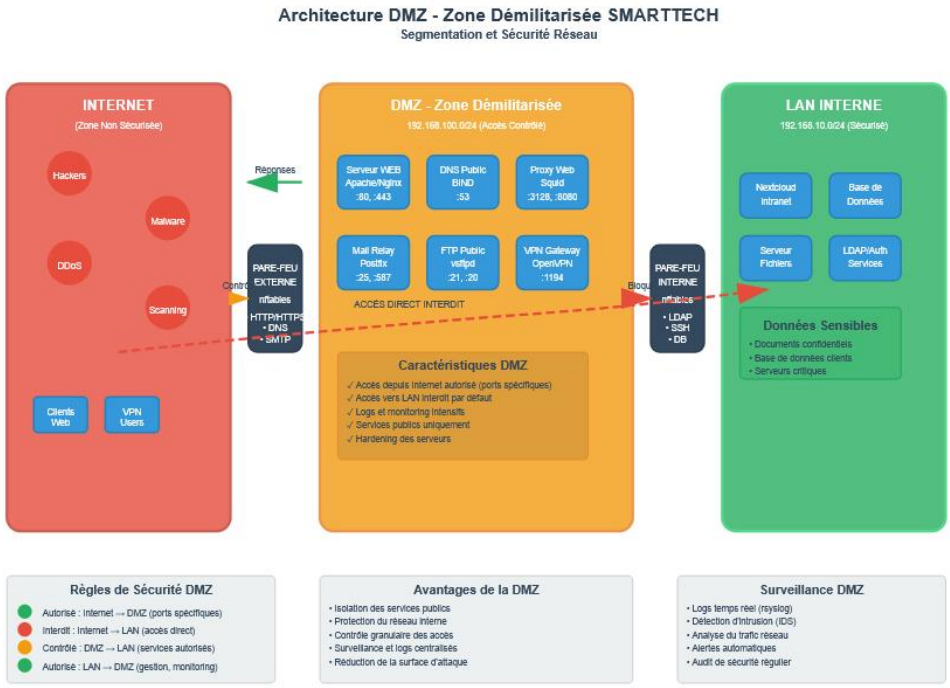
DMZ

INTRODUCTION :

La zone démilitarisée (DMZ) joue un rôle clé dans la sécurisation de l'infrastructure réseau de l'entreprise SMARTTECH.
Elle sert de zone tampon entre Internet et le réseau interne, hébergeant des services exposés comme le serveur web, le DNS secondaire, et le proxy.
L'objectif est de permettre un accès public contrôlé tout en isolant les services critiques internes grâce à un filtrage strict via le pare-feu.



---

1. Serveur Web public (Apache/Nginx)

Apres avoir exécuter la commande **apt update** on Installer le serveur Apache dans le conteneur dmz .

```
/ # apt install apache2 openssl vim -y
```

On édite le fichier /var/html/index.html pour la page web

```
/ # vi /var/www/html/index.html
```

On modifie la page d'accueil (facultatif)

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <title>Bienvenue chez SMARTTECH</title>
    <style>
        body {
            font-family: 'Segoe UI', sans-serif;
```

on edite le fichier /etc/hosts pour la résolution DNS locale

```
/ # vi /etc/hosts
```

Configurer le nom SMARTECH associé a l'adresse du dmz (192.168.100.10)

```
192.168.100.10    smartech.site
```

On fait le test **http://smartech.site** dans le navigateur du webterm (11.11.12.11)

Activons le HTTPS :

La commande **mkdir -p /etc/apache2/ssl** crée un dossier pour stocker les fichiers SSL. Ensuite, **openssl req** ... génère un certificat SSL auto-signé valable 365 jours pour le nom SMARTECH, avec sa clé privée (smartech.key) et son certificat (smartech.crt).

```
/ # mkdir -p /etc/apache2/ssl
/ # openssl req -x509 -nodes -days 365 -newkey rsa:2048 \
>   -keyout /etc/apache2/ssl/smartech.key \
>   -out /etc/apache2/ssl/smartech.crt \
>   -subj "/CN=SMARTECH"
```

Modifie le fichier /etc/apache2/sites-available/default-ssl.conf :

```
# vi /etc/apache2/sites-available/default-ssl.conf
```

Ces directives activent le chiffrement SSL (SSLEngine on) et indiquent à Apache où trouver le certificat (smartech.crt) et la clé privée (smartech.key). La directive ServerName SMARTECH associe cette configuration au nom de domaine local SMARTECH.

```
SSLCertificateFile      /etc/apache2/ssl/smartech.crt
SSLCertificateKeyFile /etc/apache2/ssl/smartech.key
ServerName smartech.site
ServerAlias www.smartech.site
```

La commande a2enmod ssl active le module SSL dans Apache pour permettre les connexions HTTPS.

```
/ # a2enmod ssl
```

La commande a2ensite default-ssl active le site par défaut configuré pour utiliser SSL (HTTPS).

