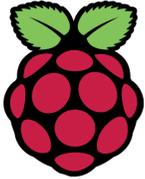




Cluster Kubernetes avec Raspberry Pi

Antonin LEFEVRE 



Introduction

La virtualisation présente un intérêt majeur dans l'informatique moderne, permettant la création d'environnements isolés et flexibles pour le déploiement d'applications et de services. De nombreux outils existent pour faciliter ce processus, chacun adapté à des besoins spécifiques. Dans le cadre de ce projet, nous avons mis en place un cluster de Raspberry Pi pour faire tourner de petits logiciels publics ainsi que des projets personnels sur le réseau.

Matériel nécessaire

Pour ce projet, nous avons utilisé plusieurs composants matériels. Tout d'abord, des Raspberry Pi, en particulier un RP3 B+ et un RP4 avec 8 Go de RAM. Un réseau internet est nécessaire pour permettre la communication entre les Raspberry Pi. De plus, des cartes Micro SD d'au moins 16 Go sont requises pour chaque Raspberry Pi. Côté logiciel, nous avons utilisé le Raspberry Pi Imager pour installer le système d'exploitation sur les cartes SD.

En complément, un rack pour empiler les Raspberry Pi avec un système de ventilation a été utilisé.

Keywords Kubernetes, K3s, Docker, Raspberry Pi

TABLE DES MATIÈRES

1. Kubernetes et K3s	3
2. Préparer les Raspberry Pi	3
2.1. Installation du rack	3
2.2. Installation des OS et configuration	4
3. Installer K3s sur les Raspberry Pi	7
3.1. Installation de K3s sur le nœud maître (RP4)	7
3.2. Ajouter les nœuds travailleurs au cluster	7
3.3. Modifier le nœud maître pour qu'il puisse aussi faire tourner des pods	8
4. Conteneuriser et publier son application	8
5. Installer une application	9
6. Ajouter des variables d'environnement	11
7. Commandes utiles	12
8. Conclusion	12

1. KUBERNETES ET K3S

Kubernetes est une plateforme qui permet de gérer et d'orchestrer des applications conteneurisées de manière automatique et à grande échelle. Il permet de déployer des applications sous forme de pods sur un cluster de nœuds, assurant le bon fonctionnement, la répartition des ressources et la haute disponibilité des applications. Un service dans Kubernetes permet d'exposer l'application aux utilisateurs, que ce soit sur Internet ou sur un réseau local.

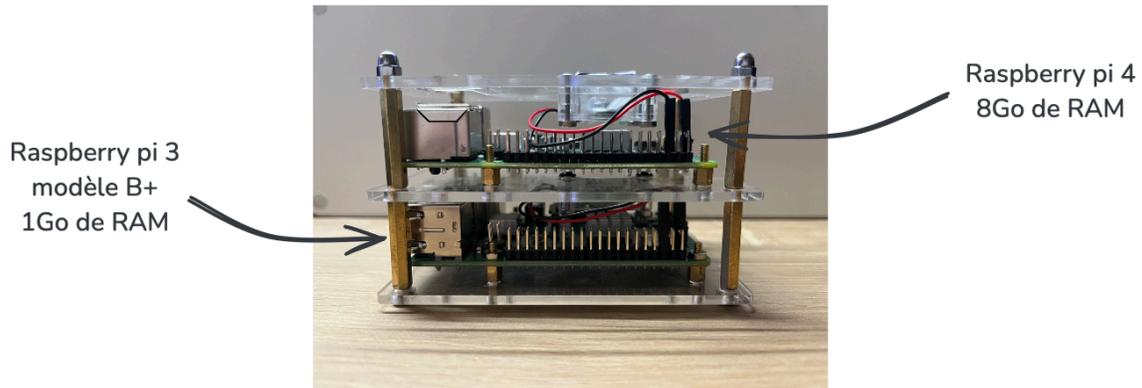
K3s est une distribution légère de Kubernetes, spécialement conçue pour les environnements contraints en ressources comme les Raspberry Pi. Il offre toutes les fonctionnalités essentielles de Kubernetes, mais avec une empreinte mémoire et CPU réduite, ce qui le rend idéal pour notre projet.

