UNIVERSITE GASTON BERGER DE SAINT LOUIS

UFR DE SCIENCES APPLIQUEES ET DE TECHNOLOGIE

CENTRE DE FORMATION PROFESSIONNELLE PLURIDISCIPLINAIRE

(CFPP)

MASTER PROFESSIONNELLE RESEAUX & TELECOMMUNICATION

(MART)

Mémoire de fin d'étude

MISE EN PLACE D'UNE SOLUTION ZÉRO CONFIANCE POUR L'ACCÈS RÉSEAU

PRÉSENTÉ ET SOUTENU PAR :

ABDOULAYE MAHAMAT BOKHIT

SOUS LA DIRECTION DE :

Pr. CHERIF DIALLO

Année Académique 2025/2026

Remerciements

Je tiens à exprimer ma profonde gratitude à mon grand frère **Ali Mht Bokhtit**, pour son soutien constant, ses conseils et ses encouragements tout au long de ce parcours.
Mes sincères remerciements vont également à mon père, **Mahamat Bokhit Erip**, pour son soutien moral, ses prières et les valeurs qu'il m'a transmises.

Je remercie tout particulièrement mon encadreur, **Pr Chérif Diallo**, pour sa disponibilité, son accompagnement et la qualité de ses orientations, qui ont largement contribué à la réussite de ce travail.

Je souhaite également adresser mes remerciements à l'ensemble des enseignants de l'**UFR SAT**, pour la qualité de la formation dispensée, ainsi qu'à l'administration du **CFPP** pour son appui et son encadrement administratif.

Je remercie aussi toute la **grande famille**, pour son soutien moral, sa solidarité et ses encouragements constants tout au long de ce parcours.

Enfin, je remercie chaleureusement mes amis, mes camarades de la promotion **Master Réseaux et Télécommunications 2025–2026**, ainsi que **l'ensemble des étudiants tchadiens de Saint-Louis**, pour leur entraide, leur solidarité et leur soutien.

Résumé

Ce mémoire présente la conception et la mise en œuvre d'une architecture **Zero Trust Network Access (ZTNA)** destinée à renforcer la sécurité des accès aux applications internes. Face aux limites des architectures traditionnelles reposant sur les VPN et la confiance implicite accordée au réseau, cette étude propose une approche fondée sur le principe **« Ne jamais faire confiance, toujours vérifier »**, où chaque demande d'accès est authentifiée, autorisée et contrôlée selon l'identité de l'utilisateur.

La solution développée repose sur une architecture modulaire composée d'un **serveur d'authentification Keycloak**, d'un **reverse proxy Zero Trust Pomerium**, d'un **serveur d'applications protégé par Nginx**, d'une **base de données MariaDB**, d'un **serveur de supervision intégrant Prometheus, Grafana et Node Exporter**, ainsi que d'un **Cloudflare Tunnel** pour l'exposition sécurisée des applications publiques. L'ensemble de ces composants communique via des protocoles sécurisés et applique des politiques d'accès basées sur les rôles des utilisateurs.

L'implémentation a permis de démontrer qu'une architecture ZTNA réduit efficacement la surface d'attaque, limite les mouvements latéraux, améliore la gestion des identités, renforce la supervision des services et offre un contrôle d'accès granulaire aux applications. Les différentes simulations réalisées ont confirmé le bon fonctionnement de la solution et sa capacité à protéger les ressources internes tout en garantissant une meilleure visibilité sur les accès et les événements de sécurité.

Mots-clés : Zero Trust Network Access (ZTNA), Keycloak, Pomerium, Nginx, Cloudflare Tunnel, Prometheus, Grafana, MariaDB, Authentification, Cybersécurité, Contrôle d'accès, Supervision.

Abstract

This thesis presents the design and implementation of a **Zero Trust Network Access (ZTNA)** architecture aimed at strengthening the security of access to internal applications. In response to the limitations of traditional security architectures based on **Virtual Private Networks (VPNs)** and the implicit trust granted to internal networks, this work proposes an approach based on the principle **"Never Trust, Always Verify,"** where every access request is authenticated, authorized, and continuously verified according to the user's identity.

The proposed solution is built on a modular architecture composed of a **Keycloak authentication server**, a **Pomerium Zero Trust reverse proxy**, an **Nginx application reverse proxy**, a **MariaDB database**, a **monitoring server integrating Prometheus, Grafana, and Node Exporter**, as well as a **Cloudflare Tunnel** for the secure exposure of public applications. These components communicate through secure protocols and enforce identity- and role-based access control policies.

The implementation demonstrates that a ZTNA architecture effectively reduces the attack surface, prevents lateral movement, improves identity and access management, enhances infrastructure monitoring, and provides fine-grained access control to protected applications. The different simulation scenarios validated the effectiveness of the proposed solution in securing internal resources while ensuring better visibility into user access and security events.

Keywords: Zero Trust Network Access (ZTNA), Keycloak, Pomerium, Nginx, Cloudflare Tunnel, Prometheus, Grafana, MariaDB, Authentication, Cybersecurity, Access Control, Monitoring.

Table des matières

Introduction générale 1

Chapitre 1 : Fondements théorique de la solution ZTNA..... 3

Introduction 3

1.1 Evolution de la sécurité des systèmes d'informations 4

1.2 Limite des architectures de sécurité traditionnelle 5

1.3 Origine et principes du ZTNA 6

1.4 Zero Trust Network Access (ZTNA) : définition et concepts clés 8

1.5 Avantages, limites et enjeux du ZTNA 9

1.5.1 Avantages du Zéro Trust Network Access 9

1.5.2 Limites du Zéro Trust Network Access 10

1.5.3 Enjeux liés à l'adoption du ZTNA 10

1.6 Étude comparative entre le ZTNA et les VPN traditionnels 11

Conclusion 13

Chapitre 2 : Analyse du besoin et conception de la solution ZTNA .. 15

Introduction 15

2.1 Présentation de l'environnement étudié 15

2.2 Description générale de l'architecture 16

2.3 Problématique de sécurité dans l'environnement cible 17

2.4 Justification de l'approche ZTNA dans cet environnement 18

2.4.1 Positionnement de l'environnement par rapport au Zero Trust 18

2.4.2 Analyse des besoins fonctionnels 18

2.4.2.1 Besoin d'authentification centralisée 19

2.4.2.2 Besoin de contrôle d'accès applicatif granulaire 19

2.4.2.3 Besoin de sécurisation des accès distants 20

2.4.2.4 Besoin de supervision et de traçabilité 20

2.4.2.5 Besoin d'intégration et d'évolutivité 21

2.4.3 Analyse des besoins non fonctionnels 21

2.4.3.1 Exigences de sécurité 21

2.4.3.2 Exigences de performance 22

2.4.3.3 Exigences de disponibilité et de fiabilité 22

2.4.3.4 Exigences de maintenabilité et d'évolutivité 23
2.4.3.5 Exigences de conformité et de traçabilité 23
2.5 Choix technologiques 24
2.5.1 Choix du serveur d'authentification : Keycloak 24
2.5.1.1 Architecture et fonctionnement de Keycloak 25
2.5.1.2 Sécurité et standards supportés 26
2.5.1.3 Intégration de Keycloak dans une architecture ZTNA 26
2.5.1.4 Avantages, limites et justification du choix 27
2.5.2 Choix du serveur d'authentification : Pomerium 28
2.5.2.1 Présentation générale et objectifs 28
2.5.2.2 Architecture et fonctionnement de Pomerium 28
2.5.2.2.1 Proxy 29
2.5.2.2.1 Authentifications 30
2.5.2.2.2 Autorisations 31
2.5.2.2.3 Courtages de données 31
2.5.2.2.4 Sécurité, politiques et standards supportés 32
2.5.2.3 Intégration de Pomerium dans une architecture ZTNA 33
2.5.2.4 Avantages, limites et justification du choix 34
2.5.3 Choix des applications web et du serveur Nginx 34
2.5.4 Cloudflared 35
2.5.4.1 Architecture et fonctionnement de Cloudflared 36
2.5.4.2 Sécurité et mécanismes de protection 37
2.5.4.3 Intégration de Cloudflared dans l'architecture ZTNA 38
2.5.4.4 Avantages, limites et justification du choix 38
2.5.5 Prometheus 39
2.5.5.1 Présentation générale et objectifs 39
2.5.5.2 Fonctionnement de Prometheus 39
2.5.5.3 Sécurité, métriques et standards 40
2.5.5.4 Intégration de Prometheus dans une architecture ZTNA 41
2.5.6 Grafana 42
2.5.6.1 Présentation générale et objectifs 42
2.5.6.2 Architecture et fonctionnement de Grafana 42
2.5.6.3 Sécurité, gestion des accès et standards 43
2.5.6.4 Intégration de Grafana dans une architecture ZTNA 44
2.5.6.5 Avantages, limites et justification du choix 45
Conclusion 45

Chapitre 3 : Mise en œuvre de la solution Zero Trust Network Access (ZTNA) 47
Introduction 47
3.1 Présentation générale de l'architecture mise en place 47
3.2 Mise en œuvre Keycloak,Mysql,Nginx 52
3.2.1 Environnement de travail 53
3.2.1.1 Configuration réseaux 53
3.2.2 Installation de keycloak en mode test 54
3.2.2.1 Copie du fichier de téléchargement vers le serveur 54
3.2.2.2 Vérifications du fichier 54
3.2.2.3 Installation de java 55
3.2.2.3.1 Vérification de la version de java 55
3.2.2.3.2 Installation java jdk 55
3.2.2.4 Décompression du fichier d'installation keycloak 56
3.2.2.5 Configuration de keycloak H2 56
3.2.3 Passage en mode Production 62
3.2.3.1 Configuration du serveur MySQL 62
3.2.4 Configuration du serveur keycloak 64
3.2.5 Configuration du proxy interne 72
3.2.6 Configuration du reverse proxy Nginx 74
3.3 Mise en œuvre du serveur web (Nginx) 80
3.3.1 Création de l'arborescence des applications 80
3.3.2 Attribution des permissions sur le répertoire RH 81
3.3.3 Création des fichiers applicatifs 81
3.3.4 Structure des fichiers de l'application RH dans /var/www/rh 82
3.3.5 Création du fichier de configuration Nginx 85
3.3.6 Configuration du virtual host Nginx pour l'application RH 86
3.3.7 Activation du site RH dans Nginx 87
3.3.8 Accès à l'application RH via le navigateur 87
3.3.9 Accès à l'application Finance 88
3.4 Mise en œuvre du reverse proxy Zero Trust Pomerium 89
3.4.1 Installation de Pomerium 89
3.4.2 Génération des certificats TLS pour les domaines internes 91
3.4.2.1 Vérification des certificats TLS générés 91
3.4.2.2 Copie des certificats dans /etc/ssl 92

3.4.2.3 Transfert sécurisé des certificats vers le serveur Pomerium 92
3.4.2.4 Intégration de l'autorité de certification locale 93
3.4.2.5 Mise à jour du magasin des certificats système 93
3.4.2.6 Vérification de l'autorité de certification 93
3.4.3 Génération d'un secret cryptographique partagé 94
3.4.4 Configuration de Keycloak pour l'intégration avec Pomerium 94
3.4.4.1 Création du client OIDC Pomerium 94
3.4.4.2 Configuration des URLs d'accès et de redirection 95
3.4.4.3 Configuration du logout et des origines web 96
3.4.4.4 Activation de l'authentification client 97
3.4.4.5* Génération du *Client Secret 97
3.4.4.6* Création du *Mapper Groups 98
3.4.4.7 Création des groupes 98
3.4.4.8 Création des utilisateurs 99
3.4.4.9 Affectation des utilisateurs aux groupes 99
3.4.4.10 Vérification des groups 100
3.4.5 Fichier de configuration pomerium 101
3.4.6 Simulation de l'accès à l'application Finance protégée par ZTNA 103
3.4.7 Simulation de l'accès à l'application RH protégée par la solution ZTNA 106
3.4.8 Simulation d'accès refusé avec l'application RH 107
3.5 Mise en place de la supervision avec Prometheus, Grafana et Node Exporter 109
3.5.1 Déploiement de Node Exporter 110
3.5.1.1 Installation du Node Exporter 110
3.5.1.2 Création du service systemd Node Exporter 112
3.5.1.3 Vérification de l'écoute sur le port 9100 et Test de l'endpoint 113
3.5.2 Déploiement prometheus 114
3.5.2.1 Installation des dépendances 114
3.5.2.2 Configuration du noyau pour le réseau 115
3.5.2.3 Création de l'utilisateur Prometheus 115
3.5.2.4 Activation du forwarding IP 115
3.5.2.5 Création des répertoires Prometheus et Attribution des droits 116
3.5.2.6 Installation prometheus 116
3.5.2.7 Fichier de configuration prometheus 118
3.5.2.8 Fichier de configuration alerte 119
3.5.3 Déploiement Grafana 121
3.5.4 Création du répertoire des clés APT 122

3.5.3.1 Ajout de la clé GPG de Grafana 122
3.5.3.2 Ajout du dépôt officiel Grafana 122
3.5.3.3 Installation de Grafana 123
3.5.3.4 Génération du certificat TLS avec mkcert 124
3.5.3.5 Fichier de configuration grafana 126
3.5.3.6 Configuration cote keycloak 127
3.5.3.7 Simulation 127
3.6 Mise en œuvre du cloudflared 129
3.6.1 Installation de cloudfared 130
3.6.2 Fichier de configuration cloudflared 133
3.6.3 Configuration Nginx 135
3.6.4 Simulation 136
Conclusion 137
Conclusion générale 139

Tables des Figures

Figure 1: architecture ztna : https://www.syteca.com/en/blog/zero-trust-security-model 17
Figure 2: Architecture keycloak : https://miro.medium.com/1*SZRfDnFdMlBYtej_Sp19fw.jpeg..... 25
Figure 3: Architecture pomerium : https://www.pomerium.com/assets/images/pomerium-system-context-552143c2ee38dde267fa514a37fb9878.svg 29
Figure 4: Architecture cloudflared :..... 36
Figure 5:Architecture grafana : https://grafana.com/static/img/docs/grafana-cloud/arch_diagrams/hostedonly.jpg 43
Figure 6: Architecture de la solution 47
Figure 7: ping keycloak vers mysql..... 53
Figure 8: ping du client vers keycloak..... 53
Figure 9:ping de mysql vers keycloak 54
Figure 10: copie de keycloak vers le serveur..... 54
Figure 11: vérification du fichier 54
Figure 12: verification de java..... 55
Figure 13: installation jdk java 55
Figure 14: verification jdk java..... 55
Figure 15: Décompression du fichier d'installation keycloak 56
Figure 16: déplacement du fichier compresse 56
Figure 17: création d'utilisateur 56
Figure 18: attribuer les fichiers à keycloak..... 56
Figure 19: changement des permissions 56
Figure 20: lacement de keycloak en mode dev..... 57
Figure 21: Vérification de l'état de service keycloak.....57
Figure 22: accès http 58
Figure 23: création compte Admin 58
Figure 24: connexion avec Admin 59
Figure 25: Interface graphique keycloak 1.....60
Figure 26: Menu keycloak 2.....61
Figure 27:Installation MySQL 62
Figure 28: Fichier de configuration mysql 62
Figure 29: vérification de port d'écoute 63
Figure 30: création de la base données keycloak 63
Figure 31: création d'utilisateur 63
Figure 32: vérification base crée 64
Figure 33: vérification utilisateur.....64
Figure 34: installation libmariadb-java 65
Figure 35:crée un lien pour le démarrage 65
Figure 36: fichier de configuration keycloak.....65
Figure 37: recharger les configurations 67
Figure 38: vérification tcp.....67
Figure 39: connexion à la base de données keycloak.....67
Figure 40: sélection 68
Figure 41: connexion via le navigateur 68
Figure 42: connexions à l'admin temp 69
Figure 43: création du nouvel utilisateur 69
Figure 44: mot de passe nouvel utilisateur 70
Figure 45: vérification de l'utilisateur.....70

Figure 46: connexion avec le nouvel utilisateur 70
Figure 47: connexion navigateur 1 71
Figure 48: connexion navigateur 2 71
Figure 49: installation de mkcert 72
Figure 50: Création du CA 72
Figure 51: signature de keycloak.internal 73
Figure 52: sécurité des certificats 73
Figure 53: installation nginx 74
Figure 54: Configuration du pare-feu 75
Figure 55: Fichier de configuration Nginx 76
Figure 56: vérification des config 78
Figure 57: connexion https 79
Figure 58:création de l'arborescence 80
Figure 59: vérification de la structure de l'arborescence 80
Figure 60: attribution des droits aux répertoire 81
Figure 61: création des fichiers applicatifs 81
Figure 62: création de fichier de configuration ngnix 85
Figure 63; fichier de configuration nginx pour l'application rh 86
Figure 64: activation du site web 87
Figure 65: test d'accès à l'application rh 87
Figure 66: l'accès à l'application de finance 88
Figure 67:installation pomerium 89
Figure 68: sécurisation du fichier conf 90
Figure 69: installation de yamllint 90
Figure 70: génération de certificat TLS avec mkcert 91
Figure 71: vérification des certificats 91
Figure 72: copie vers répertoire ssl 92
Figure 73: transfert vers le serveur pomerium 92
Figure 74: intégration de l'autorité de certificat local 93
Figure 75: mise à jour du magasin 93
Figure 76: vérification de l'autorité 93
Figure 77: génération d'une clé secrète 94
Figure 78: création pomerium-client 94
Figure 79: configuration des URLs d'accès et redirection 95
Figure 80: Configuration logout et web origin 96
Figure 81: Activation de l'authentification 97
Figure 82: génération de client-id 97
Figure 83: création du mapper 98
Figure 84: création group 98
Figure 85: création des utilisateurs 99
Figure 86: Affectation des users aux groups 99
Figure 87: vérification des groups 100
Figure 88:fichier de configuration pomerium 101
Figure 89: connexion à l'app finance 103
Figure 90: redirection vers keycloak 104
Figure 91: authentification de l'utilisateur 104
Figure 92: connexion réussie 105
Figure 93: connexion à l'app rh 106

Figure 94: authentification rh....106
Figure 95: connexion réussie l'app rh....107
Figure 96: connexion à l'app rh avec mauvais identifiant....108
Figure 97: connexion refuse....109
Figure 98: installation....110
Figure 99: décompression du fichier....110
Figure 100: vérification du répertoire....111
Figure 101: copie vers bin....111
Figure 102: vérification de la version....111
Figure 103: fichier system node exporter....112
Figure 104: démarrage du service node exporter....113
Figure 105: vérification du port et métrique....113
Figure 106: installation des dépendances....114
Figure 107: installation des dépendances....114
Figure 108: configuration du noyau réseau....115
Figure 109: création de l'utilisateur Prometheus....115
Figure 110: activation du forwarding IP....115
Figure 111: création de répertoire et attribution des droits....116
Figure 112: Transfert de l'archive Prometheus vers le serveur....116
Figure 113: extraction du fichier prometheus....116
Figure 114: Copie des binaires Prometheus et Promtool....117
Figure 115: Attribution des droits aux binaires....117
Figure 116: Création des répertoires consoles....117
Figure 117: Vérification des versions....117
Figure 118: Fichier de configuration prometheus....118
Figure 119: alerte.yml....119
Figure 120: Droits sur les fichiers de configuration....120
Figure 121: Validation de la configuration....120
Figure 122: Vérification du port d'écoute....120
Figure 123: Accès à l'interface Web Prometheus....121
Figure 124: création du répertoire des clés....122
Figure 125: ajout de la clé....122
Figure 126: ajout dépôt officiel grafana....122
Figure 127: installation grafana....123
Figure 128 : vérification de l'installation....123
Figure 129: sécurisation des ports....123
Figure 130: sécurisation des ports....124
Figure 131:Génération du certificat TLS avec mkcert....124
Figure 132: copie des certificats....125
Figure 133: Placement des certificats dans le répertoire Grafana....125
Figure 134:Correction des permissions sur les fichiers de certificats....125
Figure 135: fichier de configuration grafana....126
Figure 136: création group pour grafana....127
Figure 137: supervision proxy....128
Figure 138: supervision serveur local....128
Figure 139: alerte....129
Figure 140:copie du cloudflared vers le serveur....130
Figure 141: installation du cloudflare....130

Figure 142: authentification cloudflare 131
Figure 143: autorisation tunnel..... 131
Figure 144: autorisation du tunnel..... 131
Figure 145: copie du certificat vers le serveur 132
Figure 147: création tunnel 132
Figure 148: création répertoire cloudflare..... 132
Figure 149: fichier de configuration cloudflare..... 133
Figure 150: dns route 134
Figure 151: fichier de configuration nginx..... 135
Figure 152: connexion à l'app publique 136
Figure 153: page d'accueil d'application 137

Liste des tableaux :

Tableau 1: Comparaisons VPN et ZTNA.....12

Liste des abréviations :

ABAC (Attribute-Based Access Control)

CIDR (Classless Inter-Domain Routing)

gRPC (Google Remote Procedure Call)

HTTP (HyperText Transfer Protocol)

HTTPS (HyperText Transfer Protocol Secure)

IAM (Identity and Access Management)

IdP (Identity Provider)

JWT (JSON Web Token)

LDAP (Lightweight Directory Access Protocol)

MFA (Multi-Factor Authentication)

NIST (National Institute of Standards and Technology)

OWASP (Open Worldwide Application Security Project)

RBAC (Role-Based Access Control)

SAML (Security Assertion Markup Language)

TLS (Transport Layer Security)

SSL (Secure Sockets Layer)

SSO (Single Sign-On)

VPN (Virtual Private Network)

ZTNA (Zero Trust Network Access)

Introduction générale

Au cours de la dernière décennie, le domaine de la cybersécurité a connu une transformation profonde, stimulée par l'évolution rapide des infrastructures informatiques, l'adoption massive des services cloud, la mobilité des utilisateurs et l'augmentation de la surface d'attaque. Les organisations modernes, publiques comme privées, doivent désormais faire face à des menaces toujours plus sophistiquées : ransomwares, compromission d'identifiants, attaques internes, vol de données sensibles, exploitation des tunnels VPN classiques, etc. Dans ce contexte, les approches traditionnelles de sécurité, fondées sur la construction d'un périmètre réseau « sûr », montrent leurs limites.

Historiquement, la sécurité réseau reposait sur le paradigme du **Castle and Moat** (château et douves), selon lequel un dispositif de protection (pare-feu, VPN) était censé protéger le réseau interne. Une fois à l'intérieur, les utilisateurs et les terminaux étaient considérés comme fiables. Ce modèle, bien qu'efficace dans les infrastructures centralisées et fermées, devient obsolète dans un environnement distribué, où les ressources sont décentralisées, les utilisateurs en mobilité, et les terminaux hétérogènes. Il suffit qu'un attaquant franchisse le périmètre pour obtenir un accès étendu aux ressources internes.

Pour répondre à ces nouveaux défis, les principaux organismes spécialisés en cybersécurité – notamment le **NIST (National Institute of Standards and Technology)** – ont introduit le concept de **Zero Trust Network Access (ZTNA)**, qui repose sur un principe fondamental : Ne jamais faire confiance par défaut, toujours vérifier.

Le Zero Trust n'accorde aucune confiance implicite, même à l'intérieur du réseau. L'accès à une ressource n'est autorisé qu'après vérification de l'identité, du contexte, des permissions, et parfois de l'état du terminal. Ce modèle a été progressivement adopté par les grandes plateformes technologiques, parmi lesquelles Google avec **BeyondCorp**, et se positionne aujourd'hui comme la stratégie de protection réseau la plus pertinente pour les infrastructures modernes.

C'est dans cette dynamique que s'inscrit le présent travail, dont l'objectif principal est la **mise en place d'une solution ZTNA basée sur des technologies open-source**. La solution réalisée s'appuie sur quatre serveurs coopérant ensemble de manière sécurisée et cohérente :

- Un **serveur Keycloak**, assurant l'authentification et la gestion centralisée des identités

- Un **serveur Pomerium**, jouant le rôle d'Identity-Aware Reverse Proxy et appliquant les politiques d'accès Zero Trust ;
- Un **serveur de supervision** reposant sur **Prometheus** et **Grafana**, permettant la collecte, l'analyse et la visualisation des métriques ;
- Un **serveur d'applications** hébergeant des services internes, accessibles uniquement via le proxy Zero Trust.

Chapitre 1 : Fondements théorique de la solution ZTNA

Introduction

La sécurité des systèmes d'information constitue aujourd'hui un enjeu majeur pour les organisations, dans un contexte marqué par la transformation numérique, la mobilité accrue des utilisateurs et la multiplication des cybermenaces. Les infrastructures informatiques modernes ne sont plus limitées à un périmètre réseau clairement défini : elles intègrent des services cloud, des applications distribuées et des accès distants permanents. Cette évolution a profondément remis en question les modèles de sécurité traditionnels basés sur la confiance implicite accordée au réseau interne (NIST SP 800-207, 2020).

Pendant de nombreuses années, la protection des systèmes d'information reposait essentiellement sur des mécanismes périmétriques tels que les pares-feux, les réseaux privés virtuels (VPN) et les systèmes de détection d'intrusion. Ce modèle, souvent désigné sous l'appellation *Castle and Moat*, supposait qu'une fois l'utilisateur authentifié et connecté au réseau interne, celui-ci pouvait être considéré comme fiable. Toutefois, plusieurs études ont démontré que ce postulat n'est plus valide face aux attaques modernes, notamment celles exploitant la compromission d'identifiants et les vulnérabilités des accès distants (Kim & Solomon, *Cybersecurity and Cyberwar*, 2016).

Face à ces limites, de nouvelles approches de sécurité ont émergé, mettant l'accent sur l'identité, le contrôle d'accès granulaire et la vérification continue. Parmi elles, le modèle **Zero Trust** s'est imposé comme une référence. Introduit conceptuellement par Forrester Research, puis concrètement mis en œuvre par Google à travers l'architecture BeyondCorp, le Zero Trust a été formalisé par le National Institute of Standards and Technology dans la norme **NIST SP 800-207** (Forrester, 2010 ; Google BeyondCorp, 2014 ; NIST SP 800-207, 2020).

Ce modèle repose sur un principe fondamental : aucune entité, qu'elle soit interne ou externe au réseau, ne doit être considérée comme digne de confiance par défaut.

Dans cette logique, l'accès aux ressources ne dépend plus de la localisation réseau, mais de l'identité de l'utilisateur, de son contexte et des politiques de sécurité définies. Cette approche a donné naissance au concept de **Zero Trust Network Access (ZTNA)**, qui vise à sécuriser l'accès aux applications en appliquant le principe du moindre privilège et en empêchant les mouvements latéraux au sein du système d'information (Gartner, *ZTNA Market Guide*, 2021).

Ainsi, ce premier chapitre a pour objectif de poser les bases théoriques nécessaires à la compréhension du Zero Trust Network Access. Il présente l'évolution de la sécurité des systèmes d'information, les limites des modèles traditionnels, ainsi que les principes fondamentaux du Zero Trust et du ZTNA. Ces éléments constituent le socle conceptuel indispensable à la conception et à la mise en œuvre de la solution ZTNA développée dans les chapitres suivants.

1.1 Evolution de la sécurité des systèmes d'informations

La sécurité des systèmes d'information a évolué de manière progressive, en fonction des transformations technologiques et des nouveaux usages informatiques. À l'origine, les environnements informatiques étaient centralisés, peu connectés et utilisés par un nombre restreint d'acteurs, ce qui limitait naturellement les risques de compromission. Les mécanismes de sécurité se concentraient principalement sur le contrôle d'accès physique et la protection des équipements matériels (Stallings, *Network Security Essentials*, 2017).

Avec l'apparition des réseaux locaux puis l'essor d'Internet, les systèmes d'information se sont ouverts vers l'extérieur. Cette ouverture a rendu nécessaire la mise en place de mécanismes de sécurité réseau tels que les pares-feux, les systèmes de détection d'intrusion (IDS/IPS) et les réseaux privés virtuels (VPN). Ces solutions visaient à établir une frontière entre un réseau interne considéré comme fiable et un environnement externe jugé non sécurisé, conformément au modèle périmétrique traditionnel (Kim & Solomon, *Cybersecurity and Cyberwar*, 2016).

Cependant, cette approche a progressivement montré ses limites. Dans le contexte actuel, les utilisateurs accèdent aux ressources depuis différents lieux géographiques, à l'aide de terminaux hétérogènes et via des réseaux qui ne sont pas toujours maîtrisés par l'organisation. Le recours massif au télétravail et aux services cloud a contribué à la disparition progressive du périmètre réseau classique, rendant les mécanismes de sécurité traditionnels moins efficaces (NIST SP 800-207, 2020).

Dans le cadre de ce travail, cette évolution apparaît comme un facteur déterminant dans la redéfinition des stratégies de sécurité. Le fait de considérer le réseau interne comme un espace de confiance n'est plus réaliste, car les menaces peuvent provenir aussi bien de l'extérieur que de l'intérieur du système d'information. Un utilisateur légitime peut, par exemple, devenir un vecteur d'attaque à la suite d'une compromission de ses identifiants, rendant inefficaces les contrôles basés uniquement sur la localisation réseau.

Par ailleurs, les cyberattaques modernes exploitent davantage les faiblesses humaines et organisationnelles que les vulnérabilités techniques classiques. Le phishing, l'ingénierie sociale et le vol d'identifiants constituent aujourd'hui des vecteurs d'attaque majeurs. Une fois ces identifiants compromis, les mécanismes de sécurité périmétrique accordent un niveau de confiance excessif à l'utilisateur authentifié, facilitant ainsi les mouvements latéraux au sein du système d'information (ENISA, Threat Landscape, 2022).

Cette évolution justifie la nécessité d'adopter une approche de sécurité centrée sur l'identité et le contrôle d'accès granulaire. Dans le contexte de ce projet, la mise en place d'une solution ZTNA permet de répondre à ces nouveaux enjeux en supprimant la confiance implicite accordée au réseau et en appliquant des politiques d'accès strictes basées sur l'identité, le contexte et les permissions des utilisateurs.

1.2 Limite des architectures de sécurité traditionnelle

Les architectures de sécurité traditionnelles reposent majoritairement sur une approche périmétrique, selon laquelle le réseau interne est considéré comme un environnement de confiance, tandis que le réseau externe est perçu comme potentiellement hostile. Cette approche, souvent désignée sous le terme de *Castle and Moat*, s'appuie sur des mécanismes tels que les pares-feux, les systèmes de détection et de prévention d'intrusion (IDS/IPS), ainsi que les réseaux privés virtuels (VPN), afin de contrôler les accès aux ressources internes. (à réviser)

Dans ce modèle, le pare-feu joue un rôle central en filtrant le trafic entrant et sortant, tandis que le VPN permet aux utilisateurs distants d'accéder au réseau interne de manière sécurisée. Une fois l'utilisateur authentifié et connecté via le VPN, il bénéficie généralement d'un accès étendu aux ressources du réseau, ce qui introduit une forme de confiance implicite. Or, cette confiance repose uniquement sur la réussite de l'authentification initiale et sur la localisation réseau, sans vérification continue du contexte ou des actions effectuées (Kim & Solomon, *Cybersecurity and Cyberwar*, 2016).

L'une des principales faiblesses de ce modèle réside dans sa vulnérabilité face à la compromission des identifiants. Les attaques par phishing, aujourd'hui très répandues, permettent aux attaquants d'obtenir des identifiants valides et d'accéder au réseau interne en se faisant passer pour des utilisateurs légitimes. Une fois à l'intérieur, l'attaquant peut se déplacer latéralement, explorer le réseau, identifier des services vulnérables et compromettre des ressources critiques, sans être immédiatement détecté (ENISA, *Threat Landscape Report*, 2022).

Les VPN traditionnels constituent également un point de faiblesse majeur. Plusieurs incidents de sécurité récents ont mis en évidence l'exploitation de vulnérabilités critiques dans des solutions VPN largement utilisées, permettant des accès non autorisés aux réseaux internes. Ces failles démontrent que le VPN, bien qu'il chiffre les communications, ne garantit pas un contrôle d'accès suffisamment granulaire ni une isolation efficace des ressources.

Dans la pratique, cette architecture pose un problème fondamental : elle offre un accès au réseau plutôt qu'un accès aux applications. Ainsi, un utilisateur ayant uniquement besoin d'accéder à une application spécifique se retrouve souvent connecté à l'ensemble du réseau interne, ce qui augmente considérablement la surface d'attaque. Cette approche est en contradiction avec le principe du moindre privilège, qui recommande de limiter strictement les droits d'accès aux seules ressources nécessaires.

Par ailleurs, la gestion opérationnelle des architectures périmétriques devient de plus en plus complexe. La multiplication des règles de pare-feu, la gestion des accès VPN, ainsi que la difficulté à assurer une visibilité complète des connexions rendent ces systèmes difficiles à maintenir et à superviser efficacement. Cette complexité opérationnelle peut elle-même devenir une source de vulnérabilités (Microsoft Security Insights, 2018).

Face à ces limites, les experts en cybersécurité recommandent une refonte des stratégies de protection des systèmes d'information. Le NIST souligne notamment que la confiance implicite accordée au réseau interne constitue un risque majeur et préconise une approche dans laquelle chaque accès est systématiquement vérifié, indépendamment de l'emplacement de l'utilisateur (NIST SP 800-207, 2020). Ces constats ont favorisé l'émergence du modèle Zero Trust, qui sera présenté dans la section suivante.

1.3 Origine et principes du ZTNA

Le modèle Zero Trust est né de la remise en question des architectures de sécurité traditionnelles fondées sur la confiance implicite accordée au réseau interne. Face à la multiplication des attaques exploitant les failles des accès distants et la compromission d'identifiants, plusieurs experts en cybersécurité ont proposé une approche radicalement différente, centrée non plus sur le périmètre réseau mais sur l'identité et la vérification systématique des accès.

Le concept de Zero Trust a été formalisé pour la première fois par Forrester Research en 2010. Dans son modèle, Forrester introduit l'idée selon laquelle aucune entité utilisateur, terminal ou application ne doit être considérée comme digne de confiance par défaut, même si elle se situe

à l'intérieur du réseau de l'organisation (Forrester, *The Zero Trust Model*, 2010). Cette approche marque une rupture avec les modèles de sécurité traditionnels et pose les bases d'un contrôle d'accès plus strict et plus granulaire.

Le modèle Zero Trust a ensuite été mis en œuvre à grande échelle par Google à travers le projet BeyondCorp. Cette initiative visait à permettre aux employés d'accéder aux applications internes de Google depuis n'importe quel réseau, sans recourir à un VPN, tout en maintenant un haut niveau de sécurité. L'accès aux ressources y est conditionné par l'identité de l'utilisateur, l'état du terminal et le respect de politiques de sécurité centralisées (Google, *BeyondCorp: A New Approach to Enterprise Security*, 2014). L'expérience de Google a largement contribué à la diffusion et à l'adoption du Zero Trust dans les environnements professionnels.

Afin de standardiser cette approche, le National Institute of Standards and Technology (NIST) a publié en 2020 le document NIST Special Publication 800-207, qui définit officiellement l'architecture Zero Trust. Selon le NIST, le Zero Trust repose sur le principe selon lequel le réseau ne doit jamais être considéré comme fiable, et chaque demande d'accès doit faire l'objet d'une vérification explicite, fondée sur des politiques de sécurité dynamiques (NIST SP 800-207, 2020).

Le modèle Zero Trust s'articule autour de plusieurs principes fondamentaux. Le premier est celui de la vérification systématique (*Never Trust, Always Verify*), qui impose une authentification et une autorisation pour chaque accès, indépendamment de la localisation de l'utilisateur. Le deuxième principe est celui du moindre privilège, qui consiste à accorder uniquement les droits strictement nécessaires à l'exécution d'une tâche donnée. Enfin, le Zero Trust préconise une surveillance continue des accès et des comportements, afin de détecter rapidement toute activité suspecte (NIST SP 800-207, 2020).

Dans le cadre de ce travail, le modèle Zero Trust apparaît comme une réponse particulièrement adaptée aux environnements modernes, caractérisés par la dispersion des ressources et la mobilité des utilisateurs. En supprimant la notion de confiance implicite liée au réseau, cette approche permet de réduire significativement la surface d'attaque et de limiter les impacts d'une compromission d'identifiants. Elle favorise également une meilleure segmentation des accès, en autorisant les utilisateurs à accéder uniquement aux applications qui leur sont destinées.

L'application concrète des principes du Zero Trust dans les infrastructures modernes a conduit à l'émergence de solutions spécifiques, parmi lesquelles le **Zero Trust Network Access (ZTNA)**. Ce dernier constitue une déclinaison opérationnelle du Zero Trust, axée sur la

sécurisation des accès applicatifs. La section suivante sera consacrée à la présentation du ZTNA, à ses caractéristiques et à ses différences avec les solutions d'accès traditionnelles.

1.4 Zero Trust Network Access (ZTNA) : définition et concepts clés

Le Zero Trust Network Access (ZTNA) constitue une déclinaison opérationnelle du modèle Zero Trust, spécifiquement orientée vers la sécurisation des accès aux applications et aux services internes. Contrairement aux architectures traditionnelles qui accordent un accès global au réseau après authentification, le ZTNA vise à fournir un accès strictement contrôlé à des ressources applicatives précises, en fonction de l'identité et du contexte de l'utilisateur (NIST SP 800-207, 2020).

Selon le NIST, le ZTNA repose sur l'idée que l'accès aux ressources ne doit jamais dépendre de la localisation réseau, mais uniquement d'une décision d'autorisation fondée sur des politiques dynamiques. Chaque tentative d'accès est évaluée individuellement, en tenant compte de plusieurs critères tels que l'identité de l'utilisateur, ses rôles, les attributs de sécurité, le contexte de connexion et les règles définies par l'organisation (NIST SP 800-207, 2020). Cette approche permet d'éliminer la confiance implicite traditionnellement associée au réseau interne.

Le ZTNA se distingue fondamentalement des VPN traditionnels par sa logique d'accès. Alors qu'un VPN établit un tunnel chiffré donnant accès à une partie, voire à l'ensemble du réseau interne, le ZTNA n'expose jamais directement le réseau à l'utilisateur. Les applications sont publiées de manière contrôlée, souvent à travers un proxy ou un point d'accès intermédiaire, qui vérifie systématiquement les droits de l'utilisateur avant de permettre l'accès (Gartner, *ZTNA Market Guide*, 2021).

Dans le cadre de ce projet, l'approche ZTNA apparaît particulièrement pertinente car elle permet de sécuriser les accès aux applications internes sans exposer le réseau sous-jacent. En offrant un accès applicatif ciblé, le ZTNA réduit considérablement les risques liés aux mouvements latéraux et limite l'impact potentiel d'une compromission d'identifiants. Cette granularité est essentielle dans les environnements modernes où les utilisateurs n'ont pas besoin d'un accès réseau global pour accomplir leurs tâches.

Un autre aspect fondamental du ZTNA réside dans son intégration étroite avec les systèmes de gestion des identités. L'identité devient ainsi le pivot central de toute décision d'accès. Les

solutions ZTNA s'appuient généralement sur des protocoles standards tels que OAuth 2.0 et OpenID Connect pour assurer une authentification robuste et interopérable (RFC 6749 ; OpenID Connect Core, 2014). Cette intégration facilite la mise en œuvre de politiques de sécurité cohérentes et centralisées.

Enfin, le ZTNA favorise une meilleure visibilité et une supervision renforcée des accès. Chaque requête étant contrôlée individuellement, il devient possible de tracer précisément les connexions, d'analyser les comportements et de détecter rapidement les anomalies. Cette capacité de journalisation et d'audit constitue un avantage majeur par rapport aux architectures traditionnelles, souvent limitées en termes de visibilité (NIST SP 800-207, 2020).