```
/ # a2ensite default-ssl
```

Redémarrer Apache :

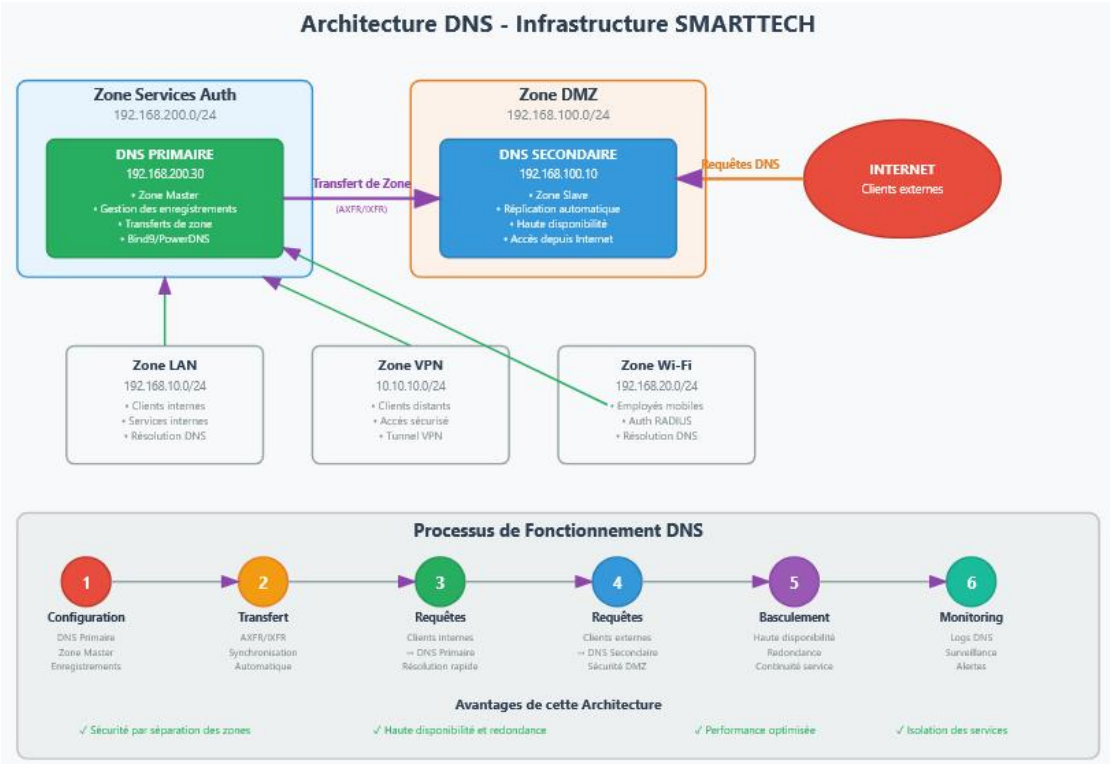
```
/ # service apache2 reload
```

Tester HTTPS dans Webterm

1. DNS PRIMAIRE ET SECONDAIRE :

Objectif

- DNS **primaire (master)** : sur la machine `192.168.200.30` (zone Auth)
- DNS **secondaire (slave)** : sur la machine `192.168.100.10` (DMZ)



- **DNS PRIMAIRE — 192.168.200.30**

Installe le service DNS Bind9 et les outils nécessaires.

```
phone1@phone:~$ sudo apt install -y bind9 bind9utils bind9-doc
```

Puis on edite le fichier tp.local

```
 GNU nano 7.2                          /etc/bind/db.tp.local
$TTL 86400
@   IN  SOA dns1.smarttech.local. admin.tp.local. (
        2025062201 ; Serial
        3600       ; Refresh
        1800       ; Retry
        1209600    ; Expire
        86400 )    ; Negative Cache TTL
;
@       IN  NS  dns1.smarttech.local.
ldap     IN  A   192.168.200.50  ;Serveur LDAP
krb     IN  A    192.168.200.50  ;Serveur Kerberos
kdc     IN  A    192.168.200.50  ;Hostname
```

Ce fichier définit la zone DNS directe **tp.local** avec un serveur maître **dns1.smarttech.local**. Il associe les noms **ldap, kdc** et **dns1** à l'adresse IP 192.168.200.50, correspondant respectivement aux services **LDAP, Kerberos** et **DNS** principal. Le champ SOA configure les paramètres de synchronisation de zone.

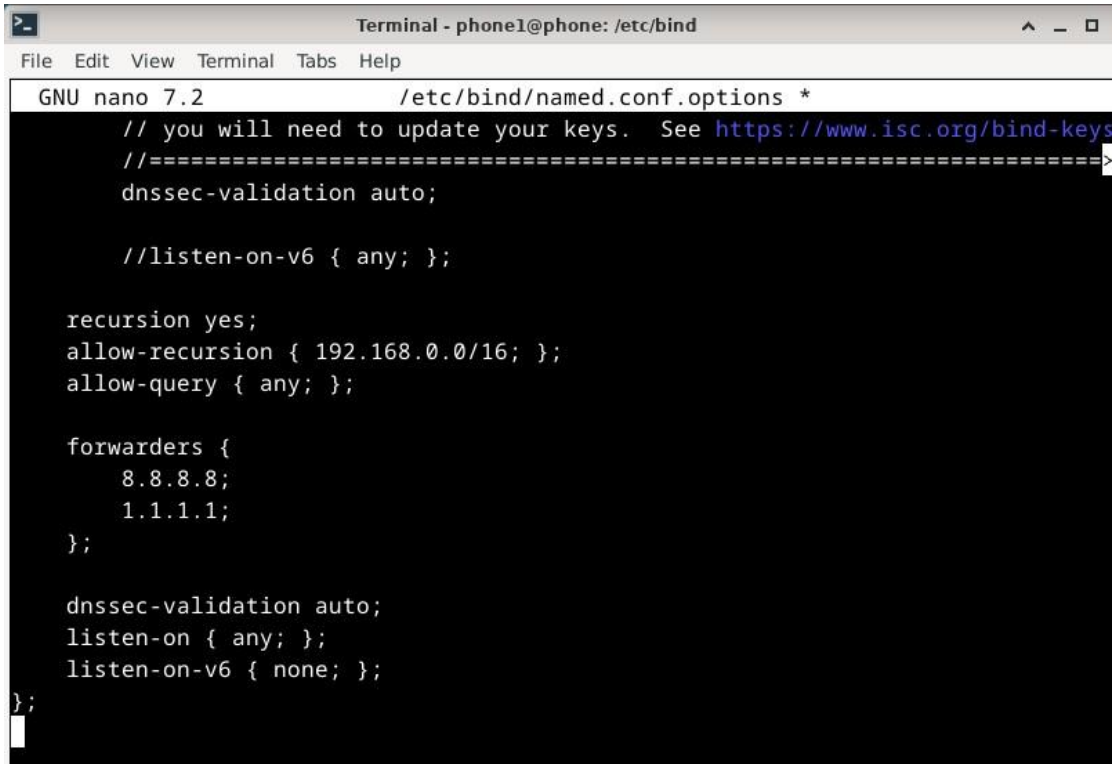
```
 GNU nano 7.2                          db.local
$TTL 86400
@   IN  SOA dns1.smarttech.local. admin.local. (
        2025062201 ; Serial
        3600
        1800
        1209600
        86400 )
;
@           IN  NS  dns1.smarttech.local.
nextcloud   IN  A   192.168.10.10
```

Dans le fichier **db.local** avec comme serveur maitre **dns1.smarttech.local, il associe nextcloud a l'adresse iP 192.168.10.10 ,** qui sera utiliser plustard dans le lan .

