’ filesystem.

Critères

Voici les quatre critères que nous avons définis pour l'évaluation des différents fournisseurs. Nous les avons choisis car nous les estimons importants pour le choix d'une plateforme.

Prix

Ce critère s'exprime sur le coût de trois instances à l'heure, en incluant les éventuels frais supplémentaires (trafic, adresse IP publique, …). La solution la moins chère aura la meilleure note et inversement pour la plus chère.

Documentation

La documentation d'une plateforme joue énormément sur son utilisabilité, car c'est à cet emplacement où l'on trouve les procédures et informations pour réaliser nos opérations. Nous évaluons la facilité à trouver l'information, son exhaustivité et son utilisabilité.

Possibilités de l'API

Nous évaluons ici la variété des actions et opérations possibles depuis l'interface de programmation en ligne de commande.

Interface utilisateur

Enfin, l'interface graphique de l'utilisateur et utilisatrice est évaluée sur sa clarté, son utilisabilité, son expérience utilisateur et sa facilité d'utilisation.

Tableau comparatif

Les notes sont de 1 à 5, avec 5 pour le meilleur et 1 pour le plus mauvais.

Cloud	**Prix (pour 3 instances, à l'heure)**	**Documentation**	**Possibilités de l'API**	**Interface utilisateur**	
AWS	0.0348$ (4/5)	4.5/5	5/5	3.5/5	17/20
GCE	0.03$ (5/5)	3/5	4/5	2/5	14/20
Azure	0.0365$ (2/5)	4/5	3/5	4.5/5	13.5/20
Switch Engine	0.0360$ (3/5)	0.5/5	3.5/5	4.5/5	11.5/20
Exoscale	0.0479$ (1/5)	3/5	5/5	1/5	10/20

Une explication des notations des critères pour chaque plateforme est fournie en tant qu'annexes.

Conclusions

Avec les critères évalués ci-dessus, nous constatons que les trois géants (Amazon, Google et Microsoft) sont en tête de classement. Ceci est cohérent par rapport à la part d'utilisation que ces services dans le Cloud. De même, entre SwitchEngine et Exoscale, qui fonctionnent sur les mêmes technologies sous-jacentes, la différence d'appréciation se fait entre le prix (en faveur de Switch Engine), la documentation (en faveur d'ExoScale) et l'interface utilisateur (en faveur de Switch Engine).

Ce travail était intéressant car nous avons pu découvrir les différents fournisseurs de service Cloud. Étant donné que chaque personne a pu se concentrer sur son fournisseur, ce qui a permis de bien explorer les possibilités puis de les partager avec ses partenaires de laboratoire.

Annexes

AWS: Feedback

Loïc Guibert

Price

Resources needs:

- 3 EC2 instances t2.micro
- No elastic addresses
- Volumes not kept at termination

Price on-demand, no spot

Total for an hour: 3 x $0.0116 / hour

Documentation

The official documentation is detailed and complete enough to create a first EC2 instance and to describe the differences. But more specialized details were difficult to find: for this, online resources from unofficial sources were really useful.

AWS: good, complete but a LOT of terms and particularities. 4/5

Boto3: Great, very complete resources but not user-friendly. 4/5

awscli: great explanation and respond to problems. 5/5

Cloud User Interface

The AWS UI is good and gets the work done, but the UX is really complicated. For a novice, it is easy to get lost, to search for information or actions.

A lot of operations can be done on the AWS UI, it is really complete.

3.5/5

API

Using boto3, the field of possible is enormous. Besides some details, the API was easy to understand and use. A lot of online knowledge can be found on forums and blogs.

5/5

Azure

Prix

Le prix des machines virtuelles est dans la norme avec 1cpu et 1 G RAM a 8.76$

Le prix d'une adresse ip static est de 0.0036$ par heure.

Documentation et API

La documentation pour l'utilisation avec la ligne de commande est très bien fournie avec beaucoup d'exemple pour différent cas d'utilisation. En revanche pour l'API python, la documentation est très incomplète. Le site officiel d'azure ne fournit que très peu d'exemple sur python et il est difficile de trouver des cas d'utilisation similaire.

Aussi certaines fonctionnalités disponibles sur la GUI et avec les lignes de commandes ne sont pas disponibles avec l'api python. Par exemple : custom data qui permet de lancer un script à la création d'une machine virtuelle.

J'ai dû contourner ce problème en appelant des lignes de commandes depuis le script python.

