Rapport de projet (PoS)

LAFORGE Samuel
SEVERAN Thomas
BAREL Loïc

Partie de Thomas

1) Découverte du projet

Au cours de cette année 2019-2020 nous avons commencé un projet qui est "le Point de Vente" (qu'on raccourcira tout au long de ce document par PoS pour Point of Sale).

En quoi consiste ce projet ?

Ce projet consiste à gérer un point de vente via l'utilisation de kubernetes (le cluster) et odoo (l'API). Ce point de vente sera géré par des raspberry PI. Par exemple une personne arrive à une machine à café, elle paye puis son café se prépare. Lors du moment où la personne a payé il y a eu une transaction, donc il a fallu un lecteur pour la carte bleue. Ce lecteur doit ensuite envoyer les informations pour accepter le paiement. Et bien notre projet consiste à gérer les données qui sont envoyées lors de l'achat.

Le cahier des charges

Pour ce projet, on nous a donné un début de cahier des charges que nous avons complété avec le temps :

- Création d'un cluster avec Kubernetes
- Création d'un point de vente avec Odoo
- Durée (Novembre 2019 - 14 juin 2020)
- Utilisation de Machines Virtuelles et Raspberry Pi
- Interconnecter les Raspberry Pi (Switch+Câbles)

2) C'est quoi Kubernetes ?

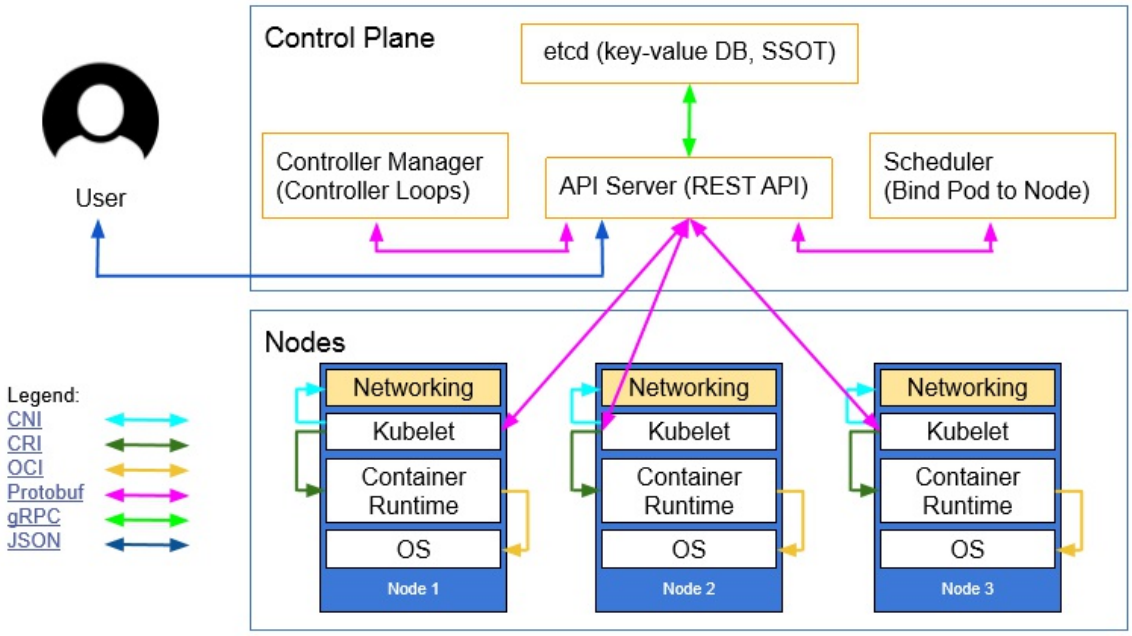
- Définition simple :
  (Source : Wikipedia)
  Kubernetes est un système open source qui vise à fournir une « plate-forme permettant d'automatiser le déploiement, la montée en charge et la mise en œuvre de conteneurs d'application sur des clusters de serveurs.
- Principales fonctionnalités :
  (Source : kubernetes.io/fr/)
  -> pas besoin de modifier une application pour la découverte de services, Kubernetes donne aux pods leurs propres adresses IP et nom DNS unique pour un ensemble de pods, et peut équilibrer la charge entre eux.
  -> monter automatiquement le système de stockage de notre choix, que ce soit à partir du stockage local, d'un fournisseur de cloud public, ou un système de stockage réseau.
- Kubernetes est de base créé par Google, mais maintenant, c'est un projet open source CNCF(=Cloud Native Computing Foundation) diplômé.

