# **Installation d'un cluster Kubernetes**

Table des matières

[Déploiement dans Kubernetes 1](#_Toc529553467)

[Installation de Rancher et Kubernetes 1](#_Toc529553468)

[Installation manuelle d’un master Kubernetes et d’un Node séparé 3](#_Toc529553469)

[Configurer Kubernetes avec le Cloud-provider AWS 4](#_Toc529553470)

[*kube-dns is always pending because of weave net crash* 8](#_Toc529553471)

[Quelques commandes du CNI Weave 8](#_Toc529553472)

[Diverses commandes Kubernetes 9](#_Toc529553473)

[Reset complet de Kubernetes et des nodes 9](#_Toc529553474)

[Installation d’un Ingress Controller Kubernetes-Nginx 11](#_Toc529553475)

[Certificat auto-signé : 12](#_Toc529553476)

[Création d’une configuration de déploiement Kubernetes 17](#_Toc529553477)

## Installation manuelle d’un master Kubernetes et d’un Node séparés

L'installation est réalisée sur 2 instances EC2 : un master-node et un Worker-node (qui hébergera les les conteneurs des applications ainsi que l'Ingress).

Le workernode peut être une t2.micro mais le master devrait être une t3.small.

*(Ce modop fonctionne bien avec la version 1.12 de kubectl et kubeadm).*

1. Les commandes suivantes sont à exécuter **sur le master et sur le worker** afin de préparer l'environnement de la VM à Kubernetes :

$ apt-get update -y

$ swapoff -a // Doit être saisi à chaque démarrage de l’instance ?

$ hostnamectl set-hostname kube-master *// ou kube-worker*

$ vi /etc/hosts *//Y declarer les hostname des 2 serveurs*

$ apt-get install apt-transport-https ca-certificates curl software-properties-common -y

$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | apt-key add -

$ add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable"

$ apt-get update -y

$ apt-get install docker-ce -y

$ curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add -

$ echo 'deb http://apt.kubernetes.io/ kubernetes-xenial main' | sudo tee /etc/apt/sources.list.d/kubernetes.list

$ apt-get update -y

$ sysctl net.bridge.bridge-nf-call-iptables=1 // Pour le CNI wave.net // Doit être saisi à chaque démarrage de l’instance ? Fonctinne avec Ubuntu 16 mais pas Ubuntu 18 ?

$ apt-get install kubelet kubeadm kubectl -y

1. Sur le **master** uniquement :

*Optionnel : pour deployer des pods sur le master (non recommandé).*

*$ kubectl taint nodes --all node-role.kubernetes.io/master-*

Initialisation du Worker node :

$ kubeadm init *--pod-network-cidr=192.168.1.0/24*

Préparation du fichier de configuration Kubernetes pour la commande Kubectl :

$ mkdir -p $HOME/.kube

$ cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

$ chown $(id -u):$(id -g) $HOME/.kube/config

OU $ *mkdir -p $HOME/.kube; cp -i /etc/kubernetes/admin.conf $HOME/.kube/config; chown $(id -u):$(id -g) $HOME/.kube/config*

$ kubeadm token create --print-join-command --ttl 0

*Note : cette commande générera un token qui n'expirera pas ainsi qu'une commande 'kubeadm join' à exécuter sur le worker node pour le joindre au master. Il faut la noter pour l'utiliser sur le worker node un peu plus tard.*

Vérification de l'initialisation des composants du master :

$ kubectl get pods --all-namespaces

*-> Attendre que tout soit 'running', sauf le service CoreDNS qui reste à 'pending' sans CNI.*

Installation du CNI pour la gestion du réseau (ici, Weaver) :

$ kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=$(kubectl version | base64 | tr -d '\n')"

*Note : pour modifier le CIDR :*

*$ kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=$(kubectl version | base64 | tr -d '\n')&env.IPALLOC\_RANGE=10.32.0.0/24"*

$ kubectl get pods --all-namespaces

-> *Attendre que tout soit 'running'*, *y compris CoreDNS.*

*Optionnel : à tester ?*

*$ kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard/master/src/deploy/recommended/kubernetes-dashboard.yaml*

1. Joindre le premier **Worker node** avec la commande 'Join' notée précédemment *(commande à passer sur le Worker uniquement)* :

$ kubectl join (cf commande copiée précédemment lors du ‘kubeadm init’) :

1. Vérifier la déclaration du worker node (sur le **master** uniquement) :

$ kubectl get nodes

*-> Le master node et le worker doivent tous les 2 être 'ready'*

### Diverses commandes du CNI Weave

DEBUGG WEAVE :

$ kubectl get pods -n kube-system -o wide *// Affiche également sur quel node le conteneur weave tourne pour le debuger par la suite*

$ kubectl get pods -n kube-system -l name=weave-net

$ kubectl logs -n kube-system <POD Weave> weave

$ kubectl exec -n kube-system <POD Weave> -c weave -- /home/weave/weave --local status