2. PRÉPARER LES RASPBERRY PI

2.1. Installation du rack

On installe les deux Raspberry Pi sur un rack, qui permet de les empiler et d'avoir un système de ventilation.

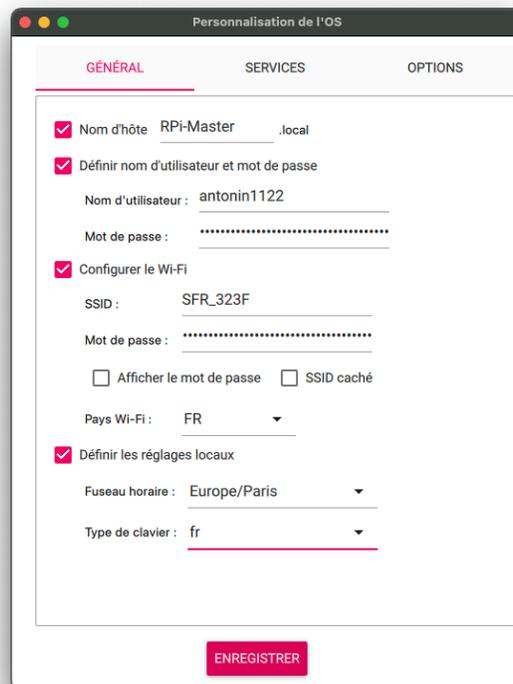




2.2. Installation des OS et configuration

Nous avons choisi d'utiliser Raspberry Pi OS Lite pour notre installation. Dans l'interface de Raspberry Pi Imager, nous avons sélectionné le modèle de Raspberry Pi et l'OS approprié. Les paramètres ont été configurés comme suit :

- Dans l'onglet General, nous avons défini le hostname sur RPi-Master pour le nœud maître et RPi-Worker pour le nœud travailleur. Les noms d'utilisateur et mots de passe ont été configurés, ainsi que la connexion Wi-Fi.
- Dans l'onglet Services, nous avons activé SSH en choisissant l'authentification par mot de passe.



Pour éviter les changements d'IP dus à l'attribution dynamique, nous définissons des IP statiques sur chaque Raspberry Pi. Dans les versions actuelles de Raspberry Pi OS, c'est NetworkManager qui est utilisé pour gérer les adresses IP.

Nous récupérons l'adresse IP du routeur avec la commande :

```
ip route | grep default | awk '{print $3}'
```

Ensuite, nous ouvrons le fichier `/etc/NetworkManager/system-connections/preconfigured.nmconnection` et modifions la section `[ipv4]` comme suit :

```
[ipv4]
method=manual
addresses=<static_address>/24;<routeur_address>
gateway=<routeur_address>
dns=<routeur_address>;
```

- `<static_address>` : l'adresse IP statique que nous souhaitons attribuer au Raspberry Pi.
- `<routeur_address>` : l'adresse IP du routeur récupérée précédemment.

Après avoir enregistré les modifications, nous redémarrons le système avec :

```
sudo reboot
```

Une fois le Raspberry Pi redémarré, nous nous reconnectons en SSH en utilisant la nouvelle adresse IP statique. Nous pouvons vérifier l'adresse IP du Raspberry Pi avec :

```
hostname -I
```

3. INSTALLER K3S SUR LES RASPBERRY PI

Avant toute chose, il faut activer `cgroup` sur les Raspberry Pi, car il ne l'est pas par défaut avec Raspberry Pi OS. Pour cela, il suffit d'ajouter `cgroup_memory=1 cgroup_enable=memory` à la fin du fichier `/boot/firmware/cmdline.txt`. Ensuite, nous redémarrons les Raspberry Pi.