Il est aussi impossible d'utiliser les clé ssh avec python.

Interface utilisateur

L'interface utilisateur d'azure est bien détaillée. Il est facile de créer un réseau de machines virtuelles en peu de temps. En revanche les prix excepté pour le prix des machines virtuelles ne sont pas indiqué directement (par exemple: le prix des adresses statiques).

La création d'une machine virtuelle simple est très bien implémentée et détaillée.

Conclusion personnelle

Ce laboratoire était très instructif car c'est la première fois que j'utilise le cloud mais j'ai trouvé le laboratoire assez flou sur ce qui nous était demandé. Ma solution est fonctionnelle mais aurais pu être mieux optimisée avec plus de temps et de connaissances.

Léonard Noth

Switch Engine

Prix

Pour une instance « small » avec 1 CPU core et 1 GO de RAM, avec 15 GO de stockage, le coût est de 0.031757 CHF par heure (22.87 CHF par mois)[1].
Chaque adresse IP élastique coûte 0.83 CHF par mois.
La bande passante n'est pas facturée sur Switch Engine.
Le stockage de snapshot coûte 0.0000254 CHF/GO/heure (0.0183 CHF par mois).

Documentation et API

La documentation de Switch engine est limitée à l'utilisation de l'interface[2], sauf une très petite section dans la FAQ qui indique les variables d'environnement à setter pour utiliser le CLI d'Openstack. Pour le reste, tout est renvoyé à la documentation d'Openstack[3]. Le lien d'Openstack contenu dans Switch Engine renvoie au client Python d'Openstack, mais celui-ci est réalisé uniquement pour un accès par ligne de commande (CLI).
La documentation d'Openstack offre un PDF complet pour l'utilisation du CLI, mais aucune indication pour utiliser le SDK Python. Pour utiliser ce SDK, il faut fouiller dans les repos d'Openstack pour trouver le client Python pour Nova[4]. Ce client permet de gérer les instances sur Switch engine, et offre notamment une méthode qui crée une instance.
Cependant, pour arriver à le faire fonctionner, il faut utiliser les instructions de configuration très sommaires (environ une page A4) présentées sur la page « usage »[5], tout en ajoutant d'autres paramètres provenant des instructions concernant les variables d'environnement de Switch Engine, qui ne sont pas prises en charge par le client nova.
De plus, il faut rajouter un paramètre lors de l'instanciation du client nova : region_name (un paramètre qui n'est pas dans la documentation officielle[6], et qu'il faut donc deviner en allant fouiller dans le code source du SDK). Sans ce paramètre, il est impossible de créer des instances à l'aide du SDK, et l'erreur retournée par le client nova n'est pas du tout équivoque.
Une fois les instances créées, il est impossible d'accéder à celles-ci à l'aide d'une IP publique, car le SDK de nova ne permet pas d'assigner une IP publique à une interface réseau.
Pour ajouter une IP publique, il faut donc utiliser le client Python de Neutron[7]. La documentation de ce client contient également une page « basic usage » extrêmement sommaire[8], mais **absolument aucune indication sur la référence des classes Python**.
C'est donc uniquement en fouillant le code source du client Neutron qu'il est possible de deviner les méthodes à utiliser : list_floatingips() pour lister les adresses IP publiques disponibles, et update_floatingip() pour les mettre à jour. Comme il est impossible de connaître le retour des méthodes, il faut utiliser le débugger et observer la réponse retournée

[1] https://www.switch.ch/export/sites/default/engines/.galleries/files/SWITCHengines-Pricing-Flyer-2021.pdf
[2] https://help.switch.ch/engines/documentation/
[3] https://docs.openstack.org/python-openstackclient/latest/
[4] https://docs.openstack.org/python-novaclient/latest/
[5] https://docs.openstack.org/python-novaclient/latest/user/python-api.html
[6] https://docs.openstack.org/python-novaclient/latest/reference/api/novaclient.client.html
[7] https://docs.openstack.org/python-neutronclient/pike/reference/index.html
[8] https://docs.openstack.org/python-neutronclient/pike/reference/index.html#basic-usage