Après avoir présenté les concepts fondamentaux du Zero Trust Network Access, il est nécessaire d'analyser les bénéfices et les limites de cette approche. La section suivante sera donc consacrée à l'étude des avantages, des contraintes et des enjeux liés à l'adoption du ZTNA dans les organisations.

1.5 Avantages, limites et enjeux du ZTNA

1.5.1 Avantages du Zéro Trust Network Access

Le Zéro Trust Network Access présente plusieurs avantages significatifs par rapport aux architectures de sécurité traditionnelles. Le premier avantage réside dans la **suppression de la confiance implicite accordée au réseau interne**. Dans une architecture ZTNA, chaque accès est soumis à une vérification explicite, indépendamment de la localisation de l'utilisateur, ce qui permet de réduire considérablement la surface d'attaque (NIST SP 800-207, 2020).

Un autre avantage majeur du ZTNA est la **segmentation fine des accès applicatifs**. Contrairement aux VPN classiques qui offrent un accès étendu au réseau, le ZTNA permet de limiter l'accès à une application ou à un service précis. Cette segmentation repose sur le principe

du moindre privilège et contribue à empêcher les mouvements latéraux en cas de compromission d'un compte utilisateur (Gartner, *ZTNA Market Guide*, 2021).

Le ZTNA favorise également une **meilleure gestion des identités et des accès**. En s'appuyant sur des systèmes d'authentification centralisés, il permet l'application cohérente de politiques de sécurité, l'activation de l'authentification multi-facteurs et une administration simplifiée des droits d'accès. Cette centralisation améliore la gouvernance de la sécurité et réduit les erreurs humaines liées à la gestion manuelle des accès (Forrester, *Zero Trust Model*, 2010).

Dans le cadre de ce projet, l'approche ZTNA s'est révélée particulièrement efficace pour sécuriser l'accès aux applications internes sans exposer le réseau sous-jacent. L'accès applicatif ciblé permet d'offrir aux utilisateurs uniquement les services nécessaires, tout en maintenant un niveau de sécurité élevé. Cette approche améliore également l'expérience utilisateur en éliminant les contraintes liées à l'utilisation d'un VPN traditionnel.

1.5.2 Limites du Zéro Trust Network Access

Malgré ses nombreux avantages, le ZTNA présente certaines limites qu'il convient de prendre en considération. L'une des principales contraintes concerne la **dépendance au système de gestion des identités**. Dans une architecture ZTNA, l'identité constitue le pilier central de toute décision d'accès. Par conséquent, une indisponibilité ou une mauvaise configuration du fournisseur d'identité peut entraîner une interruption des accès aux applications (NIST SP 800-207, 2020).

La mise en œuvre d'une solution ZTNA peut également représenter une **complexité initiale**. Elle nécessite une identification précise des applications à protéger, une définition rigoureuse des politiques d'accès et une compréhension approfondie des flux applicatifs. Cette phase de conception peut être chronophage, notamment dans les environnements complexes ou peu documentés (ENISA, *Zero Trust Adoption Challenges*, 2021).

Dans le cadre de ce travail, la phase de configuration initiale a nécessité une attention particulière afin d'assurer la cohérence entre les politiques d'accès et les besoins réels des utilisateurs. Toutefois, cette complexité initiale tend à diminuer une fois l'architecture stabilisée, et les bénéfices en termes de sécurité et de visibilité compensent largement les efforts de mise en place.

1.5.3 Enjeux liés à l'adoption du ZTNA

Les enjeux liés à l'adoption du ZTNA ne se limitent pas aux aspects techniques. Ils concernent également la **gouvernance de la sécurité**, la **gestion des identités** et la **culture organisationnelle**. Le passage à une architecture Zero Trust implique une remise en question des pratiques existantes et nécessite l'adhésion des équipes techniques et des utilisateurs (Gartner, *Security and Risk Management Trends*, 2022).

Un enjeu majeur du ZTNA réside dans la **conformité réglementaire** et la traçabilité des accès. La capacité à journaliser et à analyser précisément les connexions constitue un atout important pour répondre aux exigences réglementaires et faciliter les audits de sécurité. Toutefois, cela suppose une gestion rigoureuse des logs et des données collectées (NIST SP 800-92, 2006).

Dans le contexte de ce projet, le ZTNA représente un levier stratégique permettant d'améliorer durablement la sécurité des accès tout en préparant l'infrastructure à des évolutions futures. Il constitue une base solide pour l'intégration de nouvelles applications et pour l'adaptation aux nouveaux usages, tels que le télétravail et la mobilité, tout en maintenant un niveau de sécurité élevé.

1.6 Étude comparative entre le ZTNA et les VPN traditionnels

Les réseaux privés virtuels (VPN) ont longtemps constitué la solution de référence pour permettre aux utilisateurs distants d'accéder aux ressources internes d'une organisation. Basés sur l'établissement d'un tunnel chiffré entre le poste client et le réseau interne, les VPN visent principalement à sécuriser le transport des données sur des réseaux non fiables. Toutefois, malgré leur efficacité en matière de chiffrement, les VPN présentent des limites importantes en termes de contrôle d'accès et de segmentation, particulièrement dans les environnements modernes (NIST SP 800-207, 2020).

Le ZTNA adopte une approche fondamentalement différente. Plutôt que de fournir un accès réseau global, il propose un accès applicatif ciblé, basé sur des politiques de sécurité strictes. L'utilisateur n'est jamais directement connecté au réseau interne ; il accède uniquement aux applications pour lesquelles il est explicitement autorisé, ce qui réduit considérablement les risques liés aux mouvements latéraux et à l'exposition du réseau (Gartner, *ZTNA Market Guide*, 2021).

Tableau 1: Comparaisons VPN et ZTNA

Critère	VPN traditionnel	ZTNA
Principe d'accès	Accès au réseau	Accès à l'application
Confiance	Implicite après connexion	Aucune confiance implicite
Segmentation	Faible	Fine et granulaire
Mouvement latéral	Possible	Fortement limité
Dépendance réseau	Élevée	Faible
Visibilité des accès	Limitée	Élevée
Authentification	Souvent basique	MFA + IAM
Adaptation cloud	Limitée	Native
Expérience utilisateur	Variable	Optimisée

Dans un contexte opérationnel, le VPN offre une solution simple à déployer, mais il présente un risque majeur : une fois connecté, l'utilisateur bénéficie souvent d'un accès étendu au réseau interne, indépendamment de ses besoins réels. Cette approche va à l'encontre du principe du moindre privilège et augmente la surface d'attaque en cas de compromission des identifiants.

À l'inverse, le ZTNA permet de mieux aligner les droits d'accès avec les besoins fonctionnels des utilisateurs. En isolant les applications et en contrôlant chaque requête, il réduit considérablement l'impact potentiel d'une attaque. De plus, le ZTNA facilite la supervision des accès et améliore la traçabilité, deux éléments essentiels pour la détection des incidents et la conformité réglementaire.

Cependant, il convient de noter que le ZTNA ne remplace pas systématiquement tous les usages du VPN. Certaines situations spécifiques, telles que l'accès à des ressources non web ou à des équipements réseau, peuvent encore nécessiter l'utilisation de VPN. Le ZTNA doit donc être envisagé comme un complément ou une évolution stratégique du VPN, plutôt que comme une substitution systématique et immédiate (ENISA, *Zero Trust Adoption Challenges*, 2021).

Cette étude comparative met en évidence la pertinence du ZTNA face aux limites des VPN traditionnels. Elle justifie ainsi le choix de cette approche dans le cadre de ce projet et prépare

la transition vers la synthèse du chapitre, qui récapitulera les principaux concepts théoriques abordés.

Conclusion

Ce premier chapitre a permis d'établir le cadre conceptuel nécessaire à la compréhension des enjeux liés à la sécurisation des accès dans les systèmes d'information modernes. L'évolution rapide des technologies, combinée à la généralisation du cloud, à la mobilité des utilisateurs et à l'ouverture croissante des infrastructures, a profondément modifié les conditions dans lesquelles la sécurité doit être pensée et appliquée. Dans ce contexte, les approches traditionnelles basées sur un périmètre réseau fixe apparaissent de moins en moins adaptées aux réalités actuelles (NIST SP 800-207, 2020).

L'étude de l'évolution de la sécurité des systèmes d'information a mis en évidence que les architectures périmétriques classiques reposent sur des hypothèses devenues obsolètes, notamment celle de la fiabilité du réseau interne. Les solutions historiques, telles que les pare-feu et les VPN, ont certes permis de répondre aux besoins de protection à une certaine époque, mais elles présentent aujourd'hui des faiblesses importantes en matière de contrôle d'accès granulaire, de visibilité et de limitation des mouvements latéraux en cas de compromission (Kim & Solomon, 2016).

Face à ces limites, le modèle Zero Trust s'est progressivement imposé comme une alternative crédible et structurante. En plaçant l'identité et la vérification systématique des accès au cœur de la stratégie de sécurité, cette approche rompt avec la logique de confiance implicite et propose un contrôle continu fondé sur des politiques dynamiques. Les travaux de Forrester, l'expérience de Google avec BeyondCorp et la formalisation du modèle par le NIST ont contribué à faire du Zero Trust une référence en matière de sécurité des accès (Forrester, 2010 ; Google BeyondCorp, 2014 ; NIST SP 800-207, 2020).

La déclinaison opérationnelle de ces principes à travers le **Zero Trust Network Access (ZTNA)** permet de répondre de manière concrète aux exigences des environnements modernes. En offrant un accès ciblé aux applications plutôt qu'un accès global au réseau, le ZTNA réduit la surface d'attaque et améliore la maîtrise des droits d'accès. Toutefois, son efficacité repose sur une gestion rigoureuse des identités et sur une conception précise des politiques de sécurité, ce qui en fait une solution exigeante mais structurante (Gartner, 2021).

La comparaison entre le ZTNA et les VPN traditionnels a mis en lumière une différence fondamentale de philosophie. Alors que le VPN vise principalement à sécuriser un canal de

communication, le ZTNA s'inscrit dans une logique de contrôle d'accès contextuel et applicatif. Cette distinction justifie le choix du ZTNA comme approche privilégiée pour sécuriser les accès dans les environnements distribués, tout en reconnaissant que certaines situations spécifiques peuvent encore nécessiter l'usage de solutions VPN complémentaires (NIST SP 800-207, 2020 ; ENISA, 2021).

En définitive, ce chapitre a permis de démontrer que le Zero Trust Network Access constitue une évolution naturelle et nécessaire des modèles de sécurité traditionnels. Il fournit le socle théorique indispensable à la conception et à la mise en œuvre de la solution ZTNA présentée dans la suite de ce travail. Le chapitre suivant s'attachera à analyser le besoin réel et à définir l'architecture technique adoptée pour répondre à ces enjeux.

Chapitre 2 : Analyse du besoin et conception de la solution ZTNA

Introduction

Après avoir présenté les fondements théoriques du Zéro Trust Network Access dans le chapitre précédent, ce deuxième chapitre se concentre sur l'analyse du besoin et la conception de la solution proposée. L'objectif est d'identifier les exigences fonctionnelles et non fonctionnelles de l'environnement étudié, puis de définir une architecture ZTNA adaptée à ces contraintes. Ce chapitre présente successivement le contexte de l'environnement cible, l'analyse des besoins, le choix des technologies retenues et la conception globale de l'architecture. Il constitue une étape intermédiaire essentielle entre la théorie et l'implémentation technique détaillée dans le chapitre suivant.

2.1 Présentation de l'environnement étudié

L'environnement étudié dans le cadre de ce travail correspond à une infrastructure informatique moderne, conçue pour héberger des applications internes accessibles par des utilisateurs locaux et distants. Comme dans de nombreuses organisations actuelles, cet environnement est confronté à plusieurs contraintes majeures : la nécessité d'un accès distant sécurisé, la protection des applications sensibles et la maîtrise des accès utilisateurs dans un contexte de mobilité croissante.

Les systèmes d'information modernes ne disposent plus d'un périmètre réseau clairement délimité. Les utilisateurs peuvent se connecter depuis des réseaux externes non maîtrisés, tels que des réseaux domestiques ou publics, ce qui augmente considérablement les risques de compromission. Dans ce contexte, les organismes de normalisation recommandent de considérer le réseau comme non fiable par défaut et de renforcer les mécanismes de contrôle d'accès applicatif (NIST SP 800-207, 2020).

L'environnement cible s'inscrit dans cette logique et vise à remplacer une approche basée sur l'accès réseau par une approche centrée sur l'accès aux applications, conformément aux principes du Zero Trust Network Access.

2.2 Description générale de l'architecture

Avant la mise en place de la solution Zero Trust Network Access, l'architecture existante reposait sur une organisation classique des services, dans laquelle les différents composants du système d'information étaient accessibles selon une logique principalement réseau. Les applications internes étaient hébergées sur un serveur dédié et l'accès des utilisateurs reposait essentiellement sur des mécanismes de confiance implicite liés à la localisation réseau ou à des méthodes d'accès distantes traditionnelles.

Cette architecture, bien que fonctionnelle, présentait plusieurs limites en matière de sécurité et de gestion des accès. En effet, les utilisateurs disposaient d'un accès relativement large aux ressources une fois authentifiés, sans distinction fine entre les différentes applications. De plus, l'absence d'un point de contrôle centralisé rendait difficile l'application de politiques d'accès homogènes et la supervision précise des connexions.

Afin de mieux structurer l'environnement et de répondre aux exigences de sécurité modernes, l'architecture existante a été organisée autour de **serveurs distincts**, chacun remplissant un rôle précis. Cette séparation fonctionnelle permet de limiter les responsabilités de chaque composant, de réduire la surface d'attaque et de faciliter la maintenance de l'infrastructure.

L'architecture repose ainsi sur quatre éléments principaux :

- **Un serveur d'authentification**, chargé de la gestion des identités, de l'authentification des utilisateurs et de la fourniture des attributs nécessaires au contrôle d'accès.
- **Un serveur proxy de sécurité**, positionné comme point d'entrée unique vers les applications, chargé d'intercepter les requêtes et d'appliquer les règles d'autorisation.
- **Un serveur de supervision**, dédié à la collecte des métriques, à la surveillance des services et à l'analyse des événements de sécurité.
- **Un serveur d'applications**, hébergeant les services internes à protéger, volontairement isolé et non exposé directement à Internet.

Cette organisation permet de dissocier clairement les fonctions d'authentification, d'autorisation, d'hébergement applicatif et de supervision. Elle constitue une base structurée sur laquelle peut être appliquée une approche Zero Trust, en introduisant des contrôles explicites à chaque étape de l'accès aux ressources (NIST SP 800-207, 2020).

Afin de mieux visualiser cette architecture et les relations entre les différents composants, le schéma ci-dessous présente une vue d'ensemble de l'environnement existant.

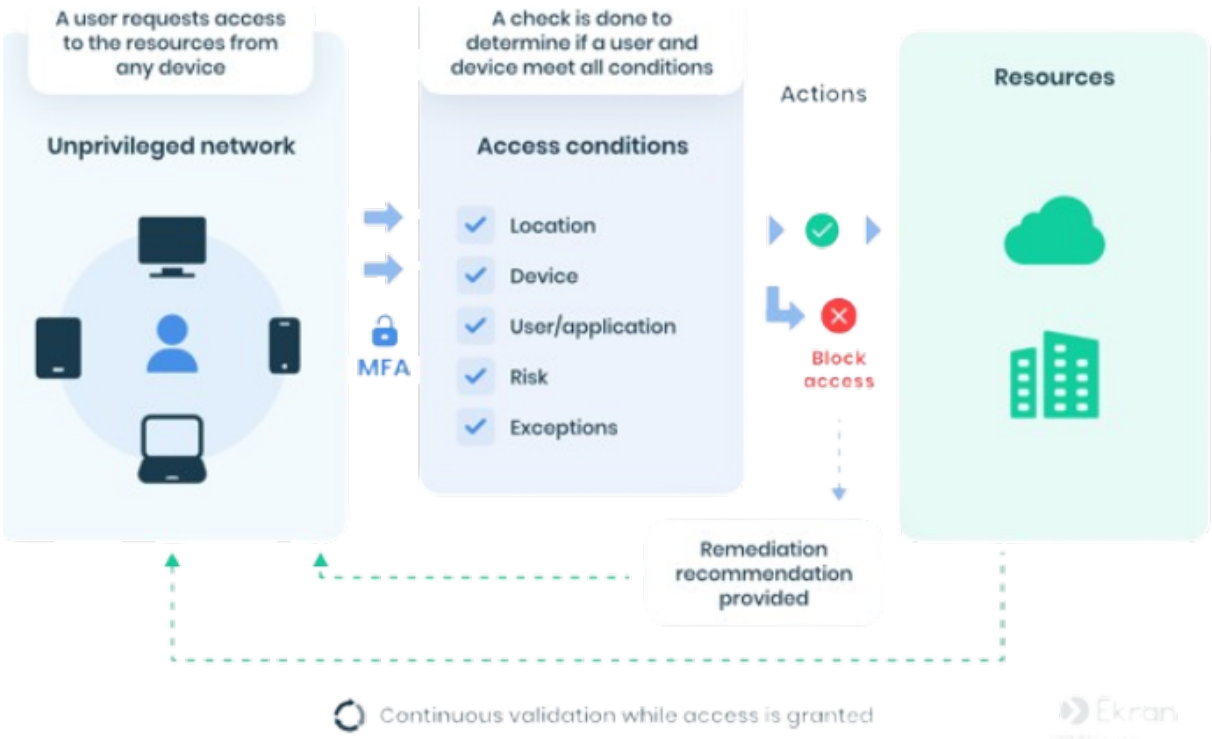


Figure 1: architecture ztna : https://www.syteca.com/en/blog/zero-trust-security-model

2.3 Problématique de sécurité dans l'environnement cible

Dans un environnement traditionnel, l'accès distant aux applications internes est souvent assuré par des VPN. Cependant, cette approche présente plusieurs limites importantes : une exposition excessive du réseau interne, une segmentation insuffisante des accès et une faible visibilité sur les actions réalisées par les utilisateurs une fois connectés.

Dans l'environnement étudié, ces limites se traduisent par :

- Une difficulté à restreindre l'accès à une application spécifique ;
- Un risque accru de mouvement latéral en cas de compromission d'un compte ;
- Une gestion complexe des droits d'accès ;
- Une supervision limitée des connexions et des événements de sécurité ;

Ces constats rejoignent les analyses réalisées par plusieurs organismes de cybersécurité, qui soulignent que les VPN traditionnels ne répondent plus pleinement aux exigences de sécurité des infrastructures modernes (Gartner, *ZTNA Market Guide*, 2021).

2.4 Justification de l'approche ZTNA dans cet environnement

Face à ces contraintes, l'adoption d'une approche **ZTNA** apparaît comme une réponse adaptée aux besoins de l'environnement cible. Le ZTNA permet de sécuriser les accès en appliquant une vérification systématique de l'identité et des permissions, tout en évitant l'exposition directe du réseau interne.

Dans cette architecture, les utilisateurs n'accèdent jamais directement au réseau ou aux serveurs d'applications. Toutes les requêtes passent par un point de contrôle centralisé, qui applique les politiques de sécurité avant d'autoriser ou de refuser l'accès. Cette approche permet de réduire la surface d'attaque, de limiter les mouvements latéraux et d'améliorer la traçabilité des accès (NIST SP 800-207, 2020).

2.4.1 Positionnement de l'environnement par rapport au Zero Trust

L'environnement étudié constitue un cas d'application concret des principes Zero Trust. Il repose sur :

Une authentification centralisée basée sur l'identité, un contrôle d'accès applicatif granulaire ; une séparation stricte des rôles des serveurs ; une supervision continue des serveurs.

Ce positionnement permet de traduire les concepts théoriques du Zero Trust en une architecture opérationnelle et mesurable. Il constitue également une base solide pour l'analyse des besoins fonctionnels et non fonctionnels, qui sera développée dans la section suivante.

2.4.2 Analyse des besoins fonctionnels

L'analyse des besoins fonctionnels constitue une étape clé dans la conception de la solution ZTNA. Elle permet d'identifier précisément les fonctionnalités que doit offrir le système afin de répondre aux exigences de sécurité et d'usage de l'environnement étudié. Dans le cadre de ce projet, les besoins fonctionnels sont directement liés à la gestion des identités, au contrôle d'accès aux applications internes, à la supervision des accès et à la sécurisation globale des flux.

2.4.2.1 Besoin d'authentification centralisée

L'un des besoins fonctionnels fondamentaux de l'environnement étudié est la mise en place d'un **mécanisme d'authentification centralisé**. Dans une architecture traditionnelle, les utilisateurs disposent souvent de multiples comptes répartis sur différentes applications, ce qui complique la gestion des identités et augmente les risques de compromission.

Une authentification centralisée permet :

- D'unifier la gestion des utilisateurs ;
- De réduire la duplication des comptes ;
- D'appliquer des politiques de sécurité homogènes ;
- De faciliter l'administration et la révocation des accès.

Les bonnes pratiques en matière de sécurité recommandent l'utilisation d'un fournisseur d'identité unique (Identity Provider – IdP) pour assurer l'authentification et la gestion des utilisateurs (NIST SP 800-53, 2020). Cette approche permet également d'intégrer des mécanismes d'authentification forte, tels que l'authentification multi-facteurs (MFA), considérée aujourd'hui comme indispensable pour la protection des accès sensibles (OWASP Authentication Cheat Sheet, 2023).

Dans le cadre de ce projet, l'authentification centralisée est un prérequis indispensable à l'implémentation du ZTNA. Elle constitue le socle sur lequel reposent les décisions d'autorisation et permet d'assurer une cohérence globale dans la gestion des accès utilisateurs.

2.4.2.2 Besoin de contrôle d'accès applicatif granulaire

Un autre besoin fonctionnel majeur concerne le contrôle d'accès granulaire aux applications internes. Contrairement aux architectures VPN traditionnelles, où l'accès est accordé au réseau dans son ensemble, la solution ZTNA doit permettre de restreindre l'accès à chaque application en fonction des droits de l'utilisateur.

Ce contrôle d'accès repose sur :

- ✓ L'identité de l'utilisateur ;
- ✓ Ses rôles et groupes ;
- ✓ Les politiques de sécurité définies ;

- ✓ Le contexte de connexion.

Les modèles de contrôle d'accès basés sur les rôles (RBAC) et sur les attributs (ABAC) sont largement recommandés pour répondre à ces exigences (NIST SP 800-162, 2014). Ils permettent de définir des règles d'accès flexibles et évolutives, adaptées aux besoins des organisations modernes.

Dans l'environnement étudié, ce besoin se traduit par la nécessité de limiter l'accès des utilisateurs aux seules applications correspondant à leurs fonctions. Cette approche réduit considérablement la surface d'attaque et renforce la sécurité globale du système d'information.

2.4.2.3 Besoin de sécurisation des accès distants

La sécurisation des accès distants constitue un enjeu central dans le contexte actuel, marqué par la généralisation du télétravail et de la mobilité. Les utilisateurs doivent pouvoir accéder aux applications internes depuis des réseaux externes sans compromettre la sécurité du système d'information.

Les organismes de normalisation recommandent de ne plus exposer directement les réseaux internes et de privilégier des mécanismes d'accès applicatif sécurisés, basés sur des points de contrôle intermédiaires (NIST SP 800-207, 2020). Le ZTNA répond précisément à cette exigence en supprimant l'accès réseau direct et en appliquant une vérification systématique des droits d'accès.

2.4.2.4 Besoin de supervision et de traçabilité

La supervision et la traçabilité des accès constituent un besoin fonctionnel essentiel pour garantir la sécurité et la conformité de l'environnement. Il est indispensable de disposer d'une visibilité complète sur :

- Les tentatives d'authentification ;
- Les accès autorisés ou refusés ;
- Les comportements anormaux ;
- L'état des composants du système.

La journalisation et la surveillance continue sont recommandées par le NIST afin de détecter rapidement les incidents de sécurité et de faciliter les audits (NIST SP 800-92, 2006).

Les outils de supervision permettent également d'anticiper les problèmes de performance et d'assurer la disponibilité des services.

Dans le cadre de ce projet, la supervision permet non seulement de renforcer la sécurité, mais aussi de mieux comprendre le fonctionnement de l'architecture ZTNA et d'identifier d'éventuels points d'amélioration.

2.4.2.5 Besoin d'intégration et d'évolutivité

Enfin, la solution ZTNA doit être **évolutive** et **facilement intégrable** avec d'autres composants du système d'information. Elle doit permettre :

- L'ajout de nouvelles applications ;
- L'intégration de nouveaux utilisateurs ;
- L'évolution des politiques de sécurité ;
- L'adaptation aux futurs besoins.

Les architectures modulaires et basées sur des standards ouverts sont recommandées pour garantir la pérennité des solutions de sécurité (ISO/IEC 27001, 2013). L'utilisation de protocoles standards tels que OAuth 2.0 et OpenID Connect favorise l'interopérabilité et la maintenabilité (RFC 6749 ; OpenID Connect Core).

Après avoir analysé les besoins fonctionnels de la solution ZTNA, il est désormais nécessaire d'étudier les besoins non fonctionnels, notamment en matière de sécurité, de performance, de disponibilité et de maintenabilité. Cette analyse fera l'objet de la section suivante.

2.4.3 Analyse des besoins non fonctionnels

Les besoins non fonctionnels définissent les **contraintes de qualité** auxquelles la solution ZTNA doit répondre. Contrairement aux besoins fonctionnels, qui décrivent ce que le système doit faire, les besoins non fonctionnels précisent comment le système doit se comporter. Dans le cadre d'une architecture de sécurité, ces besoins sont essentiels, car ils conditionnent la fiabilité, la robustesse et l'efficacité globale de la solution mise en œuvre.

2.4.3.1 Exigences de sécurité

La sécurité constitue le besoin non fonctionnel prioritaire dans toute architecture ZTNA. La solution doit garantir la confidentialité, l'intégrité et la disponibilité des données, tout en

empêchant les accès non autorisés. Les organismes de normalisation recommandent l'application de mécanismes de sécurité robustes à tous les niveaux du système, incluant l'authentification, l'autorisation, le chiffrement des communications et la journalisation des événements (NIST SP 800-53, 2020).

Dans une architecture Zéro Trust, la sécurité ne repose pas sur un périmètre réseau, mais sur des contrôles distribués et continus. Chaque composant doit être sécurisé individuellement, et les communications entre services doivent être chiffrées afin d'éviter toute interception ou altération des données (NIST SP 800-207, 2020).

Dans le cadre de ce projet, la sécurité est assurée par la combinaison d'une authentification forte, d'un contrôle d'accès granulaire et d'un chiffrement systématique des flux. Cette approche permet de réduire significativement les risques liés aux accès distants et à la compromission des identifiants.

2.4.3.2 Exigences de performance

La performance représente un enjeu important pour l'adoption d'une solution ZTNA. Les mécanismes de sécurité ne doivent pas dégrader de manière significative l'expérience utilisateur. Les délais d'authentification, de redirection et d'accès aux applications doivent rester acceptables afin de garantir l'efficacité opérationnelle des utilisateurs.

Les bonnes pratiques recommandent de concevoir des architectures capables de traiter un grand nombre de requêtes sans introduire de latence excessive, notamment en optimisant les processus d'authentification et en utilisant des mécanismes de mise en cache des tokens lorsque cela est possible (Gartner, *ZTNA Market Guide*, 2021).

Dans l'environnement étudié, la performance est un critère déterminant, car les utilisateurs doivent accéder rapidement aux applications internes. L'utilisation de composants légers et de protocoles standards permet de limiter l'impact des contrôles de sécurité sur les temps de réponse.

2.4.3.3 Exigences de disponibilité et de fiabilité

La disponibilité constitue un besoin non fonctionnel critique pour une solution ZTNA. Les services d'authentification et de contrôle d'accès doivent être accessibles en permanence, car toute indisponibilité peut bloquer l'accès aux applications pour l'ensemble des utilisateurs.

Les standards de sécurité recommandent la mise en place de mécanismes de tolérance aux pannes, de redondance et de supervision afin d'assurer la continuité de service (ISO/IEC 27001, 2013). La surveillance proactive permet également de détecter rapidement les anomalies et d'intervenir avant qu'elles n'impactent les utilisateurs.

Dans ce projet, la supervision joue un rôle clé dans la garantie de la disponibilité des services. La collecte de métriques et l'analyse des performances permettent d'anticiper les incidents et d'assurer une exploitation stable de l'architecture ZTNA.

2.4.3.4 Exigences de maintenabilité et d'évolutivité

La maintenabilité et l'évolutivité sont des critères essentiels pour garantir la pérennité de la solution ZTNA. L'architecture doit être suffisamment modulaire pour permettre l'ajout de nouvelles applications, l'intégration de nouveaux utilisateurs et l'évolution des politiques de sécurité sans remettre en cause l'ensemble du système.

Les normes ISO recommandent l'adoption d'architectures basées sur des composants indépendants et sur des standards ouverts, afin de faciliter la maintenance et l'évolution des systèmes d'information (ISO/IEC 25010, 2011).

Dans le cadre de ce travail, le choix d'outils open-source et de protocoles standards permet de garantir une grande flexibilité et une facilité d'adaptation aux besoins futurs. Cette approche contribue également à réduire les coûts et à éviter les dépendances vis-à-vis de solutions propriétaires.

2.4.3.5 Exigences de conformité et de traçabilité

Enfin, la solution ZTNA doit répondre aux exigences de conformité réglementaire et de traçabilité des accès. La journalisation des événements de sécurité est indispensable pour assurer le suivi des actions, faciliter les audits et répondre aux obligations légales en matière de protection des données.

Les recommandations du NIST soulignent l'importance de la collecte et de l'analyse des logs pour la détection des incidents et l'amélioration continue de la sécurité (NIST SP 800-92, 2006).

Dans ce projet, la traçabilité constitue un élément fondamental pour comprendre le comportement des utilisateurs et renforcer la posture de sécurité globale de l'architecture.

Après avoir analysé les besoins fonctionnels et non fonctionnels, il est désormais nécessaire de justifier les **choix technologiques** effectués pour répondre à ces exigences. La section suivante présentera les solutions retenues et les raisons de leur sélection.

2.5 Choix technologiques

Le choix des technologies constitue une étape déterminante dans la conception d'une solution **Zéro Trust Network Access (ZTNA)**. Les outils retenus doivent répondre aux besoins fonctionnels et non fonctionnels identifiés précédemment, tout en respectant les contraintes de sécurité, de performance, de maintenabilité et d'évolutivité. Dans le cadre de ce projet, une attention particulière a été portée au choix de solutions **open-source**, afin de garantir la transparence, la flexibilité et l'indépendance technologique.

2.5.1 Choix du serveur d'authentification : Keycloak

La gestion des identités est un pilier central de toute architecture ZTNA. Le choix d'un fournisseur d'identité robuste et conforme aux standards est donc essentiel. **Keycloak** a été retenu comme serveur d'authentification en raison de ses fonctionnalités avancées et de sa conformité aux bonnes pratiques de sécurité.

Keycloak est une solution **open-source de gestion des identités et des accès (Identity and Access Management – IAM)**, développée initialement par Red Hat.

Elle permet de centraliser l'authentification et l'autorisation des utilisateurs pour des applications web, des API et des services, sans que ces derniers aient à implémenter eux-mêmes les mécanismes de sécurité. Keycloak s'appuie sur des **standards ouverts**, notamment **OAuth 2.0**, **OpenID Connect (OIDC)** et **SAML**, ce qui facilite son intégration dans des architectures modernes et hétérogènes.

L'objectif principal de Keycloak est de **déléguer la gestion des identités** à un service unique et centralisé. Cette approche permet de réduire la complexité applicative, d'améliorer la sécurité globale et d'appliquer des politiques d'authentification cohérentes sur l'ensemble du système d'information. Dans les architectures Zero Trust, l'identité constitue le nouveau périmètre de sécurité ; Keycloak répond précisément à cette exigence en fournissant un socle IAM robuste et extensible. Dans le cadre de ce projet, Keycloak est utilisé comme **fournisseur d'identité central (Identity Provider – IdP)** pour l'ensemble des accès aux applications internes protégées par la solution ZTNA.

2.5.1.1 Architecture et fonctionnement de Keycloak

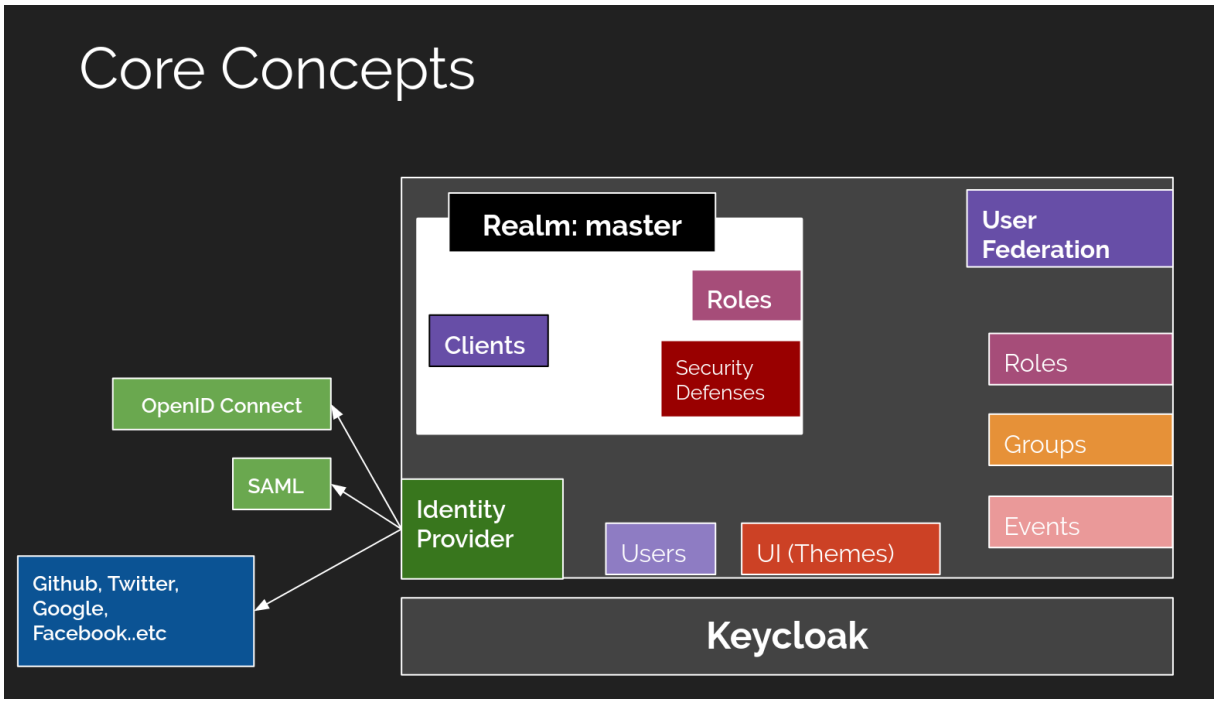


*Figure 2: Architecture keycloak : https://miro.medium.com/1*SZRfDnFdMlBYtej_Sp19fw.jpeg*

L'architecture de Keycloak repose sur plusieurs concepts clés qui structurent son fonctionnement :

Realm : espace logique isolé contenant les utilisateurs, les rôles, les clients et les politiques de sécurité.

Users : entités représentant les utilisateurs finaux.

Clients : applications ou services qui délèguent l'authentification à Keycloak.

Roles et Groups : mécanismes permettant de définir et d'organiser les droits d'accès.

Identity Providers : sources externes d'identité (LDAP, Active Directory, autres IdP).

Keycloak agit comme un **serveur central d'authentification**. Lorsqu'un utilisateur tente d'accéder à une application protégée, celle-ci redirige la requête vers Keycloak. Après authentification réussie, Keycloak émet un **jeton sécurisé** contenant les informations d'identité (claims), qui sera ensuite utilisé pour autoriser l'accès à la ressource demandée (OpenID Connect Core, 2014) Les données de configuration et les informations utilisateurs sont stockées

dans une base de données relationnelle, telle que **MariaDB ou PostgreSQL, ce qui garantit la persistance et la scalabilité du service.**

Cette architecture modulaire permet de séparer clairement la gestion des identités des applications métiers. Elle facilite également l'évolution de l'infrastructure, chaque application pouvant être ajoutée comme client sans modifier le cœur du système.

2.5.1.2 Sécurité et standards supportés

La sécurité constitue un élément central de Keycloak. La solution implémente plusieurs mécanismes conformes aux recommandations des organismes de normalisation :

- **Support des standards ouverts** : OAuth 2.0, OpenID Connect et SAML assurent l'interopérabilité et la sécurité des échanges (RFC 6749 ; OpenID Connect Core, 2014).
- **Chiffrement TLS** : toutes les communications entre les clients, Keycloak et les utilisateurs doivent être chiffrées afin de garantir la confidentialité et l'intégrité des données (NIST SP 800-52).
- **Authentification multi-facteurs (MFA)** : Keycloak permet l'ajout d'un second facteur d'authentification, renforçant considérablement la sécurité des comptes utilisateurs (OWASP Authentication Cheat Sheet).
- **Gestion fine des rôles et permissions** : les droits peuvent être définis avec précision et appliqués de manière cohérente.

Dans une architecture ZTNA, la compromission d'identifiants constitue l'une des principales menaces. L'intégration du MFA et l'utilisation de jetons à durée de vie limitée permettent de réduire fortement ce risque et d'aligner la solution avec les principes du Zero Trust.

2.5.1.3 Intégration de Keycloak dans une architecture ZTNA

Dans une architecture **Zero Trust Network Access**, Keycloak occupe une position stratégique : il devient **l'autorité de confiance pour l'identité**, sans pour autant introduire de confiance implicite. Toutes les décisions d'accès sont prises par le proxy Zero Trust (Pomerium), mais **reposent sur les informations d'identité fournies par Keycloak** (NIST SP 800-207, 2020).

Le rôle de Keycloak dans cette architecture est de :

- ✓ Authentifier les utilisateurs ;
- ✓ Fournir des attributs d'identité (rôles, groupes, claims) ;
- ✓ Permettre l'application de politiques d'accès dynamiques ;
- ✓ Centraliser la gestion des comptes et des droits.

Cette séparation entre authentification (Keycloak) et autorisation (proxy ZTNA) permet de respecter pleinement le principe Zero Trust : l'identité est vérifiée, mais l'accès n'est jamais accordé automatiquement. Chaque requête est évaluée selon une politique explicite.

2.5.1.4 Avantages, limites et justification du choix

Avantages

- Solution **open-source**, sans coût de licence.
- Support natif des **standards de sécurité modernes**.
- Large communauté et documentation complète.
- Forte intégration avec les architectures Zero Trust.

Limites

- Configuration initiale pouvant être complexe pour les environnements peu documentés.
- Dépendance forte à la disponibilité du service IAM.
- Nécessité d'une sécurisation rigoureuse (TLS, sauvegardes, haute disponibilité)

Keycloak a été retenu dans ce projet car il répond parfaitement aux exigences d'une architecture ZTNA : centralisation de l'authentification, conformité aux standards, extensibilité et intégration native avec un proxy orienté identité. Son caractère open-source permet également une maîtrise complète de l'infrastructure et une adaptation fine aux besoins spécifiques de l'environnement étudié.

Keycloak constitue un **pilier fondamental** de la solution ZTNA mise en œuvre. En fournissant une gestion centralisée, sécurisée et standardisée des identités, il permet de déplacer le périmètre de sécurité du réseau vers l'utilisateur, conformément aux recommandations du

modèle Zero Trust. Son intégration avec les autres composants de l'architecture garantit une cohérence globale et une sécurité renforcée.

