[BROUILLON] Installation infra kapsule 2020

Avant de commencer

Quelques petites choses à réfléchir avant de se lancer. Quelle version de moteur network Kubernetes, quel OS etc.

En vrac :

- Choisir son moteur network : https://kubernetes.io/fr/docs/concepts/services-networking/service/
- Matrice de compatibilité rancher 2.4.2 https://rancher.com/support-maintenance-terms/all-supported-versions/rancher-v2.4.2/
- La dernière LTS ubuntu en ce mois de mai 2020 https://wiki.ubuntu.com/Releases et à ce jour c'est la version 20.04

Installation kapsule

Dans **l'interface web scaleway** https://console.scaleway.com, aller dans la partie **kapsule** et créer son cluster kubernetes.

Les infos à saisir :

- Nom k8s-01 dans mon cas. Evidemment, mettez ce que vous voulez
- Les deux points suivant sont dans les options avancées du cluster kubernetes
- Pas de *kubernetes dashboard* car je prévois d'installer *rancher 2* qui fournit son propre dashboard
- Pas de `ingress` car je prévois d'installer le mien, à ma façon
- *A ce stade, on peut se poser la question de rattacher lecluster à un network privé, je choisis de ne pas le faire surtout que ce sera payant à un moment*
- Choisir la version de Kubernetes 1.18 dans mon cas
- Choix de 3 noeuds `DEV-M` pour limiter le cout tout en ayant quand même 3 noeuds
- Pour le moteur réseau, choix de flannel car plus performant (plus light en ressources pour mes petits DEV-M) et pas besoin de grosse sécurité

Installation serveur d'admin

Création du serveur

J'ai un PC Windows et je n'ai pas envie de devoir installer sur mon poste perso tout l'outillage pour piloter mon cluster Kubernetes. Ceci d'autant plus que je peux être amené à devoir intervenir sur le serveur sans avoir mon PC perso sous la main.

Je prévois donc de déployer en plus un simple **serveur linux** `DEV-S` chez scaleway avec tout l'outillage d'amdin installé dessus.

Pour cela, go sur la console web scaleway et ajout d'une instance avec les informations ci-dessous.

Les infos que j'utilise pour l'installation :

- Nom scw-k8s-adm-01 mais choisissez ce que vous voulez
- Ubuntu LTS 20.04

Un peu d'attente et ensuite il est possible de s'y connecter en SSH.

[INFO] On suppose ici que la partie génération de clé publique / privée pour l'accès SSH a déjà été fait et ajouté dans l'admin scaleway pour être associé au serveur déployé.

Première connection & Maj system

On se connecte en SSH avec par exemple `putty` sans oublier d'avoir préchargé sa clé dans `pagent`.

Puis, comme on est bien élevés, avant toute chose on commence par mettre à jour le système.

```
apt-get update
apt-get upgrade -y
```

Install kubectl

A ce stade, il s'agit d'installer sur le serveur l'outil `kubectl`, qui nous permettra de piloter le cluster Kubernetes en ligne de commande.

La documentation de référence :
https://kubernetes.io/fr/docs/tasks/tools/install-kubectl/#installer-kubectl-sur-linux

```
apt-get update && apt-get install -y apt-transport-https
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add
-
echo "deb https://apt.kubernetes.io/ kubernetes-xenial main" | tee -a
/etc/apt/sources.list.d/kubernetes.list
apt-get update
apt-get install -y kubectl
```

Install du fichier kubeconfig

Pour pouvoir piloter le cluster kubernetes, il faut le fichier `kubeconfig` qu'on va ensuite installer sur le serveur d'admin.

Rendez-vous dans l'interface d'admin **kapsule** scaleway pour le téléchargement du kubeconfig.

Puis installation sur le serveur après téléchargement :

```
mkdir ~/.kube
cp kubeconfig-k8s-01.yaml  ~/.kube/config
```

Pour vérifier que tout fonctionne on essaye d'obtenir les infos du cluster kapsule :

```
kubectl cluster-info
```

Install helm

Après avoir regardé un peu, il m'a semblé que le plus simple d'installation et de mise à jour dans le temps consiste à utiliser `snap`.

```
snap install helm --classic
```

Ce qui est amusant et qui n'est pas dit de manière évidente c'est que le repo helm `stable` de base n'est pas dispo out of the box. Il faut donc installer le repo. J'ai trouvé les informations ici : https://github.com/helm/charts/issues/19279

```
helm repo add stable https://kubernetes-charts.storage.googleapis.com
```

Configuration du cluster kubernetes

[IMPORTANT] A ce stade vous devez savoir que, pour des raisons de cout notamment, j'ai choisi de ne déployer **qu'un seul ingress / loadbalancer**. J'y reviens un peu plus tard. Cela implique donc des configuration séparées, ou je déploie mes back-end d'un côté et met à jour mon ingress manuellement en central de l'autre.