```
 GNU nano 7.2                  /etc/bind/db.192.168.200 *
$TTL 86400
@   IN  SOA dns1.smarttech.local. admin.smarttech.local. (
        2025062201
        3600
        1800
        1209600
        86400 )
;
@       IN  NS      dns1.smarttech.local.
50      IN  PTR     dns1.smarttech.local.
50      IN  PTR     ldp.tp.local.
10      IN  PTR     freeradius.tp.local.
```

Ce fichier configure la zone DNS inverse du réseau 192.168.200.0/24. Il associe les adresses IP aux noms **dns1.smarttech.local., 1dp.tp.local. (LDAP) et freeradius.tp.local**. via des enregistrements PTR. Le champ SOA définit les paramètres de gestion de zone pour le serveur DNS primaire.

On edite le fichier **/etc/bind/named.conf.options**



Ce fichier configure le DNS pour autoriser les requêtes internes (192.168.0.0/16) et activer la récursion. Il utilise les DNS publics 8.8.8.8 et 1.1.1.1 comme relais, et écoute sur toutes les interfaces réseau.

```
root@phone:~# cat > /etc/bind/db.smarttech.local <<'EOF'
$TTL 86400
@   IN  SOA dns1.smarttech.local. admin.smarttech.local. (
        2025062201 ; Serial
        3600       ; Refresh
        1800       ; Retry
        1209600    ; Expire
        86400 )    ; Negative TTL
;
@       IN  NS      dns1.smarttech.local.
dns1    IN  A       192.168.200.10
www     IN  A       192.168.100.20
proxy   IN  A       192.168.100.10
sip     IN  A       192.168.30.10
EOF
```

Ce fichier définit la zone DNS **smarttech.local** avec dns1.smarttech.local comme serveur principal. Il associe les noms **dns1**, **www**, **proxy** et **sip** à leurs adresses IP respectives dans les zones **Auth, DMZ** et **ToIP**.

Editons le fichier **/etc/bind/named.conf.local**

```
phone1@phone:/etc/bind$ sudo cat > /etc/bind/named.conf.local <<'EOF'
zone "smarttech.local" {
    type master;
    file "/etc/bind/db.smarttech.local";
    allow-transfer { 192.168.100.10; };
};

zone "tp.local" {
    type master;
    file "/etc/bind/db.tp.local";
    allow-transfer { 192.168.100.10; };
};

zone "local" {
    type master;
    file "/etc/bind/db.local";
    allow-transfer { 192.168.100.10; };
};

zone "200.168.192.in-addr.arpa" {
    type master;
    file "/etc/bind/db.192.168.200";
    allow-transfer { 192.168.100.10; };
};
EOF
```

Ce fichier déclare plusieurs zones DNS (**smarttech.local, tp.local, local, zone inverse 192.168.200.x**) en tant que zones maîtres, toutes autorisant le transfert vers le DNS secondaire 192.168.100.10. Chaque zone est associée à son fichier de données respectif pour la résolution directe ou inverse.

```
root@phone:~# named-checkconf
root@phone:~#
```

La commande **named-checkconf** vérifie que la configuration globale de BIND ne contient pas d'erreur.

```
root@phone:~# named-checkzone smarttech.local /etc/bind/db.smarttech.local
zone smarttech.local/IN: loaded serial 2025062201
OK
```

Ensuite, **named-checkzone** confirme que le fichier de zone **smarttech.local** est bien formaté et valide, avec le numéro de série 2025062201.

```
root@phone:~# named-checkzone tp.local /etc/bind/db.tp.local
zone tp.local/IN: loaded serial 2025062201
OK
```

Confirme que le fichier de zone **tp.local** est bien formaté et valide.

```
root@phone:~# named-checkzone 200.168.192.in-addr.arpa /etc/bind/db.192.168.200
zone 200.168.192.in-addr.arpa/IN: loaded serial 2025062201
OK
```

Confirme que le fichier de zone **zone inverse 192.168.200.x** est bien formaté et valide.

```
root@phone:~# named-checkzone local /etc/bind/db.local
/etc/bind/db.local:1: no TTL specified; using SOA MINTTL instead
zone local/IN: loaded serial 2025062201
OK
```

Confirme que le fichier de zone **local** est bien formaté et valide.

```
root@phone:~# systemctl restart bind9
```

La commande systemctl restart bind9 relance le service DNS pour appliquer les nouvelles configurations.

```
  GNU nano 7.2                    /etc/resolv.conf
# Generated by NetworkManager
search localdomain
nameserver 192.168.200.30
```

Le fichier /etc/resolv.conf permet de définir l'adresse IP du serveur DNS local utilisé 192.168.200.30 pour la résolution de noms.

