Travaux Diriges — Niveau 2

Administration Linux Avancee

Fil rouge : NovaTech — Haute disponibilite et depannage

4 TD avances - de la haute disponibilite a la resolution de pannes

Contexte du Niveau 2

Rappel : ou en sommes-nous ?

Au terme du **Niveau 1**, l'infrastructure NovaTech est operationnelle :

- `srv-prod` sert une application web via **Nginx**
- `srv-backup` assure les sauvegardes
- `srv-nagios` supervise tout le parc avec **Nagios + NRPE**

> La direction est satisfaite... mais un incident de production recent a mis en evidence un **point de defaillance unique** : si `srv-prod` tombe, toute l'application est indisponible.

Le nouveau mandat

Le DSI de NovaTech vous confie une mission :

> "Nous avons eu 4 heures d'arret la semaine derniere. Il nous faut une architecture **tolerante aux pannes**, avec une supervision capable de **detecter et localiser les incidents** rapidement. Budget : pas question de tout refaire — on etoffe l'existant."

Votre feuille de route :

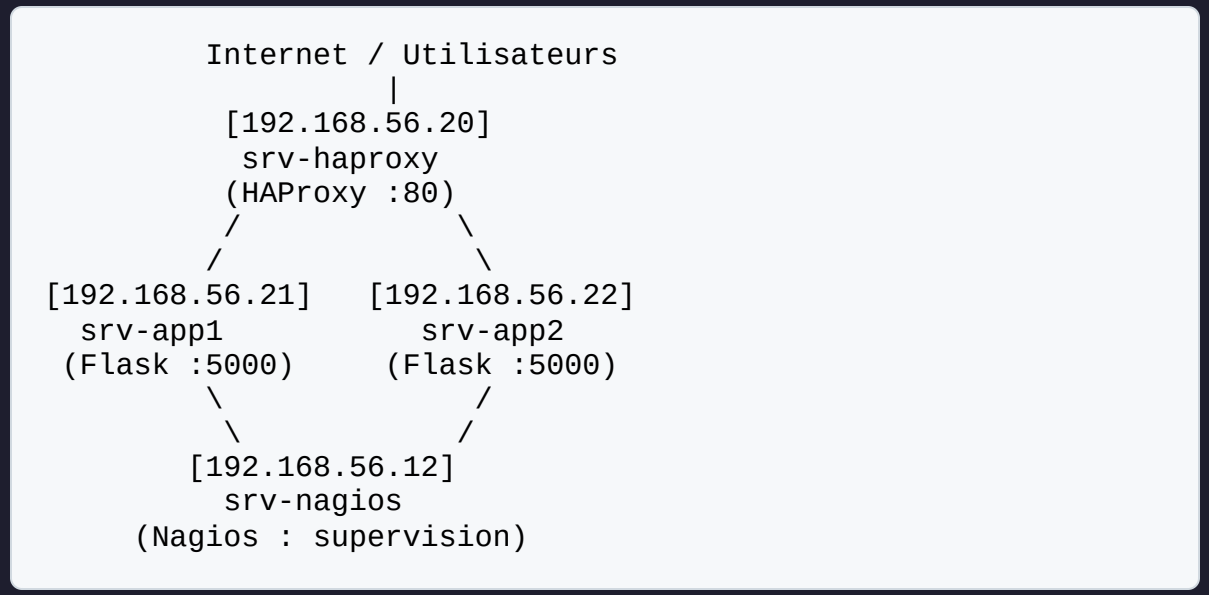
1. Deployer l'application Python sur 2 serveurs backends
2. Mettre en place **HAProxy** comme load balancer
3. Etendre **Nagios** pour superviser cette nouvelle couche
4. Simuler des pannes et appliquer des procedures de depannage

Infrastructure etendue

Machine	Role	IP	OS
`srv-prod`	Serveur existant (conserve)	192.168.56.10	Rocky Linux 9
`srv-backup`	Sauvegarde (conserve)	192.168.56.11	Rocky Linux 9
`srv-nagios`	Supervision (conserve)	192.168.56.12	Rocky Linux 9
`srv-haproxy`	Load balancer HAProxy	192.168.56.20	Rocky Linux 9
`srv-app1`	Backend applicatif 1	192.168.56.21	Rocky Linux 9
`srv-app2`	Backend applicatif 2	192.168.56.22	Rocky Linux 9

> Les 3 nouvelles VMs utilisent le meme Kickstart de base que le Niveau 1 (adaptez les IPs et noms d'hotes).

Architecture cible



> HAProxy distribue le trafic entre les deux backends. Si l'un tombe, HAProxy bascule automatiquement sur l'autre. Nagios surveille chaque composant.

Preparation : nouvelles VMs

Ajoutez dans `/etc/hosts` de **toutes les machines** :

```
192.168.56.20  srv-haproxy   srv-haproxy.novatech.lan
192.168.56.21  srv-app1      srv-app1.novatech.lan
192.168.56.22  srv-app2      srv-app2.novatech.lan
```

Verifiez la connectivite depuis `srv-haproxy` :

```
ping -c 2 srv-app1.novatech.lan
ping -c 2 srv-app2.novatech.lan
ping -c 2 srv-nagios.novatech.lan
```

> **Resultat attendu** : 0% packet loss sur les 3 pings. Si ce n'est pas le cas, verifiez le reseau virtuel (mode host-only) et les regles firewall (`firewall-cmd --list-all`).

TD 9 - Application Python Flask

TD 9 - Objectif

Contexte NovaTech : Avant de mettre en place le load balancer, il faut un service applicatif a distribuer. Vous allez deployer une application Python Flask simple sur `srv-app1` et `srv-app2`.