$ kubectl logs <POD Weave> -n kube-system weave-npc

Voir les IPs assignées aux pods des applications :

$ kubectl get pods -o wide | grep nginx

Désinstaller Weave (à faire sur TOUS les noeuds) :

*$ weave reset // ? Requière un script weave ?*

*$ curl -L git.io/weave -o /usr/local/bin/weave*

*$ chmod a+x /usr/local/bin/weave*

$ ip link delete weave

$ rm /opt/cni/bin/weave-\*

*Supprimer également les images Docker de weave*

### Diverses commandes Kubernetes

Reset de Kubeadm :

$ kubeadm reset && rm -rf ~/.kube/ && kubeadm init --pod-network-cidr=172.18.1.0/24

Changer CIDR de kube-proxy :

*Cf* [*https://github.com/kubernetes/kops/issues/4674*](https://github.com/kubernetes/kops/issues/4674)

$ kubectl create -f api-javaspringboot-small-deployment.yaml

$ kubectl delete deployment nginx-deployment

Vérifier que la commande kubectl fonctionne et dialogue bien avec le serveur :

$ kubectl version

*Ne doit pas retourner d’erreurs de connexion*

Debuger un problème de déploiement de POD :

$ kubectl describe pod <ID du pod> -n kube-system

Obtenir le status d'un déploiement :

$ kubectl rollout status deployment.v1.apps/<nom déploiement>

Supprimer un déploiement et les pods déployés :

$ kubectl describe deployment <nom déploiement>

Exposer un déploiement via un service à la volée :

$ kubectl expose deployment hello-world --type=LoadBalancer --name=my-service

Vérifier les endpoints d’un service :

$ kubectl get endpoints <service>

Tenter de démarrer kubelet avec ces infos pour eviter les evictions de pod :

$ kubectl expose deployment hello-world --type=LoadBalancer --name=my-service

### Reset complet de Kubernetes et des nodes avec le CNI Weave

*A faire sur le master uniquement :*

# Drain and delete the nodes (for each node you have)

kubectl drain kube-master --delete-local-data --force --ignore-daemonsets

kubectl delete node kube-master

# Reset the deployment

kubeadm reset && rm -rf

*A faire sur chaque nœud (master et workers) :*

## Reset the nodes and weave

sudo curl -L git.io/weave -o /usr/local/bin/weave

sudo chmod a+x /usr/local/bin/weave

sudo kubeadm reset && rm -rf ~/.kube/

sudo weave reset --force

## Clean weave binaries

sudo rm /opt/cni/bin/weave-\*

## Flush iptables rules on all nodes and restart Docker

iptables -P INPUT ACCEPT

iptables -P FORWARD ACCEPT

iptables -P OUTPUT ACCEPT

iptables -t nat -F

iptables -t mangle -F

iptables -F

iptables -X

systemctl restart docker

*+ supprimer les images Docker Weave sur les nœuds master et worker*

## Ingress Controller Kubernetes-Nginx

### Installation de l'Ingress Controller

Selon le **tutoriel** suivant :

[*https://akomljen.com/kubernetes-nginx-ingress-controller/*](https://akomljen.com/kubernetes-nginx-ingress-controller/)

1. Créer un Namespace sur le KubeMaster dédié à l'Ingress Controller :

$ kubectl create namespace ingress

1. Créer un répertoire qui contiendra les fichier de déploiement de l'Ingress :

$ mkdir -p examples/nginx-ingress/

1. Création d'un 'Default-backend' : simple POD exécutant un Nginx (aucun rapport à avec l'Ingress controller) sur lequel l'Ingress Controller renverra toutes les requêtes dont aucune configuration Ingress n'est associée ('404 not found') :

***1-default-backend-deployment.yaml*** :

apiVersion: extensions/v1beta1

kind: Deployment

metadata:

name: default-backend

spec:

replicas: 1

template:

metadata:

labels:

app: default-backend

spec:

terminationGracePeriodSeconds: 60

containers:

- name: default-backend

image: gcr.io/google\_containers/defaultbackend:1.0

livenessProbe:

httpGet:

path: /healthz

port: 8080

scheme: HTTP

initialDelaySeconds: 30

timeoutSeconds: 5

ports:

- containerPort: 8080

resources:

limits:

cpu: 10m

memory: 20Mi

requests:

cpu: 10m

memory: 20Mi

$ kubectl create -f 1-default-backend-deployment.yaml -n=ingress

1. Création du service pour le 'Default-backend' : un simple service type 'ClusterIP' (l'IP n'est pas accessible en-dehors du Xluster Kubernetes) et qui renvoie toutes les requêtes destinées à ce default-backend vers les PODs l'exécutant :

***2-default-backend-service.yaml*** :

apiVersion: v1

kind: Service

metadata:

name: default-backend

spec:

ports:

