Saé DevCloud - Compte rendu final - Matéo Lhuissier & Jules Morice & Paul Tardieu

Github publique

Étape 1 : Installation et configuration du serveur avec Almalinux et OpenNebula (durée : 3,5 jours).

Jour 1 : Découverte, documentation et installation du serveur Dell

Matéo et Jules:
Après avoir lu le cahier des charges de la saé, nous avons commencé par installer le serveur Dell mis à notre disposition en s'aidant de notre ancienne documentation créée lors d'une saé au semestre 3. Nous avons donc remis le serveur à l'état sortie d'usine via l'IDRAC (reset factory default et reset du volume). Puis nous avons configuré le réseau et le user par défaut pour avoir accès à l'IDRAC depuis nos pcs dans le réseau de la salle.
Nous avons ensuite mappé, depuis le Boot Manager, l'image de l'Almalinux 9 que nous avions téléchargée au préalable. Nous avons ensuite booté sur "Virtual Optical Drive" pour lancer l'installation de l'Almalinux.

Pendant ce temps, nous nous sommes renseigné sur OpenNebula:
- Nous avons commencé par découvrir OpenNebula, une plateforme open-source de gestion de cloud computing.
- Nous avons consulté la documentation officielle d'OpenNebula pour comprendre les prérequis matériels et logiciels.
- Nous avons décidé d'utiliser AlmaLinux 9 comme système d'exploitation pour notre serveur en raison de sa stabilité et de sa compatibilité avec les paquets requis par OpenNebula.
- Installation d'AlmaLinux 9 sur notre machine serveur.
- Mise à jour du système avec les dernières mises à jour et correctifs de sécurité via `dnf update`.

Jour 2 : Installation d'OpenNebula

Après avoir installer Almalinux sur notre serveur, nous avons suivi la documentation pour installer OpenNebula (https://docs.opennebula.io/6.8/installation_and_configuration/frontend_installation/index.html)

- Téléchargement et ajout du dépôt OpenNebula à la configuration Yum de notre serveur.

- Importation de la clé GPG pour assurer l'intégrité des paquets téléchargés.
- Installation des paquets OpenNebula requis (`opennebula-server`, `opennebula-sunstone`, `opennebula-gate`, `opennebula-flow`) via `dnf install`.
- Installation des dépendances supplémentaires telles que `mariadb-server` et `sqlite`.

Jour 3 : Configuration et Dépannage

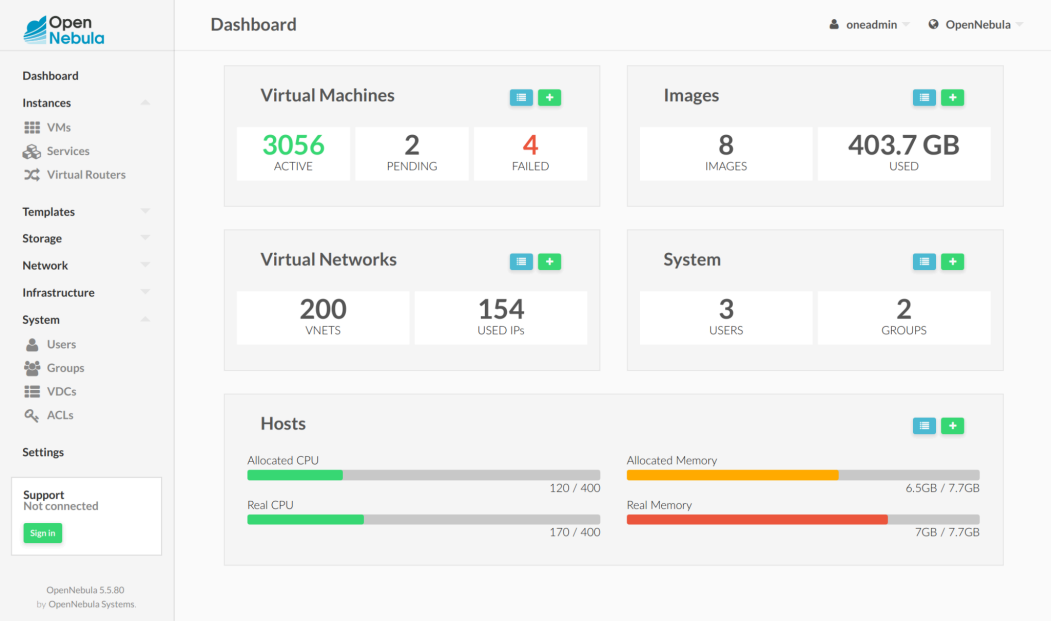
- Installation et configuration de MariaDB pour OpenNebula.
- Création d'une base de données `opennebula` et d'un utilisateur `oneadmin` avec les permissions nécessaires.
- Modification du fichier de configuration `/etc/one/oned.conf` pour utiliser MariaDB comme backend.
- Ajustement des permissions des répertoires et fichiers critiques (`/var/lib/one`, `/etc/one`, `/var/log/one`) pour s'assurer qu'ils sont accessibles par l'utilisateur `oneadmin`.

Problèmes Rencontrés :

- Erreurs de démarrage du service OpenNebula (`oned`) identifiées via `systemctl status opennebula` et `journalctl -xeu opennebula.service`.
- Résolution des erreurs de permission et des problèmes de configuration de la base de données.
- Redémarrage des services après chaque modification pour vérifier la correction des erreurs.
- Création de container: Jules à trouver qu'il fallait activer le pull développer pour récupérer l'installation github.

Jour 3,5 : Finalisation et Vérification

- Démarrage et vérification du service `opennebula-sunstone`.



- Accès à l'interface web Sunstone via le navigateur pour vérifier la configuration.
- Création d'utilisateurs OpenNebula, vérification de la connexion avec `oneadmin` et d'autres utilisateurs via l'interface web.
- Test des fonctionnalités de base d'OpenNebula pour s'assurer que tout fonctionne comme prévu (création de VMs, gestion des réseaux, etc.).

Conclusion

Le projet d'installation et de configuration d'OpenNebula sur un serveur AlmaLinux 9 a été une expérience enrichissante et complexe. Nous avons réussi à surmonter plusieurs défis techniques grâce à une documentation minutieuse, une configuration rigoureuse et une résolution méthodique des problèmes.

Résumé des problèmes et solutions :

- Problèmes de Permission : Résolus en ajustant les permissions des répertoires et fichiers critiques.
- Problèmes de configuration de Base de Données : Réglés en configurant correctement MariaDB et en modifiant le fichier de configuration d'OpenNebula.
- Service `oned` Ne Démarre pas : Résolu en examinant les journaux, en ajustant les configurations et en installant les dépendances manquantes.

Bilan

Malgré nos efforts, OpenNebula n'a pas fonctionné, impossible de relier nos containers sur notre réseau.

Étape 1 bis : Installation et configuration du serveur avec Proxmox (durée : 1 jour).

Après notre échec d'Open Nebula, nous avons décidé avec les autres groupes d'utiliser la technologie de virtualisation Proxmox que nous avions déjà mise en place lors d'une saé au semestre 3. Nous avons donc pu utiliser nos anciens compte rendu et notre expérience pour monter facilement cette solution. Les étapes d'installation sont similaires à l'installation de l'Almalinux décrites précédemment (remise à défaut du serveur, boot manager pour mapper l'iso proxmox,...).

Étape 2 : Répartition des tâches et réalisations des tâches principales de chacun (durée : environ 1 semaine)

Contexte et Objectifs

Dans le cadre de notre projet, nous avons mis en place une organisation structurée pour maximiser l'efficacité et garantir la qualité des résultats. En tant qu'équipe de trois membres (J, P, M), nous avons décidé de répartir les tâches en fonction des compétences et des intérêts de chacun, tout en assurant une communication et une gestion du projet.