A la fin de ce TD, vous aurez :

- Installe Python et Flask sur les deux serveurs backend
- Deploye une application Flask qui retourne l'identite du serveur
- Configure un service systemd pour gerer l'application
- Valide que chaque backend repond independamment

TD 9 - Etape 1 : Installer Python et Flask

1. Sur **srv-app1** ET **srv-app2**, installez les dependances :

```
dnf install -y python3 python3-pip
pip3 install flask
```

Verifiez l'installation :

```
python3 --version
python3 -c "import flask; print(flask.__version__)"
```

Resultat attendu :

```
Python 3.9.x (ou superieur)
2.x.x
```

> Flask est un micro-framework web Python. Il suffit de quelques lignes pour exposer une API HTTP.

TD 9 - Etape 2 : Creer l'application Flask

2. Sur **srv-app1** ET **srv-app2**, creez l'application :

```
mkdir -p /opt/novatech-app
cat > /opt/novatech-app/app.py << 'EOF'
from flask import Flask, jsonify
import socket
import os
import datetime

app = Flask(__name__)

@app.route('/')
def index():
    return jsonify({
        "status": "ok",
        "hostname": socket.gethostname(),
        "server_ip": socket.gethostbyname(socket.gethostname()),
        "timestamp": datetime.datetime.now().isoformat(),
        "version": "1.0.0"
    })

@app.route('/health')
def health():
    return jsonify({"status": "healthy", "hostname": socket.gethostname()}), 200

@app.route('/info')
def info():
    return f"<h1>NovaTech App</h1><p>Serveur : <strong>{socket.gethostname()}</strong></p>"

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=False)
EOF
```

TD 9 - Etape 2 : Comprendre l'application

L'application expose **3 routes** :

Route	Description	Utilisation
`/`	JSON avec hostname, IP, timestamp	API principale
`/health`	JSON `{"status":"healthy"}`	Health check HAProxy
`/info`	Page HTML avec le nom du serveur	Verification visuelle

> La route `/health` est critique : HAProxy l'utilisera pour verifier si un backend est vivant avant de lui envoyer du trafic.

TD 9 - Etape 3 : Creer le service systemd

3. Sur **srv-app1** ET **srv-app2**, creez le fichier de service :

```
cat > /etc/systemd/system/novatech-app.service << 'EOF'
[Unit]
Description=NovaTech Flask Application
After=network.target

[Service]
Type=simple
User=nobody
WorkingDirectory=/opt/novatech-app
ExecStart=/usr/bin/python3 /opt/novatech-app/app.py
Restart=on-failure
RestartSec=5
StandardOutput=journal
StandardError=journal

[Install]
WantedBy=multi-user.target
EOF
```

TD 9 - Etape 3 : Demarrer le service

4. Activez et demarrez le service :

```
systemctl daemon-reload
systemctl enable --now novatech-app
```

5. Verifiez son etat :

```
systemctl status novatech-app
```

Resultat attendu :

```
● novatech-app.service - NovaTech Flask Application
   Loaded: loaded (/etc/systemd/system/novatech-app.service; enabled)
   Active: active (running) since ...
```

> Si le service est en echec (`failed`), consultez les logs : `journalctl -u novatech-app -n 30`

TD 9 - Etape 4 : Ouvrir le pare-feu

6. Sur **srv-app1** ET **srv-app2**, ouvrez le port 5000 :

```
firewall-cmd --permanent --add-port=5000/tcp
firewall-cmd --reload
```

Verifiez :

```
firewall-cmd --list-ports
```

Resultat attendu : `5000/tcp` apparait dans la liste.

TD 9 - Etape 5 : Valider les backends

7. Testez depuis **srv-haproxy** (ou votre poste hote) :

```
# Test srv-app1
curl -s http://192.168.56.21:5000/ | python3 -m json.tool

# Test srv-app2
curl -s http://192.168.56.22:5000/ | python3 -m json.tool
```

Resultat attendu srv-app1 :

```
{
    "hostname": "srv-app1",
    "status": "ok",
    "version": "1.0.0"
}
```

Resultat attendu srv-app2 :

```
{
    "hostname": "srv-app2",
    "status": "ok",
    "version": "1.0.0"
}
```

> Les `hostname` doivent etre differents : c'est ce qui permettra de verifier que HAProxy distribue bien le trafic entre les deux serveurs.

TD 9 - Etape 5 : Test du health check

8. Testez la route `/health` :

```
curl -o /dev/null -s -w "%{http_code}\n" http://192.168.56.21:5000/health
curl -o /dev/null -s -w "%{http_code}\n" http://192.168.56.22:5000/health
```

Resultat attendu : `200` sur les deux commandes.

9. Testez la route `/info` depuis un navigateur :

```
http://192.168.56.21:5000/info
http://192.168.56.22:5000/info
```

Resultat attendu : Chaque page affiche le nom du serveur correspondant en gras.

TD 9 - Checkpoint

Avant de passer au TD 10, verifiez :

- [ ] Flask installe sur srv-app1 et srv-app2
- [ ] Service `novatech-app` actif et enabled sur les deux
- [ ] Port 5000 ouvert dans le firewall sur les deux
- [ ] `curl http://192.168.56.21:5000/` retourne `"hostname": "srv-app1"`
- [ ] `curl http://192.168.56.22:5000/` retourne `"hostname": "srv-app2"`
- [ ] Code HTTP 200 sur `/health` pour les deux backends

> Si l'un des points echoue, **ne passez pas a la suite**. Consultez `journalctl -u novatech-app` et `firewall-cmd --list-all`.