- port: 80

protocol: TCP

targetPort: 8080

selector:

app: default-backend

$ kubectl create -f 2-default-backend-service.yaml -n=ingress

1. Création d'une config-map pour activer le VTS status (?) sur le controller Nginx-Ingress :

***3-nginx-ingress-controller-config-map.yaml*** :

apiVersion: v1

kind: ConfigMap

metadata:

name: nginx-ingress-controller-conf

labels:

app: nginx-ingress-lb

data:

enable-vts-status: 'true'

$ kubectl create -f 3-nginx-ingress-controller-config-map.yaml -n=ingress

1. Création des rôles RBAC nécessaires pour activer le status sur le contrôleur :

***4-nginx-ingress-controller-roles.yaml*** :

apiVersion: v1

kind: ServiceAccount

metadata:

name: nginx

---

kind: ClusterRole

apiVersion: rbac.authorization.k8s.io/v1beta1

metadata:

name: nginx-role

rules:

- apiGroups:

- ""

resources:

- configmaps

- endpoints

- nodes

- pods

- secrets

verbs:

- list

- watch

- apiGroups:

- ""

resources:

- nodes

verbs:

- get

- apiGroups:

- ""

resources:

- services

verbs:

- get

- list

- update

- watch

- apiGroups:

- extensions

resources:

- ingresses

verbs:

- get

- list

- watch

- apiGroups:

- ""

resources:

- events

verbs:

- create

- patch

- apiGroups:

- extensions

resources:

- ingresses/status

verbs:

- update

---

kind: ClusterRoleBinding

apiVersion: rbac.authorization.k8s.io/v1beta1

metadata:

name: nginx-role

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole

name: nginx-role

subjects:

- kind: ServiceAccount

name: nginx

namespace: ingress

$ kubectl create -f 4-nginx-ingress-controller-roles.yaml -n=ingress

1. Création du contrôleur Nginx-Ingress. Il recevra toutes les requêtes arrivant via son service (déployé à la prochaine commande dans ce modop) puis à partir de l'URL appelée dans la requêtes http par le client (ex : app1.maddoudou.click) il consultera toutes les configuration ingress des applications créées pour localiser le host correspondant et rediriger la requête ver le service de l'application cible. Le(s) POD(s) de l'Ingress Controlleur écoute sur le port 80.

***5-nginx-ingress-controller-deployment.yaml*** :

apiVersion: extensions/v1beta1

kind: Deployment

metadata:

name: nginx-ingress-controller

spec:

replicas: 1

revisionHistoryLimit: 3

template:

metadata:

labels:

app: nginx-ingress-lb

spec:

terminationGracePeriodSeconds: 60

serviceAccount: nginx

containers:

- name: nginx-ingress-controller

image: quay.io/kubernetes-ingress-controller/nginx-ingress-controller:0.9.0

imagePullPolicy: Always

readinessProbe:

httpGet:

path: /healthz

port: 10254

scheme: HTTP

livenessProbe:

httpGet:

path: /healthz

port: 10254

scheme: HTTP

initialDelaySeconds: 10

timeoutSeconds: 5

args:

- /nginx-ingress-controller

- --default-backend-service=$(POD\_NAMESPACE)/default-backend

- --configmap=$(POD\_NAMESPACE)/nginx-ingress-controller-conf

- --v=2

env:

- name: POD\_NAME

valueFrom:

fieldRef:

fieldPath: metadata.name

- name: POD\_NAMESPACE

valueFrom:

fieldRef:

fieldPath: metadata.namespace

ports:

- containerPort: 80

- containerPort: 18080

$ kubectl create -f 5-nginx-ingress-controller-deployment.yaml -n=ingress

1. Création de l'ingress de la page de status :

**6-nginx-ingress.yaml**:

kind: Ingress

metadata:

name: nginx-ingress

spec:

rules:

- host: status.kubernetes.maddoudou.click

http:

paths:

- backend:

serviceName: nginx-ingress

servicePort: 18080

path: /nginx\_status

$ kubectl create -f 6-nginx-ingress.yaml -n=ingress

1. Création du service du contrôleur Nginx-Ingress (type NodePort). Il écoutera toutes les requêtes arrivant sur les IPs publiques des Workers Nodes *EC2 sur le port 30000 (forcément un ou des port(s) compris entre 30000 et 32000) (le Kube-proxy des Worker Nodes redirigera les requêtes destinées au port 30000 vers ce service*).