Outils de Gestion Utilisés

Pour organiser et suivre l'avancement des tâches, nous avons utilisé deux principaux outils :

1. **Wekan** : Un tableau Kanban pour visualiser les tâches, leurs échéances, et leurs responsables.
2. **Fichier CSV** : Pour le suivi des tâches via Nushell, permettant une analyse plus détaillée et automatisée des progrès.

Répartition des Tâches

Voici comment nous avons réparti les tâches principales du projet :

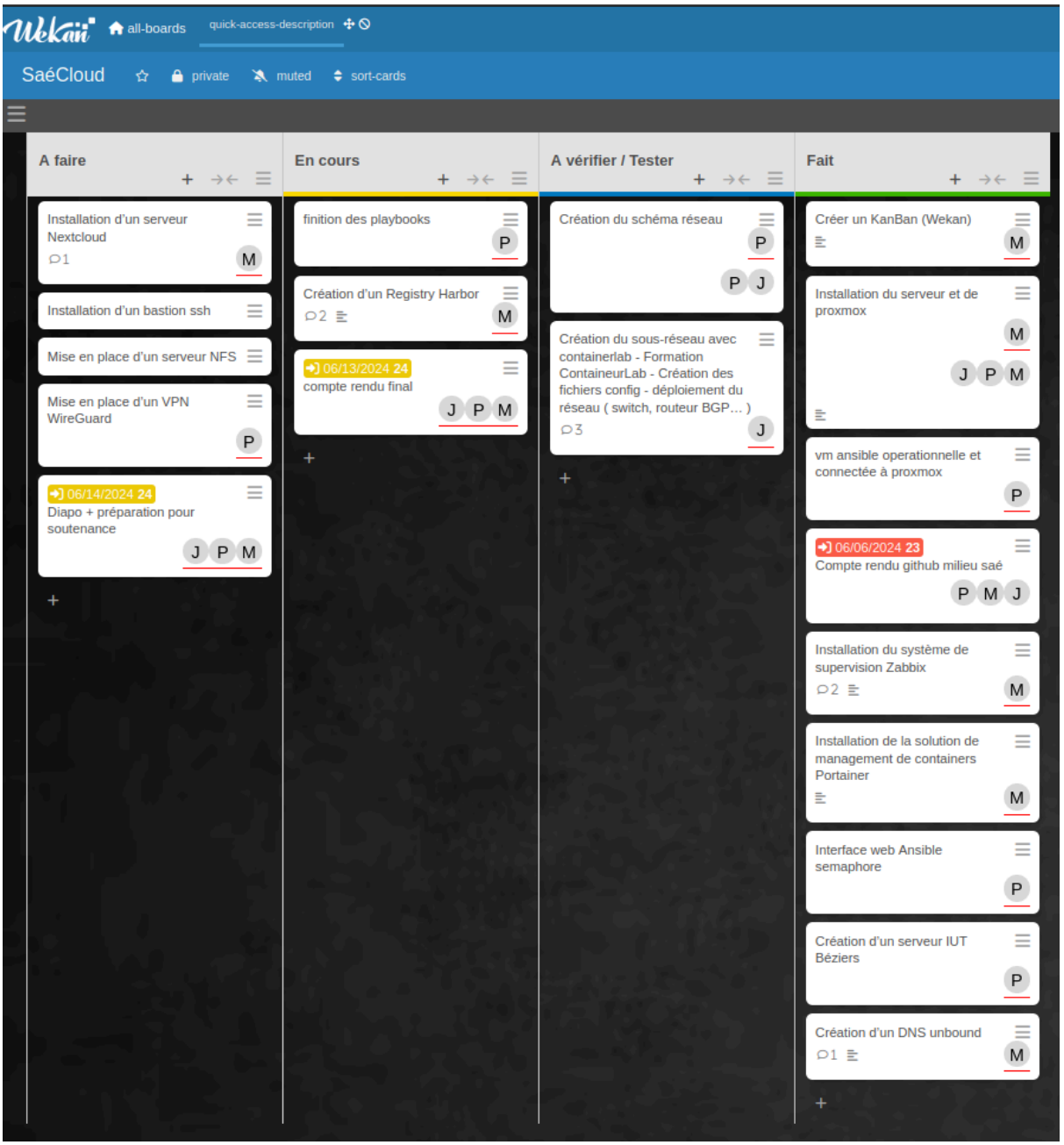
1. **Installation et Configuration de Serveurs (J) :**
    - **Sous-réseaux avec ContainerLab** : Formation, création des fichiers de configuration, et déploiement des réseaux.
    - **BGP, OSPF :**
    - **NFS Server** : Mise en place d'un serveur de fichiers.
    - **DNS Unbound** : Création et configuration d'un DNS.
2. **Ansible Semaphore Github Réseaux Microservices (P) :**
    - **Création du Schéma Réseau** : Conception du schéma réseau initial.
    - **VM Ansible** : Configuration d'une VM sur Proxmox pour Ansible.
    - **Semaphore** : Mise en place d'un environnement Sémaphore web.
    - **Wordpress et Mysql** : Réalisation de playbook Ansible d'auto création sur machine cible de container Apache2 et Mysql
    - **Playbook update** : playbook automatique avec Cron Sémaphore apt update
    - **Playbook Install Docker et Compose** : installation sur machine cible
    - **Github collaboratif :** Création du github de groupe et du dossier Ansible, vault et inventory dynamique avec synchro Github
3. **Déploiement et Supervision (M) :**
    - **Zabbix** : Installation d'un système de supervision.
    - **Portainer** : Installation de la solution de gestion de conteneurs.
    - **Wekan** : Déploiement du container Wekan et configuration du tableau collaboratif
    - **DNS Unbound** : Création et configuration d'un DNS.
    - **Nextcloud** : Installation et configuration pour le partage de fichiers.
    - **Harbor** : Déploiement d'un système de stockage d'image.

Utilisation de Wekan

Pour faciliter la gestion et la visualisation des tâches, nous avons mis en place un tableau Kanban sur Wekan. Voici comment nous l'avons structuré :

1. **Colonnes de Tâches** : Les tâches sont réparties en colonnes telles que "À Faire", "En Cours", "Terminées".
2. **Cartes de Tâches** : Chaque tâche est représentée par une carte contenant les détails, les échéances, et les responsables.
3. **Collaboration** : Les membres peuvent commenter sur les cartes, ajouter des pièces jointes, et mettre à jour le statut des tâches en temps réel.

Voici une capture d'écran de notre tableau Wekan :



Suivi avec Fichier CSV et Nushell

En complément de Wekan, nous avons utilisé un fichier CSV pour suivre l'avancement des tâches avec Nushell. Cela nous a permis de :

1. **Analyser les Données** : Utiliser Nushell pour filtrer, trier, et analyser les données des tâches.
2. **Rapports Automatisés** : Générer des rapports automatisés pour les réunions de suivi et les bilans de projet.

Étape 3 : Finalisation et regroupement des différents projets sur le réseau privé container lab (durée : 1 à 2 jours)

Après avoir travaillé sur nos tâches respectives, notre objectif final était de regrouper tous les éléments de notre projet sur un réseau privé commun, orchestré par ContainerLab. Cette étape cruciale nous a permis de valider l'intégration de nos travaux et de garantir le bon fonctionnement de l'ensemble du système.

Collaboration et Intégration
Pour cette phase, nous avons combiné nos efforts individuels pour créer un environnement cohérent et fonctionnel. Voici comment nous avons procédé :

1. **Déploiement de Conteneurs avec Ansible (Paul) :**
   - Paul a utilisé Ansible pour automatiser le déploiement de plusieurs conteneurs. Ansible a permis une gestion et un déploiement rapide des conteneurs sur le réseau.
   - Les playbooks Ansible ont été utilisés pour configurer les services et applications nécessaires sur ces conteneurs, garantissant une installation cohérente.