- Kubernetes va orchestrer et gérer des conteneurs sur des clusters de serveurs en automatisant les services de containers.

Architecture de Kubernetes : (Vu par les 3 personnes)

(on prend ici l'exemple de Docker pour gérer les conteneurs)
-le Kubernetes Master = serveur contrôlant les nodes.
-les nodes = noeuds esclaves, ce sont des machines hébergeant les Docker qui effectuent les tâches qui leur sont attribuées.
-les pods = tournent dans des nodes, c'est un environnement d'exécution d'un ou plusieurs conteneurs Docker. Tous les conteneurs d'un pod partagent les mêmes adresses IP et ports réseau.

- schéma détaillé (source : diaporama de Lucas Kaldstrom) :



  -à savoir : Kubelet est un composant exécuté sur des noeuds qui s'assure que les conteneurs démarrent et fonctionnent comme prévu à leur conception. Si un noeud est défaillant, c'est le composant Kubelet qui va le signaler au master.

Utilisez la syntaxe suivante pour exécuter des commandes kubectl depuis votre fenêtre de terminal :

`kubectl [commande] [TYPE] [NOM] [flags]`

où commande, TYPE, NOM et flags sont :

commande: Indique l'opération que vous désirez exécuter sur une ou plusieurs ressources, par exemple create, get, describe, delete.

TYPE: Indique le type de ressource. Les types de ressources sont insensibles à la casse et vous pouvez utiliser les formes singulier, pluriel ou abrégé :

`kubectl get pod pod1`

NOM: Indique le nom de la ressource. Les noms sont sensibles à la casse. Si le nom est omis, des détails pour toutes les ressources sont affichés, par exemple

`kubectl get pods.`

En effectuant une opération sur plusieurs ressources, vous pouvez soit indiquer chaque ressource par leur type et nom, soit indiquer un ou plusieurs fichiers :

- Pour indiquer des ressources par leur type et nom :
- - Pour regrouper des ressources si elles ont toutes le même type : TYPE1 nom1 nom2 nom<#>.
    Exemple: `kubectl get pod exemple-pod1 exemple-pod2`
- - Pour indiquer plusieurs types de ressources individuellement : TYPE1/nom1 TYPE1/nom2 TYPE2/nom3 TYPE<#>/nom<#>.
    Exemple: `kubectl get pod/exemple-pod1 replicationcontroller/exemple-rc1`
- - Pour indiquer des ressources avec un ou plusieurs fichiers : -f fichier1 -f fichier2 -f fichier<#>

Utilisez YAML plutôt que JSON, YAML tendant à être plus facile à utiliser, particulièrement pour des fichiers de configuration.
Exemple: `kubectl get pod -f ./pod.yaml`

flags: Indique des flags optionnels. Par exemple, vous pouvez utiliser les flags -s ou --server pour indiquer l'adresse et le port de l'API server Kubernetes.

-> on peut trouver beaucoup de commandes kubectl ci-dessous :
Source : https://kubernetes.io/fr/docs/reference/kubectl/overview/

Une commande assez utile existe pour avoir la description en détails d'un node d'une instance :
`kubectl describe nodes nom_instance`

- cette commande est en fait une version plus poussée de la commande suivante :
  `kubectl get nodes -o wide`
  qui permet de lister les nodes de toutes nos instances avec leurs status et adresses IP.

3) C'est quoi odoo ?