2.5.2 Choix du serveur d'authentification : Pomerium

2.5.2.1 Présentation générale et objectifs

Pomerium est un **reverse proxy open-source orienté identité**, conçu pour implémenter un **contrôle d'accès Zero Trust** aux applications internes. Il agit comme un **Identity-Aware Proxy (IAP)** : chaque requête d'accès est évaluée selon des politiques explicites fondées sur l'identité, le contexte et les attributs fournis par un fournisseur d'identité externe. Contrairement aux solutions VPN traditionnelles, Pomerium ne fournit pas d'accès réseau global, mais un **accès applicatif ciblé**, conforme aux principes du Zero Trust (NIST SP 800-207, 2020).

L'objectif principal de Pomerium est de **sécuriser l'accès aux applications sans exposer le réseau interne**, en supprimant toute confiance implicite liée à la localisation réseau. Les utilisateurs accèdent aux applications via leur navigateur, sans client spécifique, après authentification auprès d'un fournisseur d'identité compatible **OpenID Connect.**

Dans le cadre de ce projet, Pomerium est utilisé comme **point de contrôle centralisé** pour l'accès aux applications internes, en s'appuyant sur Keycloak comme fournisseur d'identité.

2.5.2.2 Architecture et fonctionnement de Pomerium

Pomerium fait office d'intermédiaire entre les utilisateurs finaux et les services exigeant une authentification forte. Après vérification de l'identité auprès de notre fournisseur d'identité (IdP), Pomerium utilise une politique configurable pour déterminer comment acheminer la requête de l'utilisateur et s'il est autorisé à accéder au service.

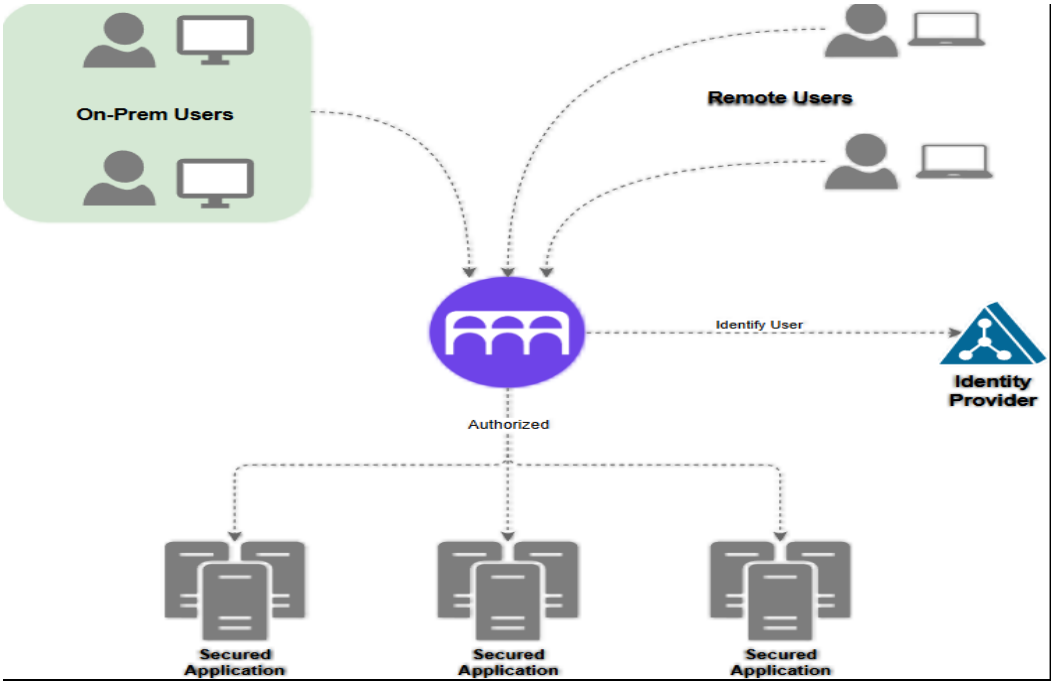


Figure 3: Architecture pomerium : https://www.pomerium.com/assets/images/pomerium-system-context-552143c2ee38dde267fa514a37fb9878.svg

L'architecture de Pomerium repose sur une séparation claire des responsabilités, alignée avec les bonnes pratiques Zero Trust. Elle se compose principalement de trois services logiques :

- **Proxy** : intercepte les requêtes entrantes vers les applications protégées.
- **Authenticate** : gère le flux d'authentification avec le fournisseur d'identité.
- **Authorize** : évalue les politiques d'accès et décide d'autoriser ou non la requête.

2.5.2.2.1 Proxy

Le service Proxy reçoit les requêtes du client et les achemine entre le client, les services en amont et les autres composants de Pomerium.

Les étapes suivantes décrivent de manière générale comment le service Proxy traite une requête :

1. Le client effectue une requête pour accéder à la ressource cible
2. Le service Proxy reçoit la requête et envoie un appel gRPC au service d'autorisation, qui évalue la politique.
3. Si le service d'autorisation ne détecte pas de cookie de session, le service proxy redirige la requête vers le service d'authentification afin de vérifier l'identité du client.

4. Une fois que le service d'authentification a vérifié l'identité du client et enregistré un cookie de session local, le service Proxy envoie les données de session au service Databroker via un appel gRPC.

5. Avant de rediriger le client vers la ressource cible, le service Proxy vérifie les autorisations auprès du service d'autorisation, puis établit un itinéraire en fonction des routes internes et externes définies dans la politique de routage.

2.5.2.2.1 Authentifications

Le service d'authentification est chargé d'authentifier les utilisateurs auprès d'un fournisseur d'identité (IdP) et d'établir des sessions utilisateur. En intégrant les protocoles OAuth 2.0 et OIDC au flux d'authentification, ce service propose une authentification unique (SSO) permettant d'extraire les informations d'identité et les données de session nécessaires à la gestion des sessions Pomerium.

De manière générale, lorsque le service d'authentification reçoit pour la première fois une requête du service proxy, il :

1. Redirige le client vers le fournisseur d'identité pour se connecter.

2. Finalise le flux d'authentification auprès du fournisseur d'identité et extrait les données de session pertinentes, telles que les identifiants de l'appareil, l'identifiant de l'utilisateur, les dates d'émission et d'expiration, les jetons OAuth et toutes les revendications OIDC (étendues) fournies par le fournisseur d'identité.

3. Enregistre les données de session dans un cookie de session local et redirige le client vers le service Proxy avec les données de session chiffrées dans les paramètres d'URL.

4. Permet à l'utilisateur de se reconnecter après l'expiration de la session.

2.5.2.2.2 Autorisations

Le service d'autorisation traite les politiques afin de déterminer les permissions du client. Chaque requête envoyée par le client doit d'abord faire l'objet d'un contrôle d'autorisation avant d'être relayée par le service proxy.

Lorsque le service d'autorisation reçoit une requête du service proxy, les actions suivantes se produisent :

1. Le service d'autorisation recherche d'abord un cookie de session contenant le jeton JWT du client. Si aucun cookie de session n'est présent, le service d'autorisation renvoie une réponse de redirection invitant le navigateur à s'authentifier auprès du service d'authentification afin d'établir l'identité de la session.
2. Une fois le cookie de session en place, le service Proxy effectue un appel gRPC au service d'autorisation afin que celui-ci puisse déterminer les autorisations en fonction des revendications JWT et de la politique.
3. Le service d'autorisation construit ensuite des en-têtes de sécurité basés sur les revendications JWT, que le service Proxy transmet aux applications en amont.
4. À chaque requête suivante, le service d'autorisation utilise la mise en cache à la demande pour interroger le service Databroker afin d'obtenir les mises à jour de l'état de la session.

2.5.2.2.3 Courtages de données

Le service Databroker gère l'état de Pomerium. Composant essentiel, il stocke les sessions , les revendications utilisateur, les jetons, les appareils, les données d'annuaire et d'autres sources de données externes. Lors de l'utilisation d'Enterprise Console, d'Ingress Controller ou de Zero, Databroker stocke également la configuration de Pomerium.

Le Databroker fournit une interface gRPC pour le stockage et la récupération des enregistrements, ainsi que les fonctionnalités supplémentaires suivantes :

- **Baux** : Garantit qu'une seule instance effectue certaines actions lorsque plusieurs instances de Pomerium sont en cours d'exécution (par exemple, un seul courtier de données actualise les sessions utilisateur).

- **Requêtes** : Prend en charge l'indexation efficace, y compris l'indexation CIDR pour les données GeoIP.
- **Limites de capacité** : Le nombre d'enregistrements par type peut être limité (par exemple, les enregistrements d'événements sont limités aux 50 derniers événements).
- **Diffusion en continu** : Les enregistrements peuvent être diffusés en continu aux clients via :
    - **Synchronisation** : Enregistrer les modifications en temps réel (y compris les suppressions).
    - **SyncLatest** : Diffusez la dernière version de chaque enregistrement
- **Gestion des versions du serveur** : les méthodes RPC incluent la version du serveur pour signaler quand les clients doivent réinitialiser leur état

Les clients utilisent des points de terminaison de flux pour conserver des copies locales de données mises en cache, évitant ainsi souvent d'avoir à effectuer des requêtes directes auprès du Databroker.

Les principales responsabilités du courtier en données comprennent :

- Une fois le client authentifié, le serveur proxy effectue un appel gRPC au Databroker pour conserver les données de session et les informations d'identité.
- Le service d'autorisation interroge le Databroker à la demande afin de maintenir la synchronisation entre les deux services.
- Il fait office de gestionnaire d'identité, actualisant les sessions utilisateur auprès du fournisseur d'identité jusqu'à expiration d'une session Pomerium.

2.5.2.2.4 Sécurité, politiques et standards supportés

La sécurité de Pomerium repose sur plusieurs mécanismes clés :

- **Contrôle d'accès basé sur des politiques déclaratives** : les règles sont définies de manière explicite (utilisateur, groupe, domaine, attributs), ce qui réduit les erreurs de configuration.

- **Support des standards ouverts** : Pomerium s'appuie sur **OAuth 2.0** et **OpenID Connect** pour l'authentification et la délégation d'identité (RFC 6749 ; OpenID Connect Core).
- **Chiffrement TLS systématique** : toutes les communications client-proxy et proxy-application sont chiffrées.
- **Absence d'exposition directe des applications** : les applications internes ne sont jamais accessibles sans passer par le proxy.

Ces mécanismes répondent aux recommandations du NIST, qui préconise l'utilisation de points de décision et de points d'application des politiques dans une architecture Zero Trust (NIST SP 800-207, 2020).

Le caractère déclaratif des politiques constitue un avantage majeur : il permet de documenter clairement les règles d'accès et d'éviter les configurations implicites souvent rencontrées dans les architectures VPN traditionnelles.

2.5.2.3 Intégration de Pomerium dans une architecture ZTNA

Dans l'architecture ZTNA mise en œuvre, Pomerium joue le rôle de **Policy Enforcement Point (PEP)**. Il s'appuie sur Keycloak pour l'authentification et récupère les attributs d'identité nécessaires à la prise de décision. Les applications internes sont placées derrière Pomerium et ne sont accessibles qu'après une autorisation explicite.

Le rôle de Pomerium dans cette architecture est de :

- Centraliser le contrôle d'accès applicatif ;
- Empêcher les mouvements latéraux en supprimant l'accès réseau global ;
- Assurer la traçabilité des accès ;
- Appliquer le principe du moindre privilège.

2.5.2.4 Avantages, limites et justification du choix

Avantages

- Mise en œuvre fidèle des principes **Zero Trust**.
- Accès **agentless**, simple pour les utilisateurs.
- Politiques d'accès **lisibles et auditables**.
- Intégration native avec les fournisseurs d'identité OIDC.

Limites

- Dépendance forte au fournisseur d'identité.
- Courbe d'apprentissage liée à la compréhension des politiques.
- Nécessite une gestion rigoureuse des certificats TLS.

Pomerium a été retenu car il permet d'appliquer un contrôle d'accès Zero Trust strict, sans exposer le réseau interne ni complexifier l'expérience utilisateur. Son intégration naturelle avec Keycloak et son approche déclarative en font un composant central et cohérent de la solution ZTNA proposée. De plus, son caractère open-source permet une maîtrise complète de l'architecture et une adaptation fine aux besoins du projet.

Pomerium constitue le **cœur du mécanisme ZTNA** de la solution mise en œuvre. En agissant comme proxy orienté identité, il permet de sécuriser l'accès aux applications internes selon des politiques explicites et vérifiables, tout en respectant pleinement les principes du Zero Trust. Son rôle est complémentaire à celui de Keycloak et essentiel pour garantir un contrôle d'accès applicatif moderne et robuste.

2.5.3 Choix des applications web et du serveur Nginx

Dans le cadre de la mise en œuvre de la solution Zero Trust Network Access, des applications web simples ont été développées à l'aide des technologies HTML, CSS et JavaScript, puis hébergées localement sur un serveur Nginx. Ces applications ont pour objectif de servir de cas d'usage fonctionnels afin de valider le bon fonctionnement de l'architecture ZTNA mise en place.

Le choix de technologies web standards telles que HTML et CSS repose sur leur simplicité, leur portabilité et leur indépendance vis-à-vis de plateformes spécifiques. Ces technologies

permettent de concevoir rapidement des interfaces accessibles via un navigateur web, sans nécessiter de dépendances complexes ou de configurations particulières côté client. Elles sont ainsi particulièrement adaptées à la démonstration et à l'évaluation des mécanismes d'accès sécurisé mis en œuvre dans ce projet (Mozilla Developer Network, 2023).

Le serveur Nginx a été retenu pour l'hébergement des applications en raison de sa légèreté, de sa performance et de sa fiabilité. Nginx est largement utilisé comme serveur web et reverse proxy dans les environnements professionnels, notamment pour la distribution de contenus statiques et la gestion des connexions HTTP et HTTPS. Sa capacité à fonctionner efficacement avec un faible impact sur les ressources système en fait un choix pertinent pour un environnement de test et de démonstration.

Dans l'architecture ZTNA proposée, le serveur Nginx n'est pas exposé directement aux utilisateurs. L'accès aux applications qu'il héberge est strictement contrôlé par le proxy Zero Trust, qui applique les politiques d'accès définies en amont. Cette configuration permet de démontrer que des applications web classiques peuvent être intégrées dans une architecture Zero Trust sans modification de leur code, ce qui constitue un avantage important en termes de compatibilité et de facilité de déploiement (NIST SP 800-207, 2020).

L'utilisation d'applications web simples hébergées localement permet de se concentrer sur les mécanismes de sécurité et de contrôle d'accès plutôt que sur la complexité applicative. Ce choix facilite la compréhension de l'architecture ZTNA et met en évidence l'efficacité du proxy Zero Trust dans la protection d'applications existantes.

2.5.4 Cloudflared

Cloudflared est un composant logiciel développé par Cloudflare, permettant d'établir un tunnel sécurisé sortant entre une application hébergée dans un réseau privé et l'infrastructure Cloudflare. Ce mécanisme, connu sous le nom de Cloudflare Tunnel, permet d'exposer une application à Internet sans ouvrir de ports entrants, sans adresse IP publique et sans modification de la topologie réseau interne (Cloudflare Tunnel Documentation).

L'objectif principal de Cloudflared est de réduire la surface d'attaque associée à l'exposition d'applications publiques. Contrairement aux méthodes traditionnelles telles que le port forwarding ou la publication directe via un reverse proxy exposé, Cloudflared fonctionne selon un principe de connexion initiée depuis l'intérieur du réseau. Cette approche s'inscrit pleinement dans les principes du Zero Trust, qui recommandent de limiter l'exposition des

ressources internes et de considérer le réseau comme non fiable par défaut (NIST SP 800-207, 2020).

Dans le cadre de ce projet, Cloudflared est utilisé pour **exposer une application en accès public**, tout en maintenant une isolation complète de l'infrastructure interne protégée par la solution ZTNA.

2.5.4.1 Architecture et fonctionnement de Cloudflared

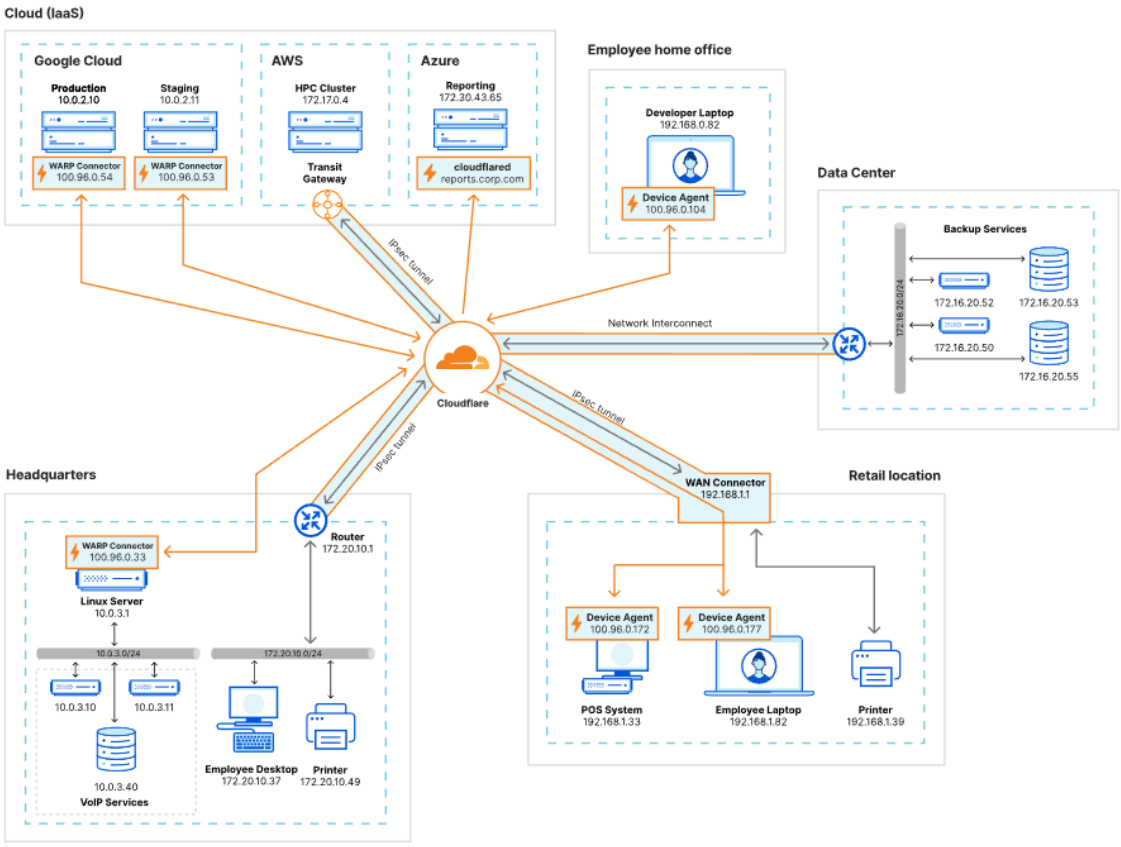


Figure 4: Architecture cloudflared :

Le fonctionnement de Cloudflared repose sur l'établissement d'un **tunnel chiffré sortant** entre le serveur hébergeant l'application et l'infrastructure Cloudflare. Le composant cloudflared, exécuté localement, agit comme un **agent** qui maintient une connexion persistante vers les points de présence Cloudflare les plus proches.

Les principales étapes de fonctionnement sont les suivantes :

1. Le service cloudflared est lancé sur le serveur interne.
2. Il établit une connexion sortante sécurisée vers Cloudflare.
3. Cloudflare associe ce tunnel à un nom de domaine public.
4. Les requêtes des utilisateurs sont reçues par Cloudflare.
5. Le trafic est acheminé via le tunnel jusqu'à l'application interne.

Ce mécanisme élimine la nécessité d'exposer directement le serveur applicatif à Internet. Le pare-feu du réseau interne n'autorise que des connexions sortantes, ce qui réduit considérablement les vecteurs d'attaque classiques tels que les scans de ports ou les attaques par exploitation de services exposés

Cette architecture est particulièrement pertinente dans un environnement pédagogique ou professionnel, car elle permet de publier rapidement une application sans modifier les règles réseau existantes, tout en maintenant un niveau de sécurité élevé.

2.5.4.2 Sécurité et mécanismes de protection

La sécurité de Cloudflared repose sur plusieurs mécanismes complémentaires :

- **Chiffrement TLS** : toutes les communications entre le service cloudflared et l'infrastructure Cloudflare sont chiffrées, garantissant la confidentialité et l'intégrité des données échangées.
- **Absence de ports entrants** : le serveur interne n'expose aucun service directement accessible depuis Internet.
- **Masquage de l'infrastructure interne** : l'adresse IP réelle du serveur applicatif n'est jamais révélée aux utilisateurs externes.

- **Protection intégrée Cloudflare** : le trafic bénéficie des mécanismes de sécurité Cloudflare, tels que la protection contre les attaques DDoS et le filtrage de trafic malveillant (Cloudflare Security Overview).

Ces mécanismes sont conformes aux recommandations des architectures Zero Trust, qui préconisent la réduction maximale de la surface d'attaque et la suppression de toute confiance implicite accordée au réseau (NIST SP 800-207, 2020).

2.5.4.3 Intégration de Cloudflared dans l'architecture ZTNA

L'intégration de Cloudflared dans l'architecture globale permet de gérer **deux types d'accès distincts** :

- **Accès interne sécurisé** : assuré par le proxy Zero Trust (Pomerium), avec authentification via Keycloak.
- **Accès public sécurisé** : assuré par Cloudflared, sans exposition directe du réseau interne.

Dans cette architecture hybride, Cloudflared ne remplace pas le ZTNA, mais le complète. Il est utilisé exclusivement pour les applications destinées à un accès public, tandis que les applications sensibles restent protégées par le proxy Zero Trust. Cette séparation permet de respecter les principes du Zero Trust tout en répondant à des besoins fonctionnels variés.

Cette approche démontre qu'il est possible de concilier exposition publique et sécurité Zero Trust, à condition de contrôler strictement les flux et de limiter l'exposition de l'infrastructure.

2.5.4.4 Avantages, limites et justification du choix

Avantages

- Aucune ouverture de port entrant.
- Aucune IP publique requise.
- Mise en œuvre rapide et simple.
- Intégration avec les mécanismes de sécurité Cloudflare

Limites

- Dépendance à l'infrastructure Cloudflare.

- Moins de contrôle direct sur certaines couches réseau.
- Nécessite une connexion sortante stable.

Cloudflared a été retenu pour ce projet afin d'exposer une application en accès public tout en respectant les principes de sécurité Zero Trust. Il permet de démontrer une approche moderne d'exposition de services, plus sécurisée que les méthodes traditionnelles, et parfaitement complémentaire au ZTNA mis en œuvre pour les accès internes.

Cloudflared constitue une solution efficace et moderne pour l'exposition sécurisée d'applications publiques. En s'appuyant sur un tunnel chiffré sortant et sur l'infrastructure Cloudflare, il permet de réduire considérablement les risques liés à l'exposition directe des serveurs internes. Son intégration dans l'architecture globale renforce la cohérence et la robustesse de la solution Zero Trust proposée.

2.5.5 Prometheus

2.5.5.1 Présentation générale et objectifs

Prometheus est un système **open-source de supervision et de collecte de métriques**, initialement développé par SoundCloud et aujourd'hui maintenu par la **Cloud Native Computing Foundation**. Il est conçu pour surveiller l'état, la performance et la disponibilité des systèmes et applications, en particulier dans des environnements modernes distribués et orientés services (Prometheus Documentation).

L'objectif principal de Prometheus est de fournir une **visibilité en temps réel** sur le fonctionnement des composants d'un système d'information. Contrairement aux outils de supervision traditionnels basés uniquement sur des checks ponctuels, Prometheus repose sur la collecte continue de **métriques temporelles**, permettant une analyse fine des comportements et des tendances (Prometheus Overview).

Dans le cadre de ce projet, Prometheus est utilisé pour superviser les différents composants de l'architecture ZTNA, notamment le serveur d'authentification, le proxy Zero Trust, le serveur d'applications et les services exposés.

2.5.5.2 Fonctionnement de Prometheus

L'architecture de Prometheus repose sur un modèle simple et robuste, articulé autour des composants suivants :

- **Prometheus Server** : cœur du système, chargé de collecter, stocker et interroger les métriques.
- **Exporters** : services ou agents exposant des métriques dans un format compatible Prometheus.
- **Base de données de séries temporelles (TSDB)** : stockage local des métriques sous forme de séries temporelles.
- **Alertmanager** : composant optionnel chargé de gérer les alertes.

Prometheus adopte un **modèle de collecte par interrogation (pull)**. Le serveur Prometheus interroge périodiquement les endpoints des services à surveiller afin de récupérer leurs métriques. Ce modèle facilite la découverte des services et renforce la sécurité, car les services surveillés n'ont pas besoin d'initier de connexions sortantes vers le serveur de supervision (Prometheus Architecture).

2.5.5.3 Sécurité, métriques et standards

La sécurité de Prometheus repose principalement sur la sécurisation des endpoints de métriques et sur le chiffrement des communications. Les bonnes pratiques recommandent :

- ✓ L'utilisation du **chiffrement TLS** pour protéger les échanges entre Prometheus et les services surveillés ;
- ✓ La restriction d'accès aux endpoints de métriques ;
- ✓ La séparation des rôles entre les composants de supervision et les applications surveillées.

Prometheus utilise un modèle de données basé sur des **paires clé/valeur (labels)**, ce qui permet de catégoriser et de filtrer finement les métriques. Le langage de requête **PromQL** offre des capacités avancées d'analyse et d'agrégation des données, facilitant l'identification d'anomalies et la détection de comportements anormaux (PromQL Documentation).

Dans le cadre de ce projet, la collecte de métriques permet de surveiller non seulement la performance, mais aussi des indicateurs liés à la sécurité, tels que le nombre de requêtes, les erreurs d'authentification ou les indisponibilités de services.

2.5.5.4 Intégration de Prometheus dans une architecture ZTNA

Dans une architecture ZTNA, la supervision joue un rôle essentiel pour assurer la **visibilité et la traçabilité** des accès. Prometheus s'intègre comme un composant transversal, chargé de collecter des métriques provenant de l'ensemble des services critiques :

- Serveur d'authentification (Keycloak) ;
- Proxy Zero Trust (Pomerium) ;
- Serveur d'applications (Nginx) ;
- Services exposés via Cloudflared.

Cette supervision permet de détecter rapidement :

- ✓ Des pics de charge anormaux ;
- ✓ Des erreurs répétées ;
- ✓ Des indisponibilités de services ;
- ✓ Des comportements pouvant indiquer une attaque ou une mauvaise configuration.

Les recommandations du NIST soulignent l'importance de la surveillance continue dans les architectures Zero Trust afin d'améliorer la posture de sécurité et de réagir rapidement aux incidents (NIST SP 800-207, 2020).

Avantages

- Solution **open-source** largement adoptée.
- Modèle de collecte simple et robuste.
- Puissantes capacités d'analyse via PromQL.
- Intégration native avec Grafana.

Limites

- Interface graphique native limitée.
- Gestion de l'authentification non intégrée par défaut.
- Nécessite une bonne compréhension des métriques pour une exploitation optimale

Prometheus constitue un pilier essentiel de la supervision de l'architecture ZTNA mise en œuvre. En offrant une collecte et une analyse efficaces des métriques, il permet d'assurer la disponibilité, la performance et la sécurité des services. Son intégration avec Grafana complète cette approche en apportant une visualisation claire et exploitable des données collectées.

2.5.6 Grafana

2.5.6.1 Présentation générale et objectifs

Grafana est une plateforme open-source de visualisation et d'observabilité permettant de transformer des données techniques (métriques, logs, traces) en tableaux de bord interactifs et exploitables. Initialement conçue pour la visualisation de séries temporelles, Grafana est aujourd'hui largement utilisée pour la supervision des systèmes, des applications et des infrastructures modernes. Elle est développée et maintenue par Grafana Labs (Grafana Documentation).

L'objectif principal de Grafana est de fournir une vision claire et centralisée de l'état d'un système d'information. En permettant la création de tableaux de bord personnalisés, Grafana facilite l'analyse des performances, la détection d'anomalies et le suivi des indicateurs clés. Dans le cadre des architectures Zero Trust, cette visibilité est essentielle pour assurer une surveillance continue des accès et des services (NIST SP 800-207, 2020).

2.5.6.2 Architecture et fonctionnement de Grafana

L'architecture de Grafana repose sur un modèle modulaire, centré autour d'un serveur Grafana qui interagit avec différentes sources de données externes. Contrairement à Prometheus, Grafana ne stocke pas les données : il interroge dynamiquement les sources configurées pour afficher les résultats sous forme graphique.

Les principaux composants de Grafana sont :

- **Grafana Server** : service central qui gère l'authentification, les tableaux de bord et les requêtes.
- **Data Sources** : connexions vers des systèmes externes (Prometheus, Elasticsearch, etc.).
- **Dashboards** : ensembles de panneaux graphiques configurables.

➢ **Alerting Engine** : moteur d'alertes basé sur des règles définies.

Grafana interroge périodiquement les sources de données afin de récupérer les informations nécessaires à l'affichage des graphiques. Cette approche permet une grande flexibilité et facilite l'intégration de multiples outils de supervision au sein d'une interface unique (Grafana Architecture Overview).

Cette séparation entre collecte (Prometheus) et visualisation (Grafana) permet d'obtenir une architecture de supervision claire, évolutive et conforme aux bonnes pratiques d'observabilité.

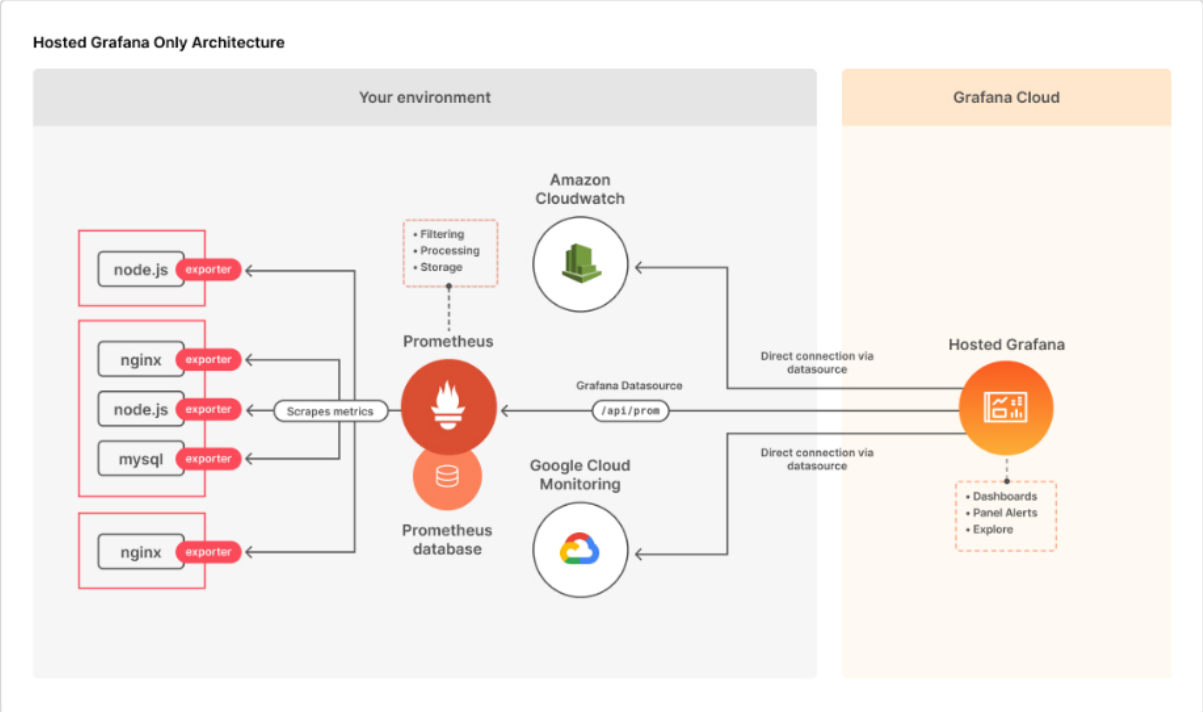


Figure 5:Architecture grafana : https://grafana.com/static/img/docs/grafana-cloud/arch_diagrams/hostedonly.jpg

2.5.6.3 Sécurité, gestion des accès et standards

La sécurité de Grafana repose sur plusieurs mécanismes destinés à protéger l'accès aux tableaux de bord et aux données visualisées :

- **Authentification des utilisateurs** : Grafana prend en charge plusieurs méthodes d'authentification, dont les comptes locaux, l'authentification via LDAP et l'intégration avec des fournisseurs d'identité externes (Grafana Security Docs).

- **Gestion des rôles (RBAC)** : les utilisateurs peuvent se voir attribuer des rôles définissant leurs permissions (administration, édition, consultation).
- **Chiffrement TLS** : les communications entre les utilisateurs et le serveur Grafana doivent être chiffrées afin de protéger les données affichées.
- **Séparation des accès** : les tableaux de bord peuvent être restreints à des groupes spécifiques.

  Ces mécanismes s'inscrivent dans les recommandations des normes de sécurité, qui préconisent la protection des outils de supervision au même titre que les applications métiers (ISO/IEC 27001).

  Dans une architecture ZTNA, les outils de supervision contiennent des informations sensibles sur l'infrastructure. Leur sécurisation est donc essentielle afin d'éviter qu'ils ne deviennent une source d'information exploitable par un attaquant.

2.5.6.4 Intégration de Grafana dans une architecture ZTNA

Grafana s'intègre naturellement dans l'architecture ZTNA comme **interface centrale de supervision**. Il permet de visualiser les métriques collectées par Prometheus concernant :

- L'état du serveur d'authentification ;
- Le trafic et les accès via le proxy Zero Trust ;
- La disponibilité des applications internes ;
- Les performances des services exposés publiquement.

Cette visibilité contribue à la **détection proactive des incidents** et à l'amélioration continue de la sécurité. Le NIST souligne l'importance de la surveillance et de l'analyse des événements pour renforcer la posture Zero Trust (NIST SP 800-207, 2020).

Grafana permet de transformer des données brutes en informations exploitables, facilitant la prise de décision et la validation du bon fonctionnement de l'architecture ZTNA.

2.5.6.5 Avantages, limites et justification du choix

Avantages

- ✓ Interface graphique riche et intuitive.
- ✓ Forte intégration avec Prometheus.
- ✓ Grande flexibilité dans la création de tableaux de bord.
- ✓ Large adoption et écosystème de plugins.

Limites

- Ne stocke pas les données (dépendance aux sources).
- Courbe d'apprentissage pour la création de dashboards complexes.
- Sécurisation indispensable pour éviter l'exposition d'informations sensibles.

Grafana a été retenu afin de fournir une visualisation claire et exploitable des métriques collectées par Prometheus. Son intégration avec l'architecture ZTNA permet de superviser efficacement les composants critiques et de démontrer la valeur ajoutée de la surveillance continue dans une approche Zero Trust.

Grafana constitue un **complément indispensable à Prometheus** dans la supervision de l'architecture ZTNA. En offrant une interface de visualisation puissante et sécurisée, il permet de suivre l'état des services, d'identifier rapidement les anomalies et de renforcer la gouvernance de la sécurité. Son rôle est essentiel pour assurer la fiabilité et la transparence de la solution proposée.

Conclusion

Ce chapitre a permis de définir et de structurer l'architecture de la solution Zero Trust Network Access (ZTNA) mise en place. À travers l'analyse des besoins et le choix des technologies, une architecture cohérente et sécurisée a été conçue, intégrant Keycloak comme fournisseur d'identité, Pomerium comme reverse proxy Zero Trust chargé de l'authentification et de l'autorisation, NGINX pour le routage applicatif, ainsi que Cloudflared pour l'exposition sécurisée des applications publiques sans ouverture d'IP publique.

L’architecture proposée repose sur les principes fondamentaux du Zero Trust, notamment la vérification systématique de l’identité, l’application du moindre privilège et la segmentation des accès par application. La séparation claire entre les applications internes et publiques, ainsi que l’intégration d’une infrastructure de supervision, renforcent la sécurité globale tout en assurant une bonne visibilité sur l’état du système.

Ainsi, ce chapitre établit une base architecturale solide et évolutive, garantissant que les choix techniques effectués répondent aux exigences de sécurité, de contrôle d’accès et de maintenabilité. Cette architecture servira de fondement à la phase d’implémentation détaillée dans le chapitre suivant.

Chapitre 3 : Mise en œuvre de la solution Zero Trust Network Access (ZTNA)

Introduction

Ce chapitre présente la mise en œuvre pratique de la solution Zero Trust Network Access (ZTNA) conçue et déployée dans le cadre de ce travail. Contrairement aux chapitres précédents, qui abordaient les aspects théoriques et conceptuels du Zero Trust, ce chapitre se concentre sur l'architecture réelle mise en place, les choix technologiques, ainsi que le fonctionnement détaillé des flux d'accès.

L'objectif principal est de démontrer comment une architecture ZTNA peut être implémentée concrètement afin de sécuriser l'accès aux applications internes d'une entreprise, tout en permettant l'exposition contrôlée de certaines applications publiques. La solution repose sur une authentification centralisée, une autorisation granulaire, une isolation réseau stricte, ainsi qu'une supervision continue.

3.1 Présentation générale de l'architecture mise en place

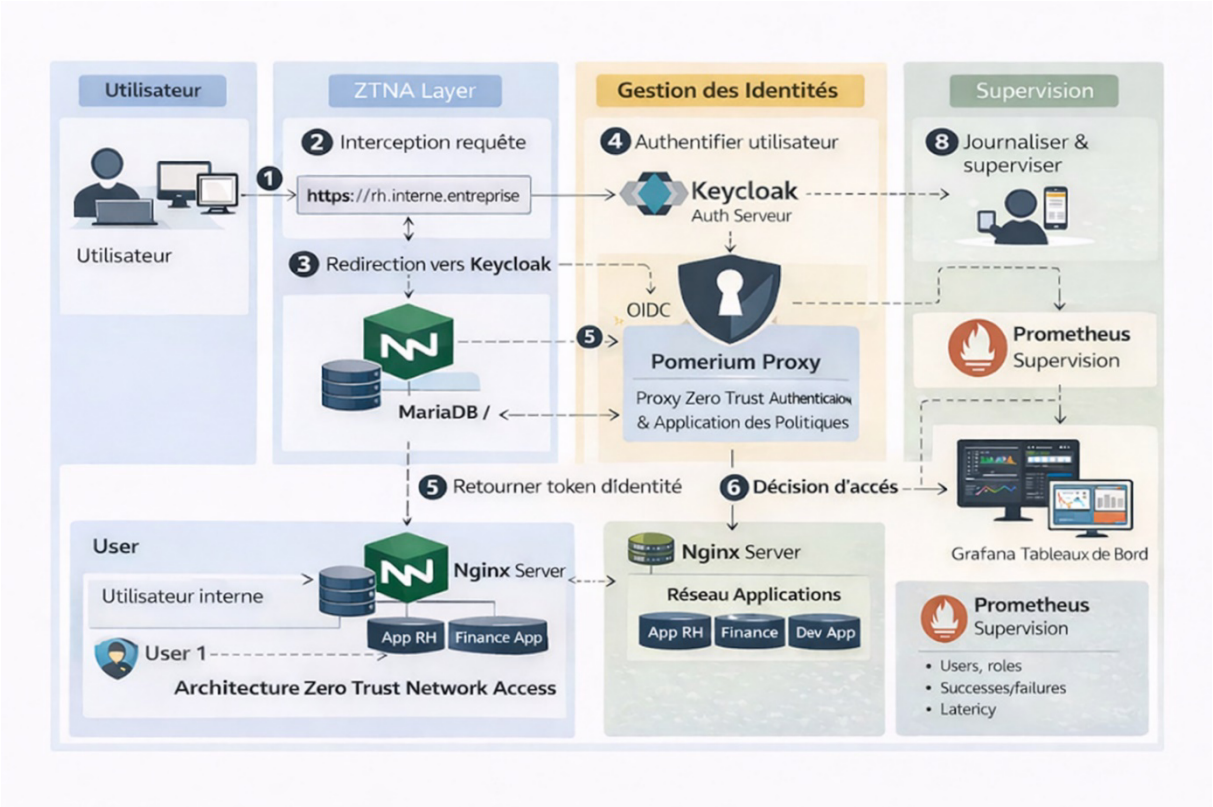


Figure 6: Architecture de la solution

L'architecture déployée repose sur le principe fondamental du Zero Trust : **ne jamais faire confiance par défaut**, même à l'intérieur du réseau. Ainsi, aucun utilisateur n'accède directement au réseau interne ou aux serveurs applicatifs.

