Raspberry Pi virtualisation légère avec Cockpit

Table of Contents

Cockpit est un projet libre, mis à jour régulièrement et disponible pour de nombreuses distributions. Concrètement, grâce à Cockpit, toutes les tâches d'administration habituelles sur les serveurs sont réalisables. Par le biais de Cockpit, on peut gérer le stockage, la mise en réseau, les comptes utilisateurs, voir les services et daemons actifs, modifier quelques paramètres systèmes (domaine, nom, configuration des mises à jour) et encore accéder aux logs. Ainsi que la partie virtualisation ...

Cet article va expliquer le déploiement d'un hyperviseur et le déploiement d'un cluster MicroK8S sur une carte Raspberry Pi.

Installation de l'hyperviseur

Installation du système hôte

Canonical propose une version ARM 64 Bits de sa distribution Ubuntu 20.04 LTS, télécharger et Utiliser l'une des méthodes décrites ici pour graver l'image sur la carte SD.

```
$ wget
http://cdimage.ubuntu.com/releases/20.04/release/ubuntu-20.04-preinstalled-s
erver-arm64+raspi.img.xz
$ xz -d ubuntu-20.04-preinstalled-server-arm64+raspi.img.xz
```

Une fois l'opération terminée, penser à créer un fichier vide nommé SSH dans la partition système, pour pouvoir se connecter à l'OS Ubuntu via le réseau local en mode headless.

Connecter alors la carte Raspberry Pi 4 au réseau local via son port Ethernet.

Lancer la découverte de l'adresse IP utilisée par l'OS via Nmap :

```
sudo nmap -p22 -v <Adressage CIDR> 1 grep open
```

On peut s'y connecter via SSH (login : ubuntu / pwd : ubuntu à changer après connexion)

Vérifier que la carte dispose d'un processeur qui supporte la virtualisation :

```
sudo apt install cpu-checker
kvm-ok
 INFO: /dev/kvm exists
 KVM acceleration can be used
```

Il est alors possible d'y installer Cockpit.

Installation de Cockpit

Installer les paquets suivants:

```
sudo apt install cockpit cockpit-bridge cockpit-dashboard cockpit-machines
cockpit-networkmanager cockpit-packagekit cockpit-storaged cockpit-system
cockpit-ws bridge-utils -y
```

Ce qui entraîne l'installation de l'hyperviseur KVM et des outils associés pour cette architecture ARM 64 Bits.

Activation de Cockpit.

```
sudo systemctl enable cockpit.socket
sudo systemctl start cockpit.socket
sudo systemctl status cockpit.socket
 cockpit.socket - Cockpit Web Service Socket
    Loaded: loaded (/lib/systemd/system/cockpit.socket; enabled; vendor
preset: enabled)
    Active: active (listening) since Sun 2020-07-05 14:33:17 UTC; 3min 24s
ago
  Triggers: cockpit.service
      Docs: man:cockpit-ws(8)
    Listen: [::]:9090 (Stream)
      Tasks: 0 (limit: 9255)
    CGroup: /system.slice/cockpit.socket
```

Activer un nouveau mot de passe pour le compte root.

Construction du cluster

Sur le port TCP 9090, on peut accéder au dashboard de Cockpit pour commencer la création de machines virtuelles

Création du noeud maître

Créer une machine virtuelle par l'utilisation d'images ISO de la distribution légère Alpine Linux ...

```
wget -c
http://di-cdn.alpinelinux.org/alpine/v3.12/releases/aarch64/alpine-virt-3.12
.0- aarch64.iso
 alpine-virt-3.12.0-aarch64.iso 100%[  in 23s
 >] 36.02M .50MB/s
 2020-07-05 18:14:30 (1.55 MB/s) - 'alpine-virt-3.12.0-aarch64.1so' saved
[37765120/37765120]
```

Connexion à la console de la machine virtuelle également par ce biais ...