Digression sur la récupération de l'ip client source

[INFO] Cette étape est clairement optionelle et ne fait pas partie de l'installation en tant que tel. J'ai préféré la noter car à un moment c'est ce qui m'a aidé pour déboguer et résoudre mes soucis d'ip client source qui n'allait pas jusqu'à mon backend wordpress.

J'avoue que cette documentation m'a été utile :
https://docs.ovh.com/gb/en/kubernetes/getting-source-ip-behind-loadbalancer/

Aussi bien pour comprendre un peu mieux ce qu'il y avait à faire mais aussi pour avoir les moyens de tester sans trop me prendre la tête.

[IMPORTANT] A noter ici que si vous suivez tout ce qui est dans ce guide vous déploierez un autre ingress. Encore une fois cette partie n'avance pas la configuration en tant que tel mes ses conclusions sont intégrées dans les autres parties de ce guide.

Tester avec ce guide et des ingress à part, c'est une chose que j'ai d'ailleurs faite au début car je pensais que mon *impossibilité* d'avoir l'ip source réelle (oui c'était une vrai guerre) venait de ma façon de déployer l'ingress. Ce qui n'était pas tout à fait vrai. Oui il y en avait un peu à cet endroit mais aussi a d'autres...

Digression sur la configuration SSL de l'ingress

[IMPORTANT] Cette section est à vocation explicative principalement car elle a été intégrée à la configuration ingress nginx citée par ailleurs dans ce guide.

Chose intéressante, j'ai remarqué qu'avec internet explorer 11 *(oui je sais c'est mal mais parfois on a pas le choix)*, mon blog n'était pas accessible en https pour une raison obscure. Après enquête *(pas facile, j'ai du déduire un peu)*, il s'avère que c'est parceque les paramètres TLS et les **algorithmes fournis de base par ma configuration TLS serveur son trop récents** et non pris en compte par ce vieux ie 11 (et d'autres config de là même époque).

Pour y voir plus clair j'avoue que https://www.ssllabs.com a été d'une grande aide pour avoir des infos sur la compatibilité et ce que mon serveur sait faire.

Quelques liens utiles:

- https://www.linode.com/docs/web-servers/nginx/tls-deployment-best-practices-for-nginx/
- https://github.com/kubernetes/ingress-nginx/blob/master/docs/user-guide/nginx-configuration/annotations.md#ssl-ciphers
- https://www.ssllabs.com/ssltest/analyze.html?d=alban.montaigu.io

Donc pour améliorer la compatibilté il suffit d'ajouter 1 ou deux `ciphers` *weak* qui sont pris en charge par les vieux navigateurs :

```
ssl-protocols: "TLSv1.2 TLSv1.3"
ssl-ciphers: "EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH;"
```

A noter que si vous voulez éviter de forcer le tls partout sur vos déploiement web quand ya des soucis de conf / compat mieux vaut désactiver la redirection *(qui se fait par défaut je crois)* :

```
ssl-redirect: "false"
```

Sinon les suites cipher ci-dessous (weak) ne sont pas dispo et ne premettent plus de supporter les vieilles config comme justement notre ami ie11.

```
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 (0xc028)   ECDH x25519 (eq. 3072 bits
RSA)   FS   WEAK    256
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)   ECDH x25519 (eq. 3072 bits
RSA)   FS   WEAK
```

Les commandes ci-dessous peuvent être utiles pour debug un peu le TLS *(a adapter en fonction de votre domaine bien sur)* :

```
curl -v --tlsv1.3 -s https://alban.montaigu.io/wp-json > /dev/null
curl -v -s http://alban.montaigu.io/wp-json > /dev/null
```

Installation & configuration de l'ingress

Après toutes ces **digressions** voici ce que cela donne.