La solution est organisée autour de deux flux principaux :

- **Un flux d'accès aux applications internes**, réservé aux utilisateurs authentifiés et autorisés
- **Un flux d'accès aux applications publiques**, exposées de manière sécurisée vers l'extérieur

Les composants principaux de l'architecture sont :

- **Keycloak** : serveur d'authentification et de gestion des identités
- **MariaDB** : base de données utilisée exclusivement par Keycloak
- **Pomerium** : proxy Zero Trust chargé de l'interception et de l'autorisation des accès internes
- **Nginx** : serveur web hébergeant les applications internes
- **Cloudflared** : tunnel sécurisé pour l'exposition des applications publiques
- **Prometheus** : collecte des métriques
- **Grafana** : visualisation et supervision

La procédure d'accès aux applications avec notre solution ZTNA de l'architecture ci-dessus se présente en plusieurs étapes comme-ci :

Étape 1 : Accès initial de l'utilisateur

À cette première étape, l'utilisateur tente simplement d'accéder à une application interne de l'entreprise à l'aide de son navigateur web**.**

L'utilisateur ouvre un navigateur (Chrome, Firefox, Edge, etc.), Il saisit l'URL de l'application interne, par exemple : https://rh.internal.ententedigital.

À ce stade :

L'utilisateur **n'est pas encore authentifié**

L'application interne **n'est pas directement accessible**

Aucune vérification d'identité n'a encore été faite

Aucune connexion au serveur applicatif n'est établie.

L'objectif de cette étape c'est d'initier une demande de connexion aux applications, aucune confiance n'est accordée à l'utilisateur, même s'il est interne au réseau.

Étape 2 : Interception par Pomerium

À cette étape, la requête envoyée par l'utilisateur est **interceptée par Pomerium**, qui joue le rôle de **point de contrôle Zero Trust**.

Pomerium agit comme :

- Un **proxy Zero Trust**
- Un **point d'entrée unique** vers les applications internes
- Un **gardien de la sécurité**

Il se situe entre l'utilisateur et les applications internes. Même si l'utilisateur est déjà dans le réseau interne, **Pomerium ne lui fait pas confiance**.

Étape 3 : Redirection vers Keycloak

À cette étape, Pomerium redirige l'utilisateur vers Keycloak afin de procéder à l'authentification.

Pomerium :

Détecte que l'utilisateur n'est pas authentifié

Il déclenche un flux d'authentification OpenID Connect (OIDC)

L'utilisateur est automatiquement redirigé vers Keycloak. Cette redirection est transparente pour l'utilisateur. Keycloak agit comme fournisseur d'identité (Identity Provider – IdP) Il est responsable uniquement de l'authentification, pas de l'autorisation.

Dans Zero Trust, **l'identité est la première barrière de sécurité**. Pomerium délègue donc cette vérification à un service spécialisé.

Étape 4 : Authentification sur Keycloak

À cette étape, l'utilisateur s'authentifie sur Keycloak, le serveur d'authentification central de l'architecture.

L'utilisateur saisit sur l'interface Keycloak :

- Son **nom d'utilisateur**
- Son **mot de passe**
- Éventuellement un **second facteur d'authentification (MFA).**

De son côté keycloak :

- ✓ Reçoit les identifiants
- ✓ Vérifie leur validité
- ✓ Consulte **MariaDB**
- ✓ Contrôle les rôles et groupes associés à l'utilisateur.

L'authentification est centralisée, forte et indépendante des applications.

Étape 5 : Génération du token d'identité

Après une authentification réussie, Keycloak génère un token d'identité sécurisé qui prouve que l'utilisateur est bien authentifié.

Keycloak :

- ✓ Confirme que l'authentification est valide
- ✓ Crée un **token d'identité OpenID Connect (OIDC)**
- ✓ Signe cryptographiquement ce token.

Le token contient plusieurs informations importantes, appelées **claims**, notamment :

- L'identité de l'utilisateur (username, email)

- Les **groupes**
- Les **rôles**
- L'identifiant du domaine (realm)
- La durée de validité du token

Ces informations serviront à décider à quelles applications l'utilisateur peut accéder. L'accès ne repose pas sur une session réseau, mais sur un jeton d'identité vérifié et temporaire.

Étape 6 Retour du token vers Pomerium et décision d'accès

Cette étape, le token d'identité généré par Keycloak est renvoyé à Pomerium, qui va prendre la décision d'autorisation.

Le retour du token se passe en :

- ✓ Le navigateur de l'utilisateur transmet le token OIDC à Pomerium
- ✓ Le token prouve que l'utilisateur est correctement authentifié.
- ✓ L'action qui se passe chez pomerium :
- ✓ Vérifie la **signature** du token
- ✓ Contrôle sa **validité temporelle**
- ✓ Lit les **attributs d'identité** (groupes, rôles).

Pomerium compare les informations du token avec les **politiques de sécurité définies**.

Exemple de règle :

- Groupe = RH → accès à rh.internal
- Groupe ≠ RH → accès refusé

Être authentifié **ne suffit pas** : chaque accès est **explicitement autorisé**.

Étape 7 : Accès à l'application interne

À cette étape, l'accès a été autorisé. L'utilisateur peut maintenant atteindre l'application interne, mais toujours via Pomerium.

Après autorisation :

- **Pomerium transmet la requête** vers le serveur applicatif
- La requête est envoyée à **Nginx**

Nginx agit comme :

- **Serveur web**
- Point d'accès aux applications internes
- Intermédiaire entre Pomerium et l'application

L'utilisateur accède à une **application**, jamais au réseau.

Étape 8 : Supervision et journalisation

À cette étape, **l'ensemble du processus d'accès est surveillé et journalisé**, afin d'assurer la visibilité, la sécurité et l'auditabilité de l'architecture.

Prometheus collecte automatiquement :

- L'état des services (Pomerium, Keycloak, Nginx)
- La disponibilité des applications
- Les performances (latence, charge)
- Les erreurs (HTTP, authentification).

Grafana utilise les métriques de Prometheus pour :

- Afficher des **tableaux de bord**
- Visualiser les accès autorisés et refusés
- Suivre l'évolution des performances
- Faciliter la détection d'anomalie

3.2 Mise en œuvre Keycloak,Mysql,Nginx

Keycloak est une solution open source de gestion des identités et des accès (IAM). Elle permet aux applications et services de gérer de manière centralisée l'authentification, l'autorisation et la gestion des utilisateurs, sans implémenter ces fonctions dans chaque application. L'objectif d'un outil IAM est de garantir que les personnes concernées au sein d'une entreprise disposent d'un accès approprié aux ressources. Conçu pour répondre aux exigences de sécurité modernes, keycloak prend en charge plusieurs standards d'authentification : OpenID Connect(OIDC), OAuth 2.0, SAML 2.0. Cette solution offre aussi plusieurs fonctionnalités qui sont :

Authentification centralisée, Gestion des utilisateurs et des rôles, fédération d'identités, l'authentification multi facteur.

Notre travail est segmenté en différentes partie ; en commençant par la configuration de l'environnement, installation de keycloak en H2 pour les tests, migration vers la production en utilisant myql MariaDB, configuration de Nginx comme reverse proxy pour les redirections et les certificats SSL.

3.2.1 Environnement de travail

Pour notre laboratoire nous avons choisis le VMware Workstation Pro comme environnement, pour nos serveurs on a pris deux serveur Ubuntu2024. Le premier serveur sera le keycloak et l'autre assura la base de données de keycloak qui est le MySQL. Kali-linux sera notre Client.

3.2.1.1 Configuration réseaux

Toutes nos machines sont en réseau interne pour faire nos tests localement. Voici les différentes ip des machines :

Serveur keycloak : 10.10.10.4/24 et 192.168.248.136 en NAT

Serveur MySQL : 10.10.10.5/24

Client :10.10.10.3/24

Mettre le nom de domaine du serveur keycloak sur toutes nos machines.

etc/hosts

10.10.10.4 keycloak.internal

1.2 vérification de config réseaux

```
serveur@keycloak:/etc$ ping 10.10.10.5
PING 10.10.10.5 (10.10.10.5) 56(84) bytes of data.
64 bytes from 10.10.10.5: icmp_seq=1 ttl=64 time=0.525 ms
64 bytes from 10.10.10.5: icmp_seq=2 ttl=64 time=0.701 ms
64 bytes from 10.10.10.5: icmp_seq=3 ttl=64 time=0.557 ms
```

Figure 7: ping keycloak vers mysql

```
┌──(kali㉿kali)-[~]
└─$ ping keycloak.internal
PING keycloak.internal (10.10.10.4) 56(84) bytes of data.
64 bytes from keycloak.internal (10.10.10.4): icmp_seq=1 ttl=64 time=0.702 ms
64 bytes from keycloak.internal (10.10.10.4): icmp_seq=2 ttl=64 time=0.504 ms
64 bytes from keycloak.internal (10.10.10.4): icmp_seq=3 ttl=64 time=0.530 ms
64 bytes from keycloak.internal (10.10.10.4): icmp_seq=4 ttl=64 time=0.527 ms
^C
```

Figure 8: ping du client vers keycloak

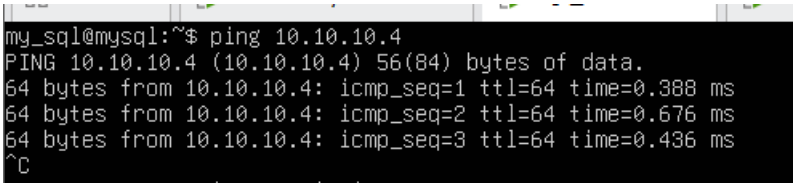


Figure 9:ping de mysql vers keycloak

Sur les images ci-dessus nous remarquons nos tests passent, ainsi nous avons une bonne configuration réseau.

3.2.2 Installation de keycloak en mode test

Le mode H2 est une configuration par défaut de keycloak qui utilise une base de données intégrée et légère conçu pour les tests. H2 fonctionne à l'intérieur du processus java de Keycloak, sans nécessiter de serveur externe. Les données peuvent être enregistrées en mémoire ou dans un simple fichier local. Pour installer keycloak nous allons télécharger via https://www.keycloak.org/ sur notre machine Windows et copier vers le serveur ou télécharger le package directement sur le serveur.

3.2.2.1 Copie du fichier de téléchargement vers le serveur

Une fois le télécharger effectuer nous avons copié le fichier sur le serveur comme l'a montré l'image ci-dessous.

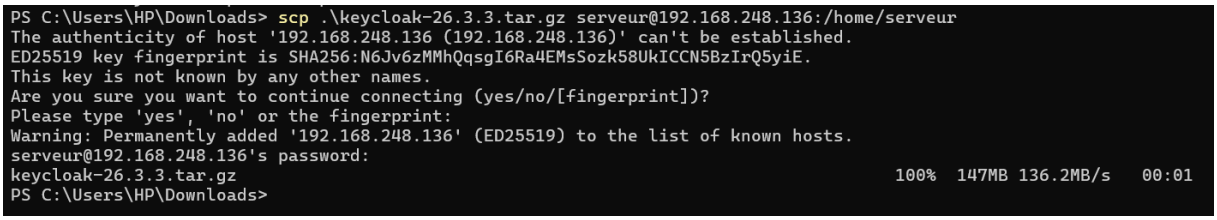


Figure 10: copie de keycloak vers le serveur

3.2.2.2 Vérifications du fichier

Nous remarquons la présence du fichier copier sur le serveur

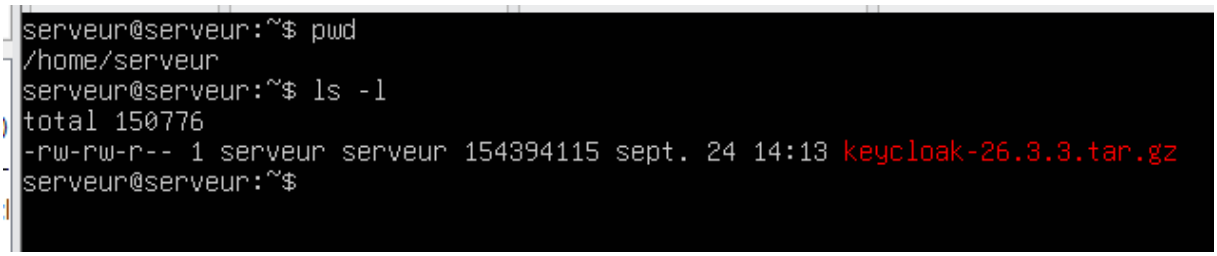


Figure 11: vérification du fichier

3.2.2.3 Installation de java

Comme H2 fonctionne à l'intérieur de processus java de keycloak, nous allons installer le jdk de java s'il n'existe pas sur notre serveur. Nous allons vérifier l'existence de la version et installer si besoin.

3.2.2.3.1 Vérification de la version de java

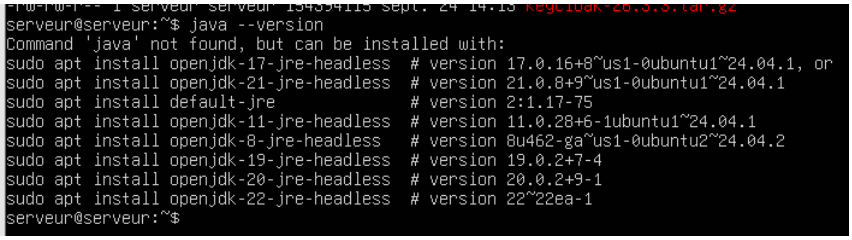


Figure 12: verification de java

Java n'est pas installé sur le serveur, nous allons installer sur le serveur

3.2.2.3.2 Installation java jdk

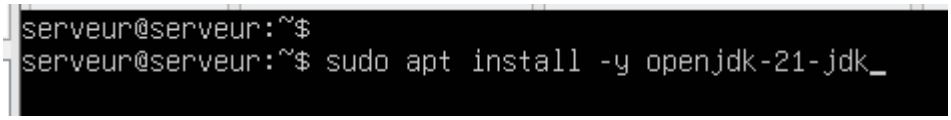


Figure 13: installation jdk java

Une fois installé nous allons revérifier

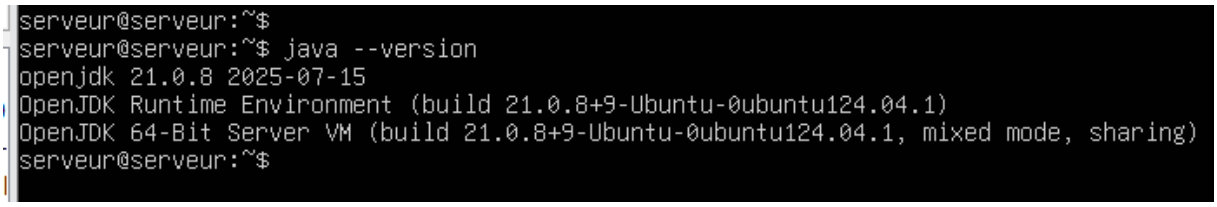


Figure 14: verification jdk java

Nous remarquons ainsi le java est bien installé sur notre serveur.

3.2.2.4 Décompression du fichier d'installation keycloak

Une fois installé la version de java appropriée nous allons décompresser le fichier d'installation qu'on avait copié sur notre serveur depuis la machine local windows .



Figure 15: Décompression du fichier d'installation keycloak

Nous allons déplacer le fichier compressé vers le répertoire /opt/keycloak(créer le répertoire keycloak s'il n'existe pas)

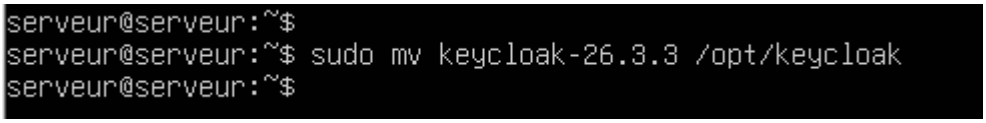


Figure 16: déplacement du fichier compresse

3.2.2.5 Configuration de keycloak H2

Dans cette partie nous créerons des utilisateurs, attribuerons des droits à l'utilisateur créer et lancer keycloak sur notre serveur.

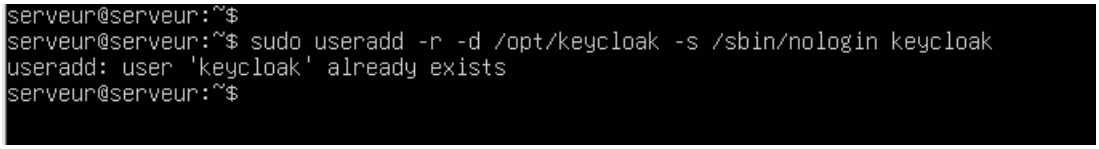


Figure 17: création d'utilisateur

Ici nous remarquons l'utilisateur existe déjà, parce que nous l'avons créé auparavant.

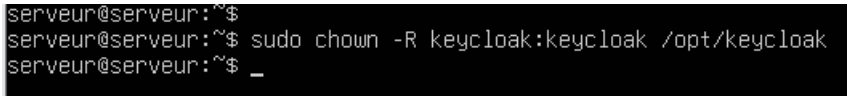


Figure 18: attribuer les fichiers à keycloak

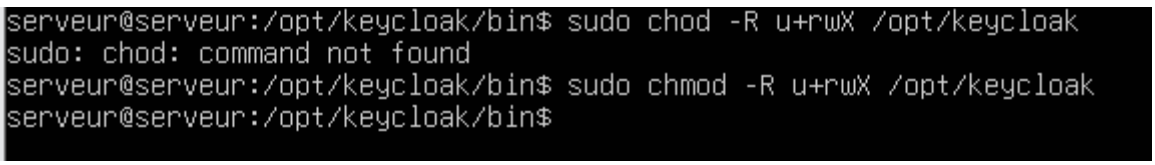


Figure 19: changement des permissions

Sur la capture des images ci-dessus nous avons attribuer tous les fichiers sur /opt/keycloak à l'utilisateur keycloak et donner des permissions de lecture, écriture et exécution sur tous les fichiers et dossiers récursivement.

Après l'attribution des différents droits nous allons lancer keycloak sur le serveur et vérifier si c'est actif sur le serveur.

```
serveur@serveur:~$
serveur@serveur:~$ cd /opt/keycloak/bin
serveur@serveur:/opt/keycloak/bin$ ./kc.sh start-dev
Updating the configuration and installing your custom providers, if any. Please wait.
```

Figure 20: lacement de keycloak en mode dev

Nous avons lancer keycloak en mode développement maintenant vérifions son état.

```
serveur@serveur:/opt/keycloak/bin$ sudo systemctl daemon-reload
serveur@serveur:/opt/keycloak/bin$ sudo systemctl enable --now keycloak
serveur@serveur:/opt/keycloak/bin$ sudo systemctl start keycloak
serveur@serveur:/opt/keycloak/bin$ sudo systemctl status keycloak
● keycloak.service - keycloak Service
     Loaded: loaded (/etc/systemd/system/keycloak.service; enabled; preset: enabled)
     Active: active (running) since Wed 2025-09-24 15:58:11 UTC; 1min 19s ago
   Main PID: 7228 (java)
      Tasks: 38 (limit: 4548)
     Memory: 422.9M (peak: 430.0M)
        CPU: 52.815s
     CGroup: /system.slice/keycloak.service
             └─7228 java -Dkc.config.built=true -Djava.util.concurrent.ForkJoinPool.common.t

sept. 24 15:58:26 serveur kc.sh[7228]: Running the server in development mode. DO NOT use th
sept. 24 15:58:31 serveur kc.sh[7228]: 2025-09-24 15:58:31,724 INFO  [org.keycloak.quarkus.r
sept. 24 15:58:36 serveur kc.sh[7228]: 2025-09-24 15:58:36,048 INFO  [org.keycloak.spi.infin
sept. 24 15:58:36 serveur kc.sh[7228]: 2025-09-24 15:58:36,186 INFO  [org.infinispan.CONTAIN
sept. 24 15:58:36 serveur kc.sh[7228]: 2025-09-24 15:58:36,342 INFO  [org.infinispan.CONTAIN
sept. 24 15:58:36 serveur kc.sh[7228]: 2025-09-24 15:58:36,733 INFO  [org.keycloak.connectio
sept. 24 15:58:36 serveur kc.sh[7228]: 2025-09-24 15:58:36,909 INFO  [org.keycloak.services]
sept. 24 15:58:38 serveur kc.sh[7228]: 2025-09-24 15:58:38,887 INFO  [io.quarkus] (main) Key
sept. 24 15:58:38 serveur kc.sh[7228]: 2025-09-24 15:58:38,888 INFO  [io.quarkus] (main) Pro
sept. 24 15:58:38 serveur kc.sh[7228]: 2025-09-24 15:58:38,888 INFO  [io.quarkus] (main) Ins
```

Figure 21: Vérification de l'état de service keycloak

Nous remarquons le service est fonctionnel sur le serveur, Nous allons accéder à l'interface graphique via notre navigateur de notre machine local.

Nous accédons à l'interface de keycloak via notre navigateur de notre machine windows avec l'adresse ip 192.168.248.136 :8080 qui est l'adresse de notre interface Nat qui sera accessible par toutes les machines connectées à notre réseau.

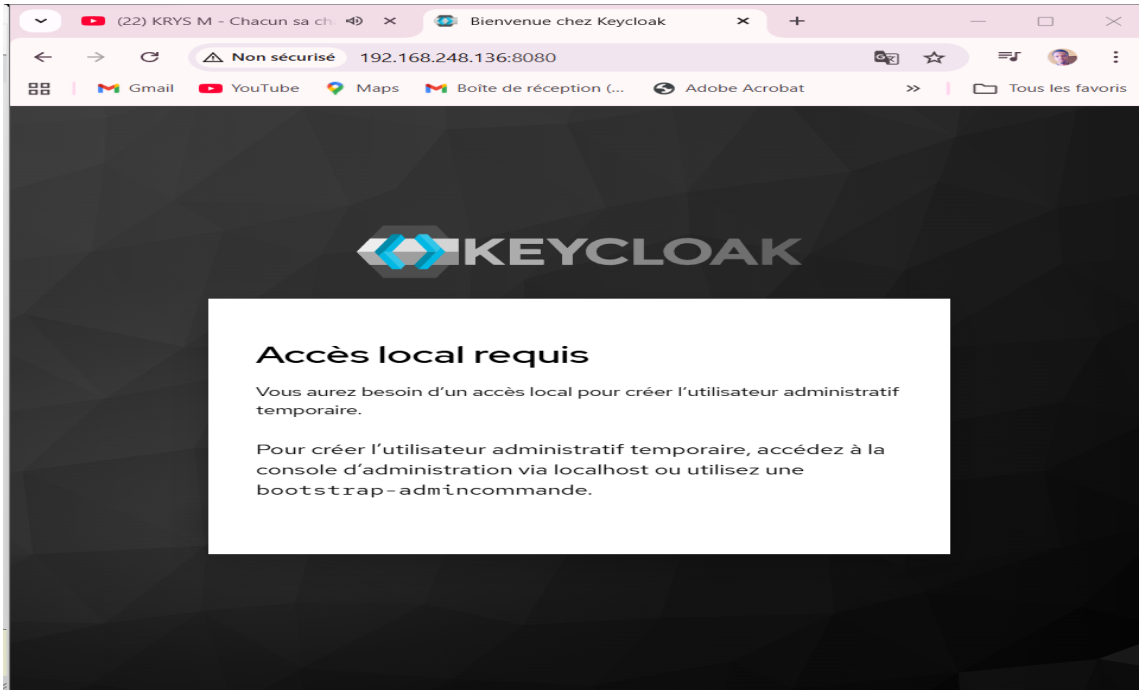


Figure 22: accès http

On constate que nous avons accéder à l'interface graphique via notre navigateur, mais nous devons créer un utilisateur administratif temporaire pour pouvoir accéder à l'administration du serveur.

Nous allons créer un compte admin avec un mot de passe sur terminal

```
serveur@serveur:/opt/keycloak/bin$
serveur@serveur:/opt/keycloak/bin$
serveur@serveur:/opt/keycloak/bin$ sudo -u keycloak KEYCLOAK_ADMIN=Bokhit KEYCLOAK_ADMIN_PASSWORD='@@235' /opt/keycloak/bin/kc.sh start-dev
```

Figure 23: création compte Admin

Sur notre serveur nous avons créé un admin qui est 'Bokhit' et un mot de passe '@@235', puis relancer le serveur. Maintenant accédons avec l'admin créer.

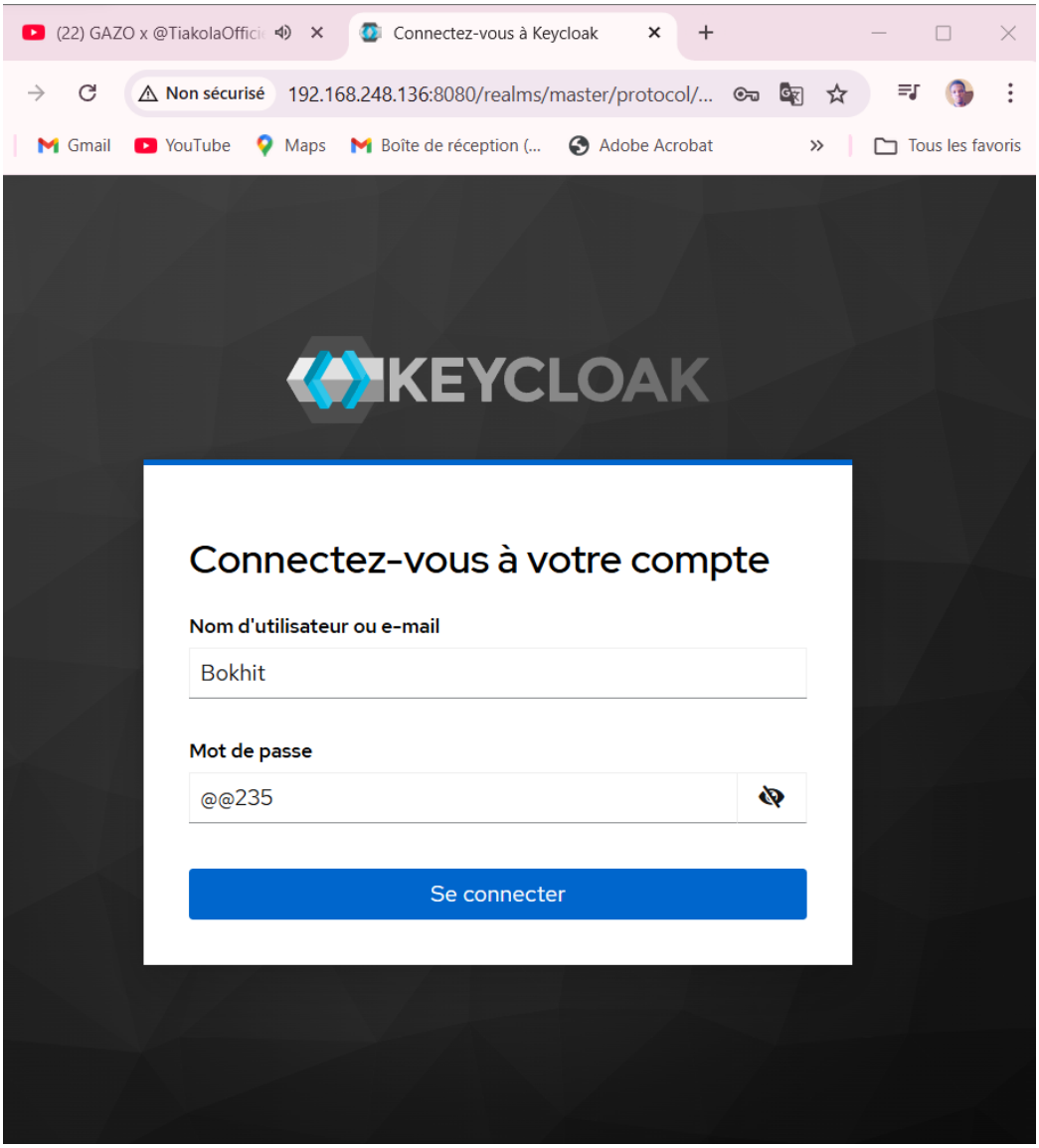


Figure 24: connexion avec Admin

Nous remarquons que nous nous somme connecter avec l'admin créer, explorons l'interface.

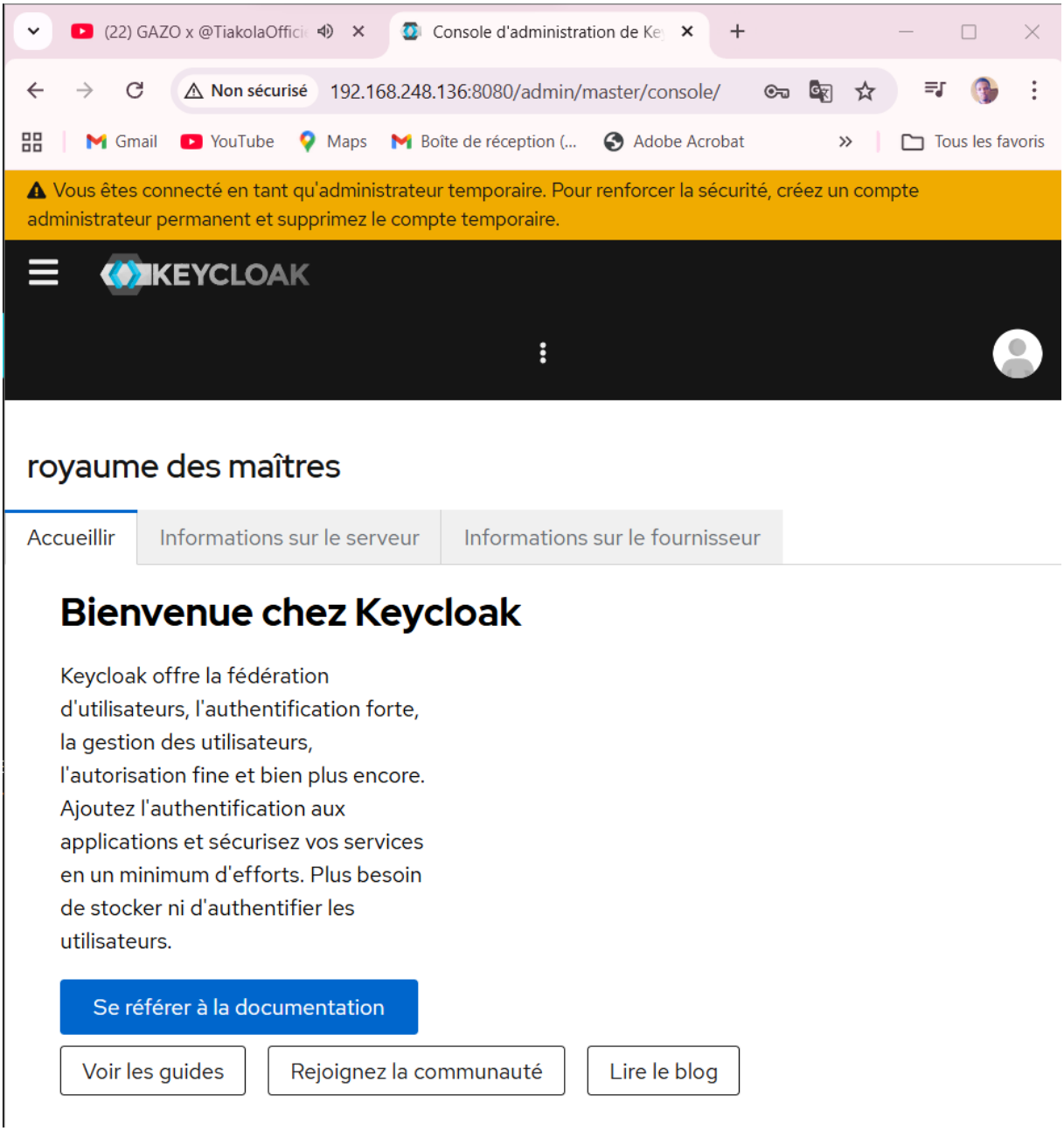


Figure 25: Interface graphique keycloak 1

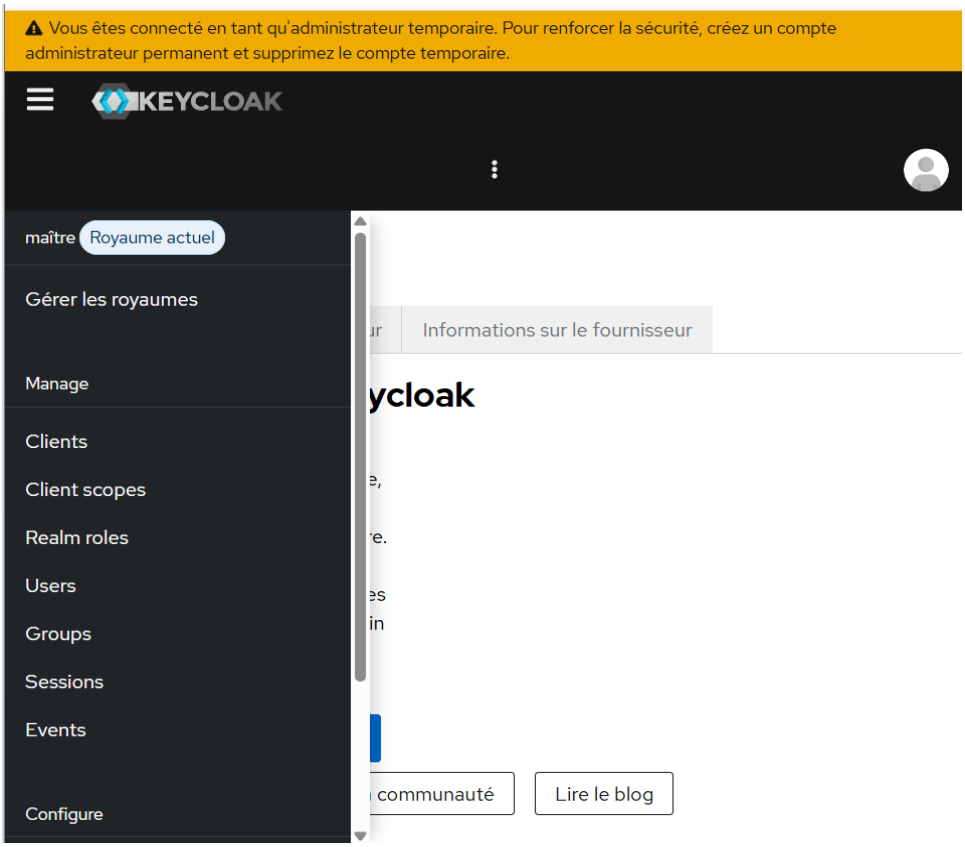


Figure 26: Menu keycloak 2

En accédant à notre serveur nous remarquons que nous avons plusieurs champs tels que le royaumes, les clients, les utilisateurs, les groups, les sessions…

Nous avons terminé avec la configuration de keycloak pour le mode test qui permet d'avoir la main sur le service et explorer les différentes parties. Dans la partie suivante notre travail nous allons passer en mode production en migrant le H2 vers la production. Par se faire nous allons configurer un serveur de base de données mysql et migrer le serveur keycloak vers la production et mettre en place un reverse proxy Nginx.

Dans la partie précédente nous avons installé le serveur en mode test, H2 est une base impliquée destinée au développement et test uniquement. Pour la production nous allons installer une base de données non éphémère, Maria DB/MYSQL est choisi comme la base de données de notre service.

Les prochaines nous montrerons les différentes étapes de passage de keycloak en mode test à la mode production.

3.2.3 Passage en mode Production

Dans cette étape nous configurerons nos deux serveurs avec leur service correspondant.

3.2.3.1 Configuration du serveur MySQL

On a choisi un serveur à part pour la base de données de notre serveur keycloak, avec une adresse IP 10.10.10.5 /24

Nous allons installer MySQL comme l'a montrera l'image ci-dessous.

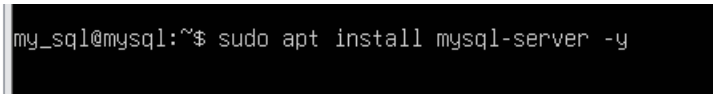


Figure 27:Installation MySQL

Après installation on va éditer le fichier de configuration /etc/mysql/mysql.conf.d/mysqld.cnf pour la configuration .

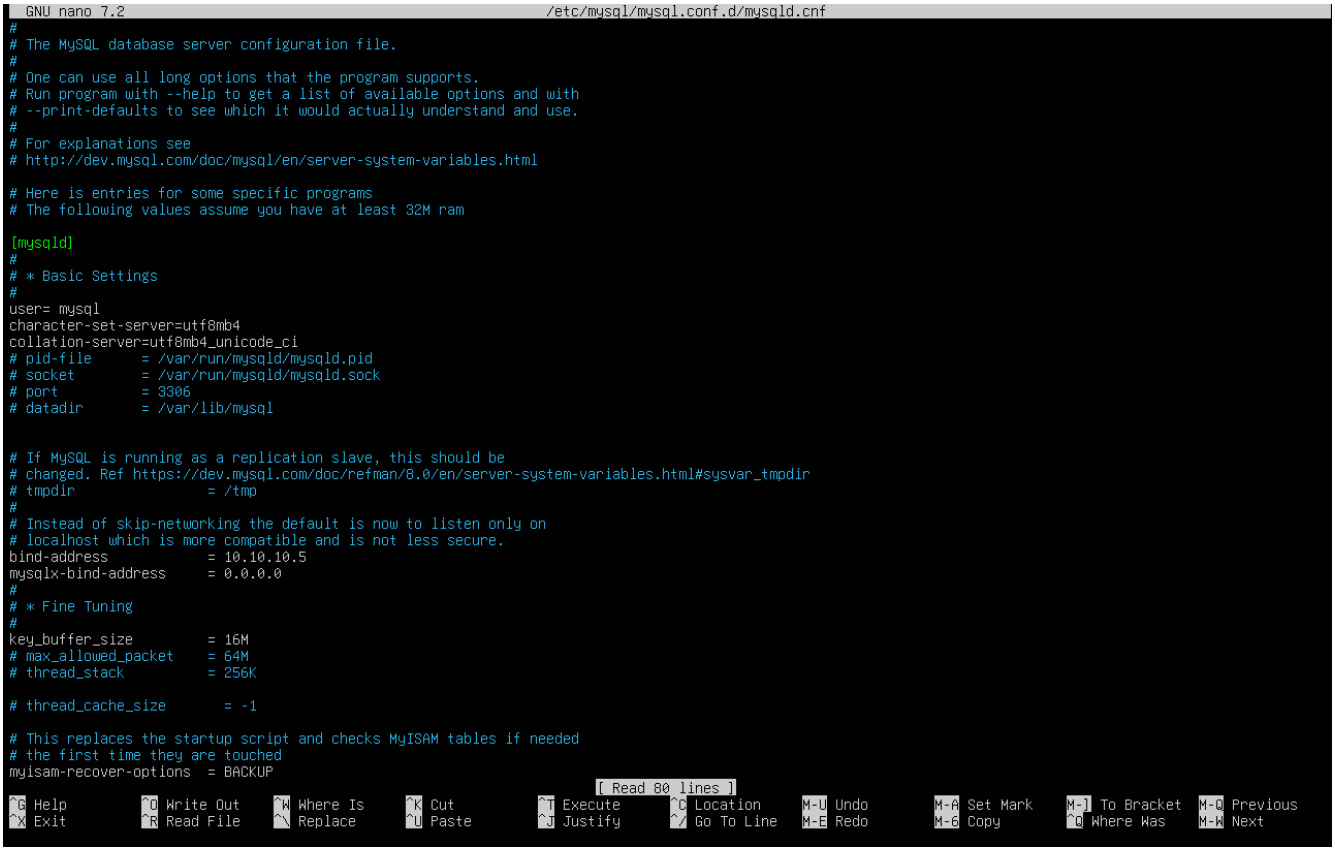


Figure 28: Fichier de configuration mysql

Nous avons édité le fichier de configuration de mysql et fait plusieurs configurations qui sont :

Défini un utilisateur système nommé **mysql,** le jeu de caractère par défaut pour toutes les bases de données, crée un support d'Unicode, l'ordre de tri pour le jeu de caractères par ordre alphabétique.