TD 10 - HAProxy : Load Balancer

TD 10 - Objectif

Contexte NovaTech : Les deux backends sont prets. Vous allez maintenant installer et configurer HAProxy sur `srv-haproxy` pour distribuer le trafic et assurer la haute disponibilite.

A la fin de ce TD, vous aurez :

- Installe et configure HAProxy sur `srv-haproxy`
- Configure un load balancing round-robin entre srv-app1 et srv-app2
- Active la page de statistiques HAProxy
- Configure les health checks automatiques
- Valide la bascule automatique en cas de panne d'un backend

TD 10 - Etape 1 : Installer HAProxy

1. Sur **srv-haproxy**, installez HAProxy :

```
dnf install -y haproxy
```

Verifiez la version :

```
haproxy -v
```

Resultat attendu : `HAProxy version 2.x.x` (version 2 ou superieure).

2. Sauvegardez la configuration par defaut :

```
cp /etc/haproxy/haproxy.cfg /etc/haproxy/haproxy.cfg.orig
```

> Toujours sauvegarder la configuration d'origine avant de la modifier — bonne pratique systemadmin.

TD 10 - Etape 2 : Configurer HAProxy

3. Remplacez la configuration HAProxy :

```
cat > /etc/haproxy/haproxy.cfg << 'EOF'
#---------------------------------------------------------------------
# Global settings
#---------------------------------------------------------------------
global
    log         /dev/log local0
    log         /dev/log local1 notice
    chroot      /var/lib/haproxy
    pidfile     /var/run/haproxy.pid
    maxconn     4000
    user        haproxy
    group       haproxy
    daemon
    stats socket /var/lib/haproxy/stats

#---------------------------------------------------------------------
# Defaults
#---------------------------------------------------------------------
defaults
    mode                    http
    log                     global
    option                  httplog
    option                  dontlognull
    option                  http-server-close
    option                  forwardfor
    option                  redispatch
    retries                 3
    timeout http-request    10s
    timeout queue           1m
    timeout connect         10s
    timeout client          1m
    timeout server          1m
    timeout http-keep-alive 10s
    timeout check           10s
    maxconn                 3000
EOF
```

TD 10 - Etape 2 : Configuration (suite)

Ajoutez la section stats et le frontend/backend :

```
cat >> /etc/haproxy/haproxy.cfg << 'EOF'

#---------------------------------------------------------------------
# Page de statistiques
#---------------------------------------------------------------------
listen stats
    bind *:8080
    stats enable
    stats uri /haproxy-stats
    stats realm HAProxy\ Statistics
    stats auth admin:novatech2024
    stats refresh 5s
    stats show-legends
    stats show-node

#---------------------------------------------------------------------
# Frontend : point d'entree
#---------------------------------------------------------------------
frontend novatech_front
    bind *:80
    default_backend novatech_backends

#---------------------------------------------------------------------
# Backend : serveurs applicatifs
#---------------------------------------------------------------------
backend novatech_backends
    balance     roundrobin
    option      httpchk GET /health
    http-check  expect status 200
    server      app1 192.168.56.21:5000 check inter 5s fall 2 rise 3
    server      app2 192.168.56.22:5000 check inter 5s fall 2 rise 3
EOF
```

TD 10 - Etape 2 : Comprendre la configuration

Directive	Valeur	Signification
`balance roundrobin`	-	Distribution alternee entre backends
`option httpchk GET /health`	-	Requete HTTP sur /health pour verifier
`http-check expect status 200`	-	Le backend doit repondre 200
`check inter 5s`	5s	Verifier toutes les 5 secondes
`fall 2`	2	2 echecs consecutifs = backend DOWN
`rise 3`	3	3 succes consecutifs = backend UP

> `fall` et `rise` evitent les basculements intempestifs dus a de brefs pics de latence.

TD 10 - Etape 3 : Valider et demarrer HAProxy

4. Validez la syntaxe de la configuration :

```
haproxy -c -f /etc/haproxy/haproxy.cfg
```

Resultat attendu :

```
Configuration file is valid
```

> Si vous voyez une erreur, HAProxy indique la ligne problematique. Corrigez avant de continuer.

5. Ouvrez les ports firewall et demarrez :

```
firewall-cmd --permanent --add-port=80/tcp
firewall-cmd --permanent --add-port=8080/tcp
firewall-cmd --reload
systemctl enable --now haproxy
```

TD 10 - Etape 4 : Verifier le load balancing

6. Depuis votre poste hote ou `srv-nagios`, testez plusieurs requetes :

```
for i in $(seq 1 6); do
    curl -s http://192.168.56.20/ | python3 -c "import sys,json; d=json.load(sys.stdin); print(d['hostname'])"
done
```

Resultat attendu :

```
srv-app1
srv-app2
srv-app1
srv-app2
srv-app1
srv-app2
```

> L'alternance reguliere confirme le fonctionnement du round-robin. Si vous voyez toujours le meme serveur, verifiez la configuration du backend.

TD 10 - Etape 5 : Interface de statistiques

7. Ouvrez dans un navigateur :

```
http://192.168.56.20:8080/haproxy-stats
```

Login : `admin` / Mot de passe : `novatech2024`

Elements a observer :

Colonne	Signification
`Status`	`UP` = backend actif, `DOWN` = inaccessible
`Check`	Resultat du dernier health check (`L7OK` = HTTP 200 recu)
`Rate`	Requetes par seconde actuellement
`Hrsp_2xx`	Total requetes avec reponse 2xx

> Cette page est votre **tableau de bord temps reel**. Gardez-la ouverte lors des prochains TDs.