3.1. Installation de K3s sur le nœud maître (RP4)

Nous installons K3s sur le nœud maître avec la commande :

```
sudo curl -sfL https://get.k3s.io | sh -
```

Pour tester la configuration :

```
sudo systemctl status k3s
```

Pour obtenir les informations du cluster :

```
sudo kubectl get nodes
```

3.2. Ajouter les nœuds travailleurs au cluster

Sur le Raspberry Pi qui servira de nœud travailleur, nous exécutons :

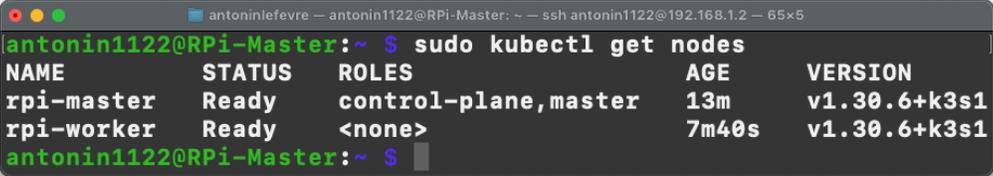
```
sudo curl -sfL https://get.k3s.io | K3S_URL=https://<master_node_ip>:6443  
K3S_TOKEN=<your_token> sh -
```

Le token peut être récupéré depuis le nœud maître avec :

```
sudo cat /var/lib/rancher/k3s/server/node-token
```

Pour vérifier la configuration du nœud sur le cluster depuis le nœud maître :

```
sudo kubectl get nodes
```



```
antonin122@RPi-Master:~$ sudo kubectl get nodes  
NAME           STATUS    ROLES                    AGE     VERSION  
rpi-master     Ready    control-plane,master    13m    v1.30.6+k3s1  
rpi-worker     Ready    <none>                  7m40s  v1.30.6+k3s1  
antonin122@RPi-Master:~$
```

3.3. Modifier le nœud maître pour qu'il puisse aussi faire tourner des pods

Pour que le nœud maître puisse également héberger des pods applicatifs, nous supprimons la "taint" par défaut :

1. Visualiser les taints appliqués :

```
sudo kubectl describe node <nom-du-noeud-maitre>
```

2. Supprimer la taint NoSchedule :

```
kubectl taint nodes <nom-du-noeud-maitre> node-role.kubernetes.io/master:NoSchedule-
```

Après cette opération, le nœud maître peut héberger des pods comme les autres nœuds du cluster.

4. CONTENEURISER ET PUBLIER SON APPLICATION

Nous publions les images des applications sur Docker Hub pour les récupérer avec Kubernetes. Après avoir conteneurisé les applications (par exemple avec `docker init`), nous testons le build :

```
docker compose up --build -d
```

Créer l'image Docker et la publier sur Docker Hub :

```
docker build -t <nom_utilisateur>/<nom_image>:tag .  
docker login  
docker push <nom_utilisateur>/<nom_image>:tag
```

5. INSTALLER UNE APPLICATION

Nous créons un dossier pour une première application, contenant les fichiers `deployment.yaml` et `service.yaml`.

`deployment.yaml` :

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: <name-app>
spec:
  replicas: 1
  selector:
    matchLabels:
      app: <name-app>
  template:
    metadata:
      labels:
        app: <name-app>
    spec:
      containers:
      - name: <name-app>
        image: <nom_utilisateur>/<app_image>:tag
        ports:
        - containerPort: 8000
```

`service.yaml` :

```
apiVersion: v1
kind: Service
metadata:
  name: <name-app>-service
spec:
  type: NodePort
  selector:
    app: <name-app>
  ports:
  - port: 8000
    targetPort: 8000
    nodePort: 30001
```

Nous déployons ensuite l'application :

```
sudo kubectl apply -f deployment.yaml
sudo kubectl apply -f service.yaml
```

Pour vérifier le déploiement :

- Déploiements :

```
sudo kubectl get deployments
```

- Pods :

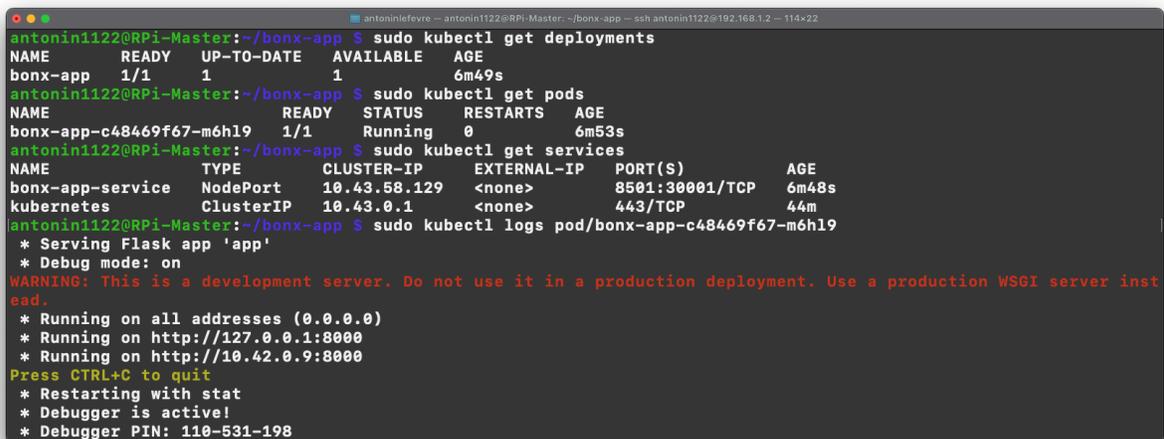
```
sudo kubectl get pods
```

- Services :