Nous avons aussi défini le port d'écoute 3306 qui est le port par défaut que nous allons changer plus tard, le bind-address qui indique sur quelle adresse IP le serveur MySQL écoute les connexions, nous avons autorisé MYSQL à écouter sur l'adresse 10.10.10.5 qui sera accessibles depuis d'autres machines du réseau. Après ces configurations on redémarre le serveur et voir le port d'écoute.

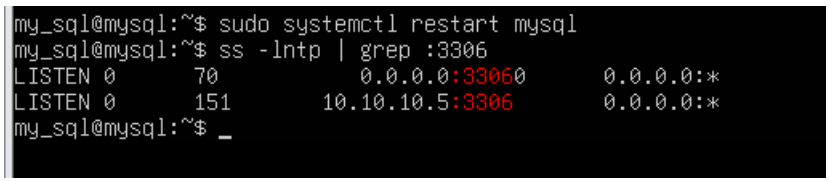


Figure 29: vérification de port d'écoute

Ici nous remarquons que notre serveur écoute sur toutes les interfaces (0.0.0.0) c'est-à-dire accessible depuis n'importe quelle IP, aussi spécifiquement sur l'IP de notre serveur DB qui est 10.10.10.5.

Une fois que notre serveur est bien configuré nous aller procéder à la création d'une base de données.



Figure 30: création de la base données keycloak

Sur la capture ci-dessus nous avons créé une base nommée keycloak, qui utilise le charset compatible avec Keyclaok et tout texte international et la collation qui respecte la casse et l'ordre Unicode standard. Maintenant nous allons créer notre utilisateur nommé kc_user



Figure 31: création d'utilisateur

On a créé un utilisateur MYSQL nommé kc_user qui pourra se connecter uniquement depuis notre serveur Keycloak, un mot de passe de connexion, donné tous les droits à notre utilisateur sur la base keycloak.

Ensuite nous allons vérifier si notre utilisateur est bien créé

```
my_sql@mysql:~$ sudo mysql -u root -p -e "SHOW DATABASES LIKE 'keycloak';"
Enter password:
+--------------------+
| Database (keycloak) |
+--------------------+
| keycloak           |
+--------------------+
my_sql@mysql:~$ _
```

Figure 32: vérification base crée

Notre base de données existe dans notre serveur mysql, donc la base de données nommé keycloak a été bien crée.

```
mysql> SELECT Host, User FROM mysql.user;
+-----------+------------------+
| Host      | User             |
+-----------+------------------+
| 10.10.10.4 | kc_user          |
| localhost  | debian-sys-maint |
| localhost  | mysql.infoschema |
| localhost  | mysql.session    |
| localhost  | mysql.sys        |
| localhost  | root             |
+-----------+------------------+
6 rows in set (0,00 sec)
```

Figure 33: vérification utilisateur

Nous constatons plusieurs utilisateur système et notre utilisateur kc_user qui sera utilisé par notre serveur keycloak pour se connecter.

3.2.4 Configuration du serveur keycloak

Nous avons mis en place un serveur de base de données pour notre serveur, maintenant nous configurerions notre serveur keycloak

Keycloak ne sait pas nativement parler à MariaDB/MYSQL on lui a ajouté le driver JDBC pour permettre la communication entre le serveur et la base de MariaDB.

```
serveur@keycloak:/opt/keycloak/providers$
serveur@keycloak:/opt/keycloak/providers$ sudo apt install -y libmariadb-java
[sudo] password for serveur:
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances... Fait
Lecture des informations d'état... Fait
libmariadb-java est déjà la version la plus récente (2.7.6-1).
0 mis à jour, 0 nouvellement installés, 0 à enlever et 115 non mis à jour.
serveur@keycloak:/opt/keycloak/providers$ _
```

Figure 34: installation libmariadb-java

```
serveur@keycloak:/$ sudo ln -sf /usr/share/java/mariadb-java-client.jar  /opt/keycloak/providers/mariadb-java-client.jar _
```

Figure 35:crée un lien pour le démarrage

Dans la capture ci-dessus de la bibliothèque qui contient le pilote de connexion on a créé un lien symbolique vers le dossier (/opt/keycloak/providers) que Keycloak analyse au démarrage.

Une fois redémarré Keycloak charge automatiquement tous les fichiers .jar dans le dossier /opt/keycloak/providers/ il détectera le pilote MariaDB au démarrage.

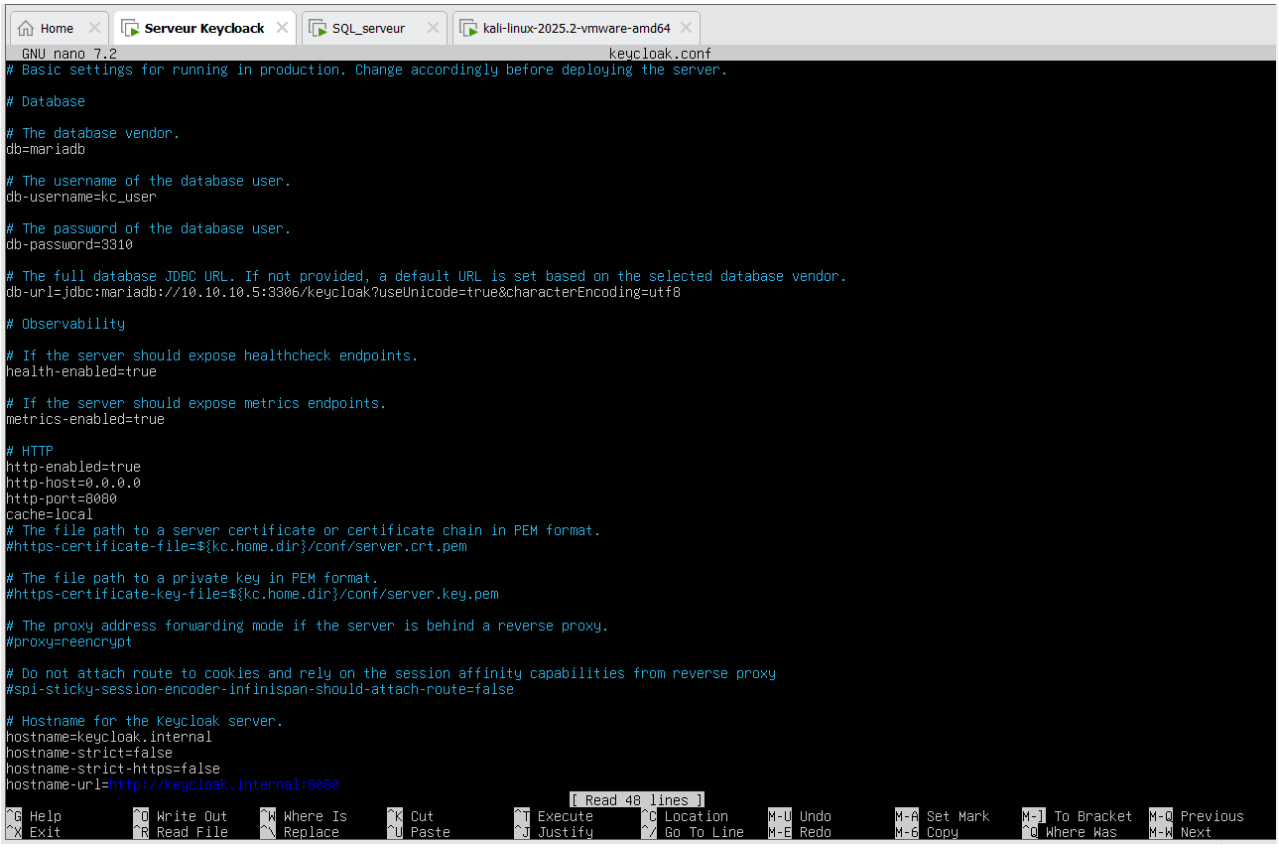


Figure 36: fichier de configuration keycloak

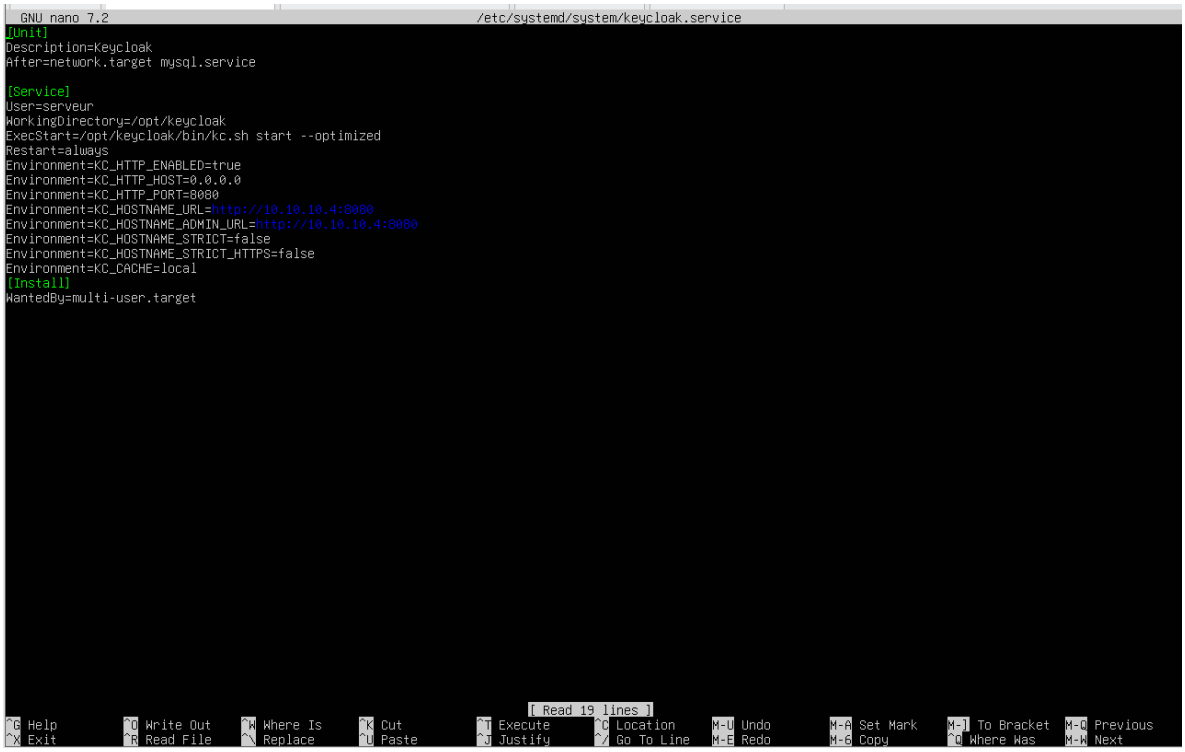
Dans le fichier ci-dessus nous avons indiqué à keyclaok d'utiliser MariaDB/MYSQL comme backend, définit les identifiants de connexions crée dans notre serveur mysql, db-url qui est

notre chemin de la connexion vers notre serveur de base de données qui contient l'adresse IP (10.10.10.5) du serveur DB, le port du serveur (3306), le nom de notre de base (keycloak).

On a activé aussi les endpoints de santé du serveur que nous allons intégrer à notre outil de santé Prometheus de vérifier si le service est vivant. Activer aussi les metriques pour surveiller les connexions, les sessions, la mémoire, etc…

Sur le bloc http nous avons activé qui permettra à keycloak d'écouter sur un port http, l'adresse et le port d'écoute.

Sur le bloc Hostname on a mis le nom de notre hôte qui est l'url que keycloak considéra valide pour les redirections et liens. Nous avons désactivé https pour l'instant et enfin l'url de la connexion.



Enfin nous avons enregistré notre serveur comme un service système via le systemd pour qu'il démarre automatiquement au boot, se redémarre en cas de crash et fonctionne en arrière-plan.

La description qui est le nom lisible du service qui s'affiche dans **systemctl status** ça affichera keyclaok.

Nous avons indiqué à systemd d'attendre que le réseau soit actif et que le service MySQL/MariaDB soit lancé avant de démarrer keyclaok. Cela évitera que Keycloak essaie de se connecter à la base avant qu'elle soit disponibles.

Pour les Service Keycloak s'exécute avec l'utilisateur serveur qui est l'utilisateur de notre mais pas avec root pour une bonne pratique de sécurité. Définit aussi le répertoire de travail par défaut. Lancer la commande que systemd exécutera pour démarrer keyclaok. Via l'option Restart= always si le service s'arrête par erreurs ou crash, le systemd le relance automatiquement.

Nous avons passés les variables d'environnement présentent dans le fichier de configuration keycloak.conf, au moment de démarrage ils remplaceront ou complèteront ceux du fichier keycloak.conf. La dernière indique que le service doit être lancer automatiquement au démarrage du système.

Une fois la configuration terminée on va recharger ces configurations du service, après on redémarre keycloak via la commande : sudo systemctl restart keycloak.

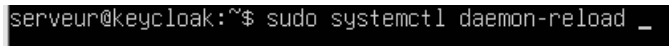


Figure 37: recharger les configurations



Figure 38: vérification tcp

Après vérification qur l'image ci-dessus nous remarquons que notre base de données est accessible en tcp depuis notre serveur keycloak.

Connectons-nous à notre base de données depuis notre serveur et faisons quelques sélections.

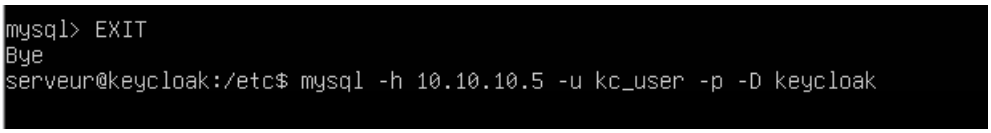


Figure 39: connexion à la base de données keycloak

Depuis le serveur keycloak on s'est connecté à notre base de données avec les identifiants :

-h 10.10.10.5 : c'est l'adresse de notre serveur mysql

-u kc_user : notre utilisateur crée dans le serveur mysql

-D keycloak : la base où on veut se connecter.

```
mysql> SELECT id, name FROM REALM;
+--------------------------------------+---------+
| id                                   | name    |
+--------------------------------------+---------+
| a5e74fd3-0b33-4895-a90e-a90d3213fd66 | Entente |
| 6e0901af-4ee9-4884-a59d-0514fcfd1fc0 | master  |
+--------------------------------------+---------+
2 rows in set (0,00 sec)
```

Figure 40: sélection

On s'est connecté à notre base de données depuis notre serveur keycloak en donnant l'IP du serveur DB, l'utilisateur et la base de données où nous voulons se connecter.

Après la connexion on a affiché les royaumes de notre serveur keycloak et nous remarquons que notre serveur utilise bel et bien comme backend le serveur mysql. Depuis notre client kali nous allons se connecter à l'interface graphique de notre serveur sur le navigateur.

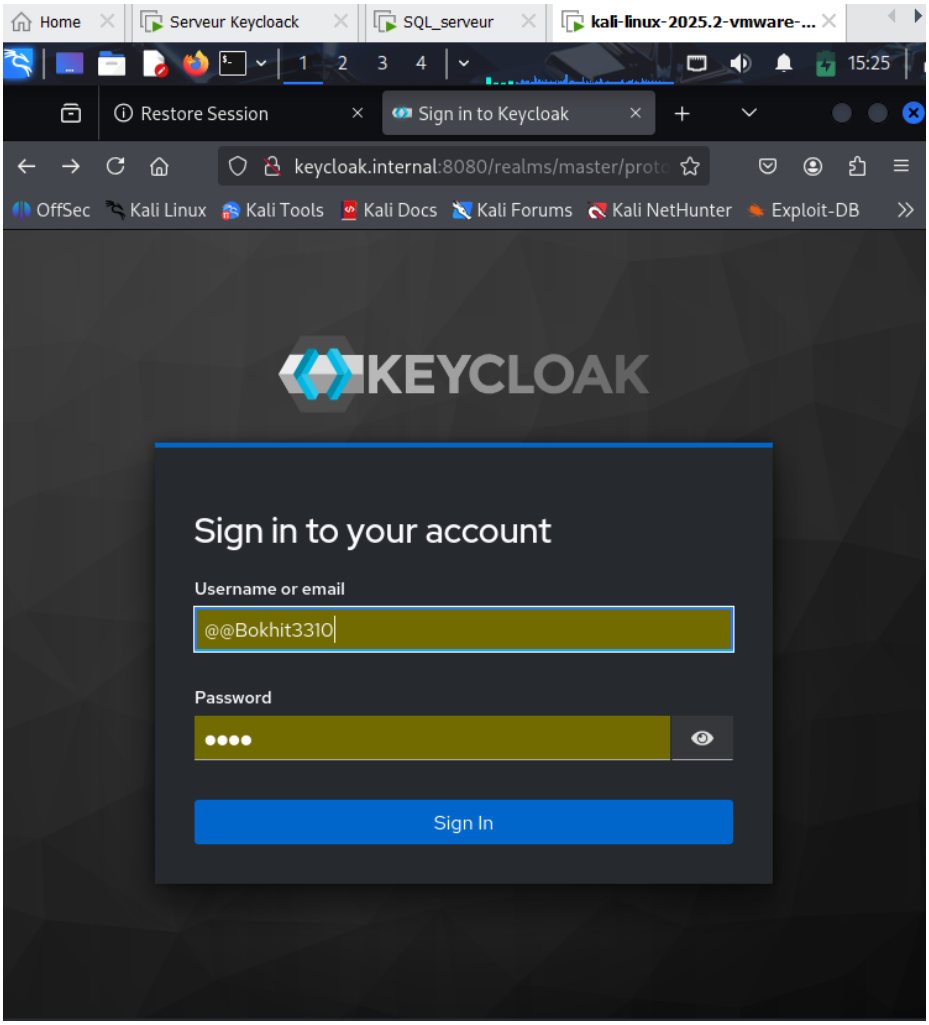


Figure 41: connexion via le navigateur

De notre client kali on s'est connecté à notre navigateur et nous remarquons aussi l'existence de notre utilisateur crée en mode H2, ainsi nous pouvons conclure que la migration s'est bien passée.

Cet utilisateur est temporaire nous allons créer un nouvel utilisateur permanent.

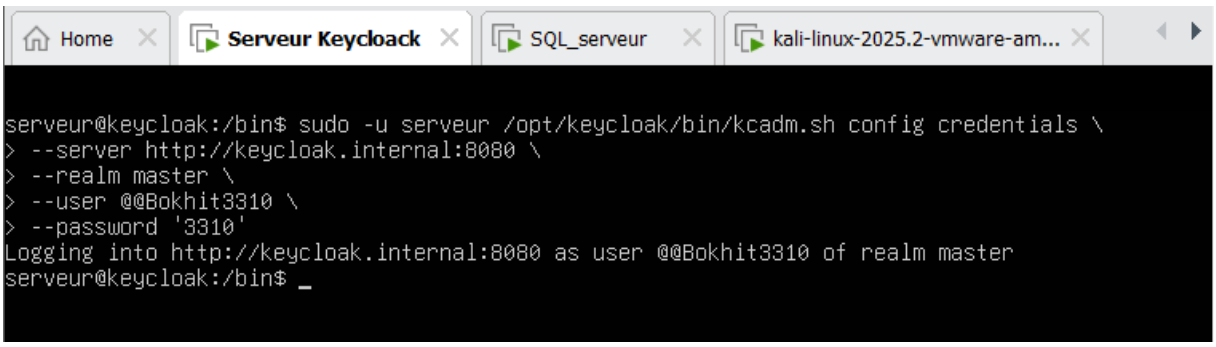


Figure 42: connexions à l'admin temp

On s'est connecté à notre utilisateur temporaire via au terminal pour créer notre administrateur permanent. Nous constatons que la connexion à l'API REST de keycloak est réussie.



Figure 43: création du nouvel utilisateur

On a créé notre nouvel utilisateur nommé @@bokhit2025, ajouté celui-ci dans le royaume master et activé le compte.

Après nous avons affiché la sortie JSON de confirmation pour confirmer l'existence de l'utilisateur créé dans notre base de données.

Nous avons assigné un mot de passe permanant à notre utilisateur.

```
User not found for username: @@Bokhit2025
serveur@keycloak:/bin$ sudo -u serveur /opt/keycloak/bin/kcadm.sh set-password -r master \
> --username @@bokhit2025 --new-password '@$Abdou235' --temporary=false
serveur@keycloak:/bin$ _
```

Figure 44: mot de passe nouvel utilisateur

En vérifiant nous constatons que notre utilisateur @@bokhit2025 existe dans base de données.

```
serveur@keycloak:/etc$ sudo -u serveur /opt/keycloak/bin/kcadm.sh config credentials \
> --server http://127.0.0.1:8080 \
> --realm master \
> --user @@Bokhit3310 \
> --password 3310
Logging into http://127.0.0.1:8080 as user @@Bokhit3310 of realm master
serveur@keycloak:/etc$ sudo -u serveur /opt/keycloak/bin/kcadm.sh get users -r master | grep username
  "username" : "@@bokhit2025",
      "name" : "username",
      "displayName" : "${username}",
        "username-prohibited-characters" : {
        "up-username-not-idn-homograph" : {
  "username" : "@@bokhit3310",
      "name" : "username",
      "displayName" : "${username}",
        "username-prohibited-characters" : {
        "up-username-not-idn-homograph" : {
```

Figure 45: vérification de l'utilisateur

```
serveur@keycloak:/etc$ sudo -u serveur /opt/keycloak/bin/kcadm.sh config credentials \
> --server http://127.0.0.1:8080 \
> --realm master \
> --user '@@bokhit2025' \
> --password '$@Abdou235'
Logging into http://127.0.0.1:8080 as user @@bokhit2025 of realm master
serveur@keycloak:/etc$
```

Figure 46: connexion avec le nouvel utilisateur

Avec le terminal de notre serveur keycloak on s'est bien connecté avec notre nouvel utilisateur, maintenant essayons de se connecter depuis notre client via le navigateur.

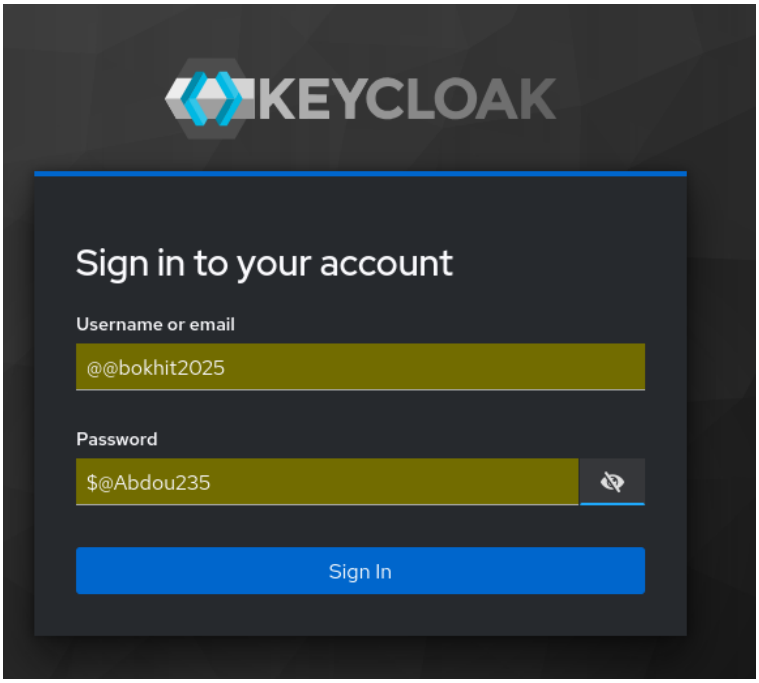


Figure 47: connexion navigateur 1

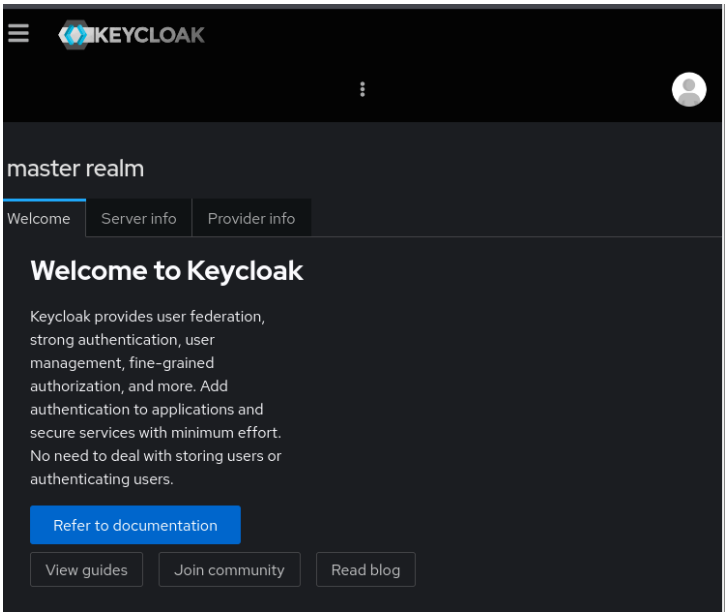


Figure 48: connexion navigateur 2

Nous constatons que le client s'est bien connecté avec le nouveau compte permanent.

Nous nous sommons accéder à notre en http, ce qui expose le trafic interne, ainsi nous allons mettre en place un proxy inverse. Nous configurerons Nginx qui recevra toutes les requêtes HTTPS externes, puis il déchiffra et relie keycloak en http interne.

3.2.5 Configuration du proxy interne

Nginx va gérer le chiffrement HTTPS, Le trafic sortant du navigateur est déchiffré par Nginx, puis envoyé à keycloak en clair sur le port 8080. Pour se faire nous allons mettre en place un CA interne qui généra des certificats TLS auto-signés, fiable localement. Les navigateurs et services internes feront confiance à notre qui sera installé en locale.

On a choisi mkcert pour créer notre CA interne, pour générer les certificats TLS locaux sur les machines de notre réseau local.

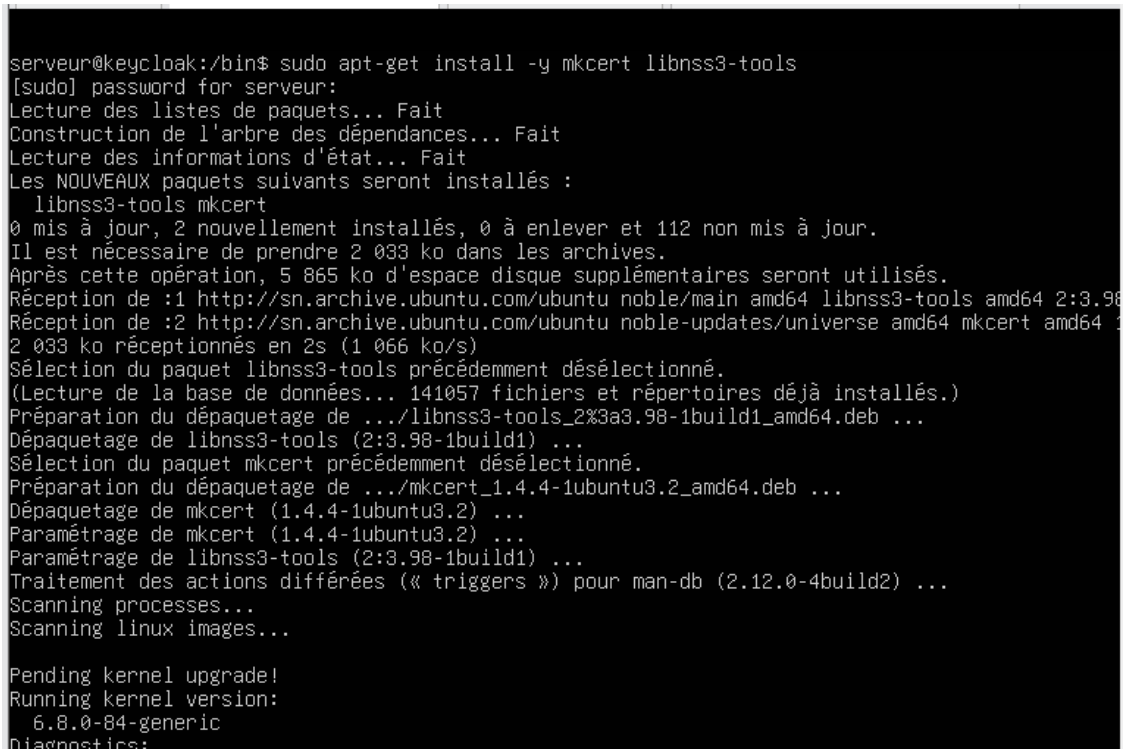


Figure 49: installation de mkcert

Ensuite nous avons créé notre Certification locale et installer dans le système pour qu'elle soit reconnue comme certification de confiance. Elle va générer une paire de clés RSA (une clé privée, une clé publique).

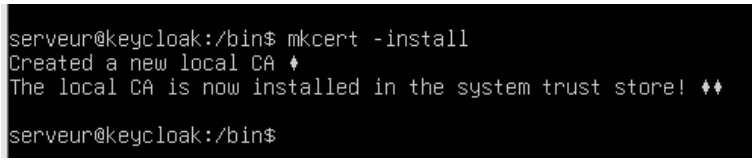


Figure 50: Création du CA

Nous avons signé le certificat par notre CA locale mkcert, après nous remarquons la durée de la validité qui est valable jusqu'au 16 janvier 2028 et pour l'instant on l'a limité que sur notre domaine keycloak.internal.

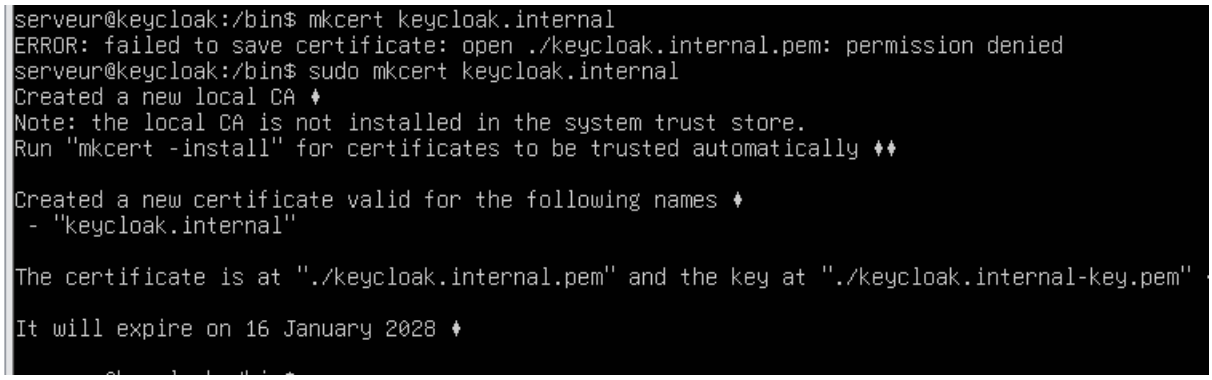


Figure 51: signature de keycloak.internal

L'isolation de nos certificats est très importante on ne doit pas les exposées pour cela on va créer un dossier et copier les deux certificats.

```
serveur@keycloak:/bin$ sudo mkdir -p /etc/ssl/keycloak
serveur@keycloak:/bin$ sudo cp keycloak.internal.pem /etc/ssl/keycloak/fullchain.pem
serveur@keycloak:/bin$ sudo cp keycloak.internal-key.pem /etc/ssl/keycloak/privkey.pem
serveur@keycloak:/bin$ _
```

Figure 52: sécurité des certificats

Nous avons copié le certificat public dans un fichier fullchain.pem est celui que Nginx utilisera pour identifer le serveur auprés de clients et la clé privée associée à notre certificat pour déchiffrer les connexions HTTPS.

On va isoler ces fichiers en renforçant la sécurité avec les attributions des droits suivant :

Sudo chmod 600 /etc/ssl/keycloak/privkey.pem : accessible uniquement par root car c'est le cœur de notre SSL. Sudo chmod 644 /etc/ssl/keycloak/fullchain.pem : le root et lisibles par les groupes et utilisateurs.

Une fois terminé avec les certificats nous allons installer et configuré notre serveur Nginx.

3.2.6 Configuration du reverse proxy Nginx

Notre Nginx interne nous servira de reverse proxy sécurisé placé devant Keycloak. Il a pour rôle d'assurer la terminaison TLS interne grâce à un certificat signé par une Autorité de Certification interne (mkcert), ce qui permet de chiffrer les échanges entre Pomerium et Keycloak. Il masque Keycloak du réseau, empêche tout accès direct au port 8080 et ne laisse transiter les requêtes que via un point d'entrée contrôlé.
On ajoutera également les en-têtes proxy nécessaires (X-Forwarded-) et réécrire les redirections afin que Keycloak fonctionne correctement derrière un proxy HTTPS.
Il permet un accès administrateur interne ou via VPN, sans passer par Pomerium, afin de sécuriser la console d'administration de Keycloak, hormis l'administrateur tous les autres doivent obligatoirement passer par le Pomerium.

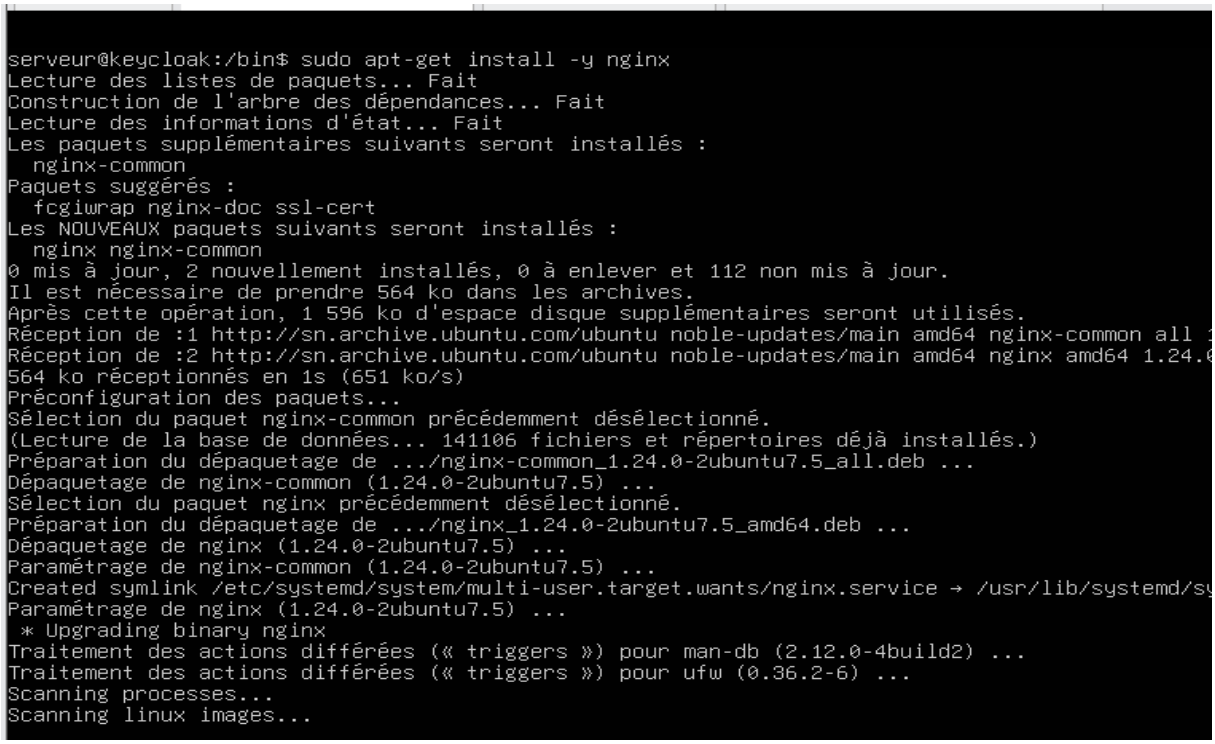


Figure 53: installation nginx

Sur le même serveur que keycloak nous avons installé notre serveur nginx et nous allons le placer devant keycloak. Comme ça c'est le nginx interne que sera l'intermédiaire entre le serveur keycloak et le Pomerium, ainsi on a isolé totalement notre serveur keycloak.

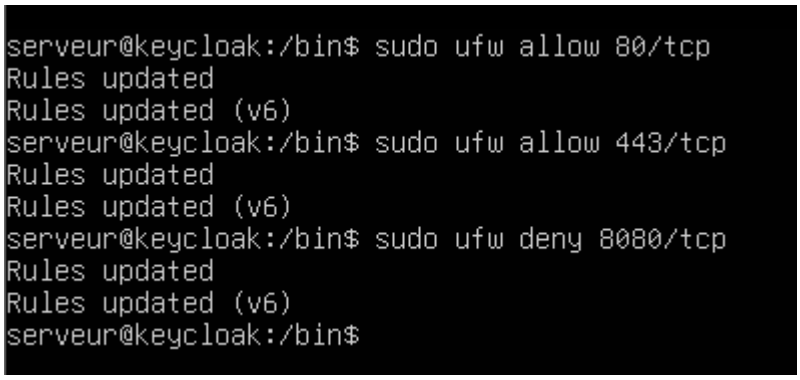


Figure 54: Configuration du pare-feu

Pour isoler notre serveur keycloak nous avons configuré trois pares-feux :

Port 80 : On a autorisé l'accès http sur le port 80 pour la redirection les requêtes vers le HTTPS, le but est uniquement de rediriger automatiquement nos utilisateurs vers un accès sécurisé en HTTPS.

Port 443 : On autorise le port 443 pour offrir un accès chiffré via HTTPS grâce à notre certificat interne, qui garantit ainsi la confidentialité des échanges vers notre proxy.

Port 8080 : Ce port est utilisé par keycloak en interne, on a explicitement bloqué ce port, ce qui empêche tout accès direct des utilisateurs au service keycloak, renforçant la sécurité en imposant le passage obligatoire par Nginx, qui joue le rôle de proxy sécurisé.

```
#
server {
   listen 443 ssl http2;

        # SSL configuration
        ssl_certificate  /etc/ssl/keycloak/fullchain.pem;
        ssl_certificate_key /etc/ssl/keycloak/privkey.pem;

        # Add index.php to the list if you are using PHP

        server_name keycloak.internal;

        location / {
                proxy_pass  http://127.0.0.1:8080;
                proxy_http_version 1.1;
                proxy_set_header Host $host;
                proxy_set_header X-Forwarded-Host $host;
                proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
                proxy_set_header X-Forwarded-proto https;
                proxy_set_header X-Forwarded-port 443;
                proxy_set_header Upgrade $http_upgrade;
                proxy_set_header Connection $connection_upgrade;
                proxy_redirect http://127.0.0.1:8080/ https://keycloak.internal/;
                proxy_redirect http://127.0.0.1:8080 https://keycloak.internal;
                proxy_redirect http://keycloak.internal:8080/ https://keycloak.internal/;
                proxy_redirect http://keycloak.internal:8080 https://keycloak.internal;
                proxy_connect_timeout 60s;
                proxy_send_timeout    120s;
                proxy_read_timeout    300s;
        }
}


server {
   listen 80;
   server_name keycloak.internal;
   return 301 https://$host$request_uri;

}
```

Figure 55: Fichier de configuration Nginx

La configuration du fichier.conf sur la capture ci-dessus s'explique comme ci :

Nous avons le bloc HTTPS du serveur frontal interne que nous avons configuré notre vhost écoute en HTTPS et active HTTP/2. Notre vhost répondra aux connexions HTTPS sur le port 443, et utilisera HTTP/2 pour optimiser les échanges.