La documentation que j'ai utilisée et qui m'a servi à un moment ou un autre :

- https://kubernetes.github.io/ingress-nginx/deploy/#using-helm
- https://www.digitalocean.com/community/tutorials/how-to-set-up-an-nginx-ingress-on-digitalocean-kubernetes-using-helm

- https://hub.helm.sh/charts/nginx/nginx-ingress
- https://github.com/kubernetes/ingress-nginx/blob/master/charts/ingress-nginx/values.yaml
- https://www.asykim.com/blog/deep-dive-into-kubernetes-external-traffic-policies
- https://www.ovh.com/blog/getting-external-traffic-into-kubernetes-clusterip-nodeport-loadbalancer-and-ingress/

[IMPORTANT] La configuration du paramètre `externalTrafficPolicy` doit être à `Local` pour éviter les hop dans le cluster kubernet et préserver l'ip client source. En effet, il ne faut pas oublier que les loadbalancer / ingress vont être en front de vos applications et si vous voulez que l'ip client passe jusqu'à votre application web il va falloir faire quelques efforts de configuration en commencant par celui là **(ce n'est pas le seul malheureusement)**.

Préparation de l'installation du ingress avec helm :

```
helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
kubectl create namespace ingress-nginx
```

Ensuite avec le fichier de configuration suivant :

ingress_nginx_helm_values.yaml

```
## nginx configuration
## Ref: https://github.com/kubernetes/ingress/blob/master/controllers/nginx/configuration.md
##
controller:

  # Will add custom configuration options to Nginx https://kubernetes.github.io/ingress-nginx/user-guide/nginx-configuration/configmap/
  config: {
    proxy-connect-timeout: "30",
    proxy-read-timeout: "1800",
    proxy-send-timeout: "1800",
    proxy-body-size: "500m",
    use-proxy-protocol: "true",
    proxy-real-ip-cidr: "51.159.75.81/32",
    compute-full-forwarded-for: "true",
    use-forwarded-headers: "true",
    ssl-redirect: "true",
    ssl-protocols: "TLSv1.2 TLSv1.3",
    ssl-ciphers: "EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH;"
  }

  ## Allows customization of the source of the IP address or FQDN to report
  ## in the ingress status field. By default, it reads the information provided
  ## by the service. If disable, the status field reports the IP address of the
```

```
  ## node or nodes where an ingress controller pod is running.
  publishService:
    enabled: true

  service:

    annotations: {
    #  service.beta.kubernetes.io/scw-loadbalancer-send-proxy-v2:
"true"
    }

    ## Set external traffic policy to: "Local" to preserve source IP on
    ## providers supporting it
    ## Ref:
https://kubernetes.io/docs/tutorials/services/source-ip/#source-ip-for-
services-with-typeloadbalancer
    externalTrafficPolicy: Local
```

[INFO] `51.159.75.81` correspond à l'ip du loadbalancer. Si elle n'est pas connue à l'avance, commencer par `0.0.0.0/32` puis la mettre à jour ensuite. C'est assez important car c'est ce paramètre qui va dire a nginx quel serveur continent la valeur ip client source a retenir dans les headers proxy.
[IMPORTANT] A ce state vous pouvez choisir d'activer ou non le **proxy protocol** et tout ce qu'il va avec. Néanmoins, dans mes expériences c'est un choix qui prête à conséquence. En effet, l'activer permettra à vos applications d'avoir à terme l'**ip source client** *(par exemple les modules de stats wordpress)*. Le **problème** de ca c'est que de ce que j'ai pu constater, activer cela faisait à minima planter certaines résolutions réseau “locales”.

Je m'explique, si jamais vous donnez un domaine associé a l'ip de l'ingress / loadbalancer pour pointer sur vos applications, j'ai constaté que les `curl` vers ce domaine depuis n'importe quel conteneur du cluster ne fonctionnaient pas *(je m'en suis rendu compte car l'outil de **santé wordpress** utilise curl pour tester le loopback et ce dernier plantait lamentablement)*.

Je n'ai pas réellement trouvé de solution à vrai dire. La seule chose que j'ai constaté c'est que ce problème n'arrive pas quand on laisse le ingress / loadbalancer **en mode tcp de base sans proxy**.