**7-nginx-ingress-controller-service.yaml**:

apiVersion: v1

kind: Service

metadata:

name: nginx-ingress

spec:

type: NodePort

ports:

- port: 80

nodePort: 30000

name: http

- port: 18080

nodePort: 32000

name: http-mgmt

selector:

app: nginx-ingress-lb

$ kubectl create -f 7-nginx-ingress-controller-service.yaml -n=ingress

### Test de l'Ingress Controller

**Test de l’ingress-controller** via le déploiement d'applications de test.

1. Créer un répertoire qui contiendra les fichier de déploiement des applications de test :

$ mkdir -p examples/test-apps/

1. Création de 2 applications (simples serveurs Nginx). Des conteneurs écoutant sur le port 80 :

***1-apps-deployment.yaml*** :

apiVersion: extensions/v1beta1

kind: Deployment

metadata:

name: app1

spec:

replicas: 1

template:

metadata:

labels:

app: app1

spec:

containers:

- name: app1

image: dockersamples/static-site

env:

- name: AUTHOR

value: app1

ports:

- containerPort: 80

---

apiVersion: extensions/v1beta1

kind: Deployment

metadata:

name: app2

spec:

replicas: 1

template:

metadata:

labels:

app: app2

spec:

containers:

- name: app2

image: dockersamples/static-site

env:

- name: AUTHOR

value: app2

ports:

- containerPort: 80

$ kubectl create -f 1-apps-deployment.yaml

1. Création des services des 2 applications écoutant également sur le port 80 (type CLusterIP) :

***2-apps-service.yaml*** :

kind: Service

metadata:

name: appsvc1

spec:

ports:

- port: 80

protocol: TCP

targetPort: 80

selector:

app: app1

---

apiVersion: v1

kind: Service

metadata:

name: appsvc2

spec:

ports:

- port: 80

protocol: TCP

targetPort: 80

selector:

app: app2

$ kubectl create -f 2-apps-service.yaml

1. Création des ingress des 2 applications écoutant sur differetes URL afin de savoir comment rediriger le trafic sur les services des applications :

***3-app-ingress.yaml***:

kind: Ingress

metadata:

annotations:

nginx.ingress.kubernetes.io/rewrite-target: /

name: app-ingress

spec:

rules:

- host: apps.kubernetes.maddoudou.click

http:

paths:

- backend:

serviceName: appsvc1

servicePort: 80

path: /app1

- backend:

serviceName: appsvc2

servicePort: 80

path: /app2

- host: app1.kubernetes.maddoudou.click

http:

paths:

- backend:

serviceName: appsvc1

servicePort: 80

path: /

- host: app2.kubernetes.maddoudou.click

http:

paths:

- backend:

serviceName: appsvc2

servicePort: 80

path: /

$ kubectl create -f 3-app-ingress.yaml

1. Exécution des tests : exécuter la commande suivante *(note : le nœud sur lequelle la commande est exécutée doit pouvoir résoudre le nom DNS, soit via un serveur DNS (ex : Route 53), soit via le /etc/hosts)* :

$ curl -X GET http://app1.kubernetes.maddoudou.click

*-> Cette commande doit retourner le contenu de la page Nginx de l'application 1.*

## Déploiement automatisé de Kuberentes sur AWS

### Présentation

L'installation est réalisée sur 2 instances EC2 : un master-node et un Worker-node (qui hébergera les les conteneurs des applications ainsi que l'Ingress).

Le workernode est une t2.micro mais le master devrait être une t3.small (au choix de l'utilisateur).

*(Ce modop fonctionne bien avec la version 1.12 de kubectl et kubeadm).*

*Le workerNode et la master Node sont déployés via des AMI qui ont été créées suivant les commandes décrites dans le chapitre précédent.*

*Sur le master Node le controlleur Ingress Nginx est déployé mais les applications de test ne sont pas déployées. En rechanche leurs fichiers de déploiement sont disponibles dans le répertoire ~/examples/test-apps/ du master.*

### Pré-requis

Le déploiement est entièrement réalisé via un fichier CloudFormation, cependant certains doivent êtres créés au-préalable afin que le déploiement se déroule correctement :

* **VPC :**
* **Launch Configuration** :

### Fichier CloudFormation

Le déploiement est entièrement réalisé via un fichier CloudFormation.

**Full-Kubernetes-env(existingVPC)-CFN.yaml** :

AWSTemplateFormatVersion: '2010-09-09'

Description: Creation automatisee d'un autoscaling group et de son load-balancer pour repartir la charge sur des Worker Nodes Kubernetes (non deployes via ce template). Creation d'une wild-card DNS pour deleguer les sous-domaines a l'Ingress Kubernetes.

# Creation automatisee d'un Groupe Autoscaling pointant sur une Launch Configuration

# - L'autoscaling Group est configure pour ne demarrer qu'une seule instance de Worker Node Kubernetes - Aucune Scaling Policy n'est associee

# Création d'un target Group référancant l'autoscaling Group pour la réaprtition de charge

# -

# Creation du load-balancer avec un Listen configuration et pontant sur le target Group pour la repartition de la charge

# - Ecoute sur le port 80

# - Plage de ports ouverts sur le Security-Group du Load-Balancer en entrée : 80, 443, 8000-10000, IMCP

# Creation d'une wild-card DNS pour deleguer les sous-domaines a l'Ingress Kubernetes.

Parameters:

# VM Kubernetes master :

DomaineDNS:

Description: HostedZone de Route53 dans laquelle déclarer l'enregistrement DNS du kube-master

Default: "maddoudou.click."

