Olivier Gabelle – FISA 2A - 21/01/2026

# RAPPORT DE TRAVAUX PRATIQUES

## NET8552 – Orchestration de Services Réseau

On souhaite orchestrer des services réseau efficacement : aujourd'hui les composants d'une infrastructure sont des machines virtuelles, ce qui signifie qu'il faut pouvoir étendre et automatiser le déploiement de ces conteneurs facilement et rapidement : c'est l'orchestration.

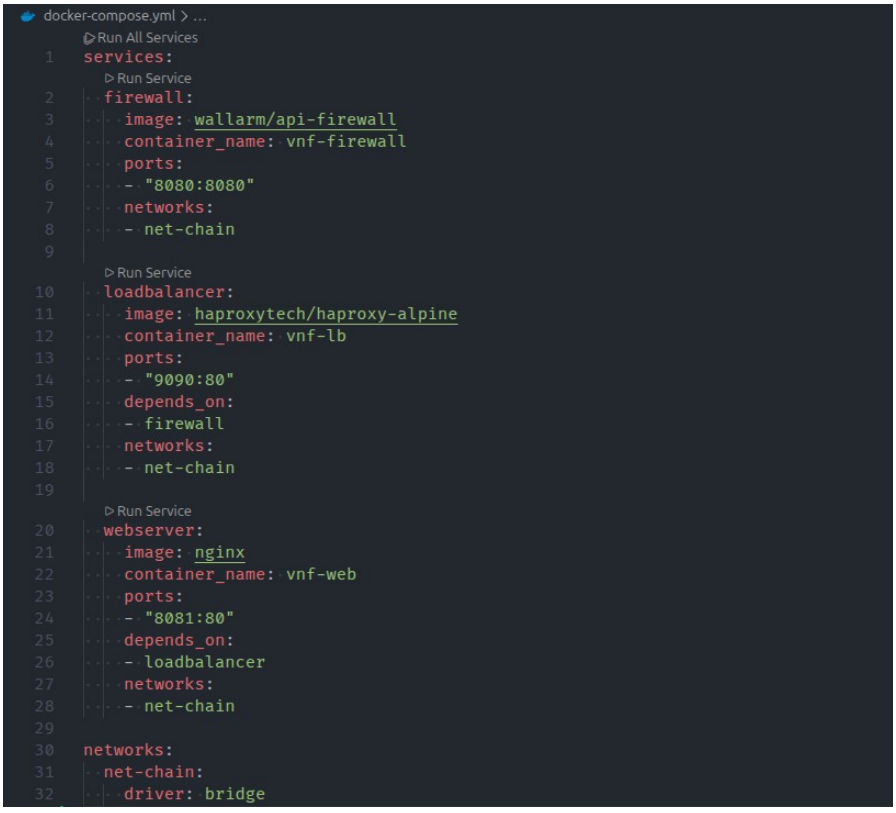
Dans le cas des réseaux, chaque conteneur est une fonction réseau, ce qui correspond à une petite unité de fonctionnement qui a un rôle précis : pare-feu, serveur web, load-balancer par exemple sont des fonctions réseau (VNF). En déployant séparément les VNFs on gagne en flexibilité.

On va déployer la chaîne de deux manières différentes :

- Docker compose
- Tosca et xOpera
- Ansible

### 4.1 – Déploiement de conteneurs Docker

Voici le fichier docker-compose.yml :



```
[~/Documents/TP_Reseaux_Orchestration]
base  olivier  main -  docker inspect vnf-web | grep IPAddress
            "SecondaryIPAddresses": null,
            "IPAddress": "",
                    "IPAddress": "172.21.0.2",

[~/Documents/TP_Reseaux_Orchestration]
base  olivier  main -  docker inspect vnf-lb | grep IPAddress
            "SecondaryIPAddresses": null,
            "IPAddress": "",
                    "IPAddress": "172.21.0.3",
```

Chaque service possède une image précise qui contient le système d'exploitation et les installations nécessaires au fonctionnement du service en question.

La clé « depends_on » permet de spécifier les dépendances entre les conteneurs : par exemple le serveur web dépend du load-balancer, et le load-balancer dépend du firewall.

La clé networks permet de créer un sous réseau virtuel pour accueillir les trois conteneur et qu'ils puissent communiquer entre eux.