Notre certificat interne et sa clé privée utilisés pour chiffrer les sessions vers keycloak.internal qui est notre FQDN interne.

Le bloc suivant est le proxy :

Location / : ce qui veut dire s'applique à toutes les URL, le chemin est la racine (/).

Proxy_pass http://127.0.0.1 :8080 : notre Nginx transmet au keycloak loacle tous les requetés reçus, le trafic ne sort jamais du serveur et aucun n'utilisateur du réseau ne peut l'atteindre directement.

Par défaut, nginx utilise http/1.0 entre lui et le backend, mais keycloak a des fonctionnalités qui nécessitent absolument http/1.1, proxy_http_version 1.1 est activé pour plus de fonctionnalité comme : le WebSockets qui n'existe pas en http/1.0, SSE(Server-Sent-Events) qui fonctionne mal en http/1.0.

Ensuite nous avons plusieurs en-têtes indispensable derrières le proxy :

X-Forwarded-Host : qui conserve le nom d'hôte keycloak.internal pour la génération correcte des URLs.

X-Forwarded-For : qui ajoute l'IP client dans la chaîne.

X-Forwarded-Proto https : indique à keycloak que le client est venu en HTTPS, sinon il pourrait produire les redirections en http.

X-Forwarded-Port 443 : évite que keycloak renvoie des URLs avec le port interne.

Les deux dernières des en-têtes autorisent la mise à niveau de la connexion, utile à l'admin console.

Nous avons sur le même bloc la réécriture des redirections, si keycloak renvoie des liens et redirections en HTPP ou avec le port interne, Nginx les réécrit en htpps://keycloak.internal/ ce qui empêchera les boucles, les cookies et les erreurs.

La dernière ligne nous avons les Timeouts qui évitent les coupures lors d'Operations lentes.

Dans notre dernier bloc on a :

L'écoute du vhost en http clair sur le port 80, son but est de capter les requêtes arrivant en http pour les rediriger. Ce vhost ne s'applique qu'au domaine keycloak.internal ce qui évite de rediriger par erreurs dans d'autres hôtes.

A la fin du bloc nous avons la redirection permanente de toute requête http vers la même URL en HTTPS. Ainsi nous avons forcé l'utilisation HTTPS pour keycloak.

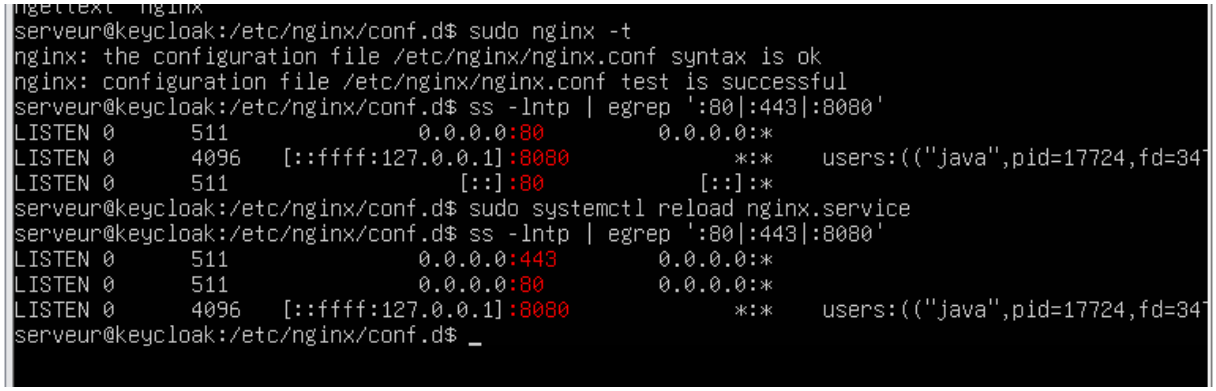


Figure 56: vérification des config

Sur l'image ci-dessus on fait un test de nos configurations avec l'option « nginx -t » nous remarquons que le test a réussi ce qui veut dire que nos configurations sont valides.

En suite la vérification des différents pares-feux que nous avons configurés via la commande ss -lntp :

:80 LISTEN 0.0.0.0 :80 Nous constatons que le http est actif sur notre serveur qui sert uniquement à rediriger vers HTTPS.

: 443 LISTEN 0.0.0.0 :443 : le HTTPS est actif donc notre certificat interne est chargé.

:8080 LISTEN 127.0.0.1 :8080 : Ce point est très important ce qui confirme l'isolation de notre serveur keycloak, le serveur écoute en locale uniquement pas exposer au réseau.

Notre reverse proxy est bien sécurisé tout passe par le serveur Nginx.

Dans ce premier rapport de l'implémentions d'une solution ZNA nous mis en place notre serveur d'identification et de gestion des accès avec ses services.

Pour se faire nous avons procédé l'installation de keycloak en mode développement cette installation nous a permis de comprendre rapidement les fonctionnalités de la palteforme. Ce mode est bien que simple pour mettre en œuvre, mais ça reste très limité en matière de sécurité car il utilise une base de données embarquée et fonctionne généralement en http sans protection du trafic.

Pour mettre en œuvre cette solution en entreprise il faut absolument passer en mode production, pour cela dans notre deuxième partie de notre travail on a migré le serveur du développement vers la production. Pour passer à la production nous avons remplacé la base de données H2 à

une base de données local robuste qui est le MariaDB/MYSQL. Nous avons stocké toutes les données sur cette base de données qui est le backend de notre serveur.

Et enfin pour sécuriser la plateforme nous avons mis en place un reverse proxy interne Nginx devant notre serveur qui assure les TLS interne, la réécriture des redirections, l'isolation totale du service keycloak en bloquant l'accès direct au port 8080, imposant ainsi un passage obligatoire par Nginx.

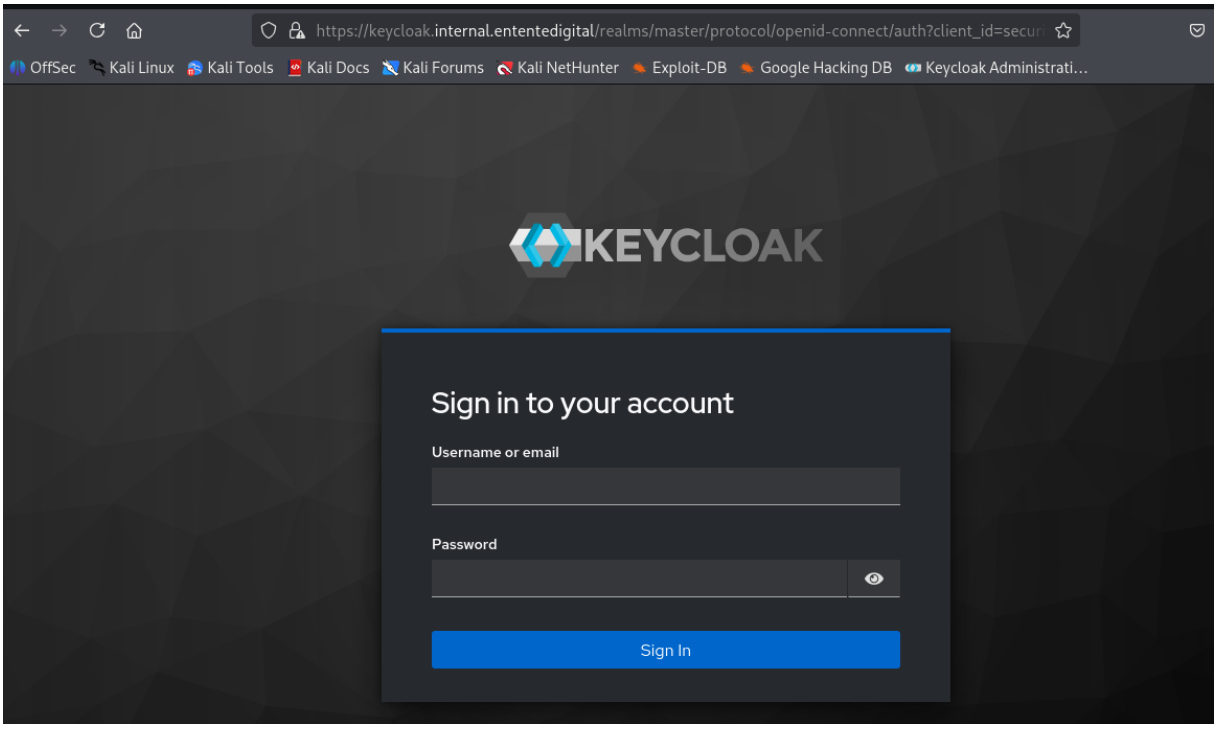


Figure 57: connexion https

3.3 Mise en œuvre du serveur web (Nginx)

Cette section décrit les différentes étapes de mise en place du **serveur web Nginx**, depuis la préparation de l'arborescence des applications jusqu'à la vérification de l'accès aux applications internes protégées par l'architecture ZTNA.

3.3.1 Création de l'arborescence des applications

La première étape consiste à créer l'arborescence nécessaire à l'hébergement de l'application RH dans le répertoire /var/www. Les dossiers assets, css, js et img sont créés afin d'organiser correctement les ressources statiques de l'application (feuilles de style, scripts JavaScript et images).

```
serveur@serveur-app:~$ sudo mkdir -p /var/www/hr/assets/{css,js,img}
[sudo] password for serveur:
serveur@serveur-app:~$
```

Figure 58:création de l'arborescence

La commande de visualisation de l'arborescence permet de confirmer la bonne création des dossiers pour les différentes applications internes (RH, Finance, Client). Cette organisation montre que chaque application dispose de son propre répertoire, tout en respectant une structure homogène.

```
serveur@serveur-app:/var/www/hr$ tree /var/www/
/var/www/
├── client
│   └── client.html
├── finance
│   ├── assets
│   │   ├── css
│   │   │   └── styles.css
│   │   ├── img
│   │   │   └── logo.png
│   │   └── js
│   │       ├── script.js
│   │       └── script.jsES
│   └── finance.html
├── hr
│   └── assets
│       ├── css
│       ├── img
│       └── js
└── html
    └── index.nginx-debian.html

13 directories, 7 files
serveur@serveur-app:/var/www/hr$ _
```

Figure 59: vérification de la structure de l'arborescence

3.3.2 Attribution des permissions sur le répertoire RH

Les permissions et le propriétaire du répertoire /var/www/rh sont modifiés afin de permettre au service Nginx (utilisateur www-data) d'accéder aux fichiers de l'application.

```
serveur@serveur-app:/var/www$ sudo chown -R www-data:www-data /var/www/hr
serveur@serveur-app:/var/www$ sudo chmod -R 755 /var/www/hr
serveur@serveur-app:/var/www$ _
```

Figure 60: attribution des droits aux répertoire

3.3.3 Création des fichiers applicatifs

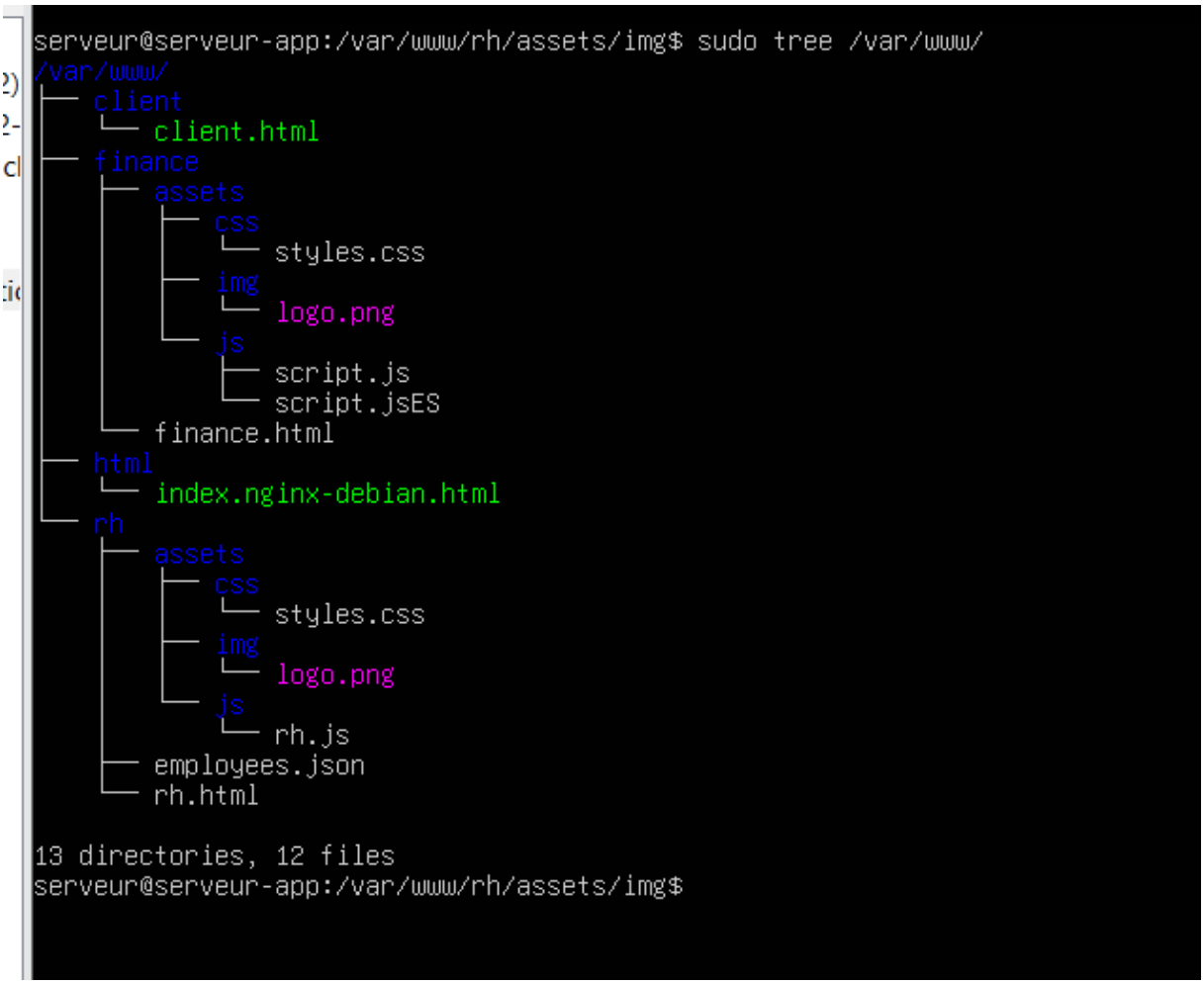
Les fichiers nécessaires au fonctionnement de l'application RH sont ensuite créés, notamment :

- Le fichier principal rh.html,
- Le fichier de données employees.json,
- Les scripts JavaScript associés.

```
serveur@serveur-app:/var/www/hr$ touch hr.html
touch: cannot touch 'hr.html': Permission denied
serveur@serveur-app:/var/www/hr$ sudo touch hr.html
serveur@serveur-app:/var/www/hr$ sudo touch employees.json
serveur@serveur-app:/var/www/hr$ sudo touch hr.js
serveur@serveur-app:/var/www/hr$ cp /var/www/finance/assets/css/styles.css /var/www/hr/assets/c
```

Figure 61: création des fichiers applicatifs

3.3.4 Structure des fichiers de l'application RH dans /var/www/rh



La figure ci-dessus présente l'arborescence complète des fichiers constituant l'application RH hébergée sur le serveur web Nginx. Cette organisation respecte les bonnes pratiques de développement web en séparant clairement les ressources selon leur fonction.

Répertoire racine /var/www/rh

Le répertoire /var/www/rh constitue la racine de l'application RH. Il contient :

- Le fichier principal de l'application (rh.html),
- Le fichier de données (employees.json),
- Le dossier assets regroupant toutes les ressources statiques.

Fichier rh.html

Le fichier rh.html contient le code HTML principal de l'application RH. Il définit :

- La structure de la page web,
- Les titres et sections de l'interface,
- Les tableaux d'affichage des employés,
- Les zones d'interaction utilisateur (recherche, actions, export).

Ce fichier fait appel aux ressources graphiques et fonctionnelles stockées dans le dossier assets, notamment les feuilles de style CSS et les scripts JavaScript.

Fichier employees.json

Le fichier employees.json contient les données des employés sous forme structurée (format JSON).
Il est utilisé par les scripts JavaScript pour :

- Charger dynamiquement la liste des employés,
- Afficher les informations (nom, poste, email),
- Simuler un comportement applicatif côté client.

Dossier assets

Le dossier assets regroupe l'ensemble des ressources statiques nécessaires au bon fonctionnement de l'application RH. Il est divisé en trois sous-dossiers distincts : css, img et js.

Dossier assets/css

Le dossier **css** contient les feuilles de style de l'application.

- **styles.css**

Ce fichier définit l'apparence graphique de l'application :

  - Couleurs,
  - Typographie,
  - Disposition des éléments,
  - Mise en page responsive.

Il permet de séparer la présentation visuelle du contenu HTML, conformément aux bonnes pratiques du développement web.

Dossier assets/img

Le dossier **img** contient les ressources graphiques utilisées par l'application.

- **logo.png**

Cette image correspond au **logo de l'application RH**.
Elle est intégrée dans le fichier rh.html afin d'identifier visuellement l'application et de renforcer son aspect professionnel.

Dossier assets/js

Le dossier **js** contient les scripts JavaScript de l'application.

- **rh.js**

Ce fichier implémente la logique fonctionnelle de l'application RH, notamment :

  - Le chargement des données depuis employees.json,
  - L'affichage dynamique du tableau des employés,
  - Les interactions utilisateur (recherche, filtrage, actions simulées).

L'utilisation de JavaScript permet de rendre l'application interactive sans recourir à une base de données côté serveur.

La figure montre également la présence d'autres répertoires tels que finance et client. Chaque application dispose de :

- Sa propre arborescence,
- Ses fichiers HTML,
- Ses ressources statiques.

Cette séparation garantit :

- Une isolation logique des applications,
- Une meilleure sécurité,
- Une intégration simple avec le proxy ZTNA.

L'organisation des fichiers de l'application RH respecte une structure claire et modulaire. Cette approche facilite l'hébergement multi-applications sur un même serveur Nginx tout en assurant une maintenance aisée. Elle s'intègre parfaitement dans l'architecture Zero Trust, où l'accès aux applications est contrôlé indépendamment du réseau sous-jacent.

3.3.5 Création du fichier de configuration Nginx

Un fichier de configuration dédié à l'application RH est créé dans le répertoire /etc/nginx/sites-available

```
serveur@serveur-app:/var/www/rh/assets/img$ sudo touch /etc/nginx/sites-available/hr.conf
serveur@serveur-app:/var/www/rh/assets/img$
```

Figure 62: création de fichier de configuration ngnix

Cette approche permet :

- ✓ Une configuration modulaire,
- ✓ Une gestion indépendante des applications,
- ✓ Une meilleure lisibilité de la configuration Nginx.

3.3.6 Configuration du virtual host Nginx pour l'application RH

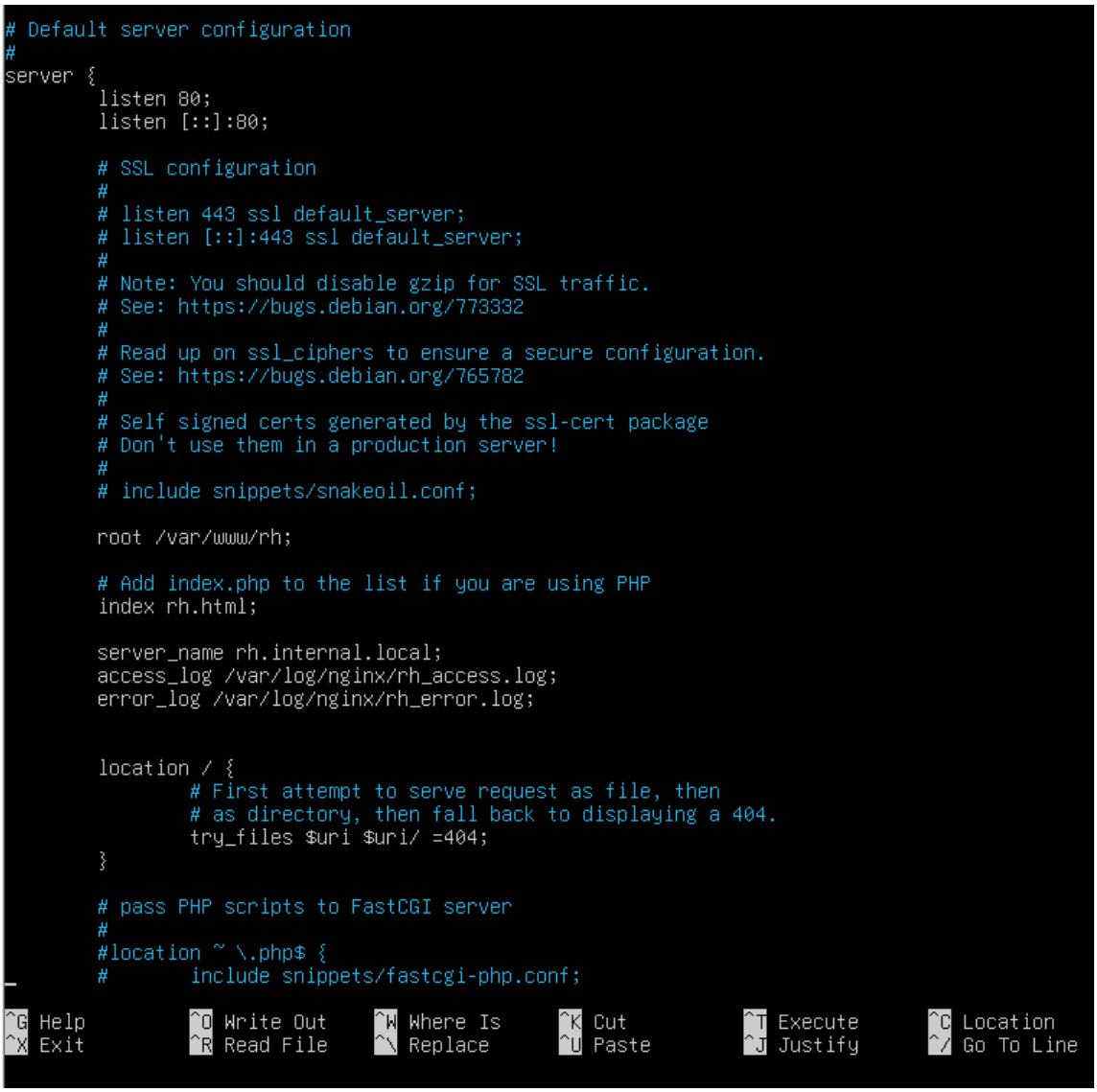


Figure 63; fichier de configuration nginx pour l'application rh

Le fichier de configuration définit :

- Le port d'écoute du serveur,
- Le répertoire racine (/var/www/rh),
- Le fichier d'index (rh.html),
- Le nom de domaine interne rh.internal.ententedigital,
- Les fichiers de logs d'accès et d'erreurs.

Cette configuration permet à Nginx de servir l'application RH uniquement via son nom de domaine interne, conformément au principe Zero Trust.

3.3.7 Activation du site RH dans Nginx

```
serveur@serveur-app:/etc/nginx/sites-available$ sudo ln -s /etc/nginx/sites-available/rh.conf /etc/nginx/sites-enabled/_
```

Figure 64: activation du site web

Le site est activé à l'aide d'un lien symbolique entre sites-available et sites-enabled. Cette méthode est recommandée par les bonnes pratiques Nginx, car elle permet d'activer ou désactiver un site sans supprimer sa configuration.

3.3.8 Accès à l'application RH via le navigateur

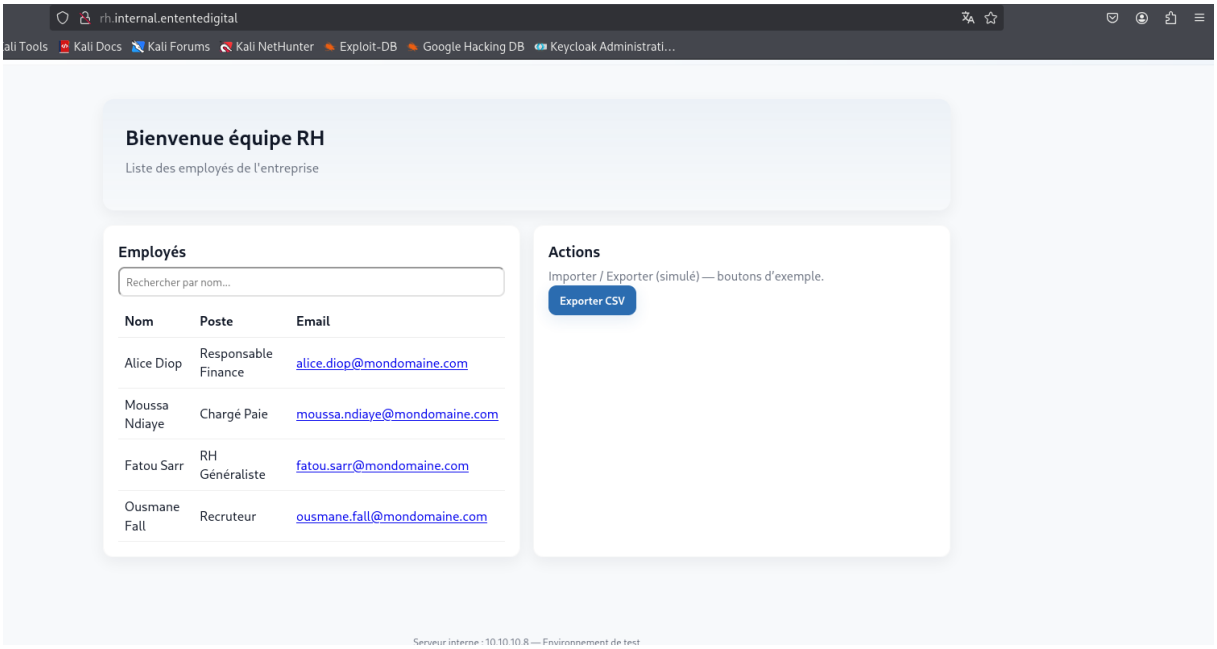


Figure 65: test d'accès à l'application rh

L'accès à l'application RH est testé via le nom de domaine interne. L'interface de l'application s'affiche correctement, confirmant :

- ✓ La bonne configuration de Nginx,
- ✓ La disponibilité de l'application,
- ✓ Le bon déploiement des ressources statiques.

3.3.9 Accès à l'application Finance

Une vérification similaire est effectuée pour l'application Finance, démontrant que plusieurs applications internes peuvent être hébergées sur le même serveur web, tout en restant logiquement séparées et protégées par la couche ZTNA.

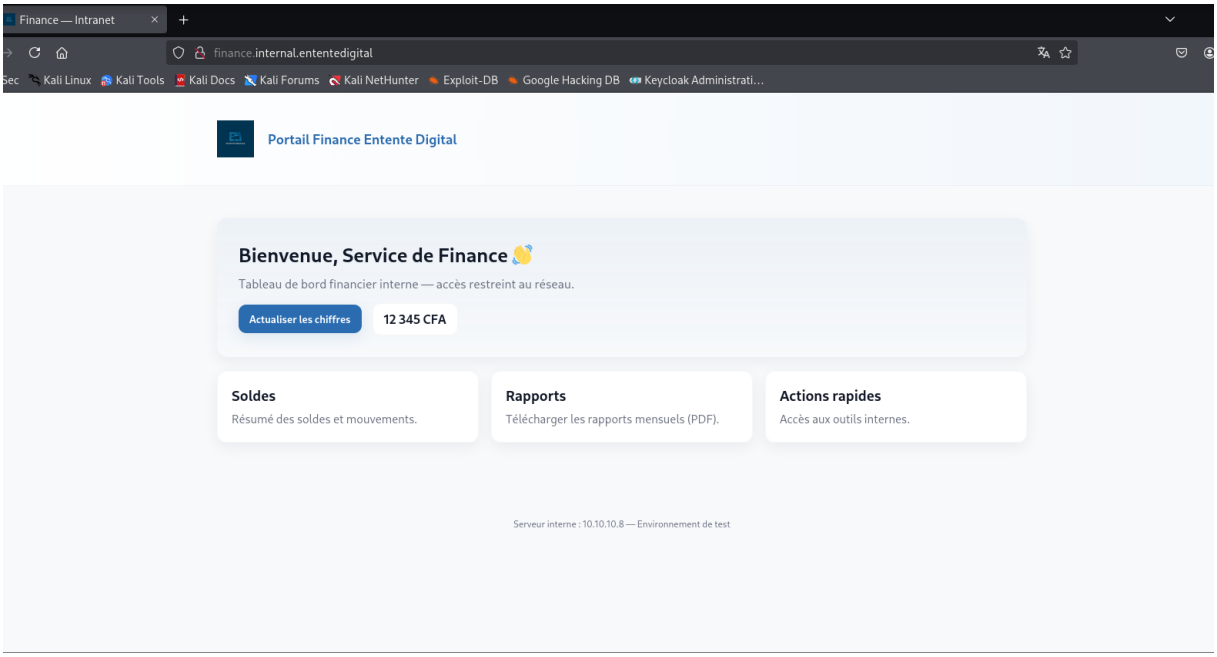


Figure 66: l'accès à l'application de finance

La mise en place du serveur web Nginx permet d'héberger efficacement les applications internes de l'entreprise. En isolant les applications dans des répertoires dédiés et en utilisant des virtual hosts distincts, cette architecture facilite l'intégration avec le proxy Zero Trust (Pomerium) et garantit que les applications ne sont jamais exposées directement aux utilisateurs.

3.4 Mise en œuvre du reverse proxy Zero Trust Pomerium

L'installation et la configuration initiale de Pomerium constituent une étape clé de l'architecture Zero Trust. Pomerium devient le point d'entrée unique pour les applications internes et garantit que :

- Aucune requête n'atteint directement les applications,
- Toute demande d'accès sera soumise à une authentification et une autorisation explicite,
- La confiance implicite est totalement éliminée.

3.4.1 Installation de Pomerium

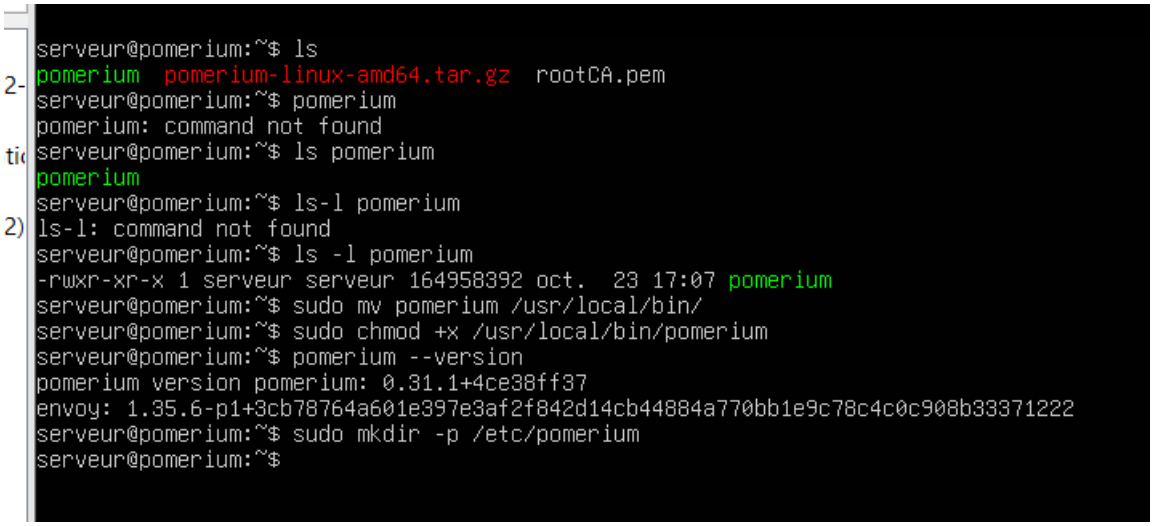


Figure 67:installation pomerium

La figure montre les différentes étapes de l'installation de Pomerium sur le serveur dédié. Dans un premier temps, le binaire de Pomerium est téléchargé sous forme d'archive (pomerium-linux-amd64.tar.gz), puis extrait afin d'obtenir l'exécutable pomerium.

Après extraction, le binaire est déplacé vers le répertoire système /usr/local/bin, ce qui permet de rendre la commande pomerium accessible globalement depuis le terminal.

Après l'installation, une vérification est effectuée en exécutant la commande de version. Le système retourne correctement la version de Pomerium installée, ce qui confirme que le binaire est fonctionnel et reconnu par le système.

Un répertoire dédié /etc/pomerium est créé afin de centraliser les fichiers de configuration de Pomerium. Cette approche respecte les conventions Linux et facilite la gestion des configurations système.

Le fichier principal de configuration config.yaml est ensuite placé dans ce répertoire. Ce fichier contiendra :

- Les paramètres réseau,
- La configuration de l'authentification (OIDC),
- Les règles de routage,
- Les politiques d'accès Zero Trust.

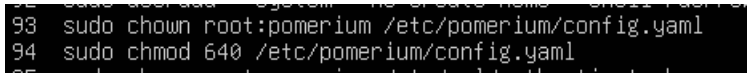


Figure 68: sécurisation du fichier conf

Les droits d'accès au fichier config.yaml sont configurés de manière restrictive :

- Le propriétaire est défini comme root:pomerium,
- Les permissions sont limitées à la lecture pour le groupe,
- L'accès est interdit aux autres utilisateurs.

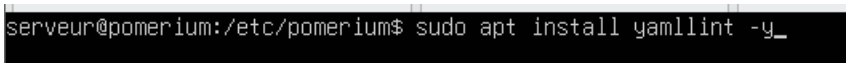


Figure 69: installation de yamllint

Afin d'éviter toute erreur de syntaxe dans les fichiers de configuration de Pomerium, l'outil **yamllint** est installé sur le serveur. Cet utilitaire permet de vérifier la structure et la conformité des fichiers YAML avant leur utilisation.

L'usage de yamllint est particulièrement pertinent dans le cadre de Pomerium, car :

- ✓ La configuration repose entièrement sur des fichiers YAML,
- ✓ Une simple erreur d'indentation peut empêcher le démarrage du service,
- ✓ Il améliore la fiabilité et la robustesse du déploiement.

3.4.2 Génération des certificats TLS pour les domaines internes

Dans cette architecture, mkcert a été installé initialement sur le serveur Keycloak, qui joue le rôle de point central d'authentification.
Les certificats TLS ont été générés sur ce serveur, puis exportés vers le serveur Pomerium afin d'assurer une chaîne de confiance cohérente entre les composants.

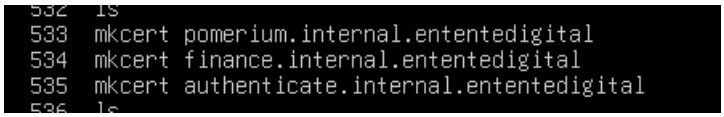


Figure 70: génération de certificat TLS avec mkcert

Les certificats TLS ont été générés à l'aide de l'outil **mkcert**, installé sur le serveur Keycloak.
Les domaines internes suivants ont été signés :

- pomerium.internal.ententedigital
- finance.internal.ententedigital
- authenticate.internal.ententedigital

3.4.2.1 Vérification des certificats TLS générés



Figure 71: vérification des certificats

Le capture ci-dessus montre le répertoire personnel du serveur Keycloak contenant plusieurs fichiers de certificats (.pem) et de clés privées (-key.pem).
Ces certificats correspondent aux différents services internes de l'architecture, notamment :

- authenticate.internal.ententedigital
- keycloak.internal.ententedigital
- pomerium.internal.ententedigital
- rh.internal.ententedigital

- finance.internal.ententedigital

Cette organisation confirme que les certificats ont été générés de manière centralisée à partir de l'autorité de certification locale créée avec mkcert.

3.4.2.2 Copie des certificats dans /etc/ssl



Figure 72: copie vers répertoire ssl

Une copie est ensuite effectuée dans le répertoire /etc/ssl afin de s'assurer de la présence :

- ✓ Des certificats TLS,
- ✓ Des clés privées associées,
- ✓ Du certificat de l'autorité de certification locale (rootCA.pem).

3.4.2.3 Transfert sécurisé des certificats vers le serveur Pomerium



Figure 73: transfert vers le serveur pomerium

Les certificats nécessaires au fonctionnement de Pomerium sont transférés depuis le serveur Keycloak vers le serveur Pomerium à l'aide de la commande scp.

Une fois déployés sur le serveur Pomerium, ces certificats sont utilisés pour :

- ➢ Activer le chiffrement HTTPS,
- ➢ Sécuriser les communications avec les utilisateurs,
- ➢ Garantir l'authenticité des services internes,
- ➢ Protéger les flux d'authentification vers Keycloak.

Pomerium référence ensuite ces certificats dans son fichier de configuration afin d'assurer un accès chiffré et fiable aux applications internes.

3.4.2.4 Intégration de l'autorité de certification locale



Figure 74: intégration de l'autorité de certificat local

Afin que le serveur Pomerium reconnaisse comme valides les certificats TLS générés par mkcert, le certificat racine (rootCA.pem) est copié dans le magasin système des autorités de certification /usr/local/share/ca-certificates.

3.4.2.5 Mise à jour du magasin des certificats système

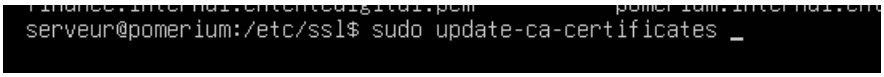


Figure 75: mise à jour du magasin

Après l'ajout du certificat racine (rootCA.pem) dans le magasin des autorités de certification locales, la commande update-ca-certificates est exécutée.
Cette opération permet au système d'exploitation de :

- Prendre en compte le nouveau certificat racine,
- Mettre à jour la base des autorités de confiance,
- Rendre les certificats internes valides pour tous les services du système.

Cette étape est indispensable pour garantir que Pomerium et les autres services reconnaissent les certificats TLS générés par mkcert.

3.4.2.6 Vérification de l'autorité de certification

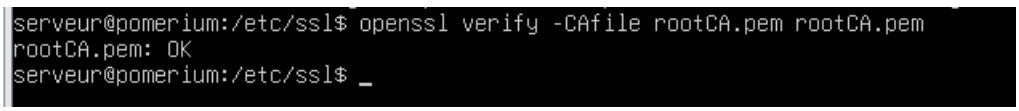


Figure 76: vérification de l'autorité

Une vérification explicite du certificat racine est réalisée à l'aide de la commande openssl verify.
Le résultat retourné confirme que le certificat racine est valide et reconnu comme autorité de certification de confiance.