TD 10 - Etape 6 : Tester la bascule automatique

8. Sur **srv-app1**, arretez l'application :

```
systemctl stop novatech-app
```

9. Depuis `srv-haproxy`, relancez la serie de requetes :

```
for i in $(seq 1 6); do
    curl -s http://192.168.56.20/ | python3 -c "import sys,json; d=json.load(sys.stdin); print(d['hostname'])"
done
```

Resultat attendu :

```
srv-app2
srv-app2
srv-app2
srv-app2
srv-app2
srv-app2
```

> HAProxy a detecte la panne via le health check et redirige **automatiquement** tout le trafic vers srv-app2. Verifiez dans l'interface stats : srv-app1 est maintenant en rouge (`DOWN`).

TD 10 - Etape 6 : Retour a la normale

10. Redemarrez l'application sur `srv-app1` :

```
# Sur srv-app1 :
systemctl start novatech-app
```

11. Observez le retour dans les stats HAProxy.

Resultat attendu : Apres 3 health checks reussis (15 secondes), `srv-app1` repasse en `UP` et le trafic est de nouveau distribue sur les deux backends.

> C'est le comportement `rise 3` : HAProxy ne reintegre pas un serveur precipitamment. Il attend 3 confirmations successives.

TD 10 - Checkpoint

- [ ] HAProxy installe et demarrage automatique (enabled)
- [ ] `haproxy -c -f /etc/haproxy/haproxy.cfg` : Configuration valide
- [ ] Round-robin visible : alternance srv-app1 / srv-app2
- [ ] Interface stats accessible sur :8080/haproxy-stats
- [ ] Bascule automatique validee : srv-app1 arrete -> tout va sur srv-app2
- [ ] Retour automatique valide : srv-app1 redemarree -> trafic redistribue

TD 11 - Supervision HAProxy avec Nagios

TD 11 - Objectif

Contexte NovaTech : L'architecture HA est en place. Mais `srv-nagios` ne surveille pas encore les nouveaux composants. Un backend silencieusement degrade passerait inaper... Vous allez etendre la supervision Nagios pour couvrir HAProxy et les backends Flask.

A la fin de ce TD, vous aurez :

- Declare les nouvelles machines dans Nagios
- Supervise les backends Flask via NRPE
- Supervise l'etat des backends via la page stats HAProxy
- Configure des alertes differenciees (WARNING / CRITICAL)
- Teste les alertes en simulant des problemes

TD 11 - Etape 1 : Installer NRPE sur les nouvelles machines

1. Sur **srv-haproxy**, **srv-app1** et **srv-app2**, installez NRPE :

```
dnf install -y epel-release
dnf install -y nrpe nagios-plugins-all
```

2. Sur chaque machine, autorisez `srv-nagios` dans NRPE :

```
sed -i 's/^allowed_hosts=.*/allowed_hosts=127.0.0.1,192.168.56.12/' \
    /etc/nagios/nrpe.cfg
```

3. Demarrez NRPE :

```
systemctl enable --now nrpe
firewall-cmd --permanent --add-port=5666/tcp
firewall-cmd --reload
```

TD 11 - Etape 2 : Plugin NRPE pour l'application Flask

4. Sur **srv-app1** et **srv-app2**, creez un plugin de check de l'app :

```
cat > /usr/lib64/nagios/plugins/check_flask_app << 'EOF'
#!/bin/bash
# Check NovaTech Flask application health

RESULT=$(curl -s -o /dev/null -w "%{http_code}" \
    --connect-timeout 5 http://localhost:5000/health 2>/dev/null)

if [ "$RESULT" = "200" ]; then
    echo "OK - Flask app reachable (HTTP $RESULT)"
    exit 0
elif [ "$RESULT" = "000" ]; then
    echo "CRITICAL - Flask app unreachable (connection refused)"
    exit 2
else
    echo "WARNING - Flask app returned HTTP $RESULT"
    exit 1
fi
EOF

chmod +x /usr/lib64/nagios/plugins/check_flask_app
```

TD 11 - Etape 2 : Enregistrer le plugin dans NRPE

5. Sur **srv-app1** et **srv-app2**, ajoutez la commande NRPE :

```
cat >> /etc/nagios/nrpe.cfg << 'EOF'
command[check_flask_app]=/usr/lib64/nagios/plugins/check_flask_app
command[check_procs_flask]=/usr/lib64/nagios/plugins/check_procs -C python3 -w 1:5 -c 1:10
EOF
```

Redemarrez NRPE :

```
systemctl restart nrpe
```

6. Testez depuis **srv-nagios** :

```
/usr/lib64/nagios/plugins/check_nrpe -H 192.168.56.21 -c check_flask_app
/usr/lib64/nagios/plugins/check_nrpe -H 192.168.56.22 -c check_flask_app
```

Resultat attendu : `OK - Flask app reachable (HTTP 200)` sur les deux.