2. **Services sur les Conteneurs (Matéo) :**
   - Matéo a installé et configuré les services critiques sur les conteneurs déployés par Ansible. Parmi ces services, nous avons inclus Nextcloud, un containeur Wekan, un DNS, et d'autres outils nécessaires à notre infrastructure.
   - Chaque service a été testé individuellement pour s'assurer qu'il fonctionnait correctement avant l'intégration finale.

3. **Réseau Privé avec ContainerLab (Jules) :**
   - Jules a conçu et mis en place le réseau privé en utilisant ContainerLab après s'être formé sur plusieurs jours. Cette infrastructure réseau a fourni une base solide pour connecter tous les conteneurs et services.
   - Le réseau privé a été configuré pour inclure des sous-réseaux, des routes dynamiques et des règles de sécurité adaptées à notre environnement.

Processus d'Intégration

Pour s'assurer que tous les composants fonctionnent ensemble de manière harmonieuse, nous avons suivi les étapes suivantes :

1. **Déploiement des Conteneurs :**
   - Paul a exécuté les playbooks Ansible pour déployer les conteneurs sur le réseau configuré par ContainerLab.
   - Chaque conteneur a été provisionné avec les configurations nécessaires pour les services à héberger grace au différents script de Jules.

2. **Configuration des Services :**
   - Matéo a installé les services sur les conteneurs déployés, en suivant les spécifications définies dans les playbooks Ansible.
   - Les services ont été configurés pour utiliser les ressources réseau et de stockage de manière optimale.

3. **Validation du Réseau :**
   - Jules a validé la configuration du réseau en s'assurant que tous les conteneurs pouvaient communiquer entre eux sans problème.
   - Des tests de connectivité et de performance ont été effectués pour garantir que le réseau répondait aux besoins de notre infrastructure.

Tests d'Intégration :
   - Une série de tests d'intégration a été réalisée pour vérifier que les services fonctionnent correctement ensemble.
   - Les tests ont inclus la vérification des connexions réseau, des accès aux services, et des performances globales.

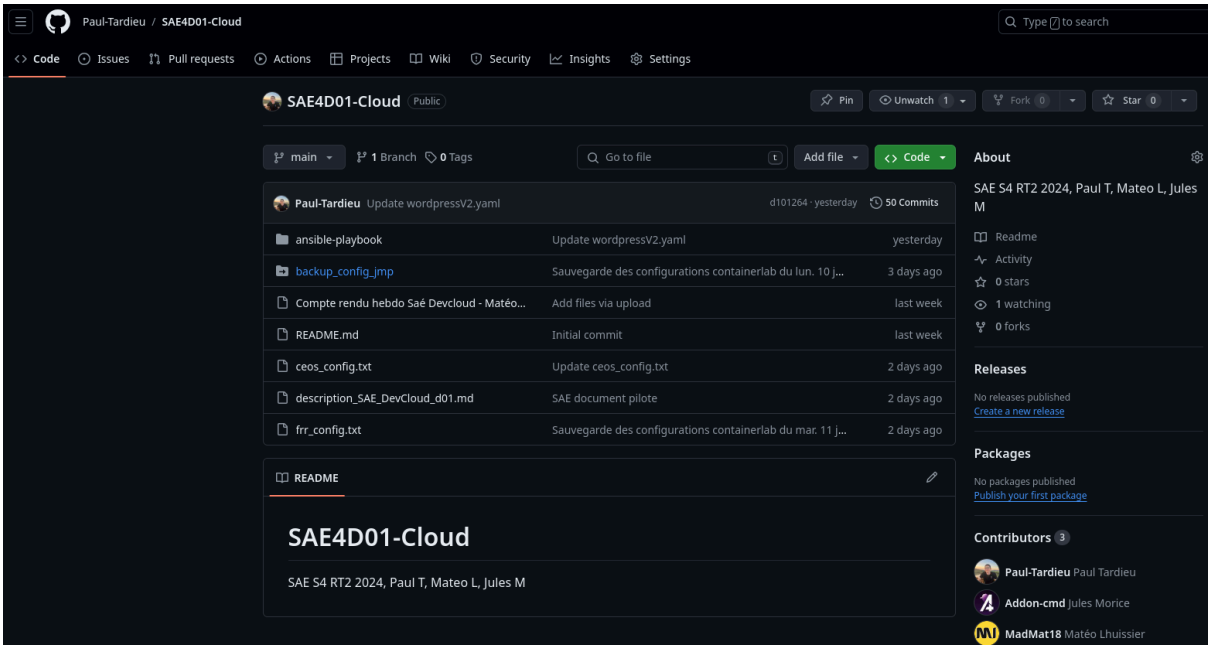
Grâce à cette étape de finalisation et de regroupement, nous avons pu constater que notre infrastructure était non seulement fonctionnelle mais qu'elle ne sont pas sécurisée par manque de temps. Les conteneurs déployés avec Ansible grâce à Paul ont parfaitement accueilli les services configurés par Matéo, et tout cela a été mis en réseau efficacement grâce au travail de Jules avec ContainerLab.

Annexe:

Rapport détaillé sur Ansible, Sémaphore et le déploiement de microservices :

Partie faite par Paul Tardieu

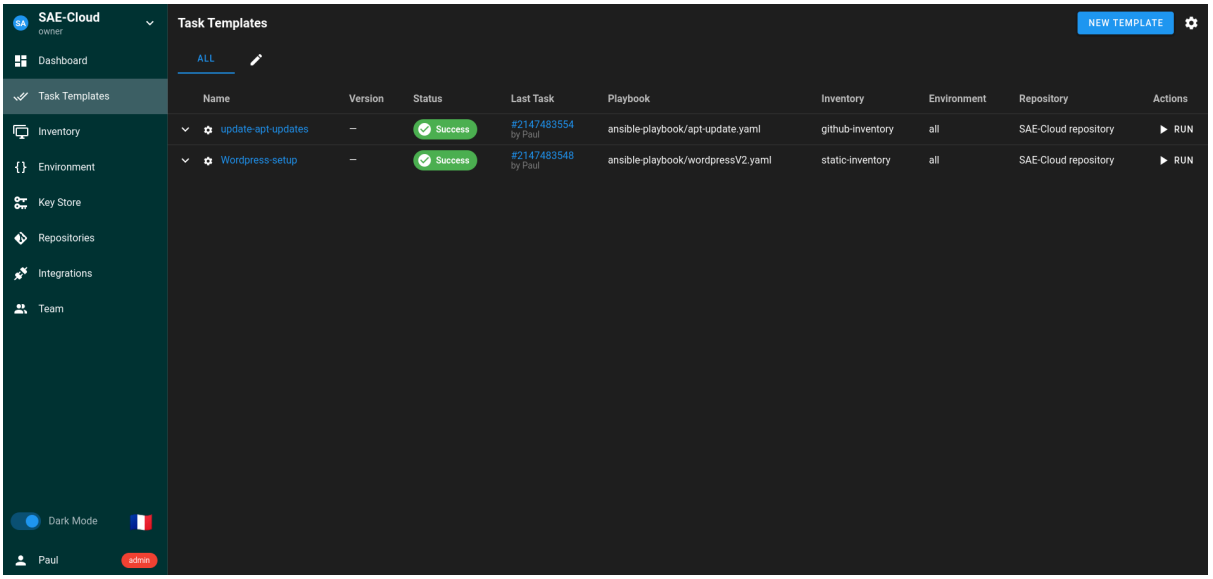
Ansible, ressources accessible sur le github commun:



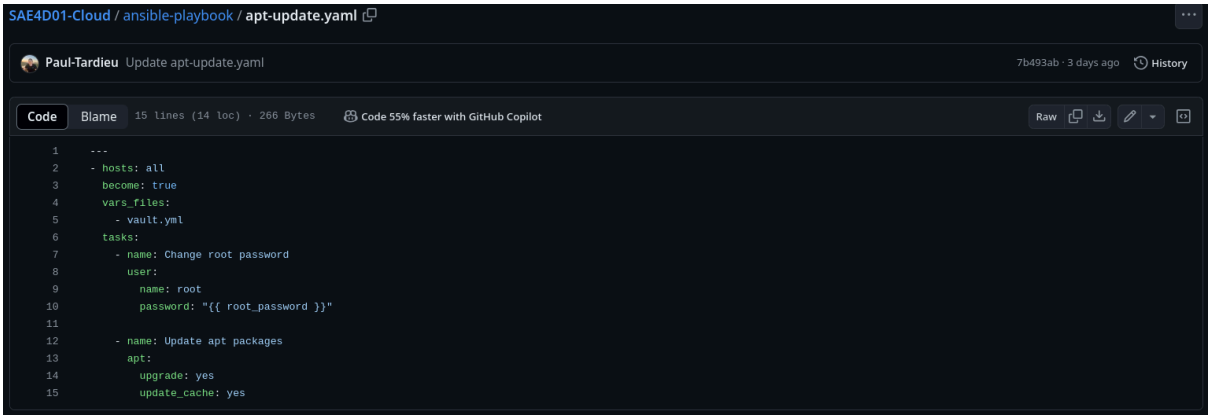
J'ai décidé de faire une VM sur Proxmox dédiée à Ansible afin de parallélisé le travail et d'avoir un environnement fixe et personnel de travail. Sur cette VM, une Debian 12.5, j'ai installé la dernière version d'Ansible et j'ai expérimenté sur une vm container lab cloné des première version faite par Jules afin de tester le déploiement de container via l'interface web Sémaphore. Mais j'ai d'abord testé via les lignes de commandes classiques:

```
root@debian:~/ansible-docker-wordpress# ansible all -m ping
10.202.202.2 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
```

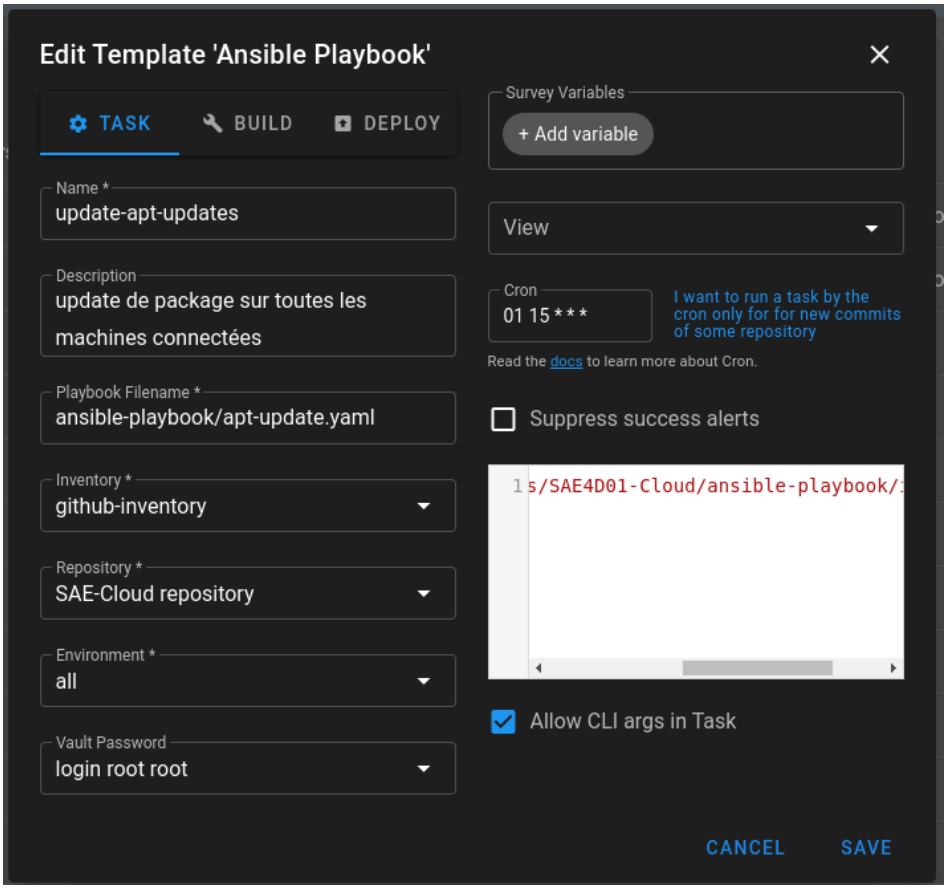
J'ai d'abord créé des fichiers ansible.cfg et un inventory en local puis je suis vite passé par le repository Github commun pour garder des traces de l'avancée du projet et avoir des playbooks dynamiques.
Afin d'expérimenter une solution que j'ai trouvé sur youtube qui me paraissait adaptée au projet, j'ai installé Sémaphore sur cette même VM, une fois l'app lancée, j'ai eu un accès à une interface web interagissant avec Ansible sur ma VM:



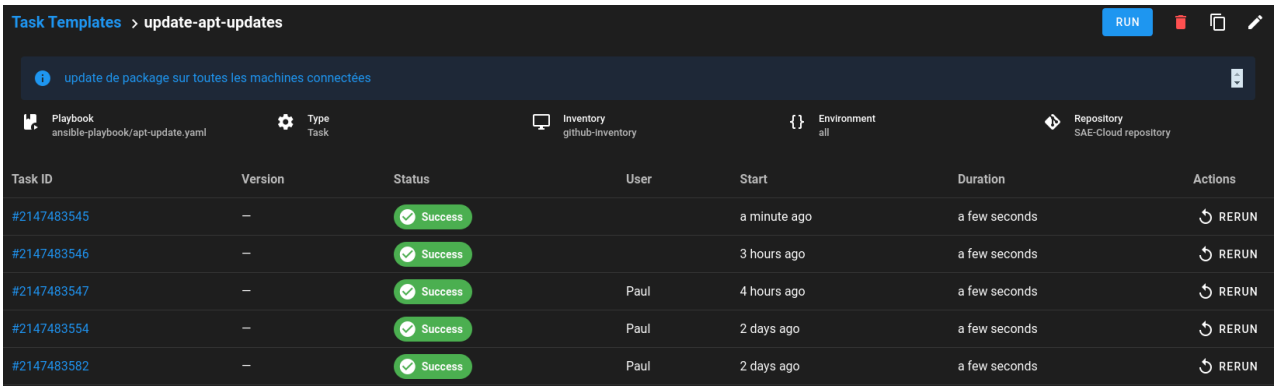
j’ai connecté mes playbooks depuis la section Repositories qui lie le repository Github à Sémaphore. J’ai d’abord testé d'exécuter un playbook simple visant à exécuter une mise à jour des paquets sur une machine cible:



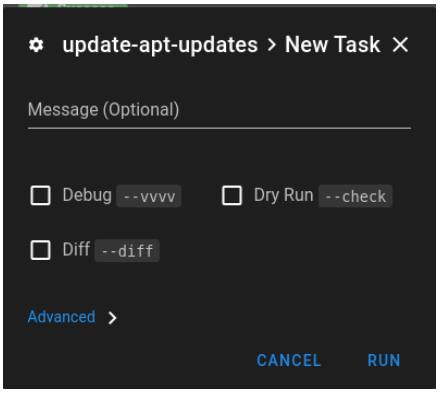
Une fois l’inventory opérationnel et la connexion à la machine cible établie avec succès grâce au partage de clé ssh entre machine, j’ai pu mettre en place une exécution automatique grâce à un paramètre Cron dans la configuration du templates suivants:



J'ai aussi accès à l'historique d'exécution des playbooks et l'utilisateur ayant lancée le playbook, si le playbook est lancée via Cron alors la section utilisateur est vide comme ceci:



Des options de lancement du playbook sont disponibles, l'option -vvvv m'as beaucoup aidé pour le détail des actions d'Ansible



Une fois le premier template mit en place j'ai pu créer un playbook pour le déploiement du conteneur wordpress et le conteneur Mysql.