```
phone1@phone:~$ dig ldap.tp.local

; <<>> DiG 9.18.33-1~deb12u2-Debian <<>> ldap.tp.local
;; global options: +cmd
;; Got answer:
;; WARNING: .local is reserved for Multicast DNS
;; You are currently testing what happens when an mDNS query is leaked to DNS
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 8873
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
; COOKIE: 9468d4694227aac30100000068588a15c58eea4a6117b203 (good)
;; QUESTION SECTION:
;ldap.tp.local.                 IN      A

;; ANSWER SECTION:
ldap.tp.local.          86400   IN      A       192.168.200.50

;; Query time: 19 msec
;; SERVER: 192.168.200.30#53(192.168.200.30) (UDP)
;; WHEN: Sun Jun 22 18:56:21 EDT 2025
```

La commande **dig ldap.tp.local** vérifie que le nom ldap.tp.local est bien résolu par le DNS primaire. La réponse reçue confirme que le nom correspond à l'IP 192.168.200.30, preuve que la zone est correctement configurée et fonctionnelle.

```
phone1@phone:~$ dig nextcloud.local

; <<>> DiG 9.18.33-1~deb12u2-Debian <<>> nextcloud.local
;; global options: +cmd
;; Got answer:
;; WARNING: .local is reserved for Multicast DNS
;; You are currently testing what happens when an mDNS query is leaked to DNS
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 60631
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
; COOKIE: f1d81000d0449bf60100000068588a3f6ceef041b3baf6e2 (good)
;; QUESTION SECTION:
;nextcloud.local.               IN      A

;; ANSWER SECTION:
nextcloud.local.        86400   IN      A       192.168.10.10

;; Query time: 15 msec
;; SERVER: 192.168.200.30#53(192.168.200.30) (UDP)
;; WHEN: Sun Jun 22 18:57:03 EDT 2025
```

la zone nextcloud.local est correctement configurée et fonctionnelle.

```
phone1@phone:~$ dig -x 192.168.200.10

; <<>> DiG 9.18.33-1~deb12u2-Debian <<>> -x 192.168.200.10
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 38201
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
; COOKIE: 670f9659e1df56960100000068588aaa54e7104930e366a9 (good)
;; QUESTION SECTION:
;10.200.168.192.in-addr.arpa.   IN      PTR

;; ANSWER SECTION:
10.200.168.192.in-addr.arpa. 86400 IN   PTR     freeradius.tp.local.
```

la zone 192.168.200.10 est correctement configurée et fonctionnelle.

```
phone1@phone:~$ ping nextcloud.local
PING nextcloud.local (192.168.10.10) 56(84) bytes of data.
64 bytes from 192.168.10.10 (192.168.10.10): icmp_seq=1 ttl=64 time=0.024 ms
64 bytes from 192.168.10.10 (192.168.10.10): icmp_seq=2 ttl=64 time=0.043 ms
64 bytes from 192.168.10.10 (192.168.10.10): icmp_seq=3 ttl=64 time=0.036 ms
```

Ici on fait un ping nextcloud.local et on voie bien que ca marche .

- **DNS SECONDAIRE --192.168.100.10 :**

Dans cette partie on a créer un script dns-secondaire.sh.

```
/ # vi dns-secondaire.sh
```

Un script bash a été utilisé pour automatiser l'installation du DNS secondaire, déclarer les zones esclaves synchronisées avec le maître, sauvegarder les configurations existantes, et redémarrer proprement le service named.

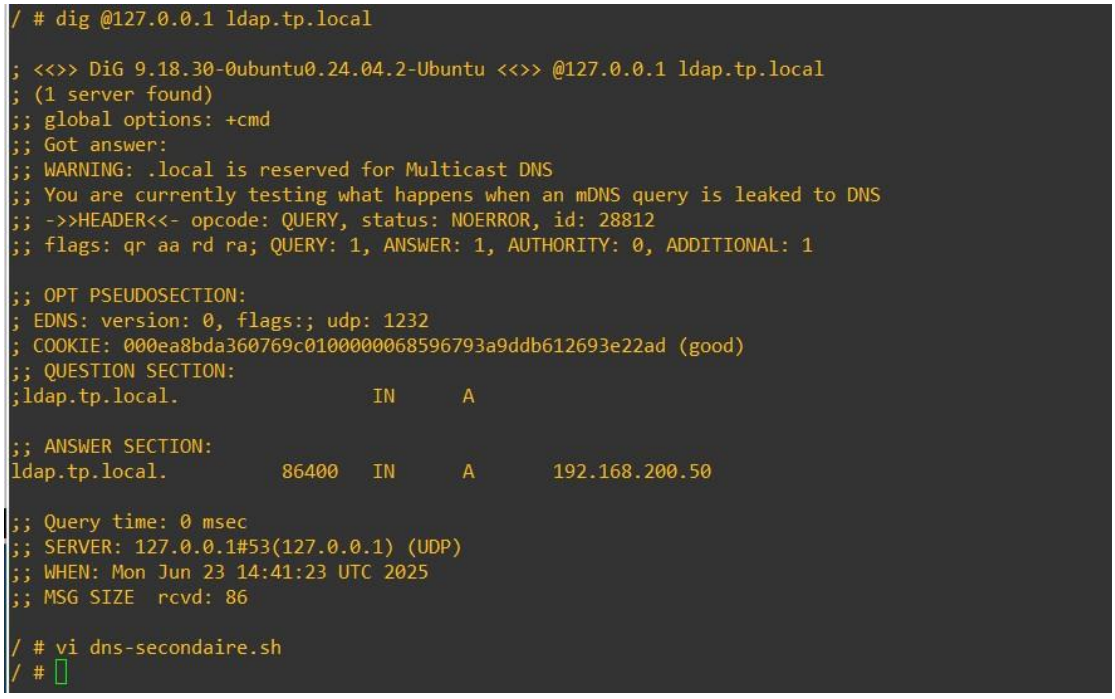
```
#!/bin/bash
set -e

# Adresse IP du DNS maître
DNS_MAITRE="192.168.200.30"

# Liste des zones à configurer en slave, correspondant aux zones master
ZONES=(
  "smarttech.local"
  "tp.local"
  "local"
  "200.168.192.in-addr.arpa"
)