(Source : Wikipedia)
Odoo, anciennement OpenERP2 et Tiny ERP, est initialement un progiciel open-source de gestion intégré comprenant de très nombreux modules permettant de répondre à de nombreux besoins de gestion des entreprises (ERP), ou de gestion de la relation client (CRM). Le logiciel est utilisé par plus de deux millions d'utilisateurs pour gérer leurs entreprises à travers le monde. Odoo est le système ERP open-source le plus populaire.

Il existe une version community gratuite sous licence LGPLv3, et une entreprise sous licence propriétaire Odoo Enterprise Edition License v1.05.

À l'origine un progiciel de gestion intégré (ERP), le logiciel s'est vu étendre ses fonctionnalités à des applications de front office (CMS, e-commerce, blogs, forums, news, événements, live chat, job offers…).

Pour résumer nous allons utiliser Odoo comme API (=Application Programming Interface) pour la gestion du point de vente.

Partie des 3 personnes

4) L'avancé dans le projet

Depuis le début de notre projet nous avons commencé par faire des recherches sur ce qui était demandé. Nous avons étudié les logiciels puis établie une carte mentale. Après avoir fait ceci nous avons fais une liste des composants dont nous avions besoin pour ce projet. Dans cette liste nous avons besoin de :

- 6 raspberry Pi si nous voulons faire de la haute disponibilité (sinon 4 suffisent)
- 1 switch pour relier nos raspberry Pi
- Des câbles Ethernet pour la connexion

Au niveau logiciel nous avons besoin de :

- Kubernetes (k3s)
- k3sup pour la gestion automatique des containers
- odoo pour la gestion du PoS

Jusqu'à maintenant nous n'avons travaillé que sur des machines virtuelles avec multipass et nous avons seulement installé les OS Debian dans les raspberry.

Qu'est-ce que Multipass ?

C'est un programme très simple d'utilisation en lignes de commandes, qui permet de créé des pods (=un équipement réseau virtualisé) dans des nodes (=noeuds) qui sont containerisés dans des vm créées par multipass.
Cela permet de ne pas utiliser les ressources de notre machine physique.

- quelques commandes de base :
  multipass find = lister les images disponibles avec multipass
  multipass list = lister les instances
  multipass launch nom_vm = lancer une vm
  multipass shell nom_instance = se connecter sur la console de l'instance
  multipass stop nom_instance = stopper l'instance
  kubectl get nodes = lister les nodes
  kubectl get pods = lister les pods
  kubectl cluster-info = voir l'état du cluster

Partie de Samuel

Installation des VMs Multipass + k3s (prérequis)

Git : https://github.com/kubernauts/multipass-k3s-rancher

Pour commencer, avant déployer nos VMs, nous devons générer une clé RSA dans notre répertoire personnel (ex: test) pour pouvoir accéder à nos VMs plus facilement :

```
ssh-keygen -t rsa
Enter file in which to save the key (/home/test/.ssh/id_rsa):
Garder le nom des clés par défaut
On entre notre mot de passe secret : mdpincroyable
On le rentre encore une fois pour éviter toute erreur
```

Maintenant il faut installer la commande kubectl pour pouvoir vérifier le fonctionnement des VMs en fin de manipulation :

```
curl -LO https://storage.googleapis.com/kubernetes-
release/release/v1.15.0/bin/linux/amd64/kubectl

chmod +x ./kubectl

mv kubectl /usr/local/bin/
```

Nous allons éditer le script 1-deploy-multipass-vms.sh de manière à ne pas corrompre l'installation de nos VM :

```
# Create containers
for NODE in ${NODES}; do multipass launch --name ${NODE} --cpus 1 --mem 1G --
disk 10G; done
```