Y installer le moteur Docker après avoir activer les dépôts nécéssaires ...

```
cat /etc/apk/repositories
#/media/cdrom/apks
 http://dl-cdn.alpinelinux.org/alpine/v3.12/main
 http://dl-cdn.alpinelinux.org/alpine/v3.12/community
 http://dl-cdn.alpinelinux.org/alpine/edge/main
 http://dl-cdn.alpinelinux.org/alpine/edge/community
 http://dl-cdn.alpinelinux.org/alpine/edge/testing

apk add docker
 OK: 372 MiB in 82 packages

rc-update add docker boot
 * service docker added to runlevel boot

service docker start
 * Mounting cgroup filesystem
                                        [ ok ]
 * /var/log/docker.log: creating file
 * /var/log/docker.log: correcting owner
 * Starting Docker Daemon ...
                                        [ ok ]

service docker status
 * status: started

docker ps -a
```

```
 CONTAINER ID IMAGE COMMAND CREATED STATUS
 PORTS NAMES

docker version
 Client:
  Version: 19.03.12
  API version: 1.40
  Go version: go1.14.4
  Git commit: 48a66213fe1747e8873f849862ff3fb981899fc6
  Built: Tue Jun 30 21:40:22 2020
  OS/Arch: linux/arm64
  Experimental: false

 Server:
  Engine:
   Version: 19.03.12
   API version: 1.40 (minimum version 1.12)
   Go version: go1.14.4
   Git commit: 48a66213fe1747e8873f849862ff3fb981899fc6
   Built: Tue Jun 30 20:58:26 2020
   OS/Arch: linux/arm64
   Experimental: false
  containerd:
   Version: v1.3.5
   GitCommit: 9b6f3ec0307a825c38617b93ad55162b5bb94234
  runc:
   Version: 1.0.0-rc91
   GitCommit: 24a3cf88a7ae5f4995f6750654c0e2ca61ef4bb2
  docker-init:
   Version: 0.19.0
   GitCommit:
```

Récupération de Rancher K3s,

```
wget -c
https://github.com/rancher/k3s/releases/download/v1.18.4%28k3s1/k3s-arm64
k3s-arm64 100%[ >] 47.06M 1.38MB/s in 30s 2020-07-05 18:33:58 (1.55 MB/s) -
'k3s-arm64' saved [49348608/49348608] chmod +rwx k3s-arm64 mv k3s-arm64
/usr/bin/k3s nohup k3s server --docker &
```

qui permet d'initier au travers de cette première machine virtuelle, un noeud maître d'un futur cluster Kubernetes ...

Création des autres noeuds du cluster

Créer deux autres machines virtuelles Alpine Linux sur le même principe.

Chargemer K3s sur ces dernières et configurer en tant que noeuds de ce cluster Kubernetes ...

```
nohup k3s agent --server https://192.168.122.10:6443 --token
Kl0d556d982bc6191f92023bf84f82a648a70959e4815fdf75b136ea7d90939b3f5::server:
67060d0 9e2e5edc448ebf6298ff2437a --docker &
```


```
nohup k3s agent --server https://192.168.122.10:6443 --token
Kl0d556d982bc6191f92023bf84f82a648a70959e4815fdf75b136ea7d90939b3f5::server:
67060d0 9e2e5edc448ebf6298ff2437a --docker &
```