# Répertoires
BIND_CONF_DIR="/etc/bind"
ZONE_CACHE_DIR="/var/cache/bind"

echo "📦 Mise à jour et installation de bind9 + dnsutils..."
apt update
apt install -y bind9 bind9utils bind9-doc dnsutils

echo "🛡️ Sauvegarde de la configuration existante (named.conf.local)..."
if [ -f "$BIND_CONF_DIR/named.conf.local" ]; then
  cp "$BIND_CONF_DIR/named.conf.local" "$BIND_CONF_DIR/named.conf.local.bak.$(date +%s)"
fi
```

- **DNS_MAITRE** : IP du serveur DNS maître (192.168.200.30).
- **ZONES** : liste des zones DNS à synchroniser depuis le maître.
- **apt install** : installe bind9 et les outils nécessaires (dnsutils, doc...).
- **Sauvegarde** : copie de sécurité de l'ancien fichier named.conf.local si présent.

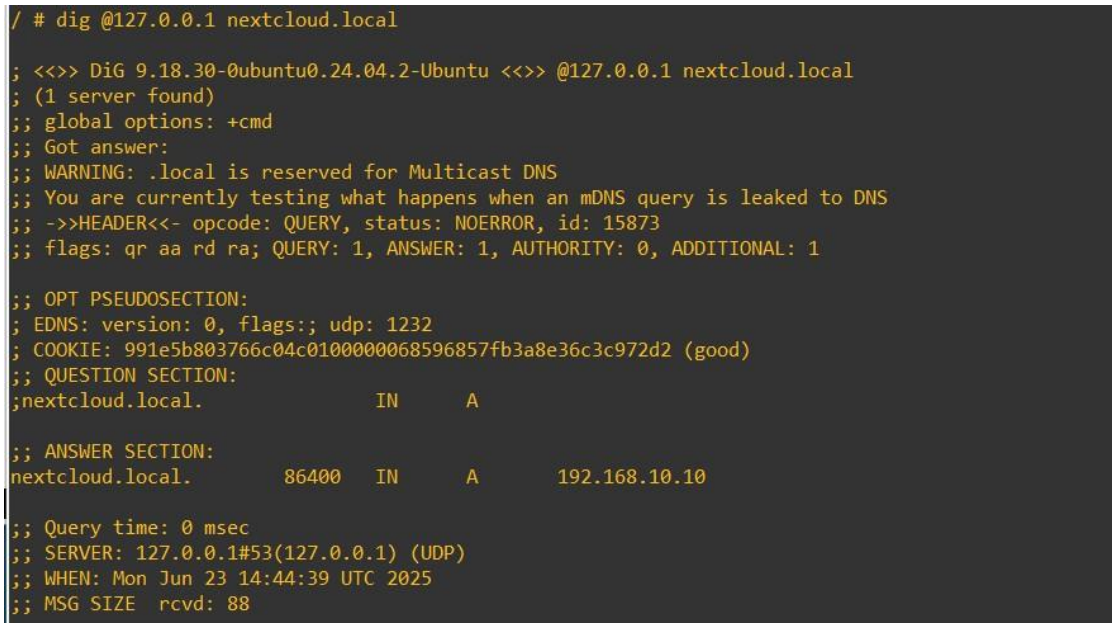
```
echo "📝 Création de named.conf.local avec les zones esclave (slave)..."
cat > "$BIND_CONF_DIR/named.conf.local" <<EOF
// Configuration des zones DNS secondaires (slaves)
EOF

for zone in "${ZONES[@]}"; do
  cat >> "$BIND_CONF_DIR/named.conf.local" <<EOF
zone "$zone" {
    type slave;
    masters { $DNS_MAITRE; };
    file "$ZONE_CACHE_DIR/db.$zone";
};

EOF
done

echo "🔍 Vérification de la configuration BIND..."
named-checkconf

echo "🔄 Redémarrage du service DNS 'named'..."
  service named restart
echo "✅ Configuration DNS secondaire terminée."
```

- **Création de named.conf.local** : initialise la config avec commentaire.

- **Boucle for** : ajoute chaque zone comme esclave (type slave) avec IP du maître et le chemin de cache.
- **named-checkconf** : vérifie que la config BIND est correcte.
- **service named restart** : redémarre le service DNS pour appliquer la config.



La commande **dig @127.0.0.1 ldap.tp.local** interroge localement le DNS secondaire pour vérifier la résolution du nom **ldap.tp.local**. La réponse correcte avec l'IP 192.168.200.50 confirme que la zone esclave a bien été synchronisée avec le DNS maître. Le serveur DNS secondaire fonctionne donc normalement.



la résolution du nom **nextcloud.local** est verifier ,la zone esclave a bien été synchronisée avec le DNS maître .

2. PROXY HTTP/HTTPS :

---

le proxy HTTP/HTTPS permet de contrôler et filtrer l'accès à Internet depuis les différentes zones du réseau (invités) Il joue aussi un rôle de sécurité en journalisant les connexions et en bloquant certains contenus ou sites non autorisés.

On installe le service **Squid**, qui jouera le rôle de proxy HTTP/HTTPS pour filtrer et gérer les connexions sortantes vers Internet

```
/ # apt install squid -y
```

On initialise les répertoires de cache de Squid pour qu'il puisse stocker les données temporaires lors de son fonctionnement.

```
/ # squid -z
```

Sauvegardons avant modification du fichier principal de configuration /etc/squid/squid.conf

```
/ # cp /etc/squid/squid.conf /etc/squid/squid.conf.bak
/ # vi /etc/squid/squid.conf
```

Le proxy écoute sur le port **3128** et autorise l'accès Internet uniquement aux machines **invitées** (**192.168.50.0/24**) et à localhost pour les tests. Toutes les autres connexions sont bloquées, et le proxy utilise 8.8.8.8 et 1.1.1.1 comme serveurs DNS.