Avec docker ps on remarque le lancement des conteneurs :

```
[~/Documents/TP_Reseaux_Orchestration]
✘ base  olivier  docker ps
CONTAINER ID   IMAGE                          COMMAND                  CREATED
   STATUS         PORTS                                      NAMES
0866f6bcf911   nginx                          "/docker-entrypoint.…"   About an hour ago
   Up 3 seconds   0.0.0.0:8081->80/tcp, [::]:8081->80/tcp   vnf-web
4066b28604e3   haproxytech/haproxy-alpine     "/docker-entrypoint.…"   About an hour ago
   Up 4 seconds   0.0.0.0:9090->80/tcp, [::]:9090->80/tcp   vnf-lb
```

## 4.2 – Modélisation TOSCA

Voici le fichier service-chain.yaml :

service-chain-tosca.yaml

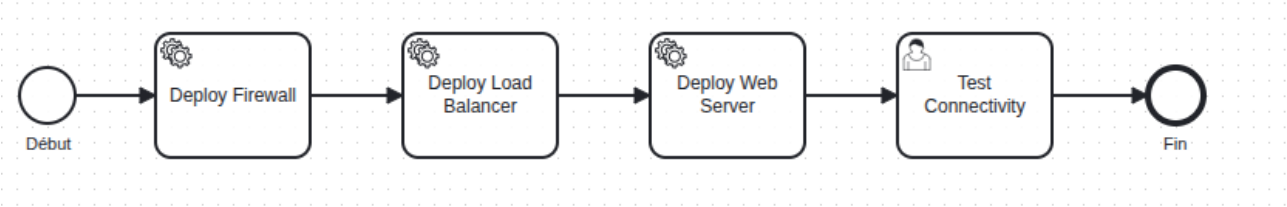
```yaml

topology_template:
  node_templates:
    Firewall:
      type: tosca.nodes.Container.Application
      properties:
        image: "wallarm/api-firewall"

    LoadBalancer:
      type: tosca.nodes.Container.Application
      properties:
        image: "haproxytech/haproxy-alpine"
      requirements:
        - dependency: Firewall
    WebServer:
      type: tosca.nodes.Container.Application
      properties:
        image: "nginx"
      requirements:
        - dependency: LoadBalancer
```

On a le même type de dépendances qu'avec docker compose, sauf que la syntaxe est différente.

Le graphe TOSCA associé à ce fichier est le suivant :




## 4.3 – Orchestration avec xOpera

On lance opera deploy et on a le résultat suivant :

```
[Worker_0]   Deploying Firewall_0
[Worker_0]     Executing create on Firewall_0
{
  "inputs": {}
}
[Worker_0] ------------
{
  "custom_stats": {},
  "global_custom_stats": {},
  "stats": {
    "localhost": {
      "changed": 1,
      "failures": 0,
      "ignored": 0,
      "ok": 2,
      "rescued": 0,
      "skipped": 0,
      "unreachable": 0
    }
  }
}
[Worker_0] ------------
[Worker_0] ============
[Worker_0]   Deployment of Firewall_0 complete
[Worker_0]   Deploying LoadBalancer_0
[Worker_0]     Executing create on LoadBalancer_0
{
  "inputs": {}
}
[Worker_0] ------------
{
  "custom_stats": {},
  "global_custom_stats": {},
  "stats": {
    "localhost": {
      "changed": 1,
      "failures": 0,
      "ignored": 0,
      "ok": 2,
      "rescued": 0,
      "skipped": 0,
      "unreachable": 0
    }
  }
}
```

Si on fait docker ps on voit les containers :

```
opera  olivier  main -  docker ps
CONTAINER ID   IMAGE                          COMMAND                  CREATED
STATUS          PORTS                                          NAMES
195ba261f991   nginx                          "/docker-entrypoint.…"   41 seconds ago
Up 40 seconds   0.0.0.0:8081->80/tcp, [::]:8081->80/tcp   vnf-web
fca3b559407d   haproxytech/haproxy-alpine     "/docker-entrypoint.…"   46 seconds ago
Up 45 seconds   0.0.0.0:9090->80/tcp, [::]:9090->80/tcp   vnf-lb
```

On peut retirer le déploiement :

```
opera  olivier  main -  opera undeploy
/home/olivier/anaconda3/envs/opera/lib/python3.10/site-packages/opera/cli.py:5: UserWa
rning: pkg_resources is deprecated as an API. See https://setuptools.pypa.io/en/latest
/pkg_resources.html. The pkg_resources package is slated for removal as early as 2025-
11-30. Refrain from using this package or pin to Setuptools<81.
  import pkg_resources
[Worker_0]   Undeploying WebServer_0
[Worker_0]   Undeployment of WebServer_0 complete
[Worker_0]   Undeploying LoadBalancer_0
[Worker_0]   Undeployment of LoadBalancer_0 complete
[Worker_0]   Undeploying Firewall_0
[Worker_0]   Undeployment of Firewall_0 complete
```