3.4.3 Génération d'un secret cryptographique partagé

```
serveur@pomerium:/etc/ssl$ sudo yamllint /etc/pomerium/config.yaml
serveur@pomerium:/etc/ssl$ python3 -c "import base64, os; print(base64.urlsafe_b64encode(os.urandom(32)).decode())"
EjanPFnNX4Te8eLZ8qPkMxXfe4lWMyhtcEviKLMm9LM=
serveur@pomerium:/etc/ssl$ _
```

Figure 77: génération d'une clé secrète

Un **secret partagé** est généré à l'aide d'un script Python utilisant le module os.urandom. Le résultat est encodé en Base64 afin d'être compatible avec la configuration de Pomerium.

Ce secret est utilisé par Pomerium pour :

- ✓ Sécuriser les sessions,
- ✓ Signer et vérifier les cookies d'authentification,
- ✓ Protéger les échanges internes entre ses composants.

L'utilisation d'un secret aléatoire fort garantit un haut niveau de sécurité cryptographique.

3.4.4 Configuration de Keycloak pour l'intégration avec Pomerium

Cette section décrit la configuration du serveur **Keycloak** afin de le lier au proxy **Pomerium** en tant que **fournisseur d'identité OpenID Connect (OIDC)**. L'objectif est de permettre à Pomerium de déléguer l'authentification des utilisateurs à Keycloak et d'obtenir les informations nécessaires à la prise de décision d'accès.

3.4.4.1 Création du client OIDC Pomerium

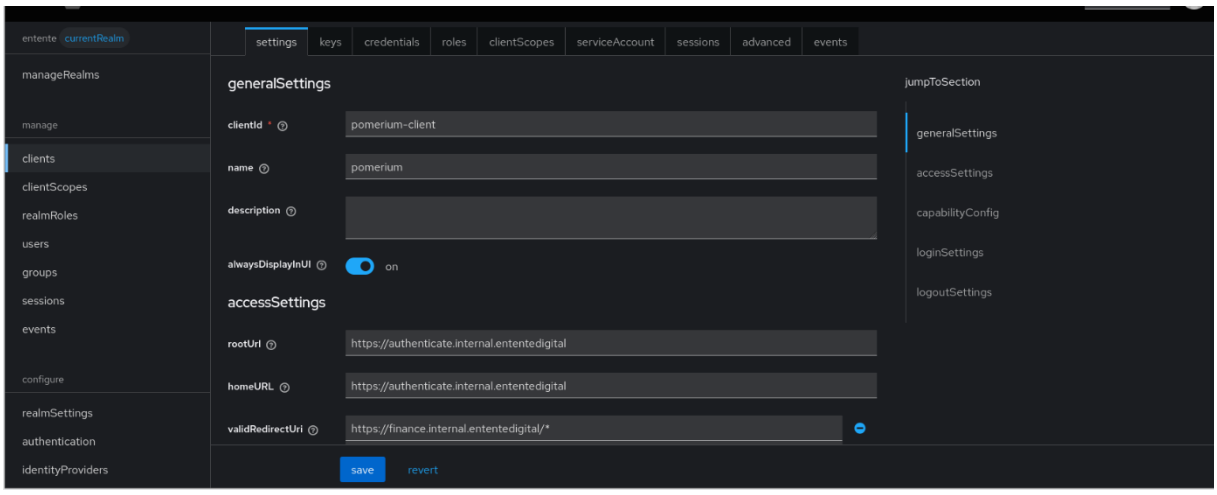


Figure 78: création pomerium-client

La première capture montre la création d'un nouveau client dans le realm **entente**.
Le client est configuré avec les paramètres suivants :

- **Client ID** : pomerium-client
- **Nom** : pomerium
- **Type de client** : OpenID Connect

Ce client représente **Pomerium** dans Keycloak et permet l'échange de tokens d'authentification selon le protocole OIDC.

L'option *Always Display in UI* est activée afin de rendre le client visible et identifiable dans l'interface d'administration.

3.4.4.2 Configuration des URLs d'accès et de redirection

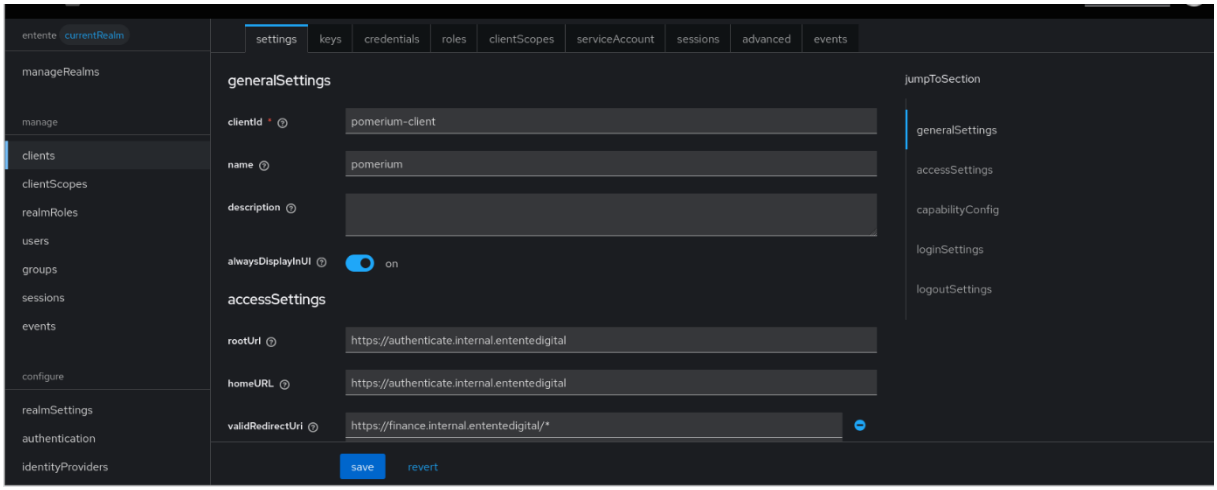


Figure 79: configuration des URLs d'accès et redirection

Les paramètres d'accès sont définis afin d'assurer le bon déroulement du flux d'authentification :

- **Root URL** : https://authenticate.internal.ententedigital
- **Home URL** : https://authenticate.internal.ententedigital
- **Valid Redirect URIs** : https://finance.internal.ententedigital/*

Ces URLs permettent à Keycloak :

- De rediriger l'utilisateur vers Pomerium après authentification,
- D'accepter uniquement des redirections provenant de domaines autorisés,

- D’empêcher toute redirection malveillante.

Cette restriction des URLs est essentielle dans un contexte Zero Trust.

3.4.4.3 Configuration du logout et des origines web

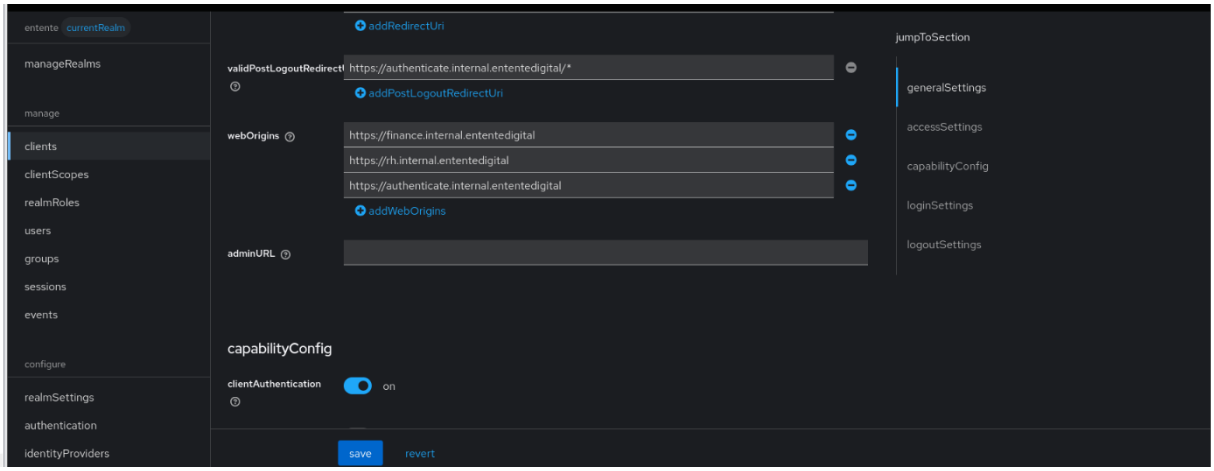


Figure 80: Configuration logout et web origin

Les paramètres de déconnexion sont configurés comme suit :

Valid Post Logout Redirect URIs :
https://authenticate.internal.ententedigital/*

Les **Web Origins** autorisées incluent :

- https://finance.internal.ententedigital
- https://rh.internal.ententedigital
- https://authenticate.internal.ententedigital

Cette configuration permet :

- La gestion correcte des sessions utilisateur,
- La prévention des erreurs CORS,
- Une communication sécurisée entre Keycloak, Pomerium et les applications internes.

3.4.4.4 Activation de l'authentification client

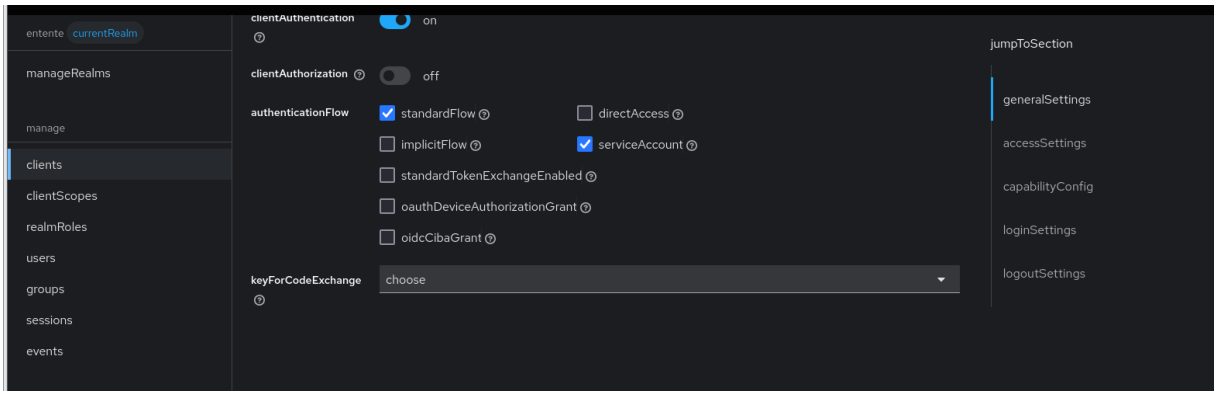


Figure 81: Activation de l'authentification

L'option **Client Authentication** est activée.

Cela signifie que Pomerium doit s'authentifier auprès de Keycloak à l'aide d'un **client secret**.

3.4.4.5 Génération du *Client Secret*

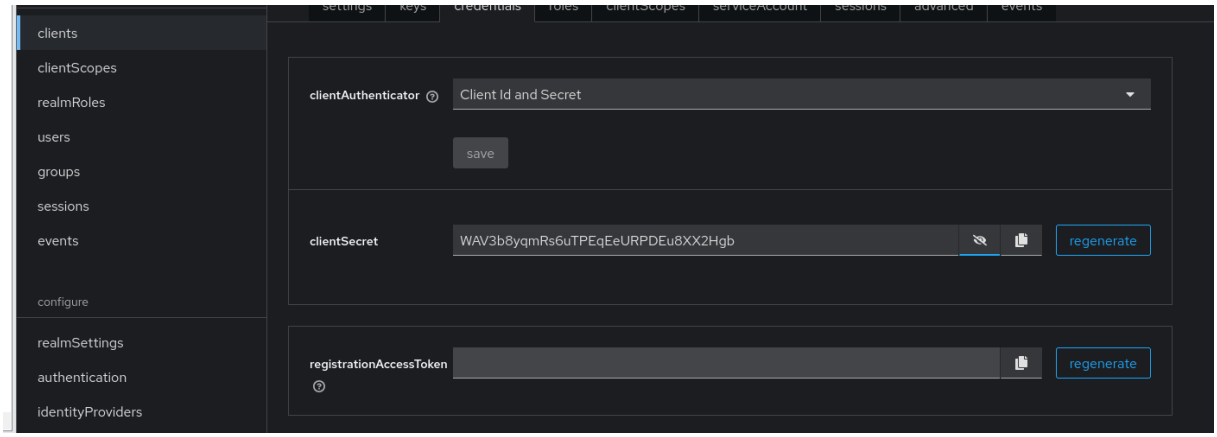


Figure 82: génération de client-id

Lors de cette étape, un client OAuth2 est configuré dans Keycloak afin de permettre l'intégration avec Pomerium. Le mécanisme d'authentification du client repose sur l'utilisation d'un Client ID et d'un Client Secret. Le secret généré par Keycloak permet à Pomerium de s'authentifier de manière sécurisée auprès du serveur d'identité lors des échanges de jetons. Ce secret sera ensuite utilisé dans le fichier de configuration de Pomerium afin d'établir une relation de confiance entre les deux services.

3.4.4.6 Création du *Mapper Groups*

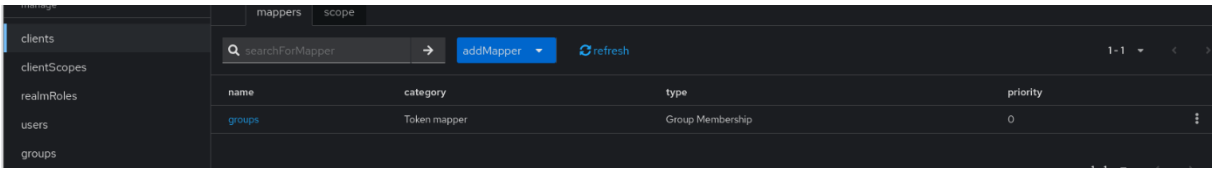


Figure 83: création du mapper

Ensuite, un mapper de type Group Membership est créé au niveau du client. Ce mapper a pour rôle d'inclure automatiquement les groupes auxquels appartient l'utilisateur dans le token JWT généré par Keycloak. Grâce à ce mécanisme, les informations de groupes sont intégrées dans le jeton d'identité et peuvent être exploitées par Pomerium pour appliquer les politiques d'accès basées sur les rôles et les groupes.

3.4.4.7 Création des groupes

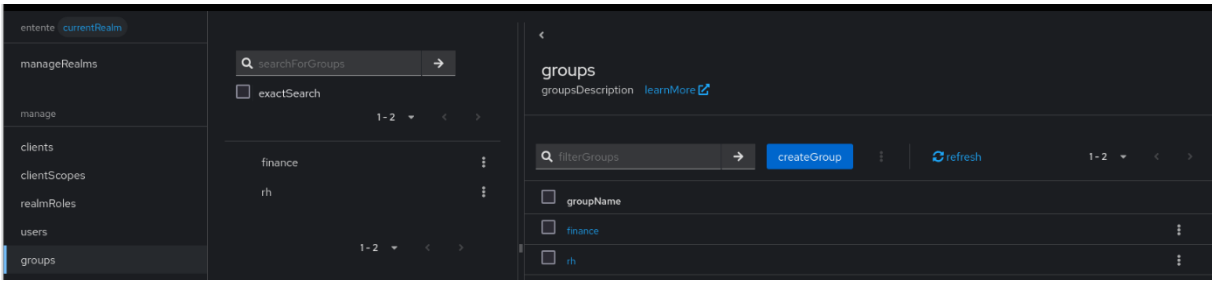


Figure 84: création group

Après cela, des **groupes métiers** sont créés dans Keycloak afin de représenter les différentes catégories d'utilisateurs de l'entreprise. Dans ce contexte, les groupes *finance* et *rh* sont définis pour correspondre respectivement aux applications internes de gestion financière et de ressources humaines. Cette organisation permet de structurer les droits d'accès selon les besoins fonctionnels de chaque service.

3.4.4.8 Création des utilisateurs

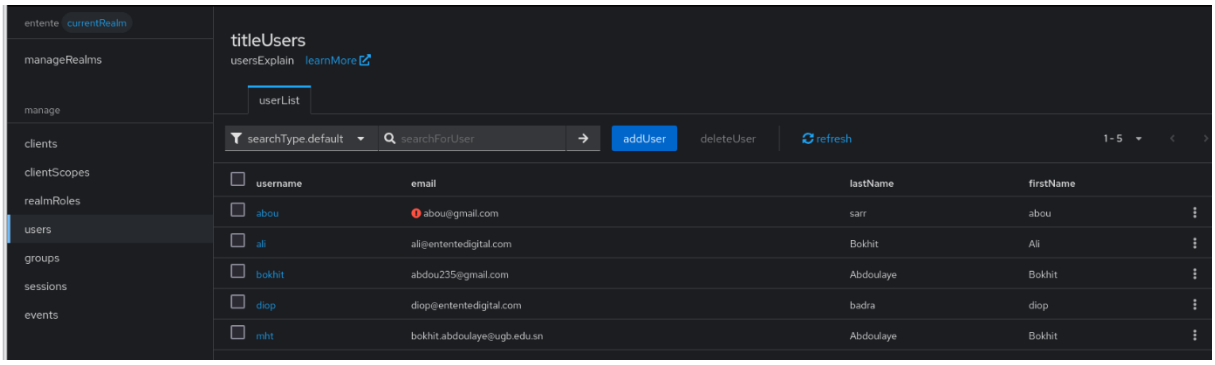


Figure 85: création des utilisateurs

Une fois les groupes créés, les **utilisateurs** sont ajoutés dans Keycloak. Chaque utilisateur représente une identité numérique unique avec ses informations personnelles telles que le nom, le prénom et l'adresse email. Ces comptes utilisateurs constituent la base d'authentification centralisée pour l'ensemble des applications protégées par Pomerium.

3.4.4.9 Affectation des utilisateurs aux groupes

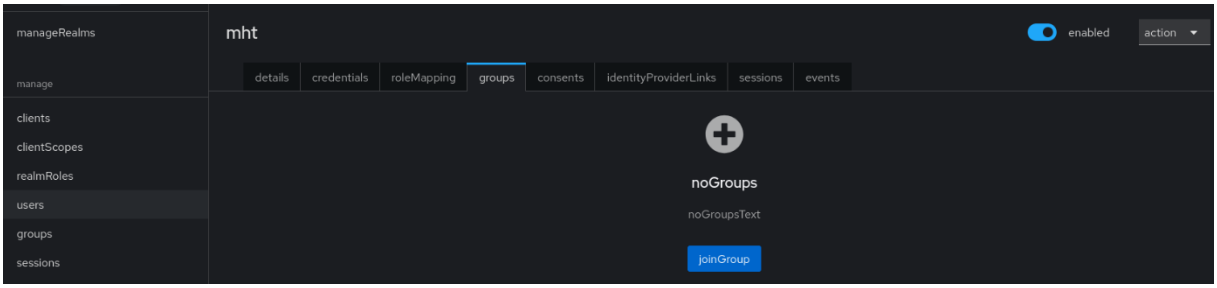


Figure 86: Affectation des users aux groups

Les utilisateurs sont **affectés aux groupes appropriés**. Cette association détermine concrètement les autorisations d'accès. Lorsqu'un utilisateur se connecte, Keycloak inclut le groupe auquel il appartient dans le token d'identité. Pomerium analyse ensuite ce token et autorise ou refuse l'accès aux applications internes en fonction du groupe détecté. Ce mécanisme garantit une gestion des accès fine, dynamique et conforme au modèle **Zero Trust Network Access (ZTNA)**.

3.4.4.10 Vérification des groups

Chaque utilisateur listé représente une identité authentifiée au sein du royaume Keycloak. Leur présence dans le groupe *finance* signifie qu'ils héritent automatiquement des droits associés à ce groupe. Ces informations d'appartenance sont intégrées dans le token d'identité généré lors de l'authentification.

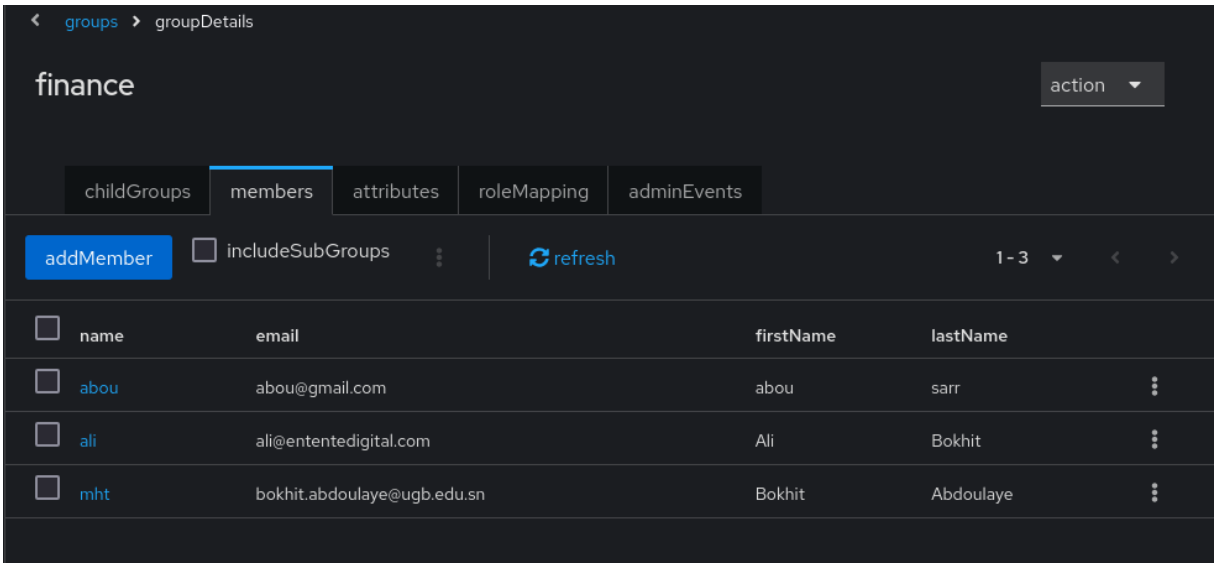


Figure 87: vérification des groups

3.4.5 Fichier de configuration pomerium



Figure 88:fichier de configuration pomerium

Ce fichier correspond au fichier de configuration principal de Pomerium, dans lequel sont définis les paramètres globaux de sécurité, l'intégration avec Keycloak ainsi que les règles de contrôle d'accès aux applications internes.

Partie écoute et TLS

address: ":443"

Cette ligne indique que Pomerium écoute les connexions HTTPS sur le port 443, qui est le port standard pour les communications web sécurisées.

certificate_file: /etc/ssl/pomerium.internal.ententedigital.pem

Cette directive spécifie le certificat TLS utilisé par Pomerium pour chiffrer les communications entrantes.

certificate_key_file: /etc/ssl/pomerium.internal.ententedigital-key.pem

Cette ligne définit la clé privée associée au certificat TLS, nécessaire pour établir une connexion sécurisée.

Service d'authentification

authenticate_service_url: https://authenticate.internal.ententedigital
Cette URL correspond au service d'authentification de Pomerium, chargé de rediriger les utilisateurs vers le fournisseur d'identité.

Intégration avec Keycloak (OIDC)

idp_provider: oidc : Cette ligne précise que le fournisseur d'identité utilisé est basé sur le protocole OpenID Connect (OIDC).

idp_provider_url: https://keycloak.internal.ententedigital/realms/entente: Cette URL indique l'endpoint OIDC du royaume Keycloak utilisé pour l'authentification des utilisateurs.

idp_client_id: pomerium-client : Cette directive correspond à l'identifiant du client OIDC créé dans Keycloak et utilisé par Pomerium.

idp_client_secret: WAV3b8yqmRs6uTPEqEeURPDEu8XX2Hgb : Cette valeur représente le secret du client OIDC, utilisé pour sécuriser la communication entre Pomerium et Keycloak.

certificate_authority_file: /etc/ssl/rootCA.pem : Cette ligne indique l'autorité de certification de confiance utilisée pour valider les certificats TLS internes.

shared_secret: EjanPFNx4Te8L2g8nKMxXFe4TMJwhtCEViKLNm9LM= : Ce secret partagé est utilisé par Pomerium pour signer et vérifier les cookies de session, garantissant leur intégrité.

Définition des routes protégées

id: finance : Cette ligne définit un identifiant unique pour la route correspondant à l'application Finance.

from: https://finance.internal.ententedigital: Cette directive précise l'URL externe par laquelle les utilisateurs accèdent à l'application Finance.

to: http://10.10.10.8:94: Cette ligne indique l'adresse interne réelle de l'application Finance hébergée sur le réseau interne.

pass_identity_headers: true : Cette option permet à Pomerium de transmettre les informations d'identité de l'utilisateur à l'application cible via des en-têtes HTTP.

policy: Cette section définit les règles d'autorisation applicables à cette route.

- allow: Cette ligne indique que la politique autorise l'accès sous certaines conditions.

or: Cette directive précise que l'accès est accordé si au moins une des conditions listées est satisfaite.

- claim/groups: finance : Cette règle autorise l'accès uniquement aux utilisateurs dont le token contient l'appartenance au groupe *finance*.

3.4.6 Simulation de l'accès à l'application Finance protégée par ZTNA

Dans cette phase de test, nous simulons l'accès d'un utilisateur à l'application Finance, protégée par la solution Zero Trust Network Access mise en place avec Pomerium et Keycloak.

Connexions via le navigateur

Tout d'abord, l'utilisateur ouvre son navigateur web et saisit l'URL de l'application Finance : https://finance.internal.ententedigital.
Cette requête HTTPS est automatiquement interceptée par Pomerium, qui agit comme point d'entrée sécurisé vers les applications internes.

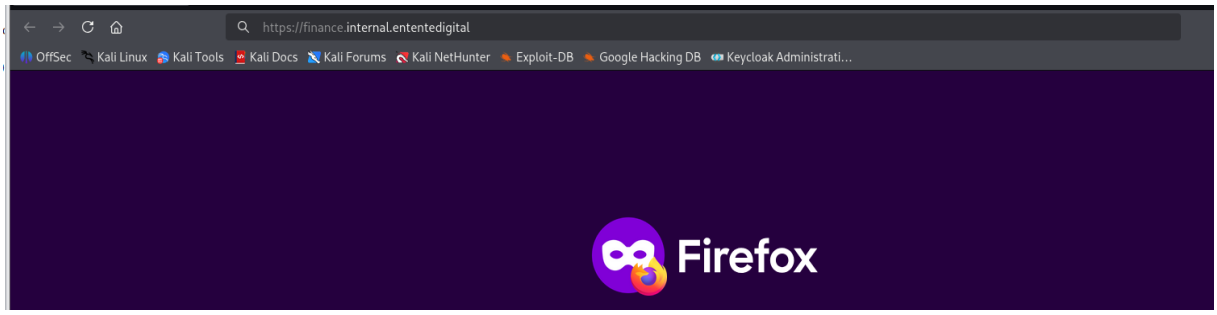


Figure 89: connexion à l'app finance

Redirection de la connexion vers pomerium

Étant donné que l'utilisateur n'est pas encore authentifié, Pomerium bloque temporairement l'accès et redirige automatiquement l'utilisateur vers le service d'authentification configuré, à savoir **Keycloak**, via le protocole **OpenID Connect**.

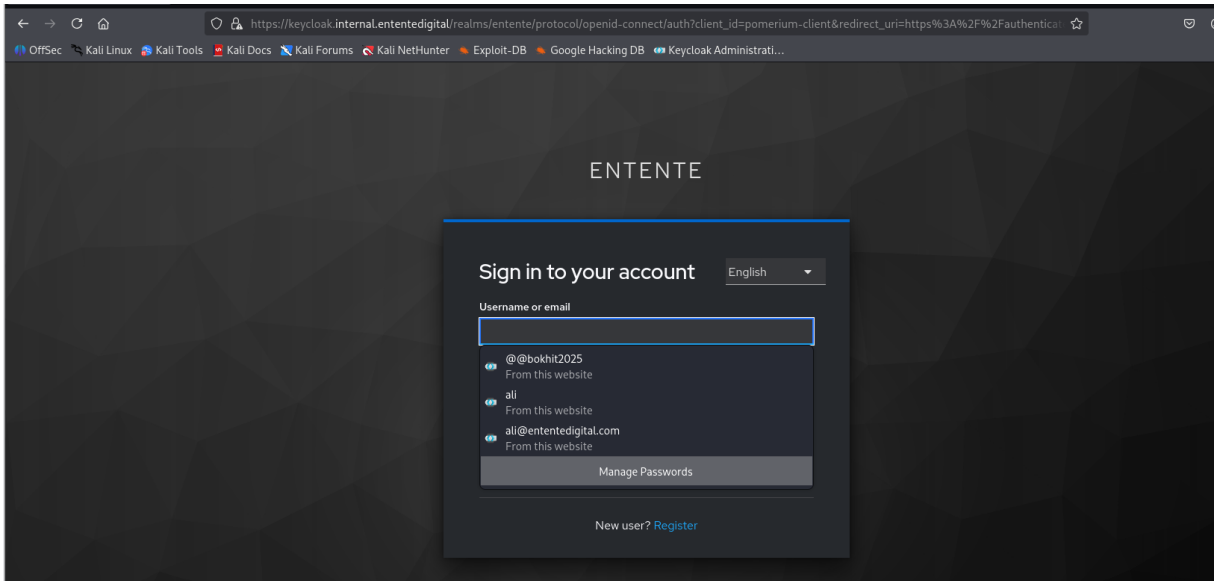


Figure 90: redirection vers keycloak

Authentification de l'utilisateur

L'utilisateur est alors présenté avec la page de connexion du royaume ENTENTE sur Keycloak. Sur cette interface, il saisit ses identifiants (nom d'utilisateur ou adresse e-mail ainsi que son mot de passe). Dans ce scénario de test, l'utilisateur ali@ententedigital.com est utilisé.

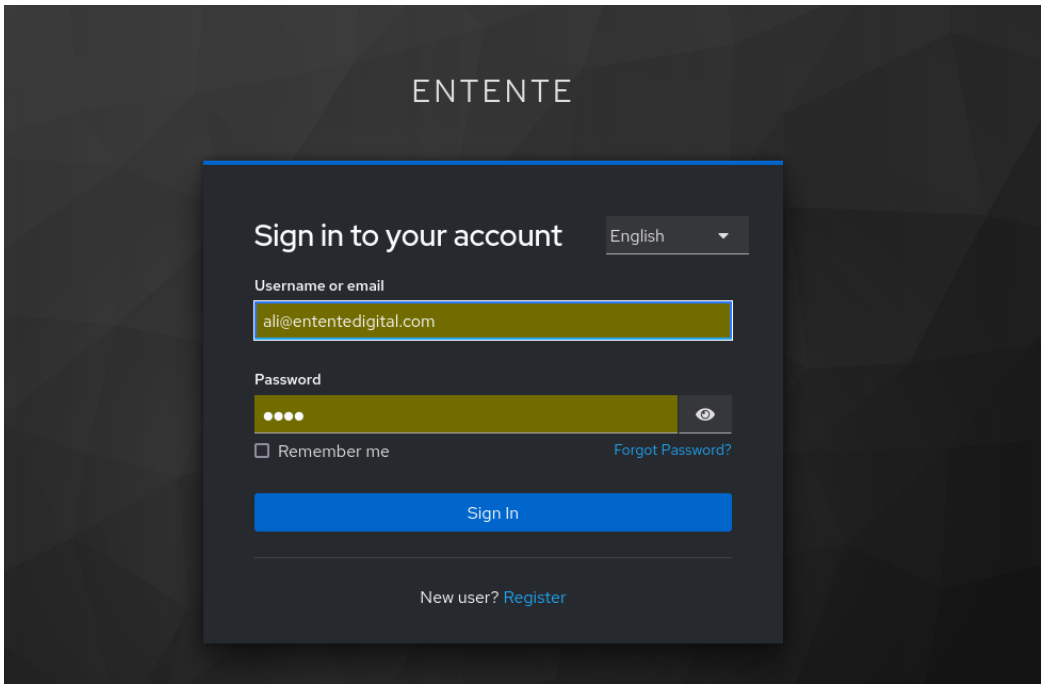


Figure 91: authentification de l'utilisateur

Après la validation des identifiants, l'authentification de l'utilisateur par Keycloak ainsi que la génération du token d'identité signé (JWT), contenant les informations de l'utilisateur et son

appartenance aux groupes, s'effectuent de manière totalement transparente et invisible pour l'utilisateur final.

De la même manière, le renvoi de ce token vers Pomerium, sa vérification (validité, intégrité) ainsi que l'évaluation des règles de sécurité définies dans le fichier de configuration, notamment la comparaison du groupe présent dans le token avec la politique d'accès associée à la route *finance*, sont réalisées en arrière-plan, sans aucune interaction ni visibilité pour l'utilisateur.

Utilisateur connecté après l'application des policy

Après une authentification réussie via Keycloak et la validation des politiques d'accès par Pomerium, l'utilisateur est automatiquement redirigé vers l'application Finance, où l'interface du portail financier interne s'affiche correctement, confirmant que l'accès a été autorisé conformément à son appartenance au groupe *finance*.

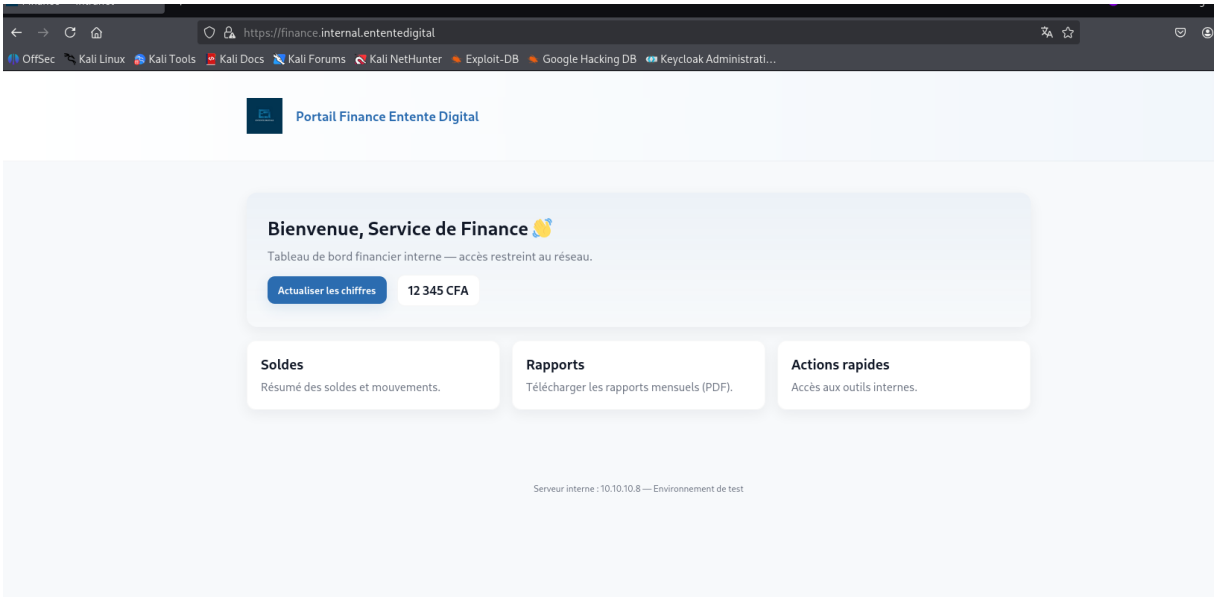


Figure 92: connexion réussie

3.4.7 Simulation de l'accès à l'application RH protégée par la solution ZTNA

Dans cette phase de test, nous simulons l'accès d'un utilisateur à l'application RH, laquelle est protégée par l'architecture Zero Trust Network Access reposant sur Pomerium et Keycloak.

Connexion via le navigateur web

L'utilisateur ouvre tout d'abord son navigateur web et saisit l'URL de l'application RH : https://rh.internal.ententedigital.

Comme pour les autres applications internes, cette requête HTTPS est interceptée par Pomerium, qui agit comme point d'entrée sécurisé.

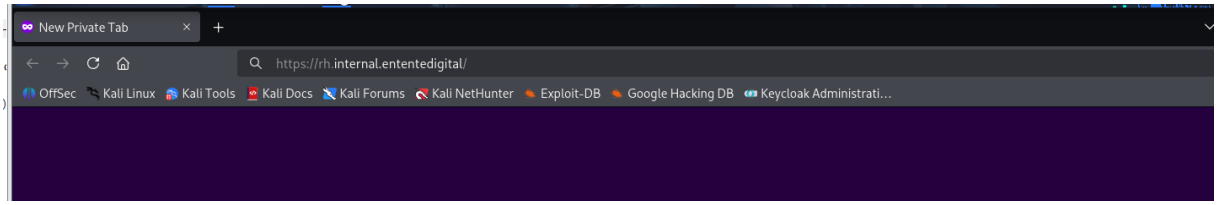


Figure 93: connexion à l'app rh

Authentification rh

Sur l'interface de connexion du royaume ENTENTE, l'utilisateur saisit ses identifiants. Dans ce scénario, l'utilisateur diop est utilisé, lequel est préalablement enregistré dans Keycloak et associé au groupe rh.

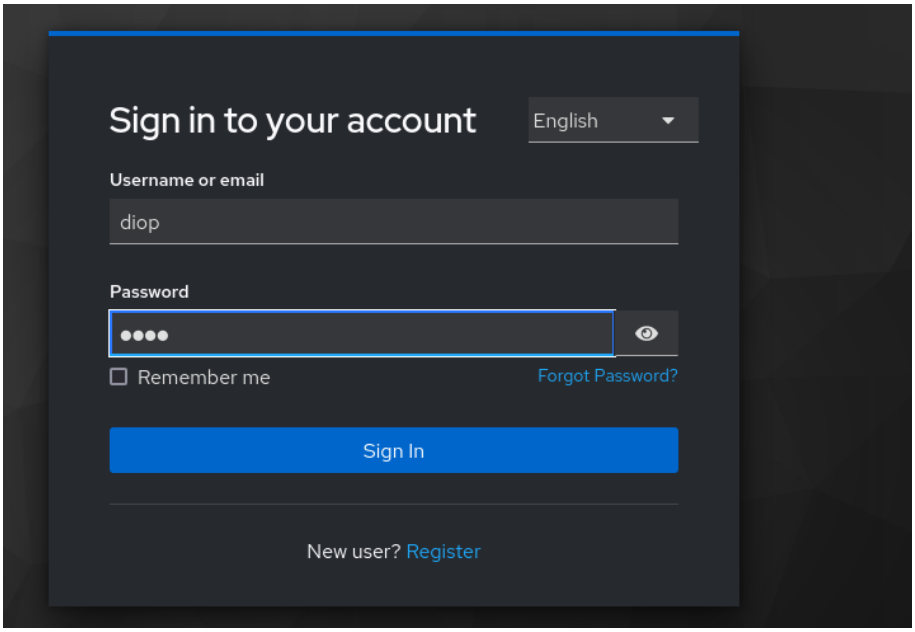


Figure 94: authentification rh

Utilisateur connecté après l'application rh des policy

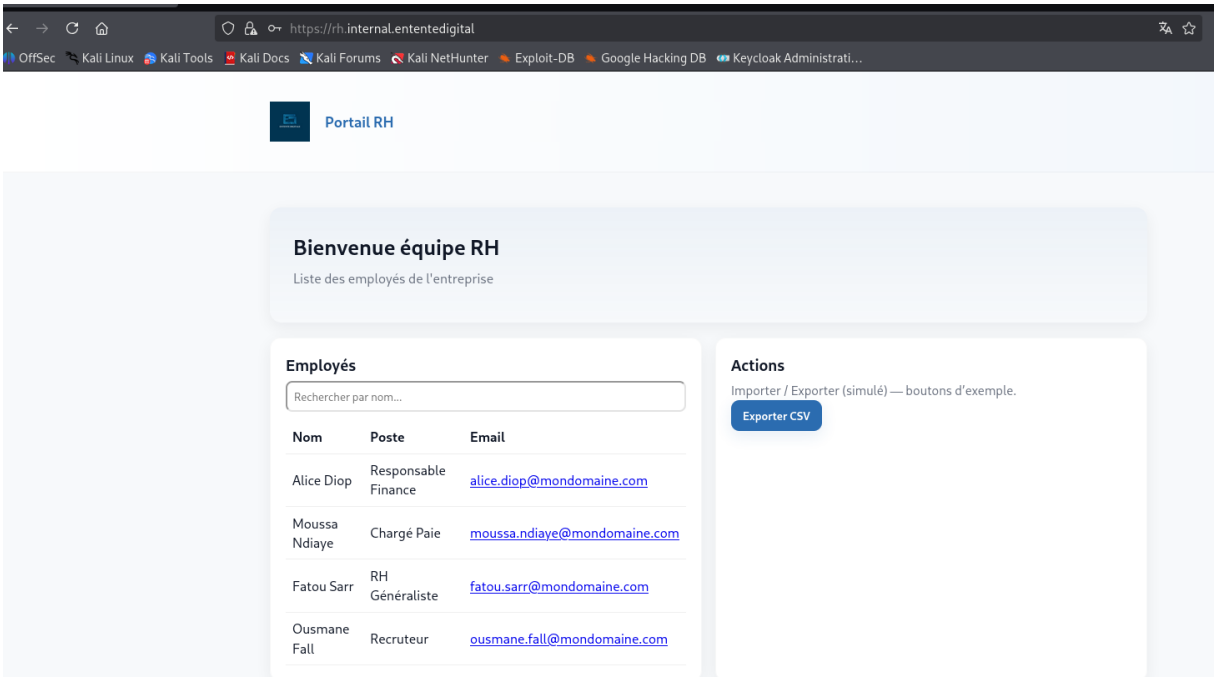


Figure 95: connexion réussie l'app rh

Comme l'utilisateur appartient bien au groupe **rh**, la politique d'accès est respectée. Pomerium autorise alors la requête et la transmet vers l'application RH hébergée sur le serveur interne via **Nginx**.