Vérifier que le nombre de cpu en notre possession et adapter si nécessaire et pareil pour la RAM. On peut installer htop via apt ou autre enrobeur afin de vérifier cela (commande: htop).

```
for NODE in ${NODES}; do
multipass transfer hosts ${NODE}:
multipass transfer ~/.ssh/id_rsa.pub ${NODE}:
multipass exec ${NODE} -- sudo iptables -P FORWARD ACCEPT
multipass exec ${NODE} -- bash -c 'sudo cat /home/multipass/id_rsa.pub >>
/home/multipass/.ssh/authorized_keys'
multipass exec ${NODE} -- bash -c 'sudo chown multipass:multipass /etc/hosts'
multipass exec ${NODE} -- bash -c 'sudo cat /home/multipass/hosts >>
/etc/hosts'
done
```

On vérifie que la clé publique tranféré à nos nodes soit bien ~/.ssh/id_rsa.pub.
Dans notre example on la remplace par /home/test/.ssh/id_rsa.pub.
Il faut nous assurer que la VM porte bien pour user multipass. Si ce n'est pas le cas il faut le modifier ou le créer (voir en lançant une des quatres VMs).

En ce qui concerne le script create-hosts.sh:

```
NODES=$(echo node{1..4})
for NODE in ${NODES}; do
# multipass exec ${NODE} -- bash -c 'echo `ip -4 addr show enp0s2 | grep -oP "
(?<=inet ).*(?=/)"` `echo $(hostname)` | sudo tee -a /etc/hosts'
multipass exec ${NODE} -- bash -c 'echo `ip -4 addr show enp0s2 | grep -oP "(?
<=inet ).*(?=/)"` `echo $(hostname)`'
# multipass exec ${NODE} -- bash -c 'echo `ip -4 addr show $nic_name | grep -oP
"(?<=inet ).*(?=/)"` `echo $(hostname)`'
# on CentOS linux on some machines the nic is named ens3
# multipass exec ${NODE} -- bash -c 'echo `ip -4 addr show ens3 | grep -oP "(?
<=inet ).*(?=/)"` `echo $(hostname)`'
done
```

Nous devons nous assurer que l'interface des VMs multipass est bien enp0s2, si ce n'est pas le cas à changer. (à regarder sur une seul VM, les autres porteront la même interface théoriquement)

Ensuite on peut lancer le script 1-deploy-multipass-vms.sh et on vérifie que tout se soit fait correctement :

```
./1-deploy-multipass-vms.sh
```

Si on veut supprimer toutes nos VMs multipass sans problème on peut utiliser la commande suivante (directement dans la console) et en même temps on peut prendre une bonne pause café:

```
for node in $(sudo multipass list|awk '{print $1}'|grep -v "Name"); do sudo
multipass delete $node; done && sudo multipass purge
```

On pourra ssh nos VMs multipass de la manière suivante grâce à la résolution de l'hôte:

```
ssh node1 -l (hôte de la VM)
Rentre le mdp secret
```

Maintenant que les VMs sont prêtes il faut déployer kubernetes dans chacunes d'entre elles. Pour se faire on utilise le script deploy k3s :

```
./8-deploy-only-k3s.sh
```

Une fois les VMs bien configurées et k3s installé on devrait pouvoir vérifier leur état sur la machine physique :

```
root@213-10:/home/test/multipass-k3s-rancher# kubectl get nodes
NAME    STATUS   ROLES    AGE     VERSION
node1   Ready    master   6m      v1.17.2+k3s1
node2   Ready    node     5m25s   v1.17.2+k3s1
node3   Ready    node     3m52s   v1.17.2+k3s1
node4   Ready    node     2m49s   v1.17.2+k3s1
```

Si cela ne fonctionne pas c'est qu'il faut exporter KUBECONFIG au bon endroit (vers le fichier k3s.yaml) :

```
export KUBECONFIG="/home/test/multipass-k3s-rancher/k3s.yaml"
```

Partie de Loïc

Installation d'odoo

Maintenant que le cluster est fonctionnel on peut installer Odoo. Il s'installe de la façon suivante (https://hub.kubeapps.com/charts/bitnami/odoo) :

```
helm repo add bitnami https://charts.bitnami.com/bitnami
helm install my-release bitnami/odoo
```

On ajoute le repository de la collection de chart de Bitnami afin que l'on puisse installer Odoo.