```
sudo kubectl get services
```

Pour afficher les logs d'une application :

```
sudo kubectl logs pod/<votre-pod>
```



```
antonin1122@RPi-Master:~/bonx-app $ sudo kubectl get deployments
NAME      READY   UP-TO-DATE   AVAILABLE   AGE
bonx-app  1/1     1             1           6m49s
antonin1122@RPi-Master:~/bonx-app $ sudo kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
bonx-app-c48469f67-m6h19            1/1     Running   0           6m53s
antonin1122@RPi-Master:~/bonx-app $ sudo kubectl get services
NAME                TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
bonx-app-service    NodePort    10.43.58.129 <none>        8501:30001/TCP   6m48s
kubernetes          ClusterIP   10.43.0.1    <none>        443/TCP          44m
antonin1122@RPi-Master:~/bonx-app $ sudo kubectl logs pod/bonx-app-c48469f67-m6h19
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:8000
* Running on http://10.42.0.9:8000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 110-531-198
```

Nous accédons ensuite à notre application depuis : <MASTER_NODE_IP>:30001.

6. AJOUTER DES VARIABLES D'ENVIRONNEMENT

Pour gérer les variables d'environnement sans les inclure dans l'image (pour des raisons de sécurité), nous créons un fichier `secret.yaml`.

`secret.yaml` :

```
apiVersion: v1
kind: Secret
metadata:
  name: myapp-secret
type: Opaque
data:
  SECRET_KEY: c2VjcmV0X2tleV92YWx1ZQ==
  DATABASE_URL: cG9zdGdyZXNxbDovL3VzZXI6cGFzc3dvcmRAMTAuMC4wLjE6NTQzMj9teWRi
```

Les valeurs du champ `data` sont encodées en base64. Pour encoder une valeur :

```
echo -n <valeur> | base64
```

Nous référençons ces valeurs dans le `deployment.yaml` :

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: <name-app>
spec:
  replicas: 1
  selector:
    matchLabels:
      app: <name-app>
  template:
    metadata:
      labels:
        app: <name-app>
    spec:
      containers:
        - name: <name-app>-container
          image: <nom_utilisateur>/<app_image>:tag
          env:
            - name: SECRET_KEY
              valueFrom:
                secretKeyRef:
                  name: <name-app>-secret
                  key: SECRET_KEY
            - name: DATABASE_URL
              valueFrom:
                secretKeyRef:
                  name: <name-app>-secret
                  key: DATABASE_URL
```

Nous appliquons les configurations :

```
sudo kubectl apply -f secret.yaml
sudo kubectl apply -f deployment.yaml
sudo kubectl apply -f service.yaml
```

7. COMMANDES UTILES

- Déboguer un pod spécifique :

```
sudo kubectl describe pod/<votre-pod>
```

- Relancer une application :

```
sudo kubectl rollout restart deployment/<votre-app>
```

- Afficher tous les pods dans le cluster et le nœud associé :

```
sudo kubectl get pods -o wide
```

- Consulter l'utilisation des ressources sur chaque nœud :

```
sudo kubectl top nodes
```

8. CONCLUSION

Nous avons ainsi mis en place un serveur sur notre réseau local, capable de faire tourner des projets personnels ou des outils téléchargés depuis GitHub, par exemple. Ce cluster de Raspberry Pi offre une plateforme flexible pour le déploiement et la gestion d'applications conteneurisées.