TD 11 - Etape 3 : Plugin pour les stats HAProxy

7. Sur **srv-haproxy**, creez un plugin de check des backends :

```
cat > /usr/lib64/nagios/plugins/check_haproxy_backends << 'EOF'
#!/bin/bash
# Check HAProxy backends via stats page
# Usage: check_haproxy_backends <backend_name>

BACKEND=${1:-"novatech_backends"}
STATS_URL="http://admin:novatech2024@localhost:8080/haproxy-stats;csv"

DATA=$(curl -s "$STATS_URL" 2>/dev/null)
if [ $? -ne 0 ]; then
    echo "CRITICAL - Cannot reach HAProxy stats page"
    exit 2
fi

DOWN_COUNT=$(echo "$DATA" | grep "^$BACKEND" | grep ",DOWN," | wc -l)
UP_COUNT=$(echo "$DATA" | grep "^$BACKEND" | grep ",UP," | wc -l)
TOTAL=$((DOWN_COUNT + UP_COUNT))

if [ "$DOWN_COUNT" -eq 0 ]; then
    echo "OK - All $TOTAL backends UP for $BACKEND"
    exit 0
elif [ "$DOWN_COUNT" -lt "$TOTAL" ]; then
    echo "WARNING - $DOWN_COUNT/$TOTAL backends DOWN for $BACKEND"
    exit 1
else
    echo "CRITICAL - All backends DOWN for $BACKEND"
    exit 2
fi
EOF

chmod +x /usr/lib64/nagios/plugins/check_haproxy_backends
```

TD 11 - Etape 3 : Enregistrer dans NRPE (HAProxy)

8. Sur **srv-haproxy**, ajoutez les commandes NRPE :

```
cat >> /etc/nagios/nrpe.cfg << 'EOF'
command[check_haproxy_backends]=/usr/lib64/nagios/plugins/check_haproxy_backends novatech_backends
command[check_haproxy_service]=/usr/lib64/nagios/plugins/check_procs -C haproxy -w 1:5 -c 1:10
EOF

systemctl restart nrpe
```

9. Testez depuis **srv-nagios** :

```
/usr/lib64/nagios/plugins/check_nrpe -H 192.168.56.20 -c check_haproxy_backends
```

Resultat attendu : `OK - All 2 backends UP for novatech_backends`

TD 11 - Etape 4 : Declarer les hotes dans Nagios

10. Sur **srv-nagios**, ajoutez les nouveaux hotes :

```
cat > /etc/nagios/objects/novatech-ha-hosts.cfg << 'EOF'
# Groupe HA
define hostgroup {
    hostgroup_name  novatech-ha
    alias           Infrastructure HA NovaTech
    members         srv-haproxy,srv-app1,srv-app2
}

define host {
    use             novatech-server
    host_name       srv-haproxy
    alias           Load Balancer HAProxy
    address         192.168.56.20
}

define host {
    use             novatech-server
    host_name       srv-app1
    alias           Backend Applicatif 1
    address         192.168.56.21
}

define host {
    use             novatech-server
    host_name       srv-app2
    alias           Backend Applicatif 2
    address         192.168.56.22
}
EOF
```

TD 11 - Etape 5 : Declarer les services dans Nagios

11. Creez les services de supervision :

```
cat > /etc/nagios/objects/novatech-ha-services.cfg << 'EOF'
# --- HAProxy ---
define service {
    use                 novatech-service
    host_name           srv-haproxy
    service_description HAProxy Backends Status
    check_command       check_nrpe!check_haproxy_backends
}

define service {
    use                 novatech-service
    host_name           srv-haproxy
    service_description HAProxy Process
    check_command       check_nrpe!check_haproxy_service
}

define service {
    use                 novatech-service
    host_name           srv-haproxy
    service_description HTTP Endpoint
    check_command       check_http!-H 192.168.56.20 -p 80 -u /
}
EOF
```

TD 11 - Etape 5 : Services backends (suite)

```
cat >> /etc/nagios/objects/novatech-ha-services.cfg << 'EOF'

# --- srv-app1 ---
define service {
    use                 novatech-service
    host_name           srv-app1
    service_description Flask App
    check_command       check_nrpe!check_flask_app
}

define service {
    use                 novatech-service
    host_name           srv-app1
    service_description Flask Process
    check_command       check_nrpe!check_procs_flask
}

# --- srv-app2 ---
define service {
    use                 novatech-service
    host_name           srv-app2
    service_description Flask App
    check_command       check_nrpe!check_flask_app
}

define service {
    use                 novatech-service
    host_name           srv-app2
    service_description Flask Process
    check_command       check_nrpe!check_procs_flask
}
EOF
```

TD 11 - Etape 6 : Valider et recharger Nagios

12. Enregistrez les nouveaux fichiers dans la configuration principale :

```
# Dans /etc/nagios/nagios.cfg, verifiez que cette ligne existe :
grep "cfg_dir=/etc/nagios/objects" /etc/nagios/nagios.cfg
```

13. Validez la configuration :

```
nagios -v /etc/nagios/nagios.cfg
```

Resultat attendu : `Things look okay - No serious problems were detected`

14. Rechargez Nagios :

```
systemctl reload nagios
```

15. Dans l'interface web Nagios, verifiez que les 3 nouveaux hotes apparaissent.

> Attendez 1-2 cycles de check (5-10 min) pour que tous les services passent au vert.

TD 11 - Etape 7 : Test d'alerte Nagios

16. Arretez l'app Flask sur **srv-app1** :

```
# Sur srv-app1 :
systemctl stop novatech-app
```

17. Depuis **srv-nagios**, forcez un check immediat :

```
/usr/lib64/nagios/plugins/check_nrpe -H 192.168.56.21 -c check_flask_app
```

Resultat attendu : `CRITICAL - Flask app unreachable (connection refused)`

18. Attendez le prochain cycle Nagios et observez :

- `srv-app1` > `Flask App` : **CRITICAL** (rouge)
- `srv-haproxy` > `HAProxy Backends Status` : **WARNING** (jaune) — 1 backend sur 2 down

19. Redemarrez l'app et verifiez le retour en vert.

TD 11 - Checkpoint

- [ ] NRPE installe sur srv-haproxy, srv-app1, srv-app2
- [ ] Plugin `check_flask_app` fonctionnel (test NRPE depuis nagios)
- [ ] Plugin `check_haproxy_backends` fonctionnel
- [ ] Les 3 nouveaux hotes visibles dans Nagios
- [ ] Tous les services en OK (vert) au repos
- [ ] Test d'alerte : arret Flask -> CRITICAL detecte dans Nagios
- [ ] Test d'alerte : WARNING HAProxy quand 1 backend down

TD 12 - Simulation de Pannes et Depannage

TD 12 - Objectif

Contexte NovaTech : L'infrastructure HA est complete et supervisee. Ce TD simule des incidents realistes de production que vous devez detecter, diagnostiquer et resoudre en suivant une methodologie structuree.