```
#squid Config pour DMZ
http_port 3128

# Autoriser les invités
acl invited_net src 192.168.50.0/24
http_access allow invited_net

# Autoriser localhost (test local)
acl localnet src 127.0.0.1/32
http_access allow localnet

# Refuser tout le reste
http_access deny all

# DNS utilisé par squid
dns_nameservers 8.8.8.8 1.1.1.1
```

On lance Squd en mode premier plan (-N) avec un niveau de débogage minimal (-d1) pour vérifier si la configuration fonctionne correctement.

```
/ # squid -N -d1
```

INVITES

Depuis la machine **invité(192.168.50.10)** on teste le fonctionnement du proxy Squid en envoyant une requête HTTPS vers Google via l'adresse du proxy (*192.168.100.10:3128*) avec **curl**.

```
/ # curl -x 192.168.100.10:3128 https://google.com
```

On voie dans la console un code html qui représente la page google.com

```
/ # curl -x 192.168.100.10:3128 https://google.com
<HTML><HEAD><meta http-equiv="content-type" content="text/html;charset=ut
f-8">
<TITLE>301 Moved</TITLE></HEAD><BODY>
<H1>301 Moved</H1>
The document has moved
<A HREF="https://www.google.com/">here</A>.
```

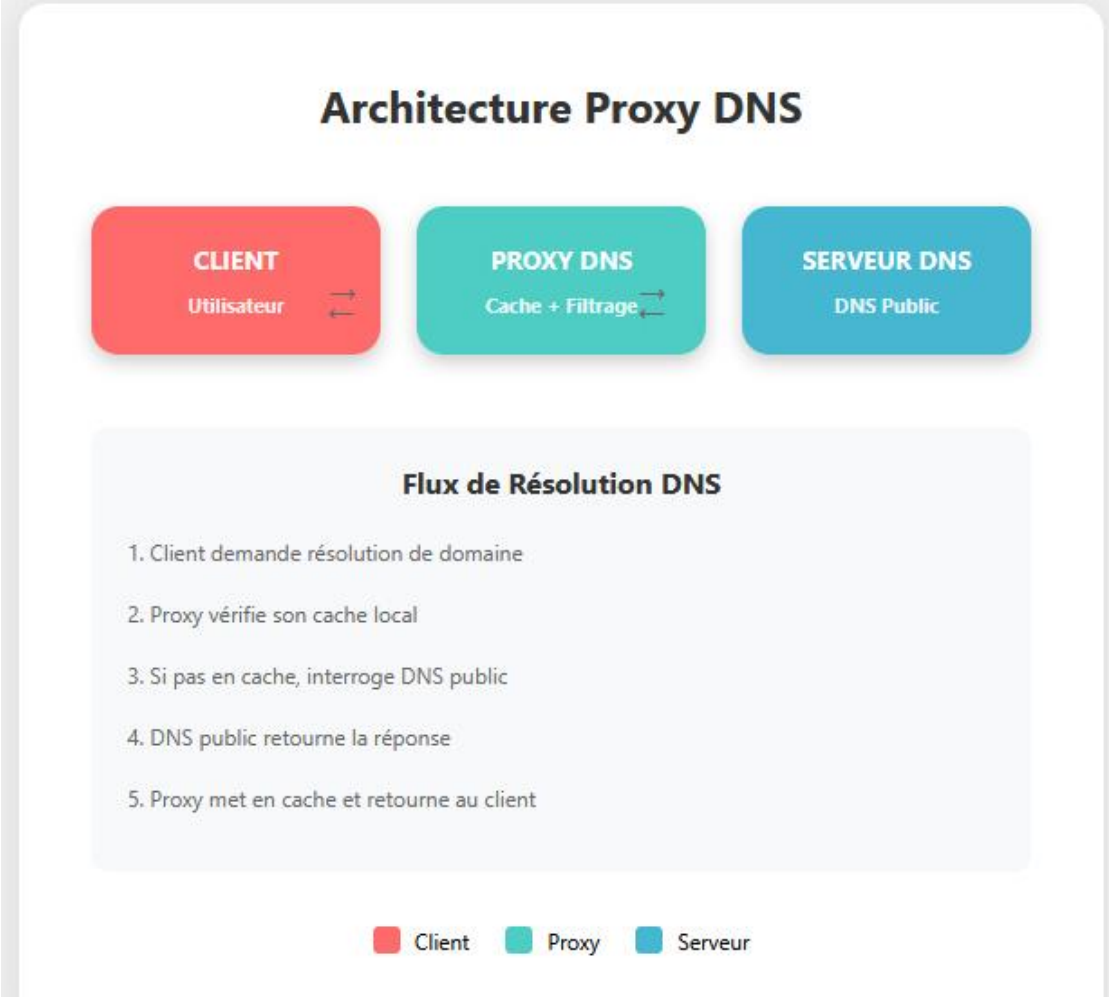
Avant de faire le test dans l'invité avec curl on avait lancé une capture wireshark dans son réseau (192.168.50.0/24)

36	46.527452	192.168.100.10	192.168.50.10	TCP	74 3128 → 33582 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=14
37	46.527753	192.168.50.10	192.168.100.10	TCP	66 33582 → 3128 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=15738
38	46.527776	192.168.50.10	192.168.100.10	HTTP	177 CONNECT google.com:443 HTTP/1.1
39	46.560160	192.168.100.10	192.168.50.10	TCP	66 3128 → 33582 [ACK] Seq=1 Ack=112 Win=65152 Len=0 TSval=269
40	46.709499	192.168.100.10	192.168.50.10	HTTP	105 HTTP/1.1 200 Connection established

- Le client invité tente de **naviguer en HTTPS** vers google.com
- Il envoie une **requête CONNECT** au proxy pour lui dire "fais le tunnel vers google.com:443"
- Le proxy **accepte la connexion** et établit un tunnel TCP sécurisé

Le proxy **joue bien son rôle** : il permet au client invité d'accéder à l'Internet en HTTPS (à condition que ça soit autorisé dans squid.conf).

3. PROXY-DNS :



Nous avons executer un script qui installe **dnsmasq**, sauvegarde l'ancienne configuration, puis écrit une nouvelle config adaptée au rôle de **proxy DNS**. Il écoute sur l'interface réseau **eth0** et sur l'adresse **192.168.100.20**. Il redirige les requêtes internes vers le DNS maître (**192.168.200.30**) et les requêtes externes vers **8.8.8.8** et **1.1.1.1**. Il active aussi le cache, les logs, et empêche l'écoute sur d'autres interfaces pour plus de sécurité.