Type: String

SousDomaineKubernetes:

Description: Le sous-domaine dedie a Kubernetes declare dans Route53 (ne pas oublier le "." a la fin)

Default: "\*.kubernetes.maddoudou.click."

Type: String

FQDNKubeMasterInstance:

Description: Le FQDN de l'instance kube-master declare dans Route53 (ne pas oublier le "." a la fin)

Default: "kubemaster.maddoudou.click."

Type: String

NomInstanceKubeMasterEC2:

Description: Nom de l'instance hebergeant le master Kubernetes.

Default: "kube-master-CFN"

Type: String

ImageKubeMaster:

Description: Image de l'instance hebergeant le master Kubernetes.

Type: AWS::EC2::Image::Id

Default: ami-0468de20342c49a99

TypeInstance:

Description: Le type d'instance pour le master Kubernetes (t3.small recommandee).

Type: String

Default: t3.small

AllowedValues: [ t2.micro, t3.small, t3.medium ]

Keypair:

Description: Keypair utilisee pour l'instance hebergeant le master Kubernetes.

Type: AWS::EC2::KeyPair::KeyName

Default: aws.ireland.keypair

# Autoscaling Group et Laod-balancer pour les worker nodes

NomLaunchConfiguration:

Description: Le nom de la "Launch Configuration" utilisee pour instancier les EC2 des Worker Nodes Kubernetes

Default: "KubeWorker-node"

Type: String

NomAutoScalingGroup:

Description: Le nom du groupe auto-scaling hebergeant les Worker Nodes Kubernetes

Default: "KubeWorker-AutoScalingGroup-CFN"

Type: String

NomLoadBalancer:

Description: Le nom du load-balancer repartissant la charge pour leGroup Autoscaling des Worker Nodes Kubernetes

Default: "KubeWorker-LB-CFN"

Type: String

NomTargetGroup:

Description: Le nom du load-balancer repartissant la charge pour leGroup Autoscaling des Worker Nodes Kubernetes

Default: "KubeWorker-TargetGroup-CFN"

Type: String

VPCKubernetes:

Description: Le VPC hebergeant le load-balancer et les instances Kubernetes. Doit être 172.32.x.x car l'image du master a ete creee dans ce subnet (doit etre accessible via Internet)

Type: AWS::EC2::VPC::Id

SubnetPublicKubeNodes:

Description: Subnet du VPC hebergeant le master et les worker nodes Kubernetes. Doit être 172.32.1.0/24 car l'image du master a ete creee dans ce subnet (accessible via Internet).

Type: AWS::EC2::Subnet::Id

SubnetsPublicLB:

Description: 2 Subnets du VPC hebergeant le load-balancer (car il necessite 2 subnets dans des AZ separees).

Type: List<AWS::EC2::Subnet::Id>

Resources:

#-----------------------------------------------------------------------------------------------

# Kube-master node -----------------------------------------------------------------------------

#-----------------------------------------------------------------------------------------------

KubemasterSecurityGroup:

Type: AWS::EC2::SecurityGroup

Properties:

VpcId: !Ref 'VPCKubernetes'

GroupDescription: Security Group AutoScalingGroup !Sub ${NomAutoScalingGroup}

SecurityGroupIngress:

- IpProtocol: icmp

FromPort: '8'

ToPort: "-1"

CidrIp: 0.0.0.0/0

- IpProtocol: tcp

FromPort: '22'

ToPort: '22'

CidrIp: 0.0.0.0/0

- IpProtocol: tcp

FromPort: '32000'

ToPort: '32767'

CidrIp: 0.0.0.0/0

- IpProtocol: udp

FromPort: '32000'

ToPort: '32767'

CidrIp: 0.0.0.0/0

- IpProtocol: tcp

FromPort: '80'

ToPort: '80'

CidrIp: 0.0.0.0/0

- IpProtocol: tcp

FromPort: '443'

ToPort: '443'

CidrIp: 0.0.0.0/0

- IpProtocol: tcp

FromPort: '10250'

ToPort: '10250'

CidrIp: 0.0.0.0/0

- IpProtocol: udp

FromPort: '8285'

ToPort: '8285'

CidrIp: 0.0.0.0/0

- IpProtocol: udp

FromPort: '8472'

ToPort: '8472'

CidrIp: 0.0.0.0/0

- IpProtocol: tcp

FromPort: '2379'

ToPort: '2380'

CidrIp: 0.0.0.0/0

- IpProtocol: tcp

FromPort: '6443'

ToPort: '6443'

CidrIp: 0.0.0.0/0

# Pour tests : A SUPPRIMER :

- IpProtocol: tcp

FromPort: '0'