On déploie l'ingress avec les bonnes options de configuration :

```
helm install ingress-nginx ingress-nginx/ingress-nginx -f
ingress_nginx_helm_values.yaml -n ingress-nginx
```

A noter qu'il est possible aussi d'appliquer des configurations **à postériori** comme décrit ci dessous.

Changer la configuration de l'ingress

[IMPORTANT] Cette étape peut être **optionelle** grâce aux helm values du fichier `ingress_nginx_helm_values.yaml` mais c'est bon à savoir que vous pouvez l'utiliser s'il faut changer quelque chose dans la configuration de votre ingress.

Un exemple du fichier de configuration en question :

ingress_nginx_config.yaml

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: ingress-nginx-ingress-nginx-controller
  namespace: ingress-nginx
data:
  proxy-connect-timeout: "30"
  proxy-read-timeout: "1800"
  proxy-send-timeout: "1800"
  proxy-body-size: "500m"
  use-proxy-protocol: "true"
  proxy-real-ip-cidr: "51.159.75.81/32"
  compute-full-forwarded-for: "true"
  use-forwarded-headers: "true"
  ssl-redirect: "true"
  ssl-protocols: "TLSv1.2 TLSv1.3"
  ssl-ciphers: "EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH;"
```

Et la commande pour l'appliquer :

```
kubectl apply -f ingress_nginx_config.yaml
```

[IMPORTANT] Le déploiement d'ingress va créer automatiquement un **loadbalancer** associé chez scaleway. C'est assez important à savoir car de mémoire il en va d'environ 9 euros par mois il vaut donc mieux le savoir avant de déployer des ingress partout.

C'est d'ailleurs un problème de fond que je n'ai jamais vu sur le net. En effet, nombre de déploiements helm comportent un ingress intégré. Quand on est innocent, on déploie sans trop se poser de question. Sauf qu'à la fin on se rend compte qu'**on a déployé une 10aine de load balancer à 9 euros par mois**. Je ne sais pas pour vous mais pour moi, c'est vraiment trop cher !

Installation rancher

A ce stade, on doit avoir un cluster kubernetes de base configuré et bien sur le serveur d'admin qui nous a permis d'y arriver. Passons maintenant a l'installation de **Ranchger** qui permettra de gérer les déploiements d'application sur kubernetes. Cela permet aussi d'avoir un outil d'admin potable.

Les documentations de référence :

- https://rancher.com/docs/rancher/v2.x/en/installation/
- https://rancher.com/docs/rancher/v2.x/en/installation/k8s-install/helm-rancher/

On prépare les repo helm :

```
helm repo add rancher-stable https://releases.rancher.com/server-
charts/stable
helm repo add jetstack https://charts.jetstack.io
helm repo update
```

On crée les namespaces qui seront utiles plus tard :

```
kubectl create namespace cattle-system
kubectl create namespace cert-manager
```

[IMPORTANT] A ce stade, je fais une déviation par rapport à l'installation officielle rancher. En effet, pour l'installation de `cert-manager` j'ai préféré me référer à https://cert-manager.io/docs/installation/kubernetes/

Donc ne pas appliquer les comdandes suivantes :