Les tasks du playbooks sont:

Stop existing WordPress and MySQL containers:
permet de stopper les conteneurs pré existants afin de toujours pouvoir créer des conteneurs sans conflit de nom.

Remove existing WordPress and MySQL containers:
suppression des conteneurs pré existants avec le nom docker-wordpress-wordpress-1 et docker-wordpress-db-1 .

Create Docker Compose directory:
dossier de création du dockerfile et du docker compose sur la machine cible

```
root@docker:~/docker-wordpress# ls
docker-compose.yml  Dockerfile
```

Create Dockerfile for custom WordPress image:
j'ai du passer par un dockerfile pour pouvoir ajouter des paquets réseaux utile pour la suite

```
dest: /root/docker-wordpress/Dockerfile
content: |
  FROM wordpress:6.5.4
  RUN apt-get update && \
      apt-get install -y iproute2 iputils-ping nmap ncat && \
      rm -rf /var/lib/apt/lists/*
```

Build custom WordPress Docker image:
création d'une image custom-wordpress

Create Docker Compose file:
création du docker compose avec spécification des variables de chaque conteneurs:

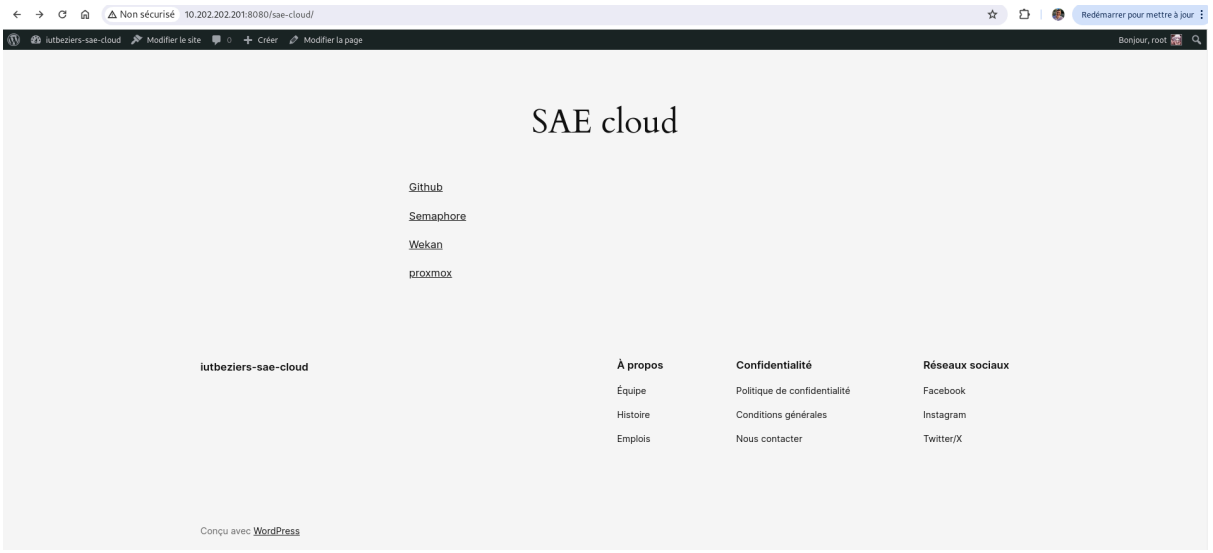
```
services:
  wordpress:
    image: custom_wordpress:latest
    restart: always
    ports:
      - 8080:80
    environment:
      WORDPRESS_DB_HOST: db
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD: wordpress
      WORDPRESS_DB_NAME: wordpress
    cap_add:
      - NET_ADMIN

  db:
    image: mysql:5.7
    restart: always
    environment:
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress
      MYSQL_ROOT_PASSWORD: somewordpress
```

Start WordPress with Docker Compose:
Exécution des conteneurs avec Docker Compose.

Voilà le playbook final pour le service Wordpress, j'ai aussi fait un playbook pour installer les dernières versions de docker et docker compose sur un hôte afin de ne pas avoir de conflit

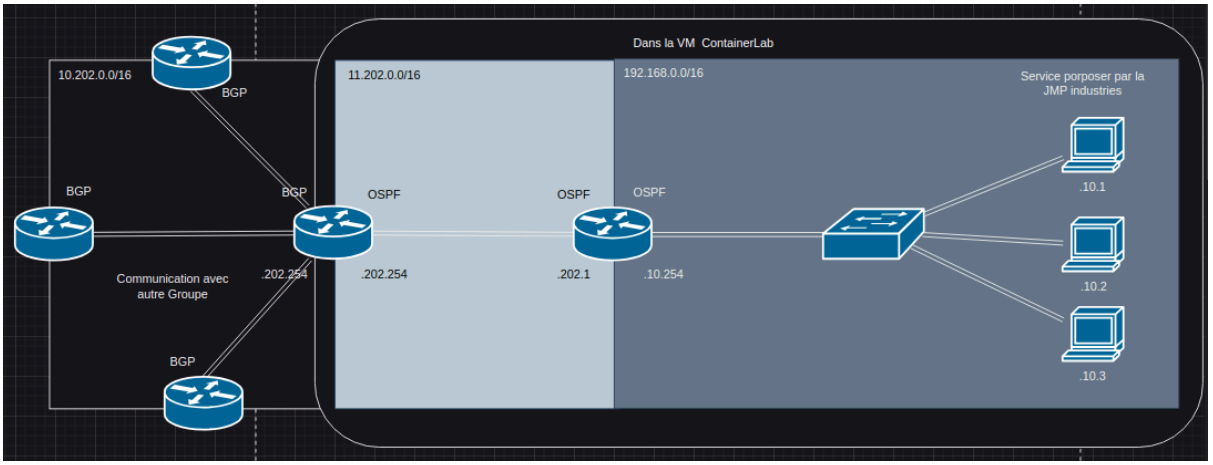
de versions de paquets logiciel entre machine. Je ne vais pas détailler le playbook mais il est sur le repository Github dans la section playbook comme on pourrait s'en douter :)
Site wordpress avec des liens utiles pour le groupe



Afin de déployer le service Wordpress sur le réseau container lab, j'ai collaboré avec Jules pour lier les conteneurs à container lab puis lier ensuite au réseau final avec la bonne ip via docker exec et les paquets installés en amont :