ToPort: '65535'

CidrIp: 0.0.0.0/0

# Pour tests : A SUPPRIMER :

- IpProtocol: udp

FromPort: '0'

ToPort: '65535'

CidrIp: 0.0.0.0/0

# Pas besoin si VXLAN :

# - IpProtocol: tcp

# FromPort: '0'

# ToPort: '65535'

# CidrIp: 0.0.0.0/0

Tags:

- Key: Name

Value: !Sub ${NomInstanceKubeMasterEC2}-SG-CFN

- Key: CloudFormationStack

Value: !Ref 'AWS::StackId'

kubemasterInstance:

Type: AWS::EC2::Instance

Properties:

ImageId: !Ref ImageKubeMaster

InstanceType: !Ref TypeInstance

KeyName:

Ref: Keypair

NetworkInterfaces:

- AssociatePublicIpAddress: 'true'

DeviceIndex: '0'

SubnetId: !Ref 'SubnetPublicKubeNodes'

GroupSet:

- !Ref 'KubemasterSecurityGroup'

PrivateIpAddresses:

- PrivateIpAddress: '172.32.1.10' # L'image du master Kubernetes a été créée avec cette adresse, elle ne doit pas changer.

Primary: 'true'

UserData:

'Fn::Base64': !Sub |

#!/bin/bash

sudo swapoff -a

sudo sysctl net.bridge.bridge-nf-call-iptables=1

Tags:

- Key: Name

Value: !Sub ${NomInstanceKubeMasterEC2}

- Key: CloudFormationStack

Value: !Ref 'AWS::StackId'

# UserData:

# "Fn::Base64": !Sub |

# #!/bin/bash -x

# sudo XXXXXX

KubemasterDnsRecord:

Type: AWS::Route53::RecordSet

Properties:

HostedZoneName: !Ref 'DomaineDNS'

Comment: Enregistrement DNS pour !Sub ${NomInstanceKubeMasterEC2}

Name: !Ref 'FQDNKubeMasterInstance'

Type: A

TTL: '900'

ResourceRecords:

- !GetAtt kubemasterInstance.PublicIp

#-----------------------------------------------------------------------------------------------

# Load-balancer --------------------------------------------------------------------------------

#-----------------------------------------------------------------------------------------------

# Utilise seulement dans la configLaunch qui n'est d'ailleurs pas creee dans ce template :

KubeWorkerNodesSecurityGroup:

Type: AWS::EC2::SecurityGroup

Properties:

VpcId: !Ref 'VPCKubernetes'

GroupDescription: Security Group AutoScalingGroup !Sub ${NomAutoScalingGroup}

SecurityGroupIngress:

- IpProtocol: icmp

FromPort: '8'

ToPort: "-1"

CidrIp: 0.0.0.0/0

- IpProtocol: tcp

FromPort: '22'

ToPort: '22'

CidrIp: 0.0.0.0/0

- IpProtocol: tcp

FromPort: '32000'

ToPort: '32767'

CidrIp: 0.0.0.0/0

- IpProtocol: udp

FromPort: '32000'

ToPort: '32767'

CidrIp: 0.0.0.0/0

- IpProtocol: tcp

FromPort: '80'

ToPort: '80'

CidrIp: 0.0.0.0/0

- IpProtocol: tcp

FromPort: '443'

ToPort: '443'

CidrIp: 0.0.0.0/0

- IpProtocol: tcp

FromPort: '10250'

ToPort: '10250'

CidrIp: 0.0.0.0/0

- IpProtocol: udp

FromPort: '8285'

ToPort: '8285'

CidrIp: 0.0.0.0/0

- IpProtocol: udp

FromPort: '8472'

ToPort: '8472'

CidrIp: 0.0.0.0/0

# Pour tests : A SUPPRIMER :

- IpProtocol: tcp

FromPort: '0'

ToPort: '65535'

CidrIp: 0.0.0.0/0

# Pour tests : A SUPPRIMER :

- IpProtocol: udp

FromPort: '0'

ToPort: '65535'

CidrIp: 0.0.0.0/0

# Pas besoin si VXLAN :

# - IpProtocol: tcp

# FromPort: '0'

# ToPort: '65535'

# CidrIp: 0.0.0.0/0

Tags:

- Key: Name

Value: !Sub ${NomLoadBalancer}-SG-CFN

- Key: CloudFormationStack

Value: !Ref 'AWS::StackId'

LBSecurityGroup:

Type: AWS::EC2::SecurityGroup

Properties:

VpcId: !Ref 'VPCKubernetes'

GroupDescription: Security Group LoadBalancer !Sub ${NomLoadBalancer}

SecurityGroupIngress:

- IpProtocol: icmp

FromPort: '8'

ToPort: "-1"

CidrIp: 0.0.0.0/0

- IpProtocol: tcp

FromPort: '8000'