```
# kubectl apply -f
https://raw.githubusercontent.com/jetstack/cert-manager/release-0.12/deploy/
manifests/00-crds.yaml
# helm install cert-manager jetstack/cert-manager --namespace cert-manager -
-version v0.12.0
```

On utilise plutot la méthode d'installation proposée par `cert-manager` *(la version peut être à mettre à jour) : <code bash> helm install cert-manager jetstack/cert-manager –namespace cert-manager –version v0.15.0 –set installCRDs=true </code> Et on vérifie que tout fonctionne bien comme cela par exemple : <code bash> kubectl get pods –namespace cert-manager </code>* ***On continue avec l'install de rancher*** (ici avec option letsEncrypt, à adapter avec vos infos à vous) *: <code bash> helm install rancher rancher-stable/rancher –namespace cattle-system –set hostname=k8s.montaigu.io –set ingress.tls.source=letsEncrypt –set letsEncrypt.email=alban.montaigu@gmail.com </code> Et on vérifie l'avancement du déploiement / que tout se passe bien avant de passer à la suite : <code bash> kubectl -n cattle-system rollout status deploy/rancher kubectl -n cattle-system get deploy rancher </code> Il n'est pas inutile non plus de vérifier les certificats `letsEncrypt` : <code bash> kubectl -n cattle-system describe certificate kubectl -n cattle-system describe issuer </code> <note info>* ***[INFO]*** *Au cas ou vous auriez besoin de reset le password rancher (parce que mal noté ou autre), utilisez la commande ci-dessous pour reset le mot de passe. </note> La commande en question pour reset le password : <code bash> kubectl -n cattle-system exec $(kubectl -n cattle-system get pods -l app=rancher | grep '1/1' | head -1 | awk '{ print $1 }') – reset-password </code> ===== L'installation de wordpress ===== ==== Installation du backend applicatif ==== La documentation / le contenu de référence : https://docs.bitnami.com/tutorials/deploy-custom-wordpress-production-helm/#step-1-define-configuration-values-for-the-bitnami-wordpress-helm-chart J'ai retenu le* ***chart bitnami*** *plutôt que celui proposé par rancher car il est plus récent. Et de toutes façons rancher s'inspire du chart bitnami. Donc a ce stade go dans la gestion des catalogues de rancher, ajouter la référence vers les charts bitnami et utiliser à la fin la fonction de launch app en choisissant `wordpress`. A noter qu'il faudra passer les bonnes valeurs de help dans le web* (soit en updloadant le fichier yaml, soit en le copian collant)*. D'ailleurs pourquoi via le web rancher plutôt qu'une simple commande helm ? Pour ma part parceque les applications déployées via rancher on des choses faites automatiquement* (un namespace, des éléments regroupement dans le web, des options d'upgrade, etc)*. Maintenant à chacun ses goûts :) ==== Configurer le (seul et unique) ingress pour causer avec le backend wordpress ==== Cette documentation comporte des informations qui peuvent être utiles :*

https://www.digitalocean.com/community/tutorials/how-to-set-up-an-nginx-ingress-with-cert-manager-on-digitalocean-kubernetes Et pour ceux qui se posent la question ***oui il est possible d'avoir un seul ingress qui envoie vers plusieurs backend dans plusieurs namespaces*** *: https://itnext.io/save-on-your-aws-bill-with-kubernetes-ingress-148214a79dcb <code bash> nano staging_issuer.yaml </code> kubectl create -f staging_issuer.yaml nano prod_issuer.yaml nano wordpress_ingress.yaml <file yaml wordpress_ingress.yaml> apiVersion: networking.k8s.io/v1beta1 kind: Ingress metadata: name: wordpress-ingress namespace: wordpress annotations: kubernetes.io/ingress.class: "nginx" cert-manager.io/cluster-issuer: "letsencrypt-prod" spec: tls: - hosts: - alban.montaigu.io secretName: wordpress-tls rules: - host: alban.montaigu.io http: paths: - backend: serviceName: wordpress servicePort: 80 </file> Commandes <code bash> kubectl create -f prod_issuer.yaml kubectl apply -f wordpress_ingress.yaml kubectl describe certificate wordpress-tls </code> Debug ingress <code bash> kubectl get ing -n wordpress kubectl describe ing wordpress-ingress -n wordpress </code> Adapt nginx Ingress configuration No more needed, since all is included in helm ingress definition. * https://stackoverflow.com/questions/54884735/how-to-use-configmap-configuration-with-helm-nginx-ingress-controller-kubernet * https://kubernetes.github.io/ingress-nginx/user-guide/nginx-configuration/annotations/#custom-max-body-size * https://github.com/kubernetes/ingress-nginx/issues/2007 Correction configuration pour x forwarded for