Par défaut Odoo va s'installer en mode LoadBalancer.

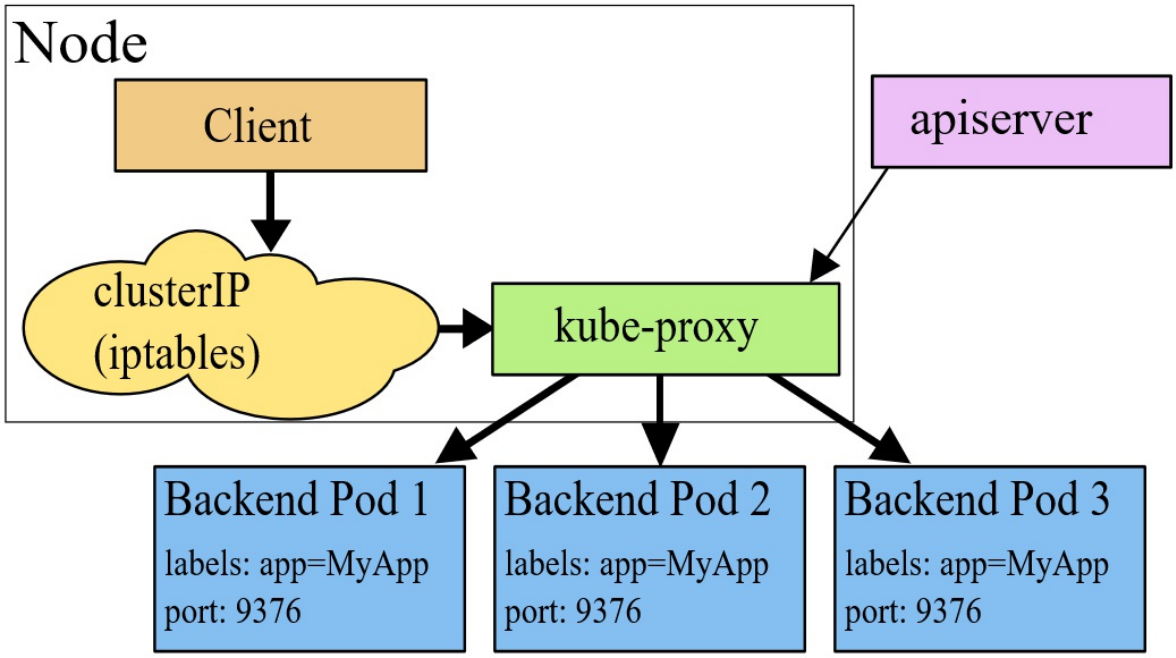
C'est un type d'architecture réseau qui va influer sur l'accès du service suite à l'installation du chart.

Chart: Regroupes des services applicatifs (Odoo et PostgreSQL (gestionnaire de base de données) dans notre cas).

Ce chart va installer ce que l'on appelle des pods qui sont un groupe de plusieurs containers qui vont se lancer.
Il faut savoir que nous pouvons avoir plusieurs pods par service et également par chart.
Chaque pod a sa propre adresse ip.

On a 3 types d'accès différents pour un service :