ToPort: '10000'

CidrIp: 0.0.0.0/0

- IpProtocol: tcp

FromPort: '80'

ToPort: '80'

CidrIp: 0.0.0.0/0

- IpProtocol: tcp

FromPort: '443'

ToPort: '443'

CidrIp: 0.0.0.0/0

Tags:

- Key: Name

Value: !Sub ${NomLoadBalancer}-SG-CFN

- Key: CloudFormationStack

Value: !Ref 'AWS::StackId'

LoadBalancer:

Type: AWS::ElasticLoadBalancingV2::LoadBalancer

Properties:

Type: application

Name: !Ref 'NomLoadBalancer'

# Scheme: internet-facing

SecurityGroups:

- Ref: LBSecurityGroup

# AvailabilityZones: !GetAZs ''

# CrossZone: 'true'

Subnets: !Ref SubnetsPublicLB

# HealthCheck:

# Target: HTTP:80/

# HealthyThreshold: '3'

# UnhealthyThreshold: '5'

# Interval: '30'

# Timeout: '5'

Tags:

- Key: CloudFormationStack

Value: !Ref 'AWS::StackId'

LoadBalancerListener:

Type: AWS::ElasticLoadBalancingV2::Listener

Properties:

DefaultActions:

- Type: forward

TargetGroupArn: !Ref 'LoadBalancerListenerTargetGroup'

LoadBalancerArn: !Ref 'LoadBalancer'

Port: 80

Protocol: HTTP

LoadBalancerListenerTargetGroup:

Type: AWS::ElasticLoadBalancingV2::TargetGroup

DependsOn: LoadBalancer

Properties:

Name: !Ref 'NomTargetGroup'

Port: 30000

Protocol: HTTP

VpcId: !Ref 'VPCKubernetes'

Tags:

- Key: CloudFormationStack

Value: !Ref 'AWS::StackId'

#-----------------------------------------------------------------------------------------------

# Autoscaling ----------------------------------------------------------------------------------

#-----------------------------------------------------------------------------------------------

KubeNodesAutoScalingGroup:

Type: AWS::AutoScaling::AutoScalingGroup

DependsOn:

- LoadBalancer

Properties:

AutoScalingGroupName: !Ref 'NomAutoScalingGroup'

# AvailabilityZones:

# Fn::GetAZs: ''

LaunchConfigurationName: !Ref 'NomLaunchConfiguration'

DesiredCapacity: '1'

MinSize: '1'

MaxSize: '1'

TargetGroupARNs:

- !Ref 'LoadBalancerListenerTargetGroup'

HealthCheckType: ELB

HealthCheckGracePeriod: 28000 # Volontairement élevé (8h) car par défaut le LB les déclare Unhealthy trop tot et l'autoscaling supprime les worker-nodes trop frequement.

VPCZoneIdentifier:

- !Ref 'SubnetPublicKubeNodes'

Tags:

- Key: CloudFormationStack

Value: !Ref 'AWS::StackId'

PropagateAtLaunch: 'true'

#-----------------------------------------------------------------------------------------------

# Sous-domaine DNS Kubernetes ------------------------------------------------------------------

#-----------------------------------------------------------------------------------------------

KubernetesSousDomaine:

Type: AWS::Route53::RecordSetGroup

Properties:

HostedZoneName: !Ref 'DomaineDNS'

Comment: Enregistrement du sous-domaine Kubernetes

RecordSets:

- Name: !Ref 'SousDomaineKubernetes'

Type: A

AliasTarget:

HostedZoneId: !GetAtt LoadBalancer.CanonicalHostedZoneID

DNSName: !GetAtt LoadBalancer.DNSName

#-----------------------------------------------------------------------------------------------

# Outputs ------------------------------------------------------------------

#-----------------------------------------------------------------------------------------------

Outputs:

IPpublicInstance:

Description: IP public de l'instance creee

Value: !Sub ${kubemasterInstance.PublicIp}

FQDNKubeMasterInstance:

Description: l'enregistrement DNS de l'instance creee

Value: !Ref 'FQDNKubeMasterInstance'

## A tester : Controlleur Igress Nginx avec certificat auto-signé :

Création du certificat auto-signé :

$ mkdir /etc/ssl/private

$ chmod 700 /etc/ssl/private

$ openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/ssl/private/nginx-selfsigned.key -out /etc/ssl/certs/nginx-selfsigned.crt

FQDN = ingress.maddoudou.click

$ openssl dhparam -out /etc/ssl/certs/dhparam.pem 2048

Modifier /etc/hosts pour ajouter :

<IP privée instance EC2> ingress.maddoudou.click

Déclaration des certificats dans un secret Kubernetes :

$ kubectl create secret tls tls-certificate --key /etc/ssl/private/ingress-nginx-selfsigned.key --cert /etc/ssl/certs/ingress-nginx-selfsigned.crt