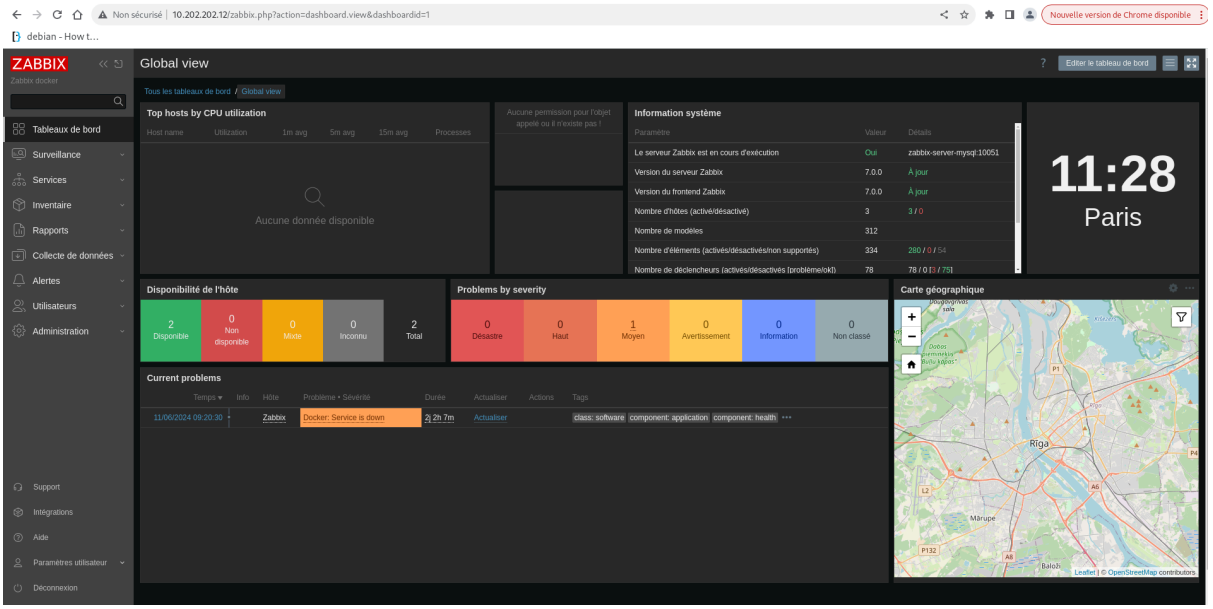
```
root@docker:~/docker-wordpress# docker exec -it 6f9887e70d15 bash
root@6f9887e70d15:/var/www/html# ip -br a
lo               UNKNOWN        127.0.0.1/8 ::1/128
eth0@if419       UP             192.168.10.1/16 fe80::a8c1:abff:fe48:ddb3/64
```

Schéma réseau final (Jules):

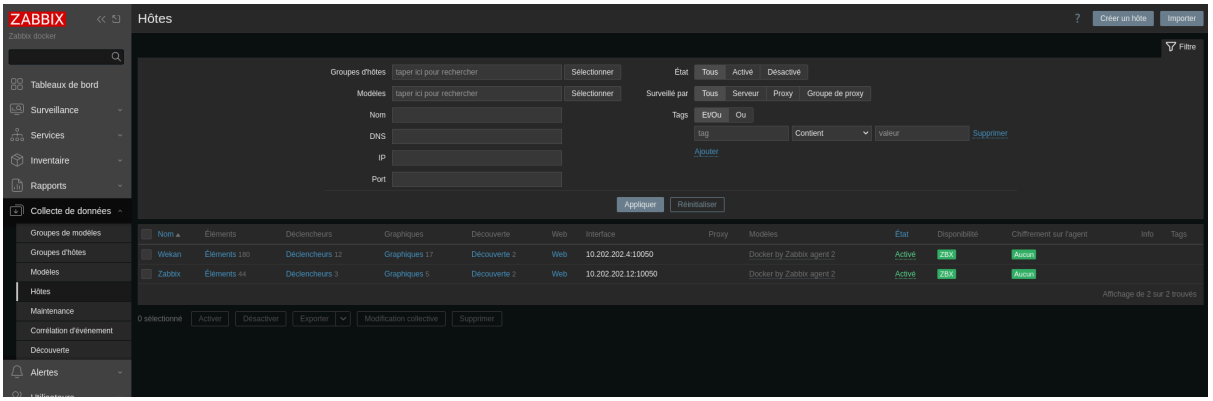


Partie faite par Matéo Lhuissier

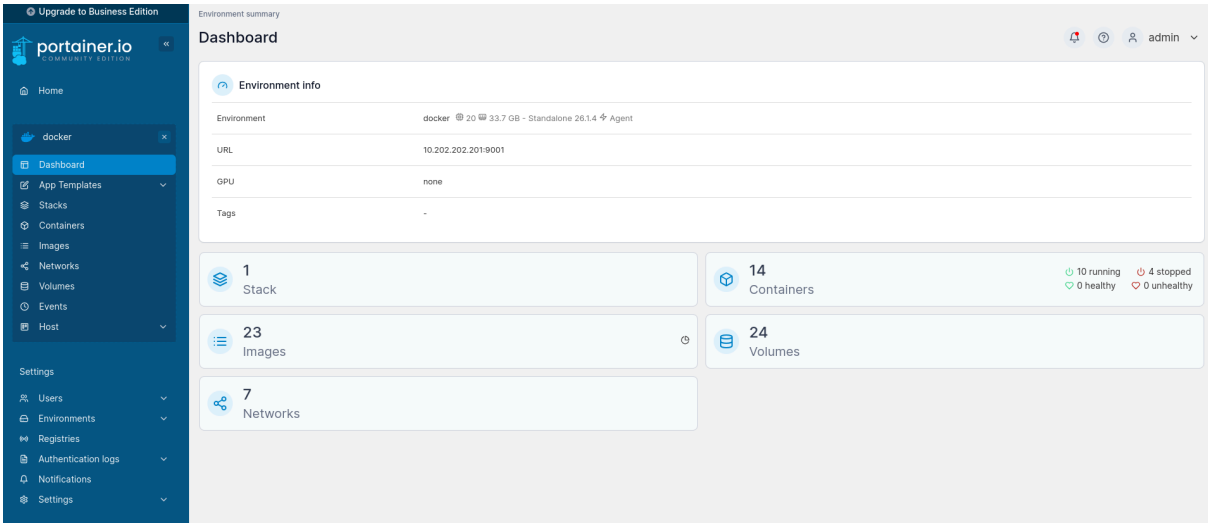
J'ai installé Zabbix à partir des containers docker mis à disposition dans la documentation officielle (https://www.zabbix.com/documentation/current/en/manual/installation/containers).
J'ai ensuite connecté mon container zabbix et wekan à l'aide d'un agent zabbix (https://hub.docker.com/r/zabbix/zabbix-agent2)
Grâce à l'interface web, j'ai pu retrouver mes containers et voir les erreurs à ceux-ci. Par exemple, quand j'ai éteint mon serveur, il y a eu une erreur comme quoi le docker était down. Zabbix permet la supervision des différents containers.



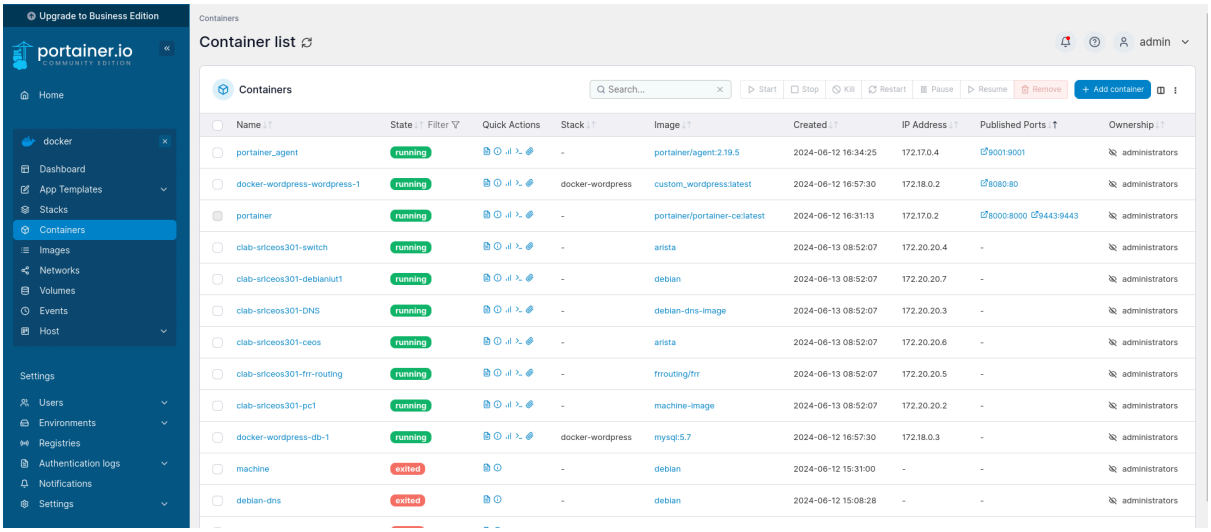
En allant un peu plus dans le détail des containers, on retrouve pour chacun leur état, leur ip avec le port d'écoute de l'agent, et d'autres informations utiles à la supervision.