## 4.4 – Workflow BPMN

Le fichier workflow.bpmn est un fichier XML qui décrit le diagramme précédent :

network-service-chain.bpmn > bpmn:definitions > bpmndi:BPMNDiagram > bpmndi:BPMNPlane > bpmndi:BPMNShape > dc:Bounds

```xml
<?xml version="1.0" encoding="UTF-8"?>
<bpmn:definitions xmlns:bpmn="http://www.omg.org/spec/BPMN/20100524/MODEL" xmlns:bpmndi="ht
  <bpmn:process id="Process_ServiceChain" isExecutable="false">
    <bpmn:startEvent id="StartEvent_1" name="Début">
      <bpmn:outgoing>Flow_1</bpmn:outgoing>
    </bpmn:startEvent>
    <bpmn:serviceTask id="Task_Firewall" name="Deploy Firewall">
      <bpmn:incoming>Flow_1</bpmn:incoming>
      <bpmn:outgoing>Flow_2</bpmn:outgoing>
    </bpmn:serviceTask>
    <bpmn:sequenceFlow id="Flow_1" sourceRef="StartEvent_1" targetRef="Task_Firewall" />
    <bpmn:serviceTask id="Task_LB" name="Deploy Load Balancer">
      <bpmn:incoming>Flow_2</bpmn:incoming>
      <bpmn:outgoing>Flow_3</bpmn:outgoing>
    </bpmn:serviceTask>
    <bpmn:sequenceFlow id="Flow_2" sourceRef="Task_Firewall" targetRef="Task_LB" />
    <bpmn:serviceTask id="Task_Web" name="Deploy Web Server">
      <bpmn:incoming>Flow_3</bpmn:incoming>
      <bpmn:outgoing>Flow_4</bpmn:outgoing>
    </bpmn:serviceTask>
    <bpmn:sequenceFlow id="Flow_3" sourceRef="Task_LB" targetRef="Task_Web" />
    <bpmn:userTask id="Task_Test" name="Test Connectivity">
      <bpmn:incoming>Flow_4</bpmn:incoming>
      <bpmn:outgoing>Flow_5</bpmn:outgoing>
    </bpmn:userTask>
    <bpmn:sequenceFlow id="Flow_4" sourceRef="Task_Web" targetRef="Task_Test" />
    <bpmn:endEvent id="EndEvent_1" name="Fin">
      <bpmn:incoming>Flow_5</bpmn:incoming>
    </bpmn:endEvent>
    <bpmn:sequenceFlow id="Flow_5" sourceRef="Task_Test" targetRef="EndEvent_1" />
  </bpmn:process>
```

Le diagramme permet de modéliser visuellement les étapes du déploiement. On peut également y trouver les dépendances entre les VNFs et aussi la distinction entre une tâche Service ou une tâche Utilisateur.

On peut utiliser Ansible pour faire le déploiement :

```
opera  olivier  main -  cat deploy_ansible.sh
#!/bin/bash

# Arrêter le script si une commande échoue
set -e

echo "========================================="
echo "   DÉMARRAGE DU DÉPLOIEMENT MANUEL (ANSIBLE)"
echo "========================================="

# 1. Déploiement du Firewall
echo "[1/3] Exécution du playbook Firewall..."
ansible-playbook playbooks/deploy_firewall.yaml

# 2. Déploiement du Load Balancer
echo "[2/3] Exécution du playbook Load Balancer..."
ansible-playbook playbooks/deploy_loadbalancer.yaml

# 3. Déploiement du Web Server
echo "[3/3] Exécution du playbook Web Server..."
ansible-playbook playbooks/deploy_webserver.yaml

echo "========================================="
echo "   DÉPLOIEMENT TERMINÉ AVEC SUCCÈS"
echo "========================================="

# Vérification finale
echo "État des conteneurs :"
docker ps --format "table {{.Names}}\t{{.Status}}\t{{.Ports}}"%
```