- **ClusterIP**

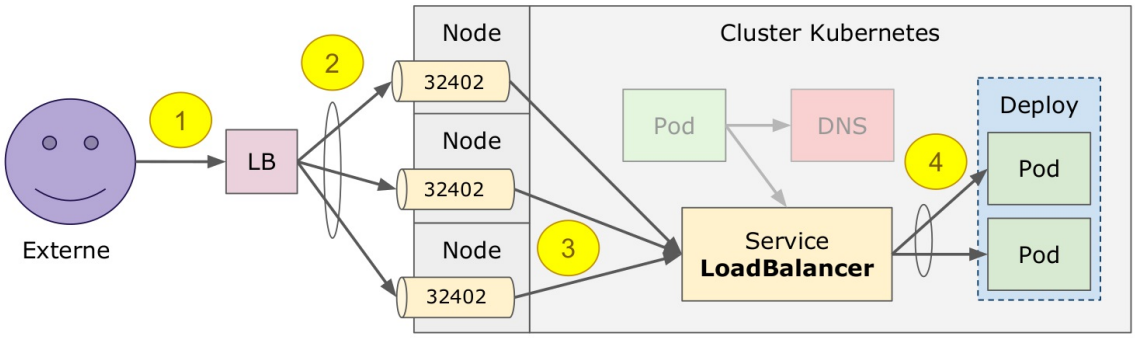


  Chaque node lance kube-proxy.
  kube-proxy correspond à l'implémentation d'une forme d'ip virtuel pour nos services.
  Pour chaque service, cela va installer des règles iptables, qui capturent le trafic vers l'ip et le port du service, et redirigent ce trafic vers l'un des ensembles principaux du service.
  Ce type de service est expose l'ip interne du cluster.
  Notre accès au service aura la forme suivante : localhost:port_redirigé

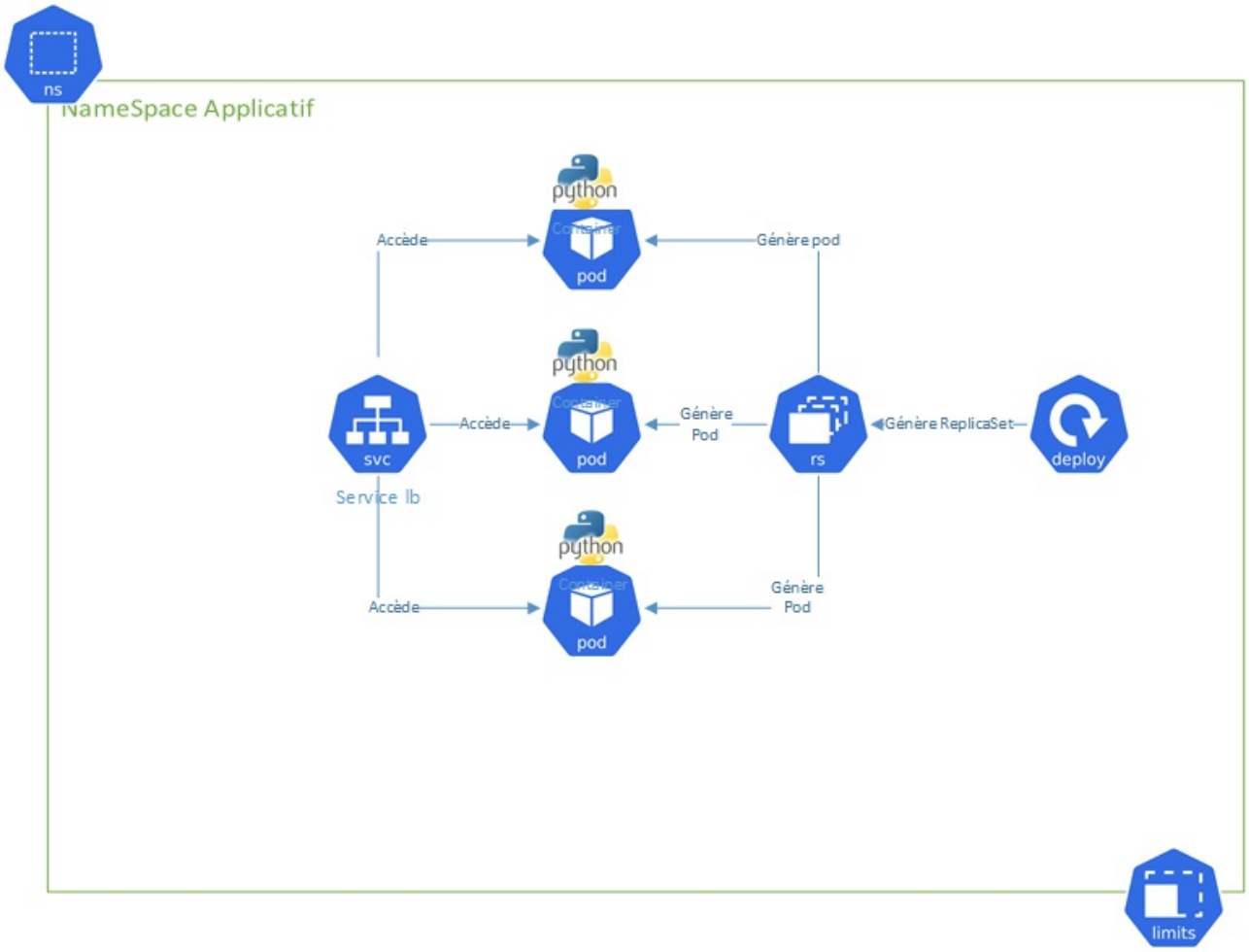
- **NodePort**
  Le controle plane va allouer un port identique sur chaque node compris dans l'intervalle 30000-32767 (= par défaut). L'intervalle de port peut être spécifié via l'argument --service-node-port-range. Pour spécifier des adresses particulières pour nos nodes, on aura l'argument --nodeport-addresses.