Configuration du contrôleur

Il reste à charger le client Kubectl sur l'OS Ubuntu avec le fichier kubeconfig correspondant pour accéder et contrôler ce cluster Kubernetes (sur la base de ces petites machines virtuelles Alpine Linux) :

```
curl -LO
https://storage.googleapis.com/kubernetes-release/release/v1.18.4/bin/linux/
arm64 /kubectl
 % Total    % Received  % Xferd Average Speed   Time    Time     Time
Current
                                 Dload  Upload Total   Spent    Left     Speed
 100 39.8M 100 39.8M    0     0 1334k    0      0:00:30 0:00:30 --:--:-- 1267k

chmod +rwx kubectl; sudo my kubectl /usr/bini; mkdir .kube; touch
.kube/config
cat .kube/config
 apiVersion: v1
 clusters:
 - cluster:
     certificate-authority-data:
LSOtLS1CRUdJTiBDRVJUSUZJINFURSOtLSOtCk1JSUJklekNCL3FBREFnRUNBZOVBTUFvRONDcUd
TTTO5OkFNOOlDTXUVEFmOmd0VkJ
BTU1HR3N6Y3kxelpYSjIKKhJdFkyRkFNVFUlTXprMOSERTFNVEFlRncweUlEOTNNRFV4TORNMUSU
RmFGdzB6TURBM01ETXhPREOxTl
RGYWNIQ014SVRBZkJnTlZCQU1NROdzemN5MXpaWEoyWlhJdFkyRkFNVFUlTXprMOSERTFNVEJaTU
JNROJ5cUdTTTUCkFnRUdDQ3FHU
0000UF3RUhBMElBC*91aXdOmtsZi8zK2cOWXVTYnp0V2tJcEpPUmhYT3BjT3EzZjhLVORnTGYKVz
ZaMGFxRGxLY1VSL1J1MWOyajBJ
VEtmZEcxS2xIeHdNRW1lUTBBalcwR2pJekFoTUEOROExVWREdOVCL3dRROpBdOlDcERBUEJnTlZI
Uk1COWY4RUJUQURBUUgvTUFvRON
DcUTITTO5C*FNUEwZOFNRVVDSURIT3VuSHk2N3RWCkR6ZVZTOnpvais3Rlc5czNBN2w0dOVXcWxN
bmRldOgrOWlFQWsOREgOSHVGQT
dFYTNodzZJaFIrSEhScTlhMAITzdoWUZxZEJBaEdDa0FRPOotLSOtLUVORCBDRVJUSUZJOOFURSO
tLSOtCg==
     server: https://192.168.122.10:6443
   name: default
 contexts:
 - context:
     cluster: default
```

```
    user: default
  name: default
current-context: default
kind: Config
preferences: {}
users:
- name: default
  user:
    password: a8a5bb53c189e4cf32d329698096eaa5
    username: admin
```

Le cluster Kubernetes est opérationnel …

```
kubectl get po,svc ,ing - - all - namespaces
```

Charger ce manifeste YAML qui va permettre de lancer le sempiternel démonstrateur FC par l'intermédiaire d'une image Docker en version ARM 64bits.

```
kubectl apply -f deployment.yml
 deployment.apps/fcdemo3 created
 service/fcdemo-service created
 ingress.networking.k8s.io/fcdemo-ingress created
```

```
 ---
 apiVersion: apps/v1
 kind: Deployment
 metadata:
   name: fcdemo3
   labels:
     app: fcdemo3
 spec:
   replicas: 2
   selector:
     matchLabels:
       app: fcdemo3
   template:
     metadata:
       labels:
         app: fcdemo3
     spec:
       containers:
       - name: fcdemo3
         image: mcas/franceconnect-agent-demo-arm64:latest
         ports:
         - containerPort: 8000
 ---
 apiVersion: v1
 kind: Service
```

```
metadata:
  name: fcdemo-service
spec:
  type: ClusterlP
  selector:
    app: fcdemo3
  ports:
  - protocol: TCP
    port: 80
    targetPort: 8000
---
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: fcdemo-ingress
  annotations: ingress.kubernetes.io/ssl-redirect: "false"
spec:
  rules:
  - http:
      paths:
      - path: /
        backend:
          serviceName: fcdemo-service
          servicePort: 80
```

Le démonstrateur FC est accessible via Traefik en tant qu'Ingress Controller dans ce cluster Kubernetes ...

Le dashboard de Cockpit permet de suivre quelques métriques de base avec le fonctionnement de ces machines virtuelles ...

Un test simple qui n'aura pas réussi à consommer la mémoire de cette version à 8Go ...