La sortie est la suivante :

```
opera  olivier  main -  ./deploy_ansible.sh
=========================================
   DÉMARRAGE DU DÉPLOIEMENT MANUEL (ANSIBLE)
=========================================
[1/3] Exécution du playbook Firewall...
[WARNING]: provided hosts list is empty, only localhost is available. Note that the
implicit localhost does not match 'all'

PLAY [Déploiement du VNF Firewall] *********************************************

TASK [Gathering Facts] *********************************************************
ok: [localhost]

TASK [Lancer le conteneur Firewall] ********************************************
changed: [localhost]

PLAY RECAP *********************************************************************
localhost                  : ok=2    changed=1    unreachable=0    failed=0    skipped
=0    rescued=0    ignored=0

[2/3] Exécution du playbook Load Balancer...
[WARNING]: provided hosts list is empty, only localhost is available. Note that the
implicit localhost does not match 'all'

PLAY [Déploiement du VNF Load Balancer] ****************************************

TASK [Gathering Facts] *********************************************************
ok: [localhost]

TASK [Lancer le conteneur Load Balancer] ***************************************
changed: [localhost]

PLAY RECAP *********************************************************************
localhost                  : ok=2    changed=1    unreachable=0    failed=0    skipped
=0    rescued=0    ignored=0

[3/3] Exécution du playbook Web Server...
[WARNING]: provided hosts list is empty, only localhost is available. Note that the
implicit localhost does not match 'all'

PLAY [Déploiement du VNF Web Server] *******************************************
```

```
==========================================
   DÉPLOIEMENT TERMINÉ AVEC SUCCÈS
==========================================
État des conteneurs :
NAMES     STATUS                  PORTS
vnf-web   Up Less than a second   0.0.0.0:8081->80/tcp, [::]:8081->80/tcp
vnf-lb    Up 5 seconds            0.0.0.0:9090->80/tcp, [::]:9090->80/tcp
```

Comme on peut le remarquer, la VNF du Firewall n'est pas lancée (il y a des erreurs). J'ai remplacé l'image du firewall par une autre image, plus facile à configurer. Il a aussi fallu ajouter un fichier haproxy.cfg pour définir correctement le comportement du load-balancer.

```
   DÉPLOIEMENT TERMINÉ AVEC SUCCÈS
==========================================
État des conteneurs :
NAMES          STATUS                  PORTS
vnf-web        Up Less than a second   0.0.0.0:8081->80/tcp, [::]:8081->80/tcp
vnf-lb         Up 5 seconds            0.0.0.0:9090->80/tcp, [::]:9090->80/tcp
vnf-firewall   Up 11 seconds           0.0.0.0:8080->8080/tcp, [::]:8080->8080/tcp
```

Grâce à ça, tous les CURLs sont fonctionnels :

```
opera  olivier  main -  curl http://localhost:9090
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

```
opera  olivier  main -  curl http://localhost:8080
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

```
opera  olivier  main -  curl http://localhost:8081
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

Si on regarde les logs des VNFs :

- Du côté du serveur web NGINX, les requêtes passent :

```
2026/01/21 10:14:39 [notice] 1#1: start worker process 44
172.22.0.3 - - [21/Jan/2026:10:15:40 +0000] "GET / HTTP/1.1" 200 615 "-" "curl/8.5.0"
"-"
172.22.0.3 - - [21/Jan/2026:10:15:42 +0000] "GET / HTTP/1.1" 200 615 "-" "curl/8.5.0"
"-"
172.22.0.1 - - [21/Jan/2026:10:15:43 +0000] "GET / HTTP/1.1" 200 615 "-" "curl/8.5.0"
"-"
```

- Pareil pour le load balancer :

```
the configuration manual).
[NOTICE]   (1) : Loading success.
Connect from 172.22.0.1:50834 to 172.22.0.3:80 (my_frontend/HTTP)
Connect from 172.22.0.2:59774 to 172.22.0.3:80 (my_frontend/HTTP)
```

## Conclusion et résultats

Nous avons exploré trois moyens différents de déployer les VNFs et de les orchestrer.

Voici les différences notables :

- Docker : utilisation de docker-compose pour décrire les images et un réseau commun aux VNFs

- xOpera : utilisation d'un fichier YAML service-chain pour préciser les dépendances. Utilisation de YAML « playbook » pour décrire les commandes docker à exécuter.

- Ansible : utilisation des YAML « playbook » mais commandes différentes pour le déploiement.

Concernant les tests réseaux, le résultat est identique.

**Différences entre TOSCA et BPMN :**

TOSCA est une approche déclarative qui dit *quoi* déployer dans l'infrastructure, en définissant une topologie dans le fichier service-chain.yaml.

BPMN est une approche impérative qui se concentre sur le processus : il décrit *quand* et *comment* les composants agissent dans l'infrastructure. Par exemple « Test Connectivity » n'est pas une brique logicielle, mais elle est présente dans le diagramme.

**Séparation xOpera et Camunda**

Séparer les deux permet de répartir les tâches, d'un côté la préparation de l'implémentation (xOpera) et d'un autre côté la spécification de l'infrastructure (Camunda).

**Limites**

La courbe d'apprentissage n'est pas la même entre Docker, xOpera et Ansible. Il est plus simple d'utiliser docker, mais avec des infrastructures complexes, cela pourrait ne pas correspondre. À l'inverse, utiliser Ansible pour un simple projet d'exemple est peu pertinent car complexe.