https://github.com/kubernetes/ingress-nginx/issues/3529 <code> compute-full-forwarded-for: "true" use-forwarded-headers: "true" </code> nano nginx_ingress_config.yaml <file yaml nginx_ingress_config.yaml> kind: ConfigMap apiVersion: v1 metadata: name: nginx-ingress-ingress-nginx-controller namespace: default data: proxy-connect-timeout: "30" proxy-read-timeout: "1800" proxy-send-timeout: "1800" proxy-body-size: "500m" use-proxy-protocol: "true" proxy-real-ip-cidr: "51.159.26.229/32" compute-full-forwarded-for: "true" use-forwarded-headers: "true" ssl-redirect: "false" ssl-protocols: "TLSv1.2 TLSv1.3" ssl-ciphers: "EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH;" </file> <code bash> kubectl apply -f nginx_ingress_config.yaml </code> Il faut utiliser l'ip du load balancer pour proxy-real-ip-cidr, on ne peut pas rester générique avec du* `0.0.0.0/32` *sinon real ip n'est pas pris au bon endroit. Voir https://github.com/kubernetes/ingress-nginx/issues/1067. Namespace default to change probably Ajout catalogue chart bitnami dans rancher * Add catalog * Global catalog * Utiliser cette adresse https://charts.bitnami.com/bitnami Utiliser plutôt cette configuration que celle de rancher ! ===== Installation wordpress ===== Ne pas utiliser le wordpress rancher, utiliser le bitnami mais via catalog bitnami dans rancher pour avoir l'app Documentation * https://github.com/bitnami/charts/tree/master/bitnami/wordpress * https://bitnami.com/stack/wordpress/helm * https://github.com/bitnami/charts/tree/master/bitnami/wordpress/#installing-the-chart Configuration maison sur base Use https://github.com/bitnami/charts/blob/master/bitnami/wordpress/values.yaml Installation via rancher catalog app lunch avec ces values Note : helm install wordpress bitnami/wordpress -f wordpress_helm_values.yaml ==== Configuration wordpress ==== Divers * Import / export wordpress media , puis pages, puis articles séparés (après configuration ingress) et penser a associer ancien auteur to new one * Rechercher remplacer les url ancien site nouveau site dans les articles (https://fr.wordpress.org/plugins/better-search-replace/ * Trouble: Souci avec certaines images (a la racine de uploads dans la source et classée parfois avec un suffixe dans la dest) https://fr.wordpress.org/plugins/regenerate-thumbnails/ + reupload images manuellement ancien new site * Changer langue to fr * Change blog slogan * Configuration ecriture URL a renregistrer * Compte alban montaigu à remplir (nom prénom, bio si nécessaire) * Changer les widgets : nuage de tag * Widgets sites que j'aime bien et mes sites * Piracy policy * Reupload des images racine upload * Rechercher replacer http://vivihome.net https://alban.montaigu.io ==== Theme ==== * Liste à puce Choix du thème twenty seventeen avec image fond du robot blanc * Suppression des autres themes * Jeu de couleurs foncées ou autre ? Modification CSS pour largeur page adequate a mettre dans le custom css Largeur max 1500 au lieu de 100 Modification taille des*

*colonnes Largeur 26% au lieu de 36% Contenu CSS <code css> @media screen and (min-width: 48em) { .wrap { max-width: 1500px; } .has-sidebar:not(.error404) #primary { width: 70%; } .has-sidebar #secondary { width: 24%; } } </code> Reglages commentaires Desactiver tout Extensions * Activer Askimet * NE PAS Installer Apocalyspe Meow (sauf si mode proxy mais il fait planter d'autres truc comme site health) * Desinstaller All-in-One WP Migration * Desinstaller AMP * Bitnami Production Console Helper * Hello Dolly * Jetpack par WordPress.com * Activer All in One SEO Pack * Activer Simple Tags * ???? WP Mail SMTP * Desinstaller wordpress importer (plus besoin) Activer apocalype meow <code bash> kubectl exec -ti -n wordpress wordpress-5498fdfdd6-b5nxw – bash cp wp-config.php wp-config.php.back chmod +w wp-config.php.back echo “” » wp-config.php.back echo “* Fix the REMOTE_ADDR value (since not possible to simply fix it in httpd with docker image)” » wp-config.php.back echo “if (isset(\$_SERVER[\”HTTP_X_REAL_IP\“])) {” » wp-config.php.back echo “\$_SERVER[\”REMOTE_ADDR\“] = \$_SERVER[\”HTTP_X_REAL_IP\“];” » wp-config.php.back3 echo “}” » wp-config.php.back chmod -w wp-config.php.back </code>

Activer W3 Total Cache

Chmod 775 pour wp-config

A voir ici : https://github.com/bitnami/bitnami-docker-wordpress/issues/203

Corriger le REMOTE_ADDR (notamment pour apocalypse now)

- https://elwpin.com/2019/03/14/quick-fix-for-cloudflare-and-php-remote_addr-ip-detector/
- https://github.com/bitnami/bitnami-docker-wordpress/issues/178

