

TÉLÉCOM SUDPARIS

Département : Informatique et Réseaux

TRAVAUX PRATIQUES

NET8552 – Orchestration de Services Réseau

ÉNONCÉ DU TP

Déploiement et orchestration d'une chaîne de fonctions réseau : Firewall → Load Balancer
→ Web Server

Objectif : Déployer un service réseau en chaîne composé de trois VNFs : Firewall, Load Balancer, Web Server.

Outils : Docker, Docker Compose, TOSCA, xOpera, BPMN (Camunda Modeler).

Environnement : VM Ubuntu 20.04 LTS.

Images Docker à utiliser :

- Firewall : wallarm/api-firewall
- Load Balancer : haproxytech/haproxy-alpine
- Web Server : nginx

Encadré par : Pr Walid GAALOUL

Réalisé par :

Table des matières

I.	Objectif du TP	2
II.	Pré-requis	2
III.	Étapes du TP	2
1.	Création des conteneurs Docker (Docker Compose)	2
2.	Modèles TOSCA pour décrire les VNFs	3
3.	Modélisation du workflow (BPMN)	3
4.	Orchestration avec TOSCA	4
4.1.	Étape 1 : Préparer l'environnement	4
4.2.	Étape 2 : Organiser ton projet TOSCA	4
4.3.	Étape 3 : Définir le modèle TOSCA (service-chain.yaml)	5
4.4.	Étape 4 : Écrire les playbooks d'exécution (Ansible YAML)	5
4.5.	Étape 5 : Lancer l'orchestration	6
4.6.	Étape 6 : Vérifier le déploiement	6
4.7.	Étape 7 : Nettoyer l'environnement	7
4.8.	Étape 8 : Bonnes pratiques / Vérifications	7
IV.	Livrable attendu	7
1.	Un rapport au format PDF/Word contenant (Voir modèle vierge) :	7
2.	Un dépôt GitHub avec :	7

TP Orchestration de Services – NET8552

Déploiement et orchestration d'une chaîne de fonctions réseau : Firewall → Load Balancer → Web Server

I. Objectif du TP

- Déployer une **chaîne de services réseau** composée de trois VNFs :
 1. **Firewall** → 2. **Load Balancer** → 3. **Web Server**
- Chaque VNF sera déployé comme **conteneur Docker**.
- Orchestrer ce service réseau avec **TOSCA** et un workflow modélisé en **BPMN**.
- Vérifier que le flux réseau traverse bien toutes les VNFs dans l'ordre défini.

II. Pré-requis

- VM Ubuntu 20.04 LTS (2 vCPU, 4 Go RAM minimum, Docker installé).
- Docker et Docker Compose installés : ([fichier installDocker.pdf](#))
- `sudo apt update && sudo apt install -y docker.io docker-compose`
- `sudo systemctl enable docker && sudo systemctl start docker`
- Python + [Opera orchestrator](#) (pour exécuter TOSCA).
- Un outil BPMN (Camunda Modeler, Bonita Studio ou BPMN.io).

Images Docker utilisées :

- **Firewall** : `wallarm/api-firewall`
- **Load Balancer** : `haproxytech/haproxy-alpine`
- **Web Server** : `nginx`

III. Étapes du TP

1. Crédit des conteneurs Docker (Docker Compose)

Créer un fichier `docker-compose.yml` :

```
version: '3.8'

services:
  firewall:
    image: wallarm/api-firewall
    container_name: vnf-firewall
    ports:
      - "8080:8080"
    networks:
      - net-chain

  loadbalancer:
    image: haproxytech/haproxy-alpine
    container_name: vnf-lb
    ports:
      - "9090:80"
    depends_on:
      - firewall
    networks:
      - net-chain
```

```

webserver:
  image: nginx
  container_name: vnf-web
  ports:
    - "8081:80"
  depends_on:
    - loadbalancer
  networks:
    - net-chain

networks:
  net-chain:
    driver: bridge

```

➡ Lancer les services : `docker-compose up -d`
 Vérifier : `docker ps`

2. Modèles TOSCA pour décrire les VNFs

Créer un fichier `service-chain.yaml` :

```

tosca_definitions_version: tosca_simple_yaml_1_3

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

```

➡ Ce modèle décrit l’interconnexion **Firewall** → **LB** → **Web Server**.

3. Modélisation du workflow (BPMN)

Le workflow BPMN doit contenir :

- **Start Event**
- **Deploy Firewall** (Service Task)
- **Deploy Load Balancer** (Service Task, dépend du Firewall)
- **Deploy Web Server** (Service Task, dépend du LB)
- **Test Connectivity** (User Task)

- **End Event**

👉 Outil conseillé : [Camunda Modeler](#).

(Fichier [installCamunda.pdf](#))

Exporter le schéma en .bpnn. (network-service-chain.bpnn)

4. Orchestration avec TOSCA

xOpera (ou **Opera orchestrator**) est un **orchestrateur TOSCA** open-source développé par **XLAB**. Il permet d'**interpréter les fichiers TOSCA YAML** et de **déployer automatiquement** les composants décrits (VMs, conteneurs, scripts, etc.).

👉 Dans le cas du TP, il servira à **lire le modèle TOSCA** (`service-chain.yaml`) et **déployer les VNFs Docker** dans le bon ordre :
Firewall → Load Balancer → Web Server.