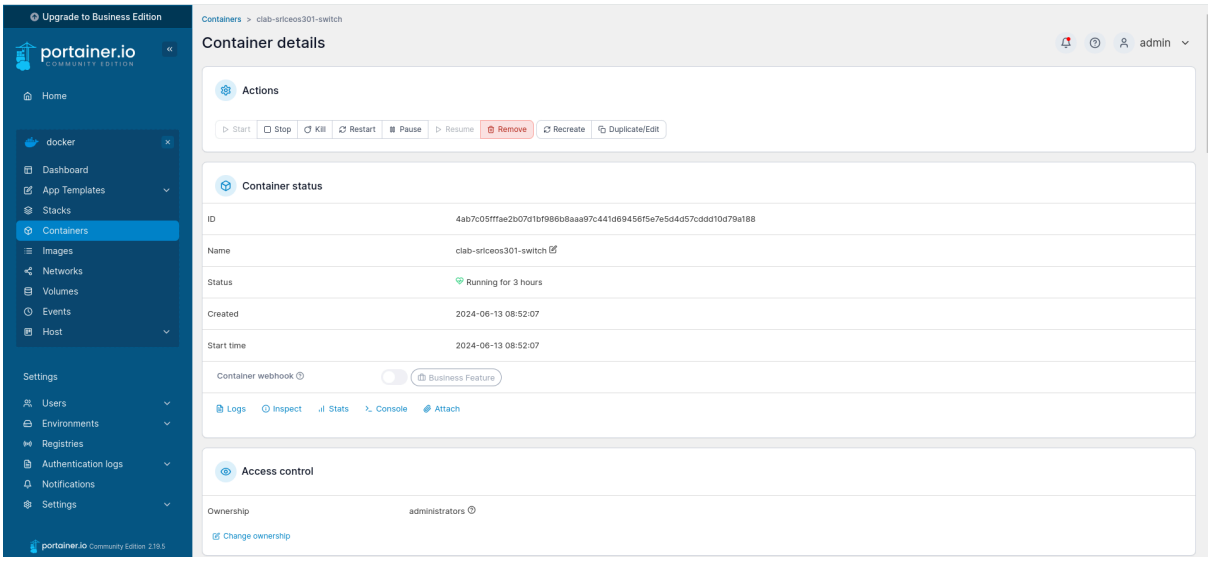
J'ai aussi installé Portainer via docker (https://docs.portainer.io/start/install-ce/server/docker/linux). J'ai d'abord appris à l'utiliser pour le déployer sur le réseau que Jules à fait. J'y ai donc connecté l'environnement docker de la VM de Jules pour avoir accès à tous les containers, images et autres, permettant ainsi de les gérer et de créer plus facilement des containers et autres.
Portainer permet aussi de gérer les volumes, images et networks de docker. Cela facilite le déploiement des containers.



On peut voir tous les dockers présents sur l'environnement importé. Pour chacun on y voit toutes ses informations, notamment le port sur lequel le service est déployé (mais aussi son état, son ip, …).



Quand on rentre dans le détail d'un container, on retrouve toutes ses caractéristiques. On peut aussi effectuer des actions sur celui-ci, comme le stoppé, le restart, le détruire,...



Quand on a voulu importer les dockers que j'avais créés sur la VM de Jules (donc dans le réseau interne), j'ai créé une image des dockers wekan pour faire un test. Je les ai ensuite envoyés sur sa VM grâce à scp pour qu'il redéploie les containers de son côté. Mais nous avons oublié qu'il fallait importer les volumes docker avec. Nous n'avons pas réussi à importer les volumes pour remonter le wekan.

```
root@wekan:~# docker ps
CONTAINER ID   IMAGE                          COMMAND                  CREATED      STATUS        PORTS
                                  NAMES
89d7f41c8e1d   portainer/agent:2.19.5         "./agent"                4 days ago   Up 23 hours   0.0.0.0:
9001->9001/tcp, :::9001->9001/tcp   portainer_agent
4d2b3620ad23   ghcr.io/wekan/wekan:latest     "bash -c 'ulimit -s …"   8 days ago   Up 23 hours   0.0.0.0:
80->8080/tcp, :::80->8080/tcp       wekan-app
f489fdc02e3b   mongo:6                        "docker-entrypoint.s…"   8 days ago   Up 23 hours   27017/tc
p                                   wekan-db
root@wekan:~# docker commit 4d2b3620ad23 wekanapp
sha256:09c146391466d8a6c4613355659646e7bfb41c238373f1a1519143bee053ccd1
root@wekan:~# docker commit f489fdc02e3b wekandb
sha256:e75c3aeaf3b498e07764f0243916571f5839093cfd217514256babce23b609b3
root@wekan:~# docker save -o wekanapp.tar wekanapp
root@wekan:~# docker save -o wekandb.tar wekandb
root@wekan:~# ls
 docker-compose.yml                              wekandb.tar
'docker-desktop-4.30.0-amd64.deb?utm_source=docker'   wget-log
 wekanapp.tar                                    zabbix-release_6.4-1+debian11_all.deb
root@wekan:~# scp wekanapp.tar test@10.202.202.201:/home/test
The authenticity of host '10.202.202.201 (10.202.202.201)' can't be established.
ECDSA key fingerprint is SHA256:NaQtvSvhJi3IW0GVXbdaVaomSJxRZ8S4ldOJXe2osbM.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.202.202.201' (ECDSA) to the list of known hosts.
test@10.202.202.201's password:
wekanapp.tar                                                  100%  933MB 143.0MB/s   00:06
root@wekan:~# scp wekandb.tar test@10.202.202.201:/home/test
test@10.202.202.201's password:
wekandb.tar                                                   100%  697MB 145.2MB/s   00:04
root@wekan:~#
```

Partie faite par Jules Morice

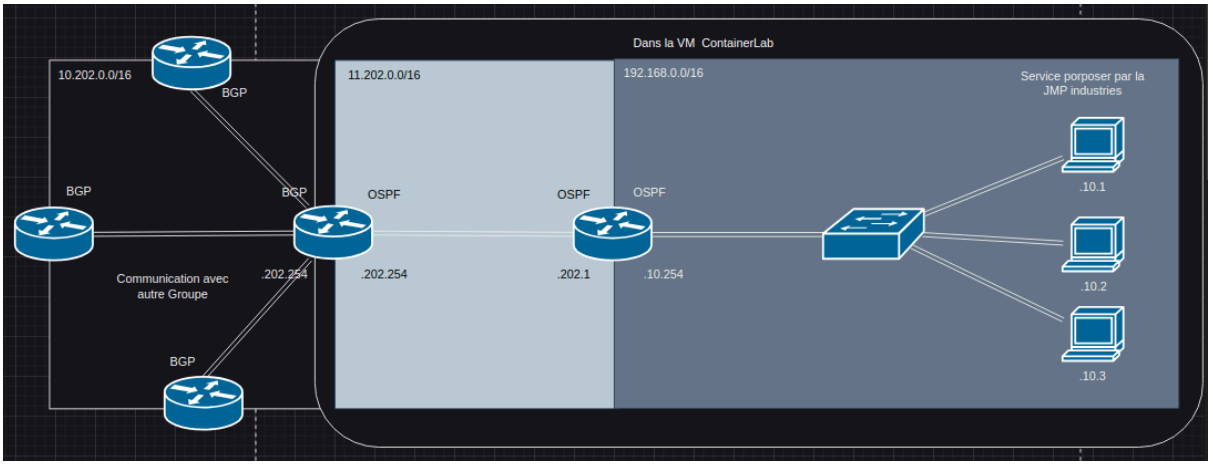
J'ai été chargé de déployer ContainerLab et de configurer l'infrastructure réseau pour notre projet de SARE. Cette tâche a impliqué la mise en place de réseaux privés, la configuration de routeurs Arista, et l'implémentation de protocoles de routage tels que OSPF et BGP. Voici un résumé détaillé de ce que j'ai fait, des scripts de configuration utilisés, des problèmes rencontrés, et des améliorations possibles.