```
set -e

echo "✎ Installation de dnsmasq..."
apt update
apt install -y dnsmasq

echo "🗎 Sauvegarde de la configuration existante..."
cp /etc/dnsmasq.conf /etc/dnsmasq.conf.bak

echo "🖊 Écriture de la nouvelle configuration..."
tee /etc/dnsmasq.conf > /dev/null << 'EOF'
##########################################
# dnsmasq.conf – Proxy DNS (192.168.100.20) ##
##########################################

# Interface réseau utilisée dans la DMZ
interface=eth0
listen-address=127.0.0.1,192.168.100.20

# Taille du cache DNS
cache-size=1000

# Ne pas utiliser /etc/resolv.conf
no-resolv

# Serveurs DNS internes
server=/tp.local/192.168.200.30


server=/smarttech.local/192.168.200.30


server=/local/192.168.200.30
```

```
# Serveurs DNS publics (pour les domaines externes)
server=8.8.8.8
server=1.1.1.1

# Empêche l'écoute sur d'autres interfaces réseau
bind-interfaces

# Log des requêtes DNS (pour débogage)
log-queries
log-facility=/var/log/dnsmasq.log
EOF

echo "🚀 Redémarrage du service dnsmasq..."
 service  dnsmasq restart

echo "☑ Installation et configuration terminées."
```

On installe et active systemd-resolved pour gérer la résolution DNS de manière centralisée avec systemd.

```
phone1@phone:~$ sudo apt install systemd-resolved
```

On lie le fichier de configuration DNS **/etc/resolv.conf** vers celui géré par **systemd** pour qu'il prenne le relais. On vérifie que le lien symbolique a bien été créé.

```
phone1@phone:~$ sudo systemctl enable systemd-resolved
phone1@phone:~$ sudo systemctl start systemd-resolved
phone1@phone:~$ sudo ln -sf /run/systemd/resolve/resolv.conf /etc/resolv.conf
phone1@phone:~$ ls -l /etc/resolv.conf
lrwxrwxrwx 1 root root 32 Jun 23 08:37 /etc/resolv.conf -> /run/systemd/resolve/
resolv.conf
phone1@phone:~$ sudo nano /etc/systemd/resolved.conf
```

*On édite le fichier de configuration de **systemd-resolved** pour ajuster les options si besoin (ex : DNS, fallbackDNS, domains, etc.).*

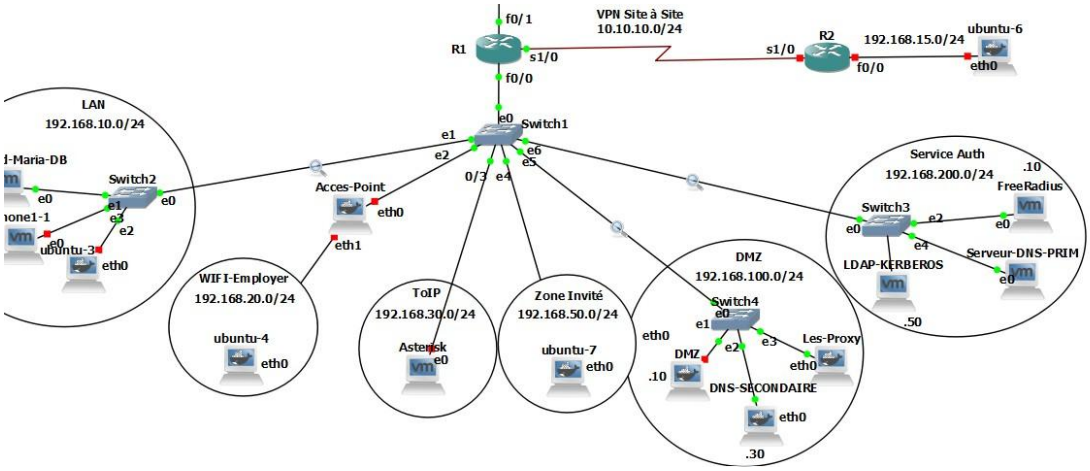
```
  GNU nano 7.2                  /etc/systemd/resolved.conf
#  systemd is free software; you can redistribute it and/or modify it under the
#  terms of the GNU Lesser General Public License as published by the Free
#  Software Foundation; either version 2.1 of the License, or (at your option)
#  any later version.
#
# Entries in this file show the compile time defaults. Local configuration
# should be created by either modifying this file, or by creating "drop-ins" in
# the resolved.conf.d/ subdirectory. The latter is generally recommended.
# Defaults can be restored by simply deleting this file and all drop-ins.
#
# Use 'systemd-analyze cat-config systemd/resolved.conf' to display the full co
#
# See resolved.conf(5) for details.

[Resolve]
DNS=192.168.100.20
FallbackDNS=8.8.8.8
Domains=tp.local smarttech.local local
```

*On configure systemd-resolved pour qu'il utilise le **proxy DNS local (**192.168.100.20**) comme serveur principal, avec* 8.8.8.8 *en secours. On précise aussi les domaines à traiter en local pour forcer la résolution interne.*

On relance *systemd-resolved* pour appliquer les nouvelles configurations.

```
phone1@phone:~$ sudo systemctl restart systemd-resolved
```

On fais une capture wireshark dans les réseaux DMZ (192.168.100.0) , SERVICE AUTH (192.168.200.0) et LAN (192.168.10.0) pour voir les paquet qui serons échanger lors des test avec la commande dig .



On utilise la commande dig pour tester la résolution du nom ldap.tp.local via le **proxy DNS (dnsmasq)** à l'adresse 192.168.100.20.