A la fin de ce TD, vous aurez :

- Applique une methodologie de depannage systematique
- Resolu 5 scenarios de pannes progressivement complexes
- Redige des procedures de resolution documentees
- Valide la detection de chaque incident par Nagios

Methodologie de depannage

Avant de toucher a quoi que ce soit, appliquez toujours cette demarche :

```
1. OBSERVER    — Que dit le monitoring ? Nagios, logs, stats HAProxy
2. FORMULER    — Quelle est l'hypothese de panne la plus probable ?
3. TESTER      — Validez l'hypothese avec une commande precise
4. CORRIGER    — Appliquez la correction minimale necessaire
5. VALIDER     — Confirmez le retour a la normale (monitoring + test fonctionnel)
6. DOCUMENTER  — Notez la cause et la resolution
```

> Cette methode evite de "bidouiller" dans tous les sens. Chaque action doit avoir un objectif precis.

TD 12 - Scenario 1 : Service Flask en echec silencieux

Description de la panne :

L'operateur de nuit signale que certains utilisateurs recoivent des erreurs 502. Nagios indique un WARNING sur `HAProxy Backends Status`.

Etat initial a reproduire (faites-le sur `srv-app1`) :

```
# Sur srv-app1 : tuez le processus Flask brutalement
pkill -9 -f "python3 /opt/novatech-app/app.py"
```

> Attention : `systemctl stop` serait trop propre. `pkill -9` simule un crash memoire, comme en production.

TD 12 - Scenario 1 : Diagnostic

Etape 1 - Observer

Depuis **srv-nagios** :

```
/usr/lib64/nagios/plugins/check_nrpe -H 192.168.56.21 -c check_flask_app
/usr/lib64/nagios/plugins/check_nrpe -H 192.168.56.20 -c check_haproxy_backends
```

Depuis **srv-haproxy** :

```
curl -s http://192.168.56.21:5000/health
```

Etape 2 - Localiser sur srv-app1

```
# Etat du service systemd
systemctl status novatech-app

# Logs du service
journalctl -u novatech-app -n 20 --no-pager

# Le processus tourne-t-il ?
ps aux | grep python3
```

TD 12 - Scenario 1 : Resolution et validation

Resultat attendu du diagnostic :

- `systemctl status` : `Active: failed` (le service a ete tue mais ne s'est pas relance)
- `journalctl` : Message `Killed` suivi d'une tentative de restart

Etape 3 - Corriger

```
# Sur srv-app1 :
systemctl start novatech-app
systemctl status novatech-app
```

Etape 4 - Valider

```
# Health check local
curl -s http://localhost:5000/health

# Depuis srv-haproxy :
curl -s http://192.168.56.20/ | python3 -c "import sys,json; d=json.load(sys.stdin); print(d['hostname'])"
```

Etape 5 - Renforcer la resilience

```
# Verifiez que Restart=on-failure est bien dans le service
grep Restart /etc/systemd/system/novatech-app.service
```

TD 12 - Scenario 2 : Mauvaise configuration HAProxy

Description de la panne :

Suite a une "optimisation" d'un collegue, HAProxy ne repond plus. Nagios indique `CRITICAL` sur `HTTP Endpoint` et `HAProxy Backends Status`.

Etat initial a reproduire (sur `srv-haproxy`) :

```
# Introduire une erreur de port dans la config
sed -i 's/192.168.56.21:5000/192.168.56.21:9999/' /etc/haproxy/haproxy.cfg
sed -i 's/192.168.56.22:5000/192.168.56.22:9999/' /etc/haproxy/haproxy.cfg
systemctl reload haproxy
```

TD 12 - Scenario 2 : Diagnostic

Etape 1 - Observer les symptomes

```
# Depuis srv-haproxy :
curl -v http://localhost/

# Stats HAProxy : les backends sont-ils UP ou DOWN ?
curl -s "http://admin:novatech2024@localhost:8080/haproxy-stats;csv" | \
    grep "novatech_backends,app"
```

Resultat attendu : Les deux backends apparaissent en `DOWN`.

Etape 2 - Lire les logs HAProxy

```
journalctl -u haproxy -n 30 --no-pager
# ou
tail -50 /var/log/haproxy.log
```

Etape 3 - Tester manuellement les backends

```
curl -v http://192.168.56.21:9999/health
# -> Connection refused : le port 9999 n'existe pas
```

TD 12 - Scenario 2 : Resolution

Etape 4 - Identifier la modification

```
# Comparer avec la sauvegarde
diff /etc/haproxy/haproxy.cfg /etc/haproxy/haproxy.cfg.orig
```

Resultat attendu :

```
< server      app1 192.168.56.21:9999 check inter 5s fall 2 rise 3
---
> server      app1 192.168.56.21:5000 check inter 5s fall 2 rise 3
```

Etape 5 - Corriger et valider la config

```
sed -i 's/192.168.56.21:9999/192.168.56.21:5000/' /etc/haproxy/haproxy.cfg
sed -i 's/192.168.56.22:9999/192.168.56.22:5000/' /etc/haproxy/haproxy.cfg

# TOUJOURS valider avant de recharger
haproxy -c -f /etc/haproxy/haproxy.cfg
systemctl reload haproxy
```

Etape 6 - Valider le retour a la normale

```
curl -s http://192.168.56.20/ | python3 -c "import sys,json; d=json.load(sys.stdin); print(d['hostname'])"
```

TD 12 - Scenario 3 : Blocage firewall

Description de la panne :

Nagios indique que `srv-app2` est injoignable (host DOWN). HAProxy reporte app2 en DOWN. Les utilisateurs remarquent une degradation des performances (un seul backend actif).