Déploiement de ContainerLab

Objectif : Mettre en place une infrastructure réseau virtualisée utilisant ContainerLab pour supporter nos services et conteneurs déployés via Ansible.

1. Installation de ContainerLab :

- J'ai commencé par installer ContainerLab sur notre serveur AlmaLinux. ContainerLab est un outil puissant qui permet de déployer des topologies réseau complexes en utilisant des conteneurs.
- Les dépendances nécessaires ont été installées, incluant Docker et les outils réseau requis.



2. **Configuration Initiale :**

- J'ai créé un fichier de configuration YAML pour définir notre topologie réseau. Ce fichier spécifie les nœuds (routeurs, switches) et les liens entre eux.

```yaml
name: srlceos400

topology:
  nodes:
    ceos:
      kind: arista_ceos
      image: arista
    frr-routing:
      kind: linux
      image: frrouting/frr
      binds:
        - router1/daemons:/etc/frr/daemons
    switch:
      kind: arista_ceos
      image: arista
    debianiut1:
      kind: linux
      image: debian
    docker-wordpress-wordpress-1:
      kind: ext-container
    DNS:
      kind: linux
      image: debian-dns-image
    pc1:
      kind: linux
      image: machine-image
    docker-wordpress-db-1:
      kind: ext-container

  links:
    - endpoints: ["ceos:eth2", "frr-routing:eth1"]
    - endpoints: ["ceos:eth1", "macvlan:ens18"]
    - endpoints: ["frr-routing:eth2", "switch:eth1"]
    - endpoints: ["switch:eth2", "debianiut1:eth1"]
    - endpoints: ["switch:eth4", "DNS:eth1"]
    - endpoints: ["switch:eth5", "pc1:eth1"]
    - endpoints: ["switch:eth6", "docker-wordpress-wordpress-1:eth0"]
    - endpoints: ["switch:eth7", "docker-wordpress-db-1:eth0"]
```

Configuration des Routeurs Arista et OSPF/BGP

Objectif : Configurer les routeurs Arista pour qu'ils communiquent en utilisant OSPF et BGP.

1. **Script de Configuration pour les Routeurs Arista :**

Voici un extrait du script de configuration pour un routeur Arista :

```
#!/bin/bash

# Configure CEOS
docker exec -i clab-srlceos301-ceos Cli << 'EOF'
enable
configure terminal
no aaa root

username admin privilege 15 role network-admin secret sha512 \$6\$i11PnoGHoTmynn0c\$/RCPMkyStp0x1u2RFENEqon7y26wP8N2f/uFjHquUdOvY.uu2CPwOaonyfe/Rc7/njZ6LDuRB9eYZ0aAeCIwI1

transceiver qsfp default-mode 4x10G

service routing protocols model multi-agent

hostname ceos

spanning-tree mode mstp

system l1
   unsupported speed action error
   unsupported error-correction action error
exit

management api http-commands
   no shutdown
exit

management api gnmi
   transport grpc default
exit

management api netconf
   transport ssh default
exit

interface Ethernet1
   mtu 1500
   no switchport
   ip address 10.202.202.254/16
exit

interface Ethernet2
   mtu 1500
   no switchport
   ip address 11.202.202.254/16
```

2. **Configuration d'OSPF :**
   - J'ai configuré OSPF pour établir les connexions internes entre les routeurs. Chaque routeur a été configuré pour annoncer ses réseaux directement connectés.

```
#!/bin/bash

# Configure FRR
docker exec -i clab-srlceos480-frr-routing vtysh << 'EOF'
configure terminal

hostname frr-routing
no ipv6 forwarding

interface eth1
 ip address 11.202.202.1/16
 ip ospf network point-to-point
exit

interface eth2
 ip address 192.168.10.254/16
 ip ospf passive
exit

router ospf
 network 11.202.0.0/16 area 0
 network 192.168.0.0/16 area 0
 redistribute bgp
exit

end
write memory
EOF

# Set MTU for the interfaces
docker exec -it clab-srlceos301-frr-routing ip link set eth1 mtu 1500
docker exec -it clab-srlceos301-frr-routing ip link set eth2 mtu 1500

# Restart the interfaces to apply MTU change
docker exec -it clab-srlceos301-frr-routing ip link set eth1 down
docker exec -it clab-srlceos301-frr-routing ip link set eth1 up
docker exec -it clab-srlceos301-frr-routing ip link set eth2 down
docker exec -it clab-srlceos301-frr-routing ip link set eth2 up

# Verify MTU settings
docker exec -it clab-srlceos301-frr-routing ip link show eth1 | grep mtu
docker exec -it clab-srlceos301-frr-routing ip link show eth2 | grep mtu
docker exec -it clab-srlceos301-frr-routing vtysh << 'EOF'
show ip interface brief
show ip route
```

3. **Configuration de BGP :**
   - BGP a été configuré pour l'échange de routes entre différents systèmes autonomes (AS). Les voisins BGP ont été définis pour chaque routeur.

```
!
router bgp 23
   neighbor 10.202.3.20 remote-as 22
   network 11.202.0.0/16
   redistribute ospf
!
```

Implémentation des Règles de Sécurité

Objectif : Assurer la sécurité du réseau privé en utilisant des règles NAT et d'autres mécanismes de sécurité.

1. NAT pour Isoler le Réseau Privé :
   - J'ai implémenté le NAT pour permettre aux conteneurs de communiquer avec le réseau externe tout en isolant le réseau privé.
   - Voici un exemple de configuration NAT sur un routeur :

```
sysctl -w net.ipv4.ip_forward=1
```

```
iptables -t nat -A POSTROUTING -o eth1 -j MASQUERADE
iptables -A FORWARD -i eth2 -o eth1 -j ACCEPT
iptables -A FORWARD -m state --state RELATED,ESTABLISHED -j ACCEPT
```

Problèmes Rencontrés

1. **Problèmes de Connectivité :**
   - J'ai rencontré des problèmes de connectivité initiale entre les routeurs, principalement dus à des erreurs de configuration dans les adresses IP et les masques de sous-réseau.

2. **Erreurs de Configuration OSPF/BGP :**
   - Il y a eu des erreurs dans la configuration OSPF et BGP, notamment des erreurs de syntaxe et des paramètres incorrects, qui ont empêché l'établissement des adjacences.

3. **Problèmes de NAT :**
   - La mise en place du NAT a été délicate, avec des problèmes de correspondance des règles NAT qui ont empêché la communication entre les réseaux interne et externe.

Améliorations Possibles

1. Automatisation des Configurations :
- Utiliser des outils d'automatisation comme Ansible pour déployer et configurer les routeurs de manière plus efficace et réduire les erreurs humaines.

2. Documentation Plus Détaillée :
- Créer une documentation plus détaillée pour chaque étape de la configuration, incluant des schémas de topologie et des exemples de configurations.

3. Tests de Connectivité :
- Mettre en place des scripts de tests automatiques pour vérifier la connectivité et les routes après chaque changement de configuration.

Travail de groupe :

Le groupe à réussi à collaborer facilement et à travailler en parallèle en partie grâce au Wekan mis en place tôt dans la SAE.
Chacun avait une partie bien claire et pouvait interroger le groupe afin d'avoir un avis extérieur sur comment gérer certaines tâches et comment régler certains problèmes.
La mise en commun finale à rencontrer des difficultés sur certains points empêchant la mise en place de certains services sur le réseau final container lab.