  Il va exposer le service sur chaque node avec une adresse ip différente et un port statique pour chacune d'entre elles.
  On peut contacter le service NodePort depuis l'extérieur du cluster, en demandant NodeIP:NodePort.

- **LoadBalancer**



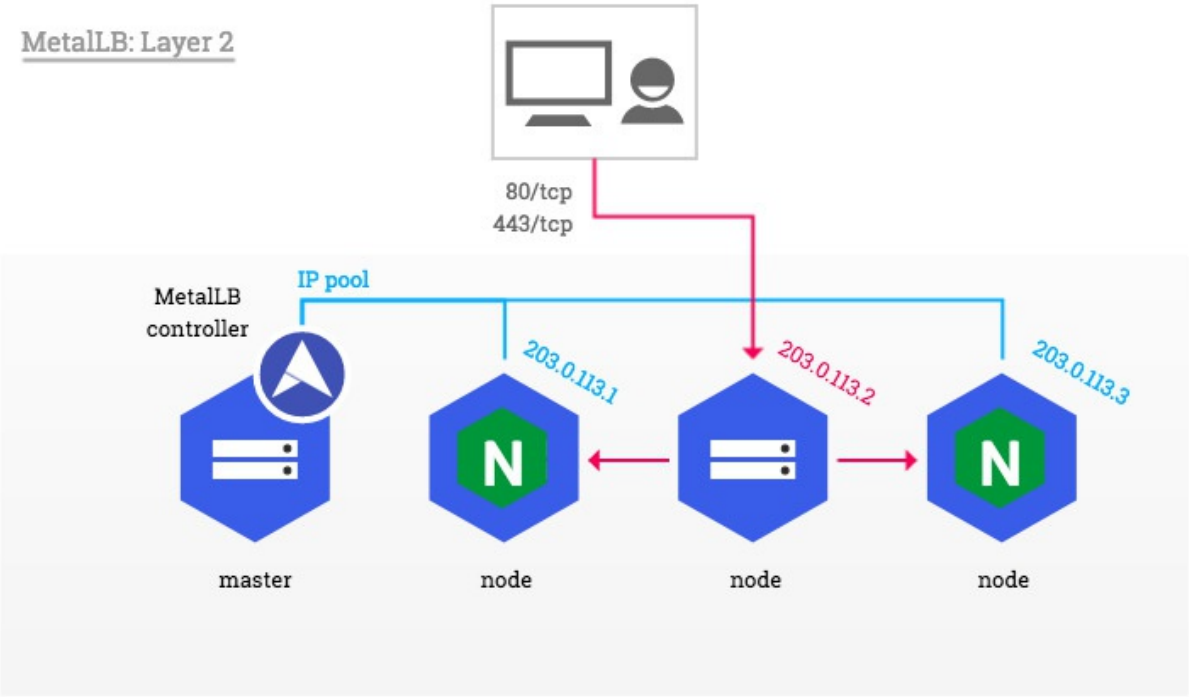
Le LoadBalancer va être équiper de ce que l'on appelle un Cloud Provider.
Cela permet de générer un pool d'adresse IP au service utilisant le type LoadBalancer.
Chaque service en LoadBalancer se verra attribuer une IP externe qui sera accessible directement depuis le réseau dans lequel l'on est (si celui-ci comporte les mêmes caractéristiques que le pool d'adresse IP (CIDR)).
On pourra donc accéder à notre service de la manière suivante : ip_externe:80.