```
phone1@phone:~$ dig ldap.tp.local
```

La réponse correcte avec l'IP 192.168.200.50 prouve que le proxy DNS redirige bien les requêtes internes vers le DNS maître.

```
; <<>> DiG 9.18.33-1~deb12u2-Debian <<>> ldap.tp.local
;; global options: +cmd
;; Got answer:
;; WARNING: .local is reserved for Multicast DNS
;; You are currently testing what happens when an mDNS query is leaked to DN
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 59848
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
; COOKIE: c4e5fbeda8f0b02e01000000685acb4766d33450f3905cc5 (good)
;; QUESTION SECTION:
;ldap.tp.local.                 IN      A

;; ANSWER SECTION:
ldap.tp.local.          86400   IN      A       192.168.200.50

;; Query time: 28 msec
;; SERVER: 192.168.100.20#53(192.168.100.20) (UDP)
;; WHEN: Tue Jun 24 11:59:03 EDT 2025
;; MSG SIZE  rcvd: 86
```

1	0.000000	192.168.10.10	192.168.100.20	DNS	96 Standard query 0x394c A ldap.tp.local OPT
2	0.000779	192.168.100.20	192.168.200.30	DNS	96 Standard query 0x24f3 A ldap.tp.local OPT
3	0.020385	192.168.200.30	192.168.100.20	DNS	128 Standard query response 0x24f3 A ldap.tp.local A 192.168.200.50 OPT
4	0.021179	192.168.100.20	192.168.10.10	DNS	128 Standard query response 0x394c A ldap.tp.local A 192.168.200.50 OPT
5	5.174366	02:42:b0:2f:a7:00	c4:01:20:08:00:00	ARP	42 Who has 192.168.100.1? Tell 192.168.100.20
6	5.192321	c4:01:20:08:00:00	02:42:b0:2f:a7:00	ARP	60 192.168.100.1 is at c4:01:20:08:00:00

La capture montre une requête DNS envoyée depuis 192.168.10.10 vers le proxy DNS 192.168.100.20 pour résoudre ldap.tp.local. Le proxy a bien redirigé la requête vers le DNS principal, qui a répondu avec l'adresse 192.168.200.50. Cela confirme que la chaîne de résolution fonctionne entre client, proxy et DNS maître.

Cette requête dig google.com montre que le proxy DNS (192.168.100.20) peut aussi résoudre des noms **externes** grâce aux serveurs DNS publics configurés (8.8.8.8, 1.1.1.1).

```
phone1@phone:~$ dig google.com

; <<>> DiG 9.18.33-1~deb12u2-Debian <<>> google.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 28095
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
;; QUESTION SECTION:
;google.com.                    IN      A

;; ANSWER SECTION:
google.com.             282     IN      A       142.250.185.14

;; Query time: 52 msec
;; SERVER: 192.168.100.20#53(192.168.100.20) (UDP)
;; WHEN: Mon Jun 23 08:58:35 EDT 2025
;; MSG SIZE  rcvd: 55
```

La réponse avec l'IP 142.250.185.14 confirme le bon fonctionnement du proxy DNS pour l'Internet.

20	460.736839	192.168.10.10	192.168.100.20	DNS	93 Standard query 0x6dbf A google.com OPT
21	460.737681	192.168.100.20	8.8.8.8	DNS	93 Standard query 0xfcc8 A google.com OPT
22	460.737723	192.168.100.20	1.1.1.1	DNS	93 Standard query 0xfcc8 A google.com OPT
23	460.767524	1.1.1.1	192.168.100.20	DNS	97 Standard query response 0xfcc8 A google.com A 142.250.185.14 OPT
24	460.768064	192.168.100.20	192.168.10.10	DNS	97 Standard query response 0x6dbf A google.com A 142.250.185.14 OPT
25	460.818470	8.8.8.8	192.168.100.20	DNS	97 Standard query response 0xfcc8 A google.com A 142.250.179.78 OPT

La capture montre que le client 192.168.10.10 envoie une requête DNS pour google.com au proxy 192.168.100.20. Ce dernier interroge à la fois 8.8.8.8 et 1.1.1.1, puis transmet la réponse reçue (142.250.185.14) au client. Cela confirme que le **proxy DNS relaie correctement les requêtes externes** vers les DNS publics et renvoie la réponse au poste client.

87	372.183068	VMware_5a:f3:5f	c4:01:20:08:00:00	ARP	60 Who has 192.168.10.1? Tell 192.168.10.10
88	372.191480	c4:01:20:08:00:00	VMware_5a:f3:5f	ARP	60 192.168.10.1 is at c4:01:20:08:00:00
89	432.714436	192.168.10.1	224.0.0.9	RIPv2	166 Response
90	460.736350	192.168.10.10	192.168.100.20	DNS	93 Standard query 0x6dbf A google.com OPT
91	460.786779	192.168.100.20	192.168.10.10	DNS	97 Standard query response 0x6dbf A google.com A 142.250.185.14 OPT

La capture montre d'abord une requête ARP de 192.168.10.10 pour trouver l'adresse MAC de 192.168.10.1, suivie de la réponse. Ensuite, on observe un message RIP v2 et une requête DNS envoyée par le client vers le proxy DNS 192.168.100.20 pour résoudre google.com, ce qui confirme le bon routage et la communication inter-réseaux.

CONCLUSION :

La mise en place de la DMZ renforce significativement la sécurité du réseau de l'entreprise en cloisonnant les services accessibles depuis Internet.
Grâce à une configuration rigoureuse du pare-feu et des services (Apache, Squid, DNS), seuls les flux autorisés circulent entre les zones.
Ce dispositif réduit les risques d'intrusion tout en assurant la disponibilité des services publics de manière sécurisée.