Etat initial a reproduire (sur `srv-app2`) :

```
# Bloquer le trafic entrant sur le port 5000
firewall-cmd --add-rich-rule='rule family="ipv4" port port="5000" protocol="tcp" drop'
```

TD 12 - Scenario 3 : Diagnostic

Etape 1 - Tester la connectivite reseau

```
# Depuis srv-haproxy :
# Test ping (ICMP) : est-ce que la machine repond ?
ping -c 3 192.168.56.22

# Test port TCP : le port 5000 est-il accessible ?
nc -zv 192.168.56.22 5000
# ou
curl -v --connect-timeout 5 http://192.168.56.22:5000/health
```

Resultat attendu :

- `ping` : repond (la machine est allumee)
- `nc` sur port 5000 : timeout (port bloque ou ferme)

Etape 2 - Sur srv-app2 : l'app tourne-t-elle ?

```
systemctl status novatech-app
curl -s http://localhost:5000/health
```

Resultat attendu : L'app tourne et repond localement. Le probleme est donc **reseau**, pas applicatif.

TD 12 - Scenario 3 : Resolution

Etape 3 - Inspecter le firewall sur srv-app2

```
# Lister toutes les regles actives
firewall-cmd --list-all

# Voir les rich-rules en particulier
firewall-cmd --list-rich-rules
```

Resultat attendu :

```
rule family="ipv4" port port="5000" protocol="tcp" drop
```

Etape 4 - Supprimer la regle bloquante

```
firewall-cmd --remove-rich-rule='rule family="ipv4" port port="5000" protocol="tcp" drop'
firewall-cmd --runtime-to-permanent
```

Etape 5 - Valider

```
# Depuis srv-haproxy :
nc -zv 192.168.56.22 5000
curl -s http://192.168.56.22:5000/health
```

> Patientez 15 secondes (3 x `rise 3` checks) pour que HAProxy reintegre app2.

TD 12 - Scenario 4 : Application retournant des erreurs HTTP 500

Description de la panne :
Les utilisateurs signalent des erreurs sporadiques. Nagios est vert, mais les erreurs arrivent quand meme. Le health check `/health` est OK, mais la route principale `/` crashe.

Etat initial a reproduire (sur `srv-app1`) :

```
# Modifier l'app pour qu'elle plante sur la route principale
cat > /opt/novatech-app/app.py << 'PYEOF'
from flask import Flask, jsonify
import socket, datetime

app = Flask(__name__)

@app.route('/')
def index():
    # Bug intentionnel : division par zero
    result = 1 / 0
    return jsonify({"status": "ok"})

@app.route('/health')
def health():
    return jsonify({"status": "healthy", "hostname": socket.gethostname()}), 200

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=False)
PYEOF

systemctl restart novatech-app
```

TD 12 - Scenario 4 : Diagnostic

Etape 1 - Reproduire l'erreur

```
# Depuis srv-haproxy :
for i in 1 2 3 4; do
    curl -s -o /dev/null -w "Code: %{http_code}\n" http://192.168.56.20/
done
```

Resultat attendu : Alternance de `Code: 200` (srv-app2) et `Code: 500` (srv-app1).

Etape 2 - Isoler le backend fautif

```
# Tester chaque backend directement
curl -s -o /dev/null -w "app1: %{http_code}\n" http://192.168.56.21:5000/
curl -s -o /dev/null -w "app2: %{http_code}\n" http://192.168.56.22:5000/
```

Etape 3 - Lire les logs applicatifs

```
# Sur srv-app1 :
journalctl -u novatech-app -n 30 --no-pager | grep -i "error\|traceback\|exception"
```

Resultat attendu : `ZeroDivisionError: division by zero` dans les logs.

TD 12 - Scenario 4 : Resolution

Etape 4 - Corriger le bug applicatif

```
# Sur srv-app1, restaurer l'app correcte :
cat > /opt/novatech-app/app.py << 'PYEOF'
from flask import Flask, jsonify
import socket, datetime

app = Flask(__name__)

@app.route('/')
def index():
    return jsonify({
        "status": "ok",
        "hostname": socket.gethostname(),
        "timestamp": datetime.datetime.now().isoformat(),
        "version": "1.0.0"
    })

@app.route('/health')
def health():
    return jsonify({"status": "healthy", "hostname": socket.gethostname()}), 200

@app.route('/info')
def info():
    return f"<h1>NovaTech App</h1><p>Serveur : <strong>{socket.gethostname()}</strong></p>"

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=False)
PYEOF

systemctl restart novatech-app
```

Etape 5 - Valider

```
for i in 1 2 3 4; do
    curl -s -o /dev/null -w "Code: %{http_code}\n" http://192.168.56.20/
done
```

Resultat attendu : Tous les codes sont `200`

TD 12 - Scenario 5 : Panne totale (challenge)

Description de la panne :

Alerte majeure : **toute l'application est inaccessible**. Nagios signale `CRITICAL` partout. Vous avez 15 minutes pour tout retablir.