Un namespace quant à lui sert à supporter de multiple cluster virtuel sur un même cluster physique.
On peut avoir plusieurs services sur un même namespace comme dans ce cas présent où l'on aura Postgres et Odoo.

Le ReplicaSet va voir pour but de maintenir les pods de manière stable en cours d'exécution à n'importe quel moment. Il est utilisé pour garantir la disponibilité d'un nombre spécifié de pods identiques. Il va servir à générer nos pods.

Principe de fonctionnement de metallb.yml :



Metallb va nous servir de cloud provider et va venir attribuer une adresse ip décrit dans le pool d'addresse mentionné dans le fichier de configuration metallb.yml.
Il va donner des adresses externes aux nodes du cluster (qui utilise le service de type LoadBalancer). Metallb joue un rôle majeur sur la couche 2 (Liaison), ce qui permet au client de se connecter au service s'il est dans le même réseau que celui préciser dans le pool d'ip.
Le service va être implémenté en annonçant que l'adresse de couche 2 (MAC) correspondant à l'ip externe va être l'adresse MAC de la node.

Pratique:
Dès lors que notre cluster est bien construit grâce à nos scripts, nous pouvons procéder à l'installation de Odoo.
Celui-ci nécessite l'installation de Helm qui est un paquet spécialement utilisé pour installer des services sur un cluster.
On peut l'installer sur notre machine physique en téléchargeant le binary.
(https://helm.sh/docs/intro/install/)

```
tar -zxvf helm-v3.0.0-linux-amd64.tar.gz
mv linux-amd64/helm /usr/local/bin/helm
```

(Penser à exporter la KUBECONFIG avant l'installation de chart)

Nous devons maintenant installer metallb de la manière suivante :
On applique le manifest de metallb.

```
kubectl apply -f
https://raw.githubusercontent.com/google/metallb/v0.8.3/manifests/metallb.yaml
```

On génère un fichier de configuration metallb.yml qui contient les lignes suivantes et on change notre pool d'adresse en fonction du réseau dans lequel on se trouve.
Personnellement, je suis en /24 dans mon réseau local donc il faut que le pool soit de 192.168.1.150-192.168.1.200.

```
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    address-pools:
    - name: default
      protocol: layer2
      addresses:
      - 192.168.1.150-192.168.1.200
```

Une fois qu'il est généré, on peut l'appliquer au cluster.

```
kubctl apply -f mettalb.yml
```

On peut installer notre chart Odoo en utilisant la commande suivante :

```
helm install my-odoo stable/odoo
```

On peut voir si le service a pris la configuration en compte (metallb):

```
kubectl get svc --namespace default -w my-release

NAME       TYPE           CLUSTER-IP     EXTERNAL-IP    PORT(S)        AGE
my-release   LoadBalancer   10.96.44.100  192.168.1.150   80:31615/TCP   2m22s
```

On pourra accéder via un navigateur web au service de la manière suivante :
http://192.168.1.150/

Il n'y a plus qu'à se loguer:

```
EMAIL
user@example.com
MDP
echo Password: $(kubectl get secret --namespace default my-release -o
jsonpath="{.data.odoo-password}" | base64 --decode)
```