Dans cette partie on va faire les actions suivantes :

ACTION	COMMANDÉ CLE
INSTALLER OPERA	<code>pip install opera</code>
CREER MODELE TOSCA	<code>service-chain.yaml</code>
AJOUTER PLAYBOOKS ANSIBLE	<code>playbooks/deploy_*.yaml</code>
DEPLOYER	<code>opera deploy service-chain.yaml</code>
VERIFIER	<code>docker ps, curl</code>
SUPPRIMER	<code>opera undeploy</code>

4.1. 🌱 Étape 1 : Préparer l'environnement

Sur ta VM **Ubuntu 20.04 LTS**, installe Python et Opera :

```
sudo apt update
sudo apt install -y python3-pip git
pip install opera
```

Vérifie l'installation : `opera --version`

Tu devrais voir une version du type xOpera 1.7.15.

4.2. 📁 Étape 2 : Organiser ton projet TOSCA

Crée une arborescence comme suit :

```
/network-orchestration/
├── docker-compose.yml
├── service-chain.yaml
├── network-service-chain.bpnn
└── service-chain.yaml          # Le modèle TOSCA principal
    ├── playbooks/
    │   ├── deploy_firewall.yaml
    │   ├── deploy_loadbalancer.yaml
    │   └── deploy_webserver.yaml
    └── screenshots/
        ├── docker_ps.png
        ├── curl_tests.png
        └── workflow_bpnn.png
```

4.3. Étape 3 : Définir le modèle TOSCA (service-chain.yaml)

Voici un exemple complet de ton modèle TOSCA :

```
tosca_definitions_version: tosca_simple_yaml_1_3

description: >
    Chaîne de services réseau : Firewall -> Load Balancer -> Web Server
    orchestrée avec xOpera.

topology_template:
    node_templates:
        Firewall:
            type: tosca.nodes.SoftwareComponent
            properties:
                image: "wallarm/api-firewall"
            interfaces:
                Standard:
                    create:
                        implementation: playbooks/deploy_firewall.yaml

        LoadBalancer:
            type: tosca.nodes.SoftwareComponent
            properties:
                image: "haproxytech/haproxy-alpine"
            requirements:
                - dependency: Firewall
            interfaces:
                Standard:
                    create:
                        implementation: playbooks/deploy_loadbalancer.yaml

        WebServer:
            type: tosca.nodes.SoftwareComponent
            properties:
                image: "nginx"
            requirements:
                - dependency: LoadBalancer
            interfaces:
                Standard:
                    create:
                        implementation: playbooks/deploy_webserver.yaml
```

 Ce modèle indique à Opera **dans quel ordre déployer les services et quel playbook exécuter** pour chaque VNF.

4.4. Étape 4 : Écrire les playbooks d'exécution (Ansible YAML)

Chaque VNF sera déployé par un petit script Ansible

◆ **playbooks/deploy_firewall.yaml**

```
- name: Déploiement du VNF Firewall
  hosts: localhost
  tasks:
    - name: Lancer le conteneur Firewall
      command: docker run -d --name vnf-firewall -p 8080:8080 wallarm/api-firewall
```

◆ **playbooks/deploy_loadbalancer.yaml**

```
- name: Déploiement du VNF Load Balancer
  hosts: localhost
  tasks:
    - name: Lancer le conteneur Load Balancer
      command: docker run -d --name vnf-lb -p 9090:80 haproxytech/haproxy-alpine
```

◆ **playbooks/deploy_webserver.yaml**

```
- name: Déploiement du VNF Web Server
  hosts: localhost
  tasks:
    - name: Lancer le conteneur Web Server
      command: docker run -d --name vnf-web -p 8081:80 nginx
```

4.5. 🚀 Étape 5 : Lancer l'orchestration

Depuis le dossier `service-chain/`, exécute : `opera deploy service-chain.yaml`
xOpera va :

1. Lire ton modèle TOSCA.
2. Exécuter les playbooks dans l'ordre défini (Firewall → LB → WebServer).
3. Créer les conteneurs correspondants.

4.6. 🔎 Étape 6 : Vérifier le déploiement

Vérifie les conteneurs : `docker ps`

Tu devrais voir les trois VNFs en cours d'exécution.

Teste la connectivité :

- Accéder au **webserver via le firewall et le load balancer** :
 - `curl :8080` passe par le firewall
 - `curl :9090` passe par le load balancer

```
curl http://localhost:8080
curl http://localhost:9090
curl http://localhost:8081
```

→ Capturer les logs pour vérifier le cheminement :

```
docker logs vnf-firewall  
docker logs vnf-lb  
docker logs vnf-web
```

4.7. 🧹 Étape 7 : Nettoyer l'environnement

Quand tu veux supprimer les VNFs : `opera undeploy`

Cela exécute les étapes inverses des playbooks (si tu les as définies avec `delete`:).

Sinon, tu peux aussi faire :

```
docker rm -f vnf-firewall vnf-lb vnf-web
```

4.8. 🌐 Étape 8 : Bonnes pratiques / Vérifications

- Vérifie les logs d'exécution : `opera logs`
- Si un conteneur échoue : `docker logs vnf-firewall`
- Tu peux aussi faire un **graph de dépendances TOSCA** avec :
`opera validate service-chain.yaml`

IV. Livrable attendu

1. Un rapport au format PDF/Word contenant (Voir modèle vierge) :

- Objectif + architecture de la chaîne de services
- Le fichier `docker-compose.yml` et le modèle **TOSCA**
- Capture d'écran du **workflow BPMN**
- Captures des conteneurs en exécution (`docker ps`)
- Vérification avec `curl` montrant que le flux passe bien par Firewall → LB → Web Server.

2. Un dépôt GitHub avec :

- `docker-compose.yml`
- `service-chain.yaml` (TOSCA)
- `workflow.bpmn`
- Dossier `screenshots/` contenant les preuves d'exécution