afin de trier les données qu'on veut obtenir (en l'occurrence, les adresses IP qui ont un statut « down »).
Pour la méthode update_floatingip(), les commentaires dans le code source ne sont **pas suffisants** pour connaître les paramètres à envoyer, car le code source transfère le paramètre « body » tel quel à un appel HTTP. Il faut donc aller dans la documentation de l'API REST Neutron d'Openstack[9] pour obtenir la structure du dict à envoyer à la méthode update_floatingip(), et deviner que le port_id à spécifier est celui de l'interface réseau liée à l'instance précédemment créée.
Cependant, pour récupérer cette interface, il faut attendre que l'instance soit démarrée, ce qui implique qu'il faut régulièrement appeler la méthode server_manager.interface_list après avoir lancé la création de l'instance, jusqu'à ce que cette méthode ne retourne pas un tableau vide. Cette procédure n'est pas notée non plus dans la documentation, elle a été devinée en faisant des tests en mode débug avec le SDK.
L'API permet de réaliser tout ce que l'interface peut faire.

Interface utilisateur

L'interface utilisateur de Switch Engine est cohérente et assez intuitive. Elle n'a pas de bugs visibles, contrairement à celle d'Exoscale. Elle est assez robuste et fiable. De plus, il est possible d'obtenir une interface simplifiée pour la gestion des instances, dans la section « quickstart ».
Cependant, il faut deviner l'URL de cette interface[10], car aucun lien n'y fait référence depuis l'interface générale d'administration[11]. La documentation concernant l'interface est, quant à elle, complète et mise à jour.

[9] https://docs.openstack.org/api-ref/network/v2/index.html#update-floating-ip
[10] https://engines.switch.ch/
[11] https://engines.admin.switch.ch/

Exoscale

Prix

Pour une instance « tiny » avec 1 CPU core et 1 GO de RAM, avec 15 GO de stockage, le coût est de 0.0159861$ par heure[1] (11.51$ par mois).
Chaque adresse IP élastique coûte 0.01389$ par heure (10$ par mois).
1 TO de bande passante est inclue par instance, et chaque GO supplémentaire coûte 0.02$.
Le stockage de snapshot coûte 0.00014$/GO/heure (0.1008$ par mois).

Documentation et API

La documentation d'Exoscale[2] fournit les définitions de toutes les classes et méthodes qu'il est possible d'utiliser en Python, ce qui permet, avec un peu de recherche, d'utiliser l'API de manière complète.
Toutefois, il existe très peu d'exemples quant à la configuration et à l'utilisation de l'API. La page des exemples[3] contient trois scripts pèle-mèle sans commentaires, et la page de configuration[4] se limite à une section de la taille d'une page A4.
Pour utiliser le SDK Python, il faut donc suivre le tutoriel de configuration puis fouiller l'API pour trouver les bonnes méthodes à utiliser pour déployer des instances.
L'API permet de réaliser tout ce que l'interface peut faire.

Interface utilisateur

L'interface utilisateur[5] d'Exoscale est fonctionnelle, mais très minimaliste et pas forcément intuitive. La gestion d'erreur dans l'interface est très sommaire, et peut souvent amener à ne pas comprendre les problèmes qui surviennent. L'exemple le plus marquant est dans l'interface de gestion des templates.
Dans cette interface, un formulaire basique nous demande un fichier QCOW2 et une checksum, puis une fois validé, la page des templates s'affiche, ajoutant le template en cours de création. Mais si on raffraîchit cette page, plus aucun template en cours de création n'apparaît. Le template apparaît ensuite au bout de quelques minutes supplémentaires lorsqu'on a rechargé à nouveau cette même page.
De même, si on laisse la page affichée une fois la création du template sans la recharger et que la création du template produit une erreur, une petite alerte rouge s'affiche sur l'interface pendant environ 5 secondes, au bout de quelques minutes. Pour lire le contenu de l'alerte, il faut donc regarder l'interface pendant plusieurs minutes et faire une capture d'écran dans une fenêtre de temps de 5 secondes afin de corriger l'erreur.
De plus, il existe des bugs dans les liens qu'il est possible de cliquer sur l'interface. Par exemple, l'accès à une instance depuis l'interface des template affiche une page blanche. Il faut donc aller manuellement dans le menu et rechercher la bonne instance, ce qui rend la présence de liens inutile et non-fonctionnelle.

---

[1] https://www.exoscale.com/calculator/
[2] https://exoscale.github.io/python-exoscale/
[3] https://exoscale.github.io/python-exoscale/usage.html
[4] https://exoscale.github.io/python-exoscale/configuration.html
[5] https://exoscale.com