Etat initial a reproduire (operateur secret, ne regardez pas !) :

```
# A executer par l'instructeur / un binome
systemctl stop novatech-app   # sur srv-app1
systemctl stop novatech-app   # sur srv-app2
```

Votre check-list de depannage (a completer en autonomie) :

- [ ] 1. Verifier le statut HAProxy (`systemctl status haproxy`)
- [ ] 2. Checker les backends via les stats HAProxy
- [ ] 3. Tester la connectivite reseau vers app1 et app2
- [ ] 4. Verifier l'etat des services sur app1 et app2
- [ ] 5. Consulter les logs (`journalctl`)
- [ ] 6. Appliquer la correction
- [ ] 7. Valider avec Nagios et test curl

TD 12 - Scenario 5 : Grille d'evaluation

Action	Points
Identifier la couche fautive (reseau / appli / LB) en < 3 min	3
Utiliser les outils dans le bon ordre (obs -> diag -> fix)	2
Ne pas rebooter les VMs sans diagnostic prealable	2
Resoudre sans aide exterieure	3
Valider avec Nagios ET curl (pas juste un des deux)	2
Documenter la cause et la resolution (1 paragraphe)	3

> **Bareme total : 15 points.** Un vrai incident de production ne vous donne pas de seconde chance — la methode compte autant que le resultat.

TD 12 - Outils de depannage essentiels

Outil	Usage	Exemple
`systemctl status`	Etat d'un service	`systemctl status novatech-app`
`journalctl -u`	Logs d'un service	`journalctl -u haproxy -n 50`
`curl -v`	Test HTTP detaille	`curl -v http://IP:port/health`
`nc -zv`	Test connectivite TCP	`nc -zv IP port`
`haproxy -c -f`	Validation config	`haproxy -c -f /etc/haproxy/haproxy.cfg`
`firewall-cmd --list-all`	Regles firewall	-
`ps aux | grep`	Processus actifs	
`diff`	Comparer configs	`diff haproxy.cfg haproxy.cfg.orig`

TD 12 - Checkpoint final

- [ ] Scenario 1 resolu : service Flask crash -> systemd restart
- [ ] Scenario 2 resolu : mauvaise config HAProxy identifiee et corrigee
- [ ] Scenario 3 resolu : blocage firewall detecte et supprime
- [ ] Scenario 4 resolu : bug applicatif 500 isole et corrige
- [ ] Scenario 5 resolu en autonomie avec documentation
- [ ] Nagios : tous les services en OK a l'issue du TD

Synthese du Niveau 2

Recapitulatif de l'infrastructure finale

Composant	TD	Ce qui a ete fait
Application Python Flask	TD 9	Deploy 2 backends + service systemd
Load Balancer HAProxy	TD 10	Round-robin + health checks + bascule automatique
Supervision HA	TD 11	Nagios + NRPE etendu aux nouveaux composants
Depannage	TD 12	5 scenarios de pannes + methodologie

Architecture finale NovaTech

```
          [Utilisateurs]
                |
    [srv-haproxy :80]        <- HAProxy (LB + failover)
    [srv-haproxy :8080]      <- Stats page
          /        \
[srv-app1 :5000]  [srv-app2 :5000]   <- Flask backends
          \          /
    [srv-nagios]                      <- Supervision Nagios
    - HAProxy Backends Status
    - Flask App (x2)
    - HTTP Endpoint
    - Flask Process (x2)
```

Competences acquises au Niveau 2

- **Architecturer** : Comprendre et deployer une architecture HA avec load balancer
- **Containeriser la logique** : Encapsuler une app dans un service systemd
- **Superviser** : Etendre Nagios a de nouveaux composants avec des plugins custom
- **Depanner** : Appliquer une methodologie structuree face a une panne inconnue
- **Lire les logs** : Utiliser `journalctl`, les stats HAProxy, les codes HTTP
- **Securiser** : Ne pas corriger sans valider la config (`haproxy -c -f`)

Differences entre Niveau 1 et Niveau 2

Aspect	Niveau 1	Niveau 2
Disponibilite	Serveur unique (SPOF)	Architecture HA (failover auto)
Application	Nginx statique	Flask dynamique + identite
Supervision	Checks standards Nagios	Plugins custom + stats HAProxy
Depannage	Procedures documentees	Scenarios realistes en autonomie
Resilience	RAID1 (stockage)	Load balancing (applicatif)

Pour aller plus loin

- **Keepalived** : VIP flottante sur HAProxy pour eliminer le SPOF du LB lui-meme
- **Prometheus + Grafana** : Metriques temps reel avec dashboards (complementaire a Nagios)
- **Ansible** : Automatiser le deploiement de l'ensemble de cette infrastructure
- **Docker / Podman** : Containeriser l'application Flask pour des deploiements reproductibles
- **Let's Encrypt + Certbot** : HTTPS sur le frontend HAProxy
- **HAProxy ACLs** : Routage intelligent (par URL, header, methode HTTP)

> L'infrastructure NovaTech est maintenant tolerante aux pannes. La prochaine etape : l'**observabilite** — savoir non seulement que quelque chose est casse, mais **pourquoi**.

Checklist globale — Niveau 2

- [ ] **TD 9** : App Flask deployee et active sur srv-app1 et srv-app2
- [ ] **TD 10** : HAProxy configure, round-robin valide, bascule testee
- [ ] **TD 11** : Nagios etendu, tous les services en OK au repos
- [ ] **TD 12** : Les 5 scenarios de pannes resolus et documentes

Fin du Niveau 2

NovaTech est desormais tolerante aux pannes.

"La haute disponibilite, ce n'est pas eliminer les pannes — c'est s'assurer qu'elles ne font plus mal."