```
if (isset($_SERVER["HTTP_X_REAL_IP"])) {
  $_SERVER['REMOTE_ADDR'] = $_SERVER["HTTP_X_REAL_IP"];
}
```

Changement DNS vivihome.net

- Ancienne adresse ip vivihome.net 163.172.154.80
- New redirection visible

Troubleshoot

- Les permaliens semblent sauter a chaque reboot: go admin enregistrer a nouveau conf permalien
- Apocalypse meow ne prend pas le xforwarded for

Commandes utiles en vrac

Excecuter un petit shell dans son conteneur wordpress à des fin de debug par exemple :

```
kubectl exec -ti -n wordpress wordpress-5498fdfdd6-hrhfp /bin/bash
```

Exectuer un conteneur oneshot pour avoir shell :

```
kubectl run my-shell --rm -i --tty --image ubuntu -- bash
```

Install JIRA / CONFLUENCE

Ajouter le catalogue helm mox dans rancher (via gestion des catalogues, ajout d'un catalogue helm 3) https://helm.mox.sh.

- https://hub.helm.sh/charts/mox/jira-software
- https://hub.helm.sh/charts/mox/confluence-server
- https://github.com/javimox/helm-charts
- https://github.com/javimox/helm-charts/releases
- https://github.com/javimox/helm-charts/tree/master/charts

Comme d'habitude pour avoir l'ip source du client qui redescend jusqu'à l'application (attention ce n'est pas le seul critère hélas), ne pas oublier de mettre le `externalTraficPolicy` à `Local`.

Le drame avec les ressources :

Ne pas oublier que les déploiements sont fait à l'origine sur des nodes `DEV-M` avec de mémoire `4Go` de RAM. Sachant que JIRA dans la config helm en requiert `2G` de réservés. Ces besoins assez importants qui devront être partagé avec les éventuels autres conteneurs déployés sur le node qui sera élu. Or, la litérature sur le net doit normalement le confirmer, java a du mal (moins avec les versions récentes) avec les conteneurs notamment sur la gestion de la mémoire. Et la le police kubernetes arrive avec du `OOMkill` ou autre.

C'est un peu ce qui a du m'arriver à ce stade, j'ai eu la joie d'avoir des https://sysdig.com/blog/debug-kubernetes-crashloopbackoff/ ou autres joyeusetées.

Autre problème à corriger le `liveness` et le `readiness` qu'il faut adapter car au premier démarrage jira et confluence mettent un temps infini a démarrer et se préparer (phase d'install). Donc par défaut kubernetes perds patience et des timeout se déclenchent pour voir finalement son conteneur redémarré par kubernetes sans que le démarrage soit terminé.

J'ai pu m'en sortir avec des adaptations de readiness et liveness plus **ajout d'un nouveau pool de nodes avec des conf à 8Go de RAM* sans oublier** des regles d'affinité **pour que les conteneurs jira et confluence aillent sur les nodes les plus généreux en RAM d'une part et d'autre part qu'ils ne soient pas déployés ensemble sur le même noeud (et ainsi se faire la guerre sur les ressources).** Conclusion **c'est là que je me suis arrêté et que j'ai renoncé pour être franc. Cela tournait mais pas de manière si rapide d'une part et d'autre part mon** porte monaie commencait à pleurer** car j'avais ajouté 2 noeuds à 8Go de RAM qui valent bien plus cher que mes noeuds de départ (que je devais conserver pour mes autres déploiements). Bref, trop cher, trop compliqué toujours pas fini, tant pis...

LEs trucs encore en vrac à formaliser

- Corriger le http://alban.montaigu.io dans la conf wordpress vers https ? Attention fait bug dans wp config et fait bug health check kubernetes
- Xforwarded for pour notamment akismet ?
  - Empeche apocalyspe meow de fonctionner et proablement les google analytics
  - https://github.com/kubernetes/ingress-nginx/issues/3529
  - A tester : https://github.com/bitnami/bitnami-docker-wordpress/issues/117

- https://docs.ovh.com/gb/en/kubernetes/getting-source-ip-behind-loadbalancer/
- https://atodorov.me/2019/11/21/getting-started-with-scaleways-managed-kubernetes-service-kapsule/
- https://github.com/nginxinc/kubernetes-ingress/blob/master/examples/proxy-protocol/README.md
- https://www.scaleway.com/en/docs/configure-proxy-protocol-with-a-load-balancer/