$ kubectl create secret generic tls-dhparam --from-file=/etc/ssl/certs/dhparam.pem

Création du contrôleur nginx :

**nginx-controller.yaml** :

apiVersion: extensions/v1beta1

kind: Deployment

metadata:

name: nginx-ingress-controller

spec:

replicas: 1

revisionHistoryLimit: 3

template:

metadata:

labels:

k8s-app: nginx-ingress-lb

spec:

containers:

- args:

- /nginx-ingress-controller

- "--default-backend-service=$(POD\_NAMESPACE)/default-http-backend"

- "--default-ssl-certificate=$(POD\_NAMESPACE)/tls-certificate"

env:

- name: POD\_NAME

valueFrom:

fieldRef:

fieldPath: metadata.name

- name: POD\_NAMESPACE

valueFrom:

fieldRef:

fieldPath: metadata.namespace

image: "gcr.io/google\_containers/nginx-ingress-controller:0.9.0-beta.5"

imagePullPolicy: Always

livenessProbe:

httpGet:

path: /healthz

port: 10254

scheme: HTTP

initialDelaySeconds: 10

timeoutSeconds: 5

name: nginx-ingress-controller

ports:

- containerPort: 80

name: http

protocol: TCP

- containerPort: 443

name: https

protocol: TCP

volumeMounts:

- mountPath: /etc/nginx-ssl/dhparam

name: tls-dhparam-vol

terminationGracePeriodSeconds: 60

volumes:

- name: tls-dhparam-vol

secret:

secretName: tls-dhparam

---

apiVersion: v1

kind: Service

metadata:

name: nginx-ingress

spec:

type: LoadBalancer

ports:

- name: http

port: 80

targetPort: http

- name: https

port: 443

targetPort: https

selector:

k8s-app: nginx-ingress-lb

## Installation de Rancher et Kubernetes

*Note : tutoriel d’installation de Rancher :*

<https://www.weave.works/blog/how-to-supercharge-your-kubernetes-cluster-with-rancher-weave-cloud>

1. Instancier une instance EC2 **'Ubuntu Server 16**' dédiée à Rancher *(T2.medium, sinon pb de mémoire avec les micro et les small pour héberger le serveur + un host (Worker node Kubernetes))*. **Y associer également une EIP**.
2. Mettre à jour les références apt-get et installer Docker :

$ apt-get update

$ apt-get install -y docker.io

1. Installer un conteneur Rancher :

$ docker run -d --restart=unless-stopped -p 8080:8080 rancher/server

1. Se connecter à la console d’administration de Rancher via l’URL http://<IP publique instance EC2> :8080
2. Dans la console web de Rancher, créer un environnement Kubernetes et ajouter un worker node :
   * Cliquer sur "Default"-> "Manage Environments", puis sur le bouton "Add environnment"
   * Entrer un nom et une description, sélectionner "Kubernetes" et cliquer sur Ok
   * Dans le menu suivant, cliquer sur "Default"-> *<Nom de l'environnement créé>*
   * Un hôte doit être créé : dans la bannière qui apparait, cliquer sur "Add a host". **Bien vérifier qu'on est dans l'environnement Kubernetes** (et non dans 'Default', cf en haut à gauche).
   * Dans l'hôte à créer, choisir 'Custom', puis saisir l'EIOP dans l'adresse.
   * Copier la commande générée par Rancher et la coller dans l'instance EC2 exécutant Rancher (en gros on créé un Worker node Kubernetes sur le serveur lui-même). Elle doit ressembler à :
   * $ sudo docker run -e CATTLE\_AGENT\_IP="52.210.251.84" --rm --privileged -v /var/run/docker.sock:/var/run/docker.sock -v /var/lib/rancher:/var/lib/rancher rancher/agent:v1.2.11 http://52.210.251.84:8080/v1/scripts/B5FBD6EE765F0FE57896:1514678400000:KID50OYIQ0Rc5dDvYaxGXNrCFk
   * Suivre également l'évolution de la création de l'hôte dans le menu "Infrastructure" -> "Hosts"
3. Une fois l'hôte créé et configuré, configurer la commande kubectl sur l’instance EC2 hébergeant Rancher :
   * Installer la commande ‘kubectl’ :

$ curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -

$ echo "deb http://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee -a /etc/apt/sources.list.d/kubernetes.list

$ sudo apt-get update

$ sudo apt-get install -y kubectl

* + Dans la console web de Rancher, lancer "Kubernetes" -> "CLI".
  + Cliquer sur le bouton « Generate configuration » et copier la configuration.
  + Sur l’instance EC2 hébergeant Rancher, créer le fichier « ~/.kube/config » et y coller la configuration copiée précédemment.
  + Tester avec la commande "kubectl get nodes" sur l’instance EC2 hébergeant Rancher et Jenkins : l'host créé doit apparaitre et être 'Ready'.