L'utilisateur est finalement redirigé vers l'interface de l'application RH, où le portail RH s'affiche correctement, confirmant que l'accès a été accordé conformément aux règles définies, sans exposition directe du réseau interne.

3.4.8 Simulation d'accès refusé avec l'application RH

Lorsqu'un utilisateur ne disposant pas de l'appartenance au groupe requis tente d'accéder à une application interne, Pomerium rejette automatiquement la requête après vérification du token d'identité, et l'accès est refusé conformément aux politiques de sécurité définies, sans exposer l'application ni le réseau interne.

Connexion avec mauvais identifiant

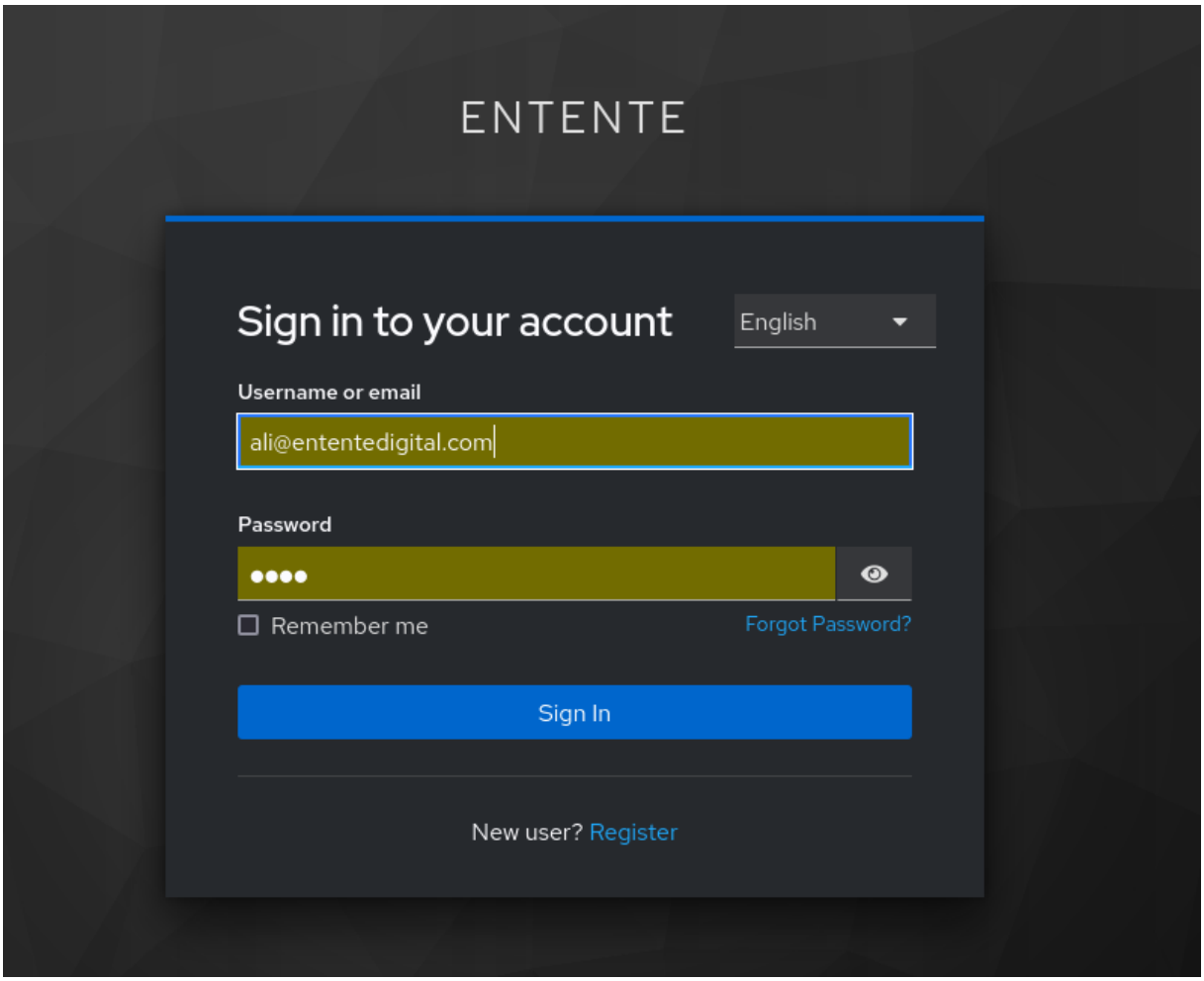


Figure 96: connexion à l'app rh avec mauvais identifiant

L'utilisateur tente d'accéder à l'application **RH** en utilisant des identifiants appartenant au **groupe finance**, ce qui signifie qu'il est correctement authentifié par Keycloak mais ne dispose pas des autorisations requises pour accéder à cette application spécifique.

Connexion refusée

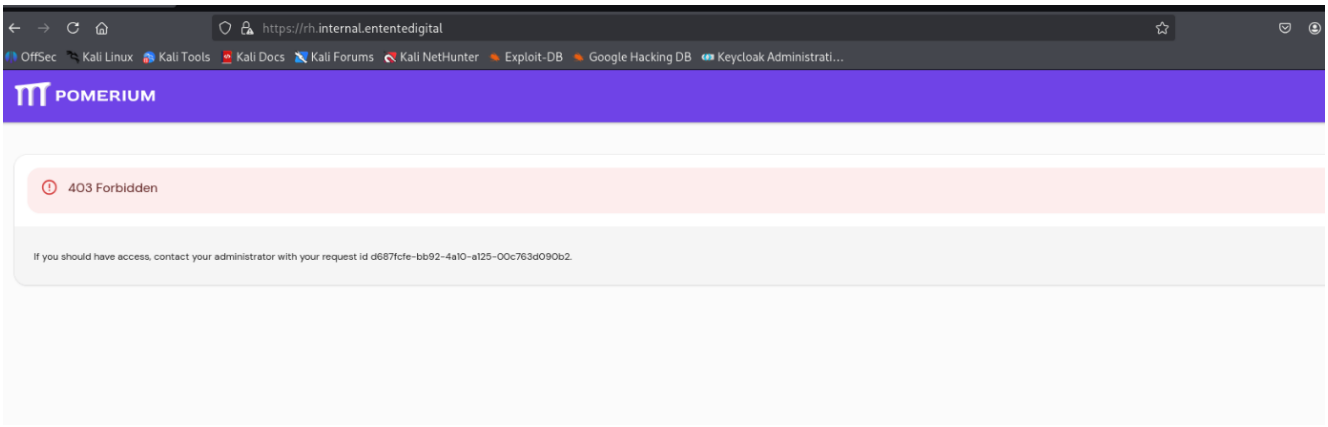


Figure 97: connexion refuse

Après l'analyse du token d'identité et la comparaison des groupes contenus avec les politiques de sécurité définies, **Pomerium refuse l'accès** à l'application RH et affiche une erreur *403 Forbidden*, confirmant que l'autorisation est strictement contrôlée conformément aux principes du Zero Trust

La mise en place du proxy Pomerium a permis de sécuriser l'accès aux applications internes en appliquant un modèle Zero Trust, dans lequel toutes les requêtes sont interceptées, authentifiées via Keycloak et autorisées selon des règles basées sur l'identité et l'appartenance aux groupes. Les différentes simulations d'accès ont confirmé que seuls les utilisateurs disposant des droits appropriés peuvent accéder aux applications, tandis que toute tentative non conforme est automatiquement bloquée, sans exposer le réseau interne ni les serveurs applicatifs.

3.5 Mise en place de la supervision avec Prometheus, Grafana et Node Exporter

Afin d'assurer une **supervision continue** de l'infrastructure mise en place, une solution de monitoring basée sur **Prometheus**, **Grafana** et **Node Exporter** a été déployée. Cette supervision permet de surveiller l'état des serveurs, les performances des services et d'anticiper d'éventuels incidents, ce qui constitue un élément essentiel dans une architecture sécurisée de type Zero Trust.

3.5.1 Déploiement de Node Exporter

Node Exporter a été installé sur les différents serveurs de l'architecture afin de collecter les métriques système. Il permet de remonter des informations détaillées telles que l'utilisation du processeur, la consommation de mémoire, l'espace disque, le trafic réseau et l'état général du système d'exploitation. Ces métriques sont exposées via un endpoint HTTP et servent de source de données pour Prometheus.

L'utilisation de Node Exporter permet d'obtenir une visibilité fine sur la santé des serveurs hébergeant les différents composants de la solution, notamment Keycloak, Pomerium, Nginx, Cloudflared et les serveurs applicatifs.

3.5.1.1 Installation du Node Exporter

Nous commençons par télécharger l'archive officielle de **Node Exporter** depuis le dépôt GitHub de Prometheus à l'aide de la commande wget, afin d'obtenir la version Linux compatible avec notre serveur.

```
serveur@vpn:/tmp$ wget https://github.com/prometheus/node_exporter/releases/download/v1.8.2/node_exporter-1.8.2.linux-amd64.tar.gz
```

Figure 98: installation

Une fois le fichier téléchargé, nous procédons à l'extraction de l'archive.tar.gz, ce qui permet de décompresser les fichiers nécessaires au fonctionnement de Node Exporter.

```
vmware-root_764-2999067644
serveur@vpn:/tmp$ tar -xvf node_exporter-1.8.2.linux-amd64.tar.gz
node_exporter-1.8.2.linux-amd64/
node_exporter-1.8.2.linux-amd64/NOTICE
node_exporter-1.8.2.linux-amd64/node_exporter
node_exporter-1.8.2.linux-amd64/LICENSE
serveur@vpn:/tmp$
```

Figure 99: décompression du fichier

Après l'extraction, nous accédons au répertoire généré et vérifions son contenu afin de confirmer la présence du binaire node_exporter ainsi que des fichiers de licence et de documentation.



Figure 100: vérification du répertoire

Nous copions ensuite le binaire node_exporter dans le répertoire système /usr/local/bin, ce qui permet de le rendre accessible globalement depuis n'importe quel emplacement du système.



Figure 101: copie vers bin

Enfin, nous exécutons la commande node_exporter --version afin de vérifier que l'installation s'est correctement déroulée et que la version installée correspond bien à celle téléchargée.



Figure 102: vérification de la version

À ce stade, notre Node Exporter est correctement installé et prêt à être configuré comme service systemd, puis supervisé par Prometheus.

3.5.1.2 Création du service systemd Node Exporter

Nous avons créé un service systemd dédié à Node Exporter afin de garantir son exécution automatique au démarrage du système et une meilleure gestion du processus. Le service est configuré pour lancer Node Exporter depuis /usr/local/bin, activer les principaux collecteurs système (CPU, mémoire, disque, réseau, charge, etc.) et exposer les métriques sur le port 9100, qui est le port standard utilisé par Prometheus.

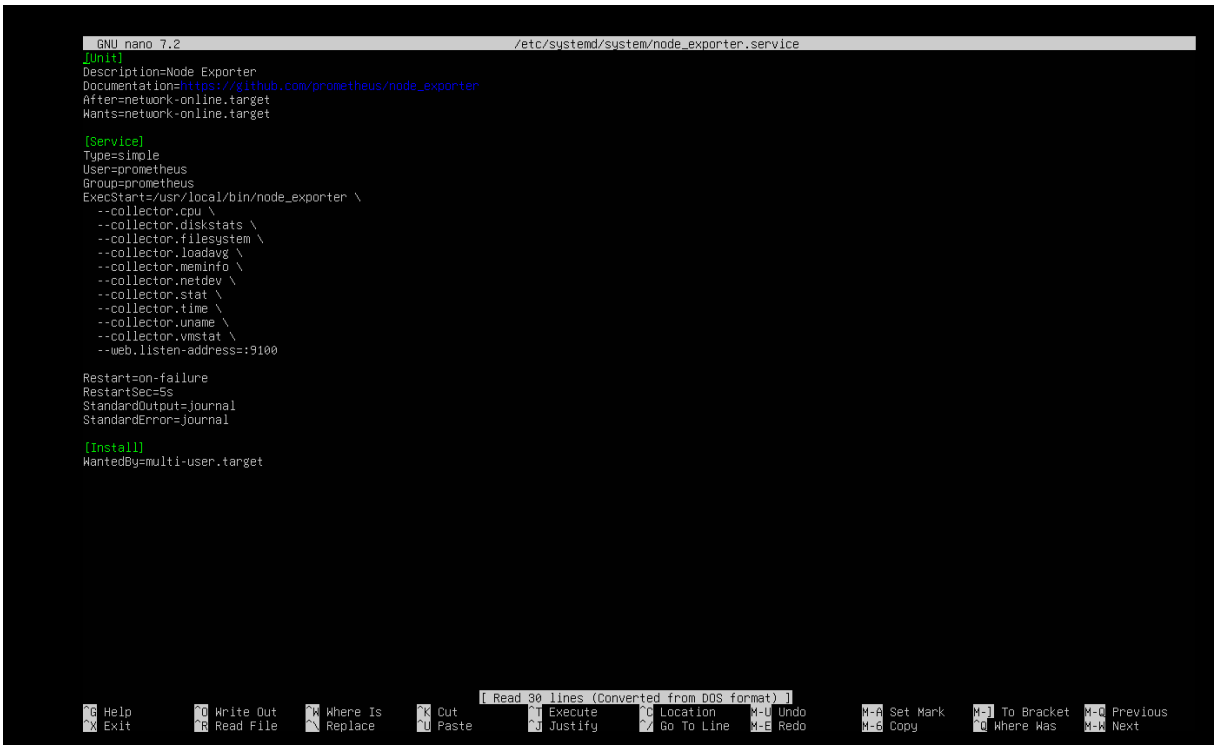


Figure 103: fichier system node exporter

Après la création du fichier de service, nous avons rechargé la configuration de systemd afin de prendre en compte le nouveau service.
Nous avons ensuite démarré Node Exporter et vérifié son état, confirmant que le service est actif et en cours d'exécution, sans erreur, et correctement attaché au processus systemd.

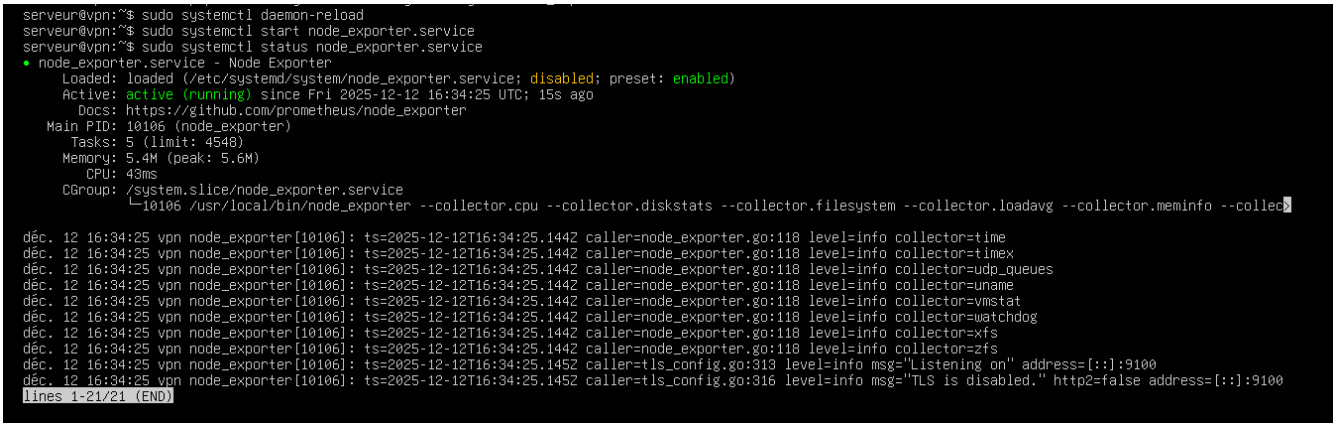


Figure 104: démarrage du service node exporter

3.5.1.3 Vérification de l'écoute sur le port 9100 et Test de l'endpoint

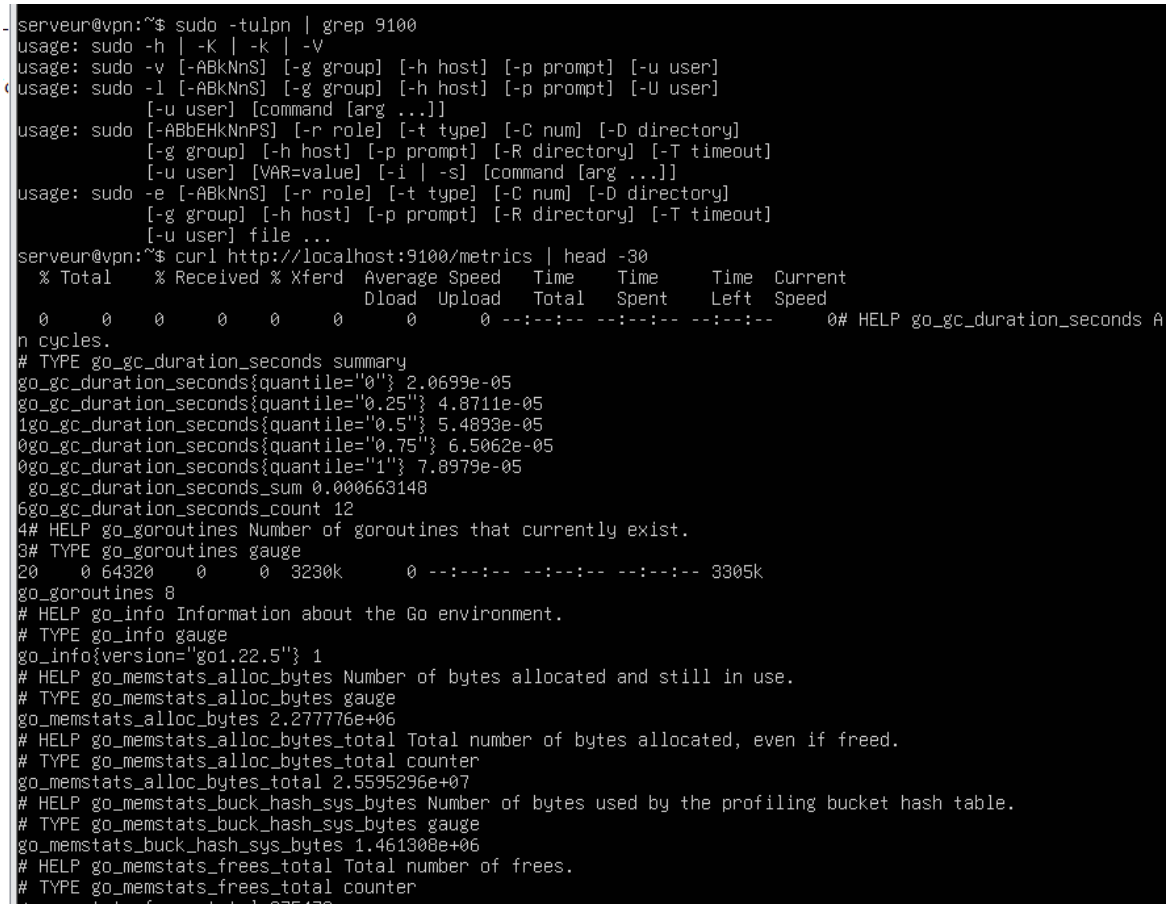


Figure 105: vérification du port et métrique

Nous avons contrôlé que Node Exporter écoute bien sur le port 9100, ce qui confirme que le service est accessible localement pour la collecte des métriques.

Cette vérification permet de s'assurer que Prometheus pourra interroger l'exporter sans problème réseau ou de configuration.

Enfin, nous avons testé l'accès à l'endpoint http://localhost:9100/metrics afin de valider la disponibilité des métriques.
La réponse retournée confirme que Node Exporter expose correctement les données système (CPU, mémoire, runtime Go, statistiques internes), ce qui signifie que le serveur est prêt à être supervisé par Prometheus.

À ce stade, **Node Exporter est correctement installé, démarré et fonctionnel**. Le serveur expose désormais ses métriques système de manière standardisée, ce qui constitue la base essentielle pour l'intégration avec **Prometheus**, puis la visualisation dans **Grafana**.

3.5.2 Déploiement prometheus

Prometheus a été mis en place comme **serveur central de collecte des métriques**. Il interroge régulièrement les endpoints exposés par Node Exporter afin de récupérer les données de supervision. Les cibles à superviser sont définies dans le fichier de configuration de Prometheus, ce qui permet d'ajouter ou de retirer facilement des serveurs à surveiller.

Grâce à son modèle de données temporelles, Prometheus permet de conserver l'historique des métriques et de suivre l'évolution des performances dans le temps. Cette capacité est particulièrement utile pour détecter des anomalies, identifier des surcharges ou analyser le comportement du système après un incident.

3.5.2.1 Installation des dépendances

Nous commençons par installer les dépendances requises telles que wget, curl, apt-transport-https et software-properties-common, indispensables pour le téléchargement sécurisé des paquets et la gestion des dépôts nécessaires à l'installation de Prometheus.

Ensuite, nous installons des outils supplémentaires comme wireguard, wireguard-tools, qrencode, curl et jq, qui seront utilisés pour la communication réseau, le traitement des données JSON et certaines intégrations liées à l'environnement de supervision.

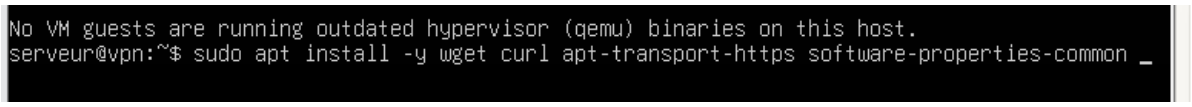


Figure 106: installation des dépendances



Figure 107: installation des dépendances

3.5.2.2 Configuration du noyau pour le réseau

Nous modifions le fichier /etc/sysctl.conf afin d'activer le forwarding IPv4 et IPv6, ce qui permet au système de router correctement le trafic réseau, une étape essentielle pour assurer le bon fonctionnement de Prometheus dans un environnement distribué.

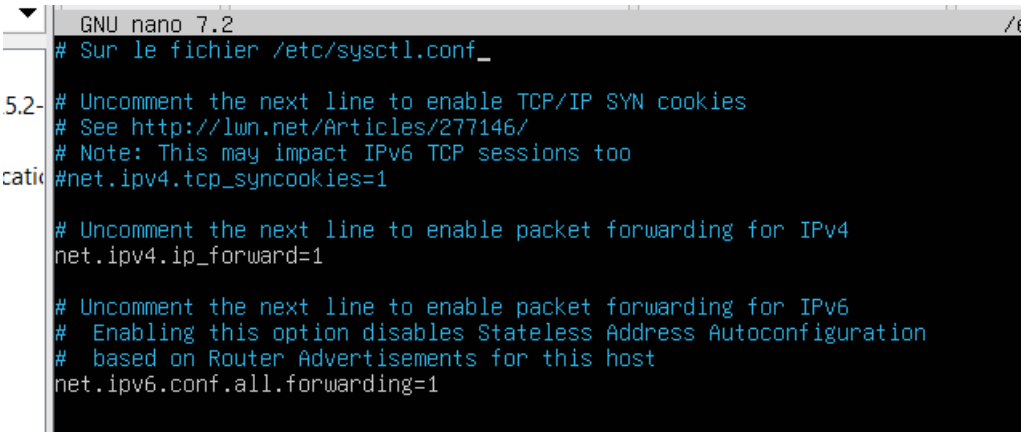


Figure 108: configuration du noyau réseau

3.5.2.3 Création de l'utilisateur Prometheus

Nous avons créé un utilisateur système dédié nommé prometheus, sans répertoire personnel et sans accès shell, afin d'exécuter le service Prometheus de manière sécurisée et isolée.



Figure 109: création de l'utilisateur Prometheus

3.5.2.4 Activation du forwarding IP

Nous avons activé le transfert de paquets IPv4 et IPv6 au niveau du noyau Linux, ce qui permet au serveur de superviser correctement les flux réseau et d'assurer la communication entre les différents services de monitoring.

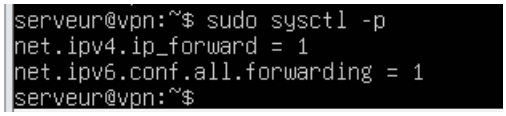


Figure 110: activation du forwarding IP

3.5.2.5 Création des répertoires Prometheus et Attribution des droits

On a créé les répertoires /etc/prometheus pour les fichiers de configuration et /var/lib/prometheus pour le stockage des données de métriques.

Nous avons attribué la propriété de ces répertoires à l'utilisateur prometheus afin de garantir que le service puisse lire et écrire ses fichiers sans privilèges excessifs.

```
serveur@vpn:~$ sudo mkdir -p /etc/prometheus
serveur@vpn:~$ sudo mkdir -p /var/lib/prometheus
serveur@vpn:~$ sudo chown prometheus:prometheus /etc/prometheus
serveur@vpn:~$ sudo chown prometheus:prometheus /var/lib/prometheus
serveur@vpn:~$
```

Figure 111: création de répertoire et attribution des droits

3.5.2.6 Installation prometheus

Nous avons copié l'archive de Prometheus depuis notre machine locale vers le serveur de supervision à l'aide de la commande scp.

```
C:\Users\HP\Downloads>scp -r prometheus-3.7.3.darwin-amd64.tar.gz serveur@192.168.248.146:/home/serveur/
The authenticity of host '192.168.248.146 (192.168.248.146)' can't be established.
ED25519 key fingerprint is SHA256:1Zq9w6CTDFIr2C2kUnPy2/ZGNipKwAiAntLuspXtN8U.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])?
Warning: Permanently added '192.168.248.146' (ED25519) to the list of known hosts.
serveur@192.168.248.146's password:
prometheus-3.7.3.darwin-amd64.tar.gz                          100%  138MB  60.0MB/s   00:02
```

Figure 112: Transfert de l'archive Prometheus vers le serveur

On a extrait le contenu de l'archive téléchargée afin d'accéder aux binaires et aux fichiers de configuration fournis par Prometheus.

```
serveur@vpn:~$ ^C
serveur@vpn:~$ cd /home/serveur/
serveur@vpn:~$ ls
prometheus-3.7.3.darwin-amd64.tar.gz
serveur@vpn:~$ tar -xvf prometheus-3.7.3.darwin-amd64.tar.gz
prometheus-3.7.3.darwin-amd64/
prometheus-3.7.3.darwin-amd64/NOTICE
prometheus-3.7.3.darwin-amd64/LICENSE
prometheus-3.7.3.darwin-amd64/prometheus.yml
prometheus-3.7.3.darwin-amd64/promtool
prometheus-3.7.3.darwin-amd64/prometheus
serveur@vpn:~$
```

Figure 113: extraction du fichier prometheus

Nous avons copié les binaires prometheus et promtool dans le répertoire /usr/local/bin pour les rendre accessibles globalement sur le système.

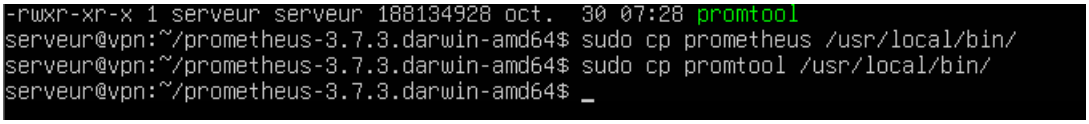


Figure 114: Copie des binaires Prometheus et Promtool

Nous avons changé le propriétaire des binaires Prometheus et Promtool pour l'utilisateur prometheus, garantissant ainsi une exécution sécurisée du service.

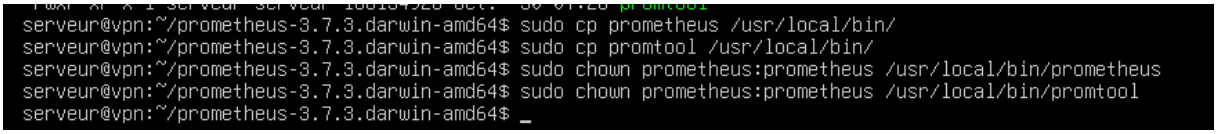


Figure 115: Attribution des droits aux binaires

Nous avons créé les répertoires **consoles** et **console_libraries**, utilisés par Prometheus pour l'affichage avancé et les dashboards internes.

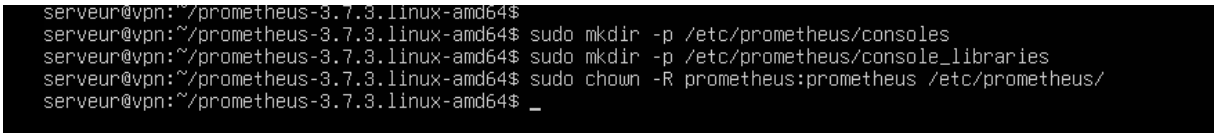


Figure 116: Création des répertoires consoles

Nous avons vérifié la version de **Prometheus** et de **Promtool**, confirmant que l'installation correspond bien à la version prévue et qu'elle est fonctionnelle.

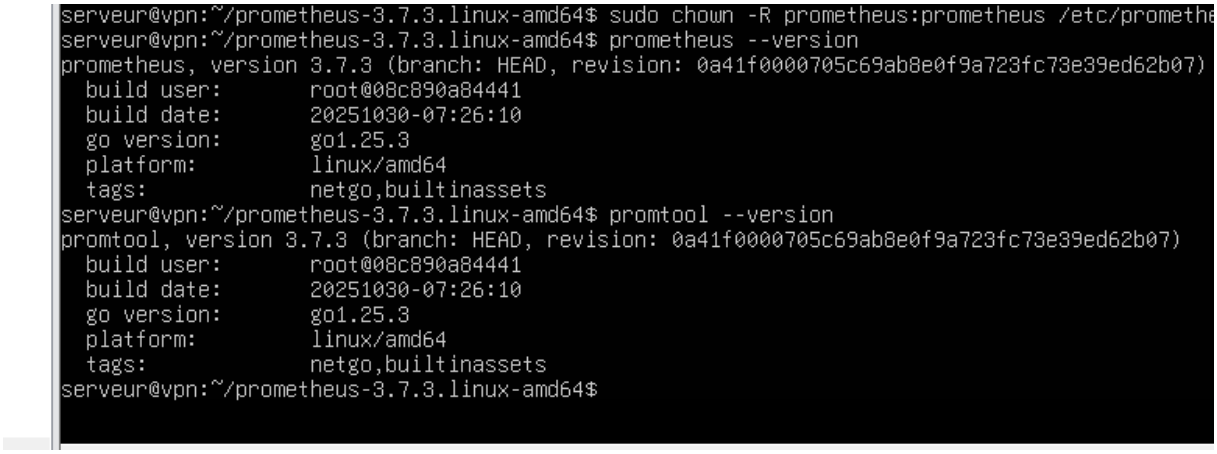


Figure 117: Vérification des versions

3.5.2.7 Fichier de configuration prometheus

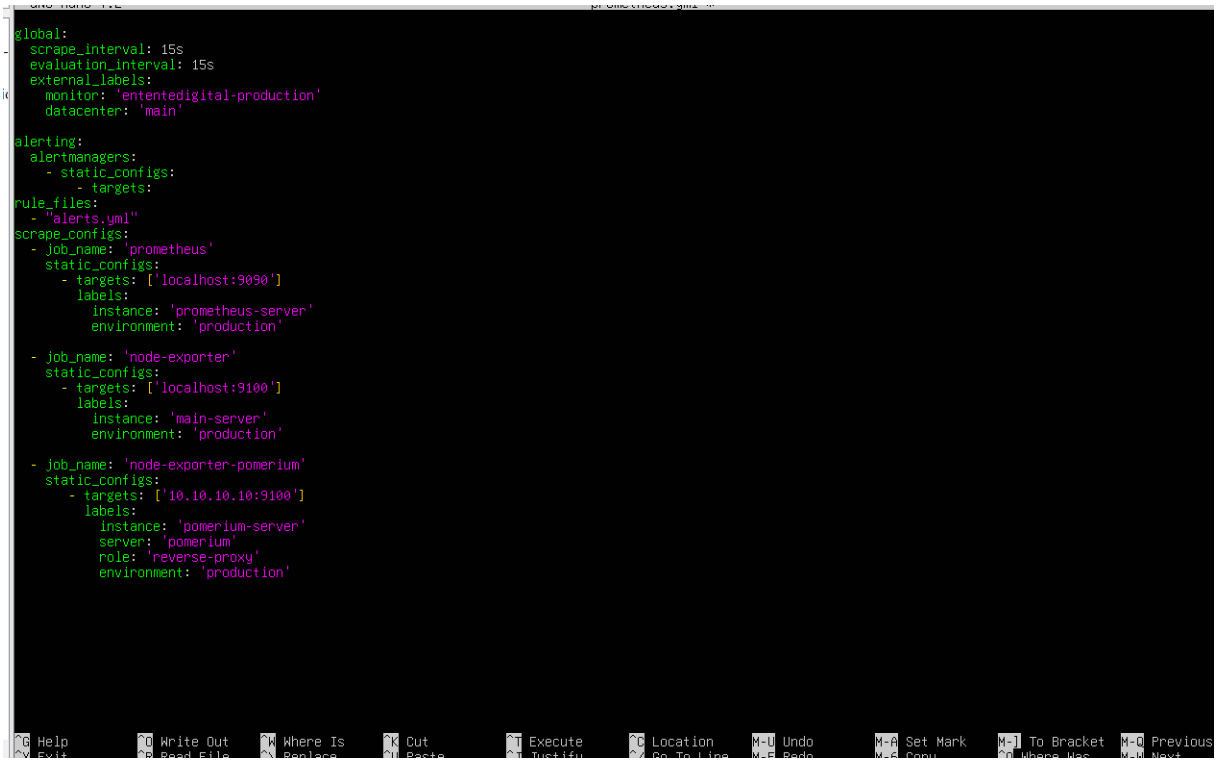


Figure 118: Fichier de configuration prometheus

Dans ce fichier, **nous définissons la configuration globale de Prometheus**, notamment l'intervalle de collecte (scrape_interval) et d'évaluation des règles (evaluation_interval), afin d'assurer une supervision régulière et cohérente de l'infrastructure.

Nous y **déclarons les règles d'alerte** via le fichier alerts.yml, ce qui permet à Prometheus de charger automatiquement les alertes personnalisées que nous avons définies.

Ensuite, **nous configurons les différentes cibles de supervision (scrape_configs)**, en commençant par Prometheus lui-même sur le port 9090, afin de surveiller son propre état de fonctionnement.

Nous ajoutons également **le job node-exporter**, qui permet de collecter les métriques système (CPU, mémoire, disque, réseau) du serveur principal via le port 9100.

Enfin, **nous déclarons un second Node Exporter pour le serveur Pomerium**, en précisant son adresse IP et en ajoutant des labels (instance, role, environment) afin de faciliter l'identification et l'exploitation des métriques dans Grafana.

3.5.2.8 Fichier de configuration alerte

```
groups:
  - name: system_alerts
    interval: 30s
    rules:
      - alert: InstanceDown
        expr: up == 0
        for: 5m
        labels:
          severity: critical
        annotations:
          summary: "Instance {{ $labels.instance }} est down"
          description: "{{ $labels.instance }} du job {{ $labels.job }} est down depuis plus de 5 minutes."

      - alert: HighCPUUsage
        expr: 100 - (avg by(instance) (rate(node_cpu_seconds_total{mode="idle"}[5m])) * 100) > 80
        for: 10m
        labels:
          severity: warning
        annotations:
          summary: "CPU élevé sur {{ $labels.instance }}"
          description: "L'utilisation CPU est supérieure à 80% depuis plus de 10 minutes. Valeur actuelle: {{ $value }}%"

      - alert: HighMemoryUsage
        expr: (1 - (node_memory_MemAvailable_bytes / node_memory_MemTotal_bytes)) * 100 > 85
        for: 10m
        labels:
          severity: warning
        annotations:
          summary: "Mémoire élevée sur {{ $labels.instance }}"
          description: "L'utilisation mémoire est supérieure à 85% depuis plus de 10 minutes. Valeur actuelle: {{ $value }}%"

      - alert: DiskSpaceLow
        expr: (1 - (node_filesystem_avail_bytes{fstype!="tmpfs",mountpoint="/"} / node_filesystem_size_bytes{fstype!="tmpfs",mountpoint="/"})) * 100 > 85
        for: 5m
        labels:
          severity: warning
        annotations:
          summary: "Espace disque faible sur {{ $labels.instance }}"
          description: "L'espace disque sur {{ $labels.mountpoint }} est supérieur à 85%. Valeur actuelle: {{ $value }}%"

      - alert: HighLoadAverage
        expr: node_load1 > 4
        for: 5m
        labels:
          severity: warning
        annotations:
```

Figure 119: alerte.yml

Dans ce fichier, **nous définissons un groupe d'alertes système** destiné à détecter les incidents critiques sur les serveurs supervisés.

Nous commençons par **l'alerte InstanceDown**, qui se déclenche lorsqu'une instance ne répond plus pendant plus de 5 minutes, ce qui permet de détecter rapidement une panne ou une coupure réseau.

Nous configurons ensuite **une alerte de forte utilisation CPU**, déclenchée lorsque l'usage du processeur dépasse 80 % sur une durée prolongée, afin d'anticiper les problèmes de surcharge.

Une alerte similaire est mise en place pour **la mémoire**, afin d'être notifiés lorsque l'utilisation RAM dépasse un seuil critique et risque d'affecter la stabilité du système.

Nous ajoutons également **une alerte sur l'espace disque**, qui surveille le taux d'occupation des systèmes de fichiers et prévient en cas de risque de saturation.

Enfin, **une alerte de charge système élevée (load average)** est définie pour identifier les situations où le serveur est soumis à une charge excessive sur une période donnée.

Nous avons attribué les bons droits aux fichiers **prometheus.yml** et **alerts.yml**, afin que Prometheus puisse les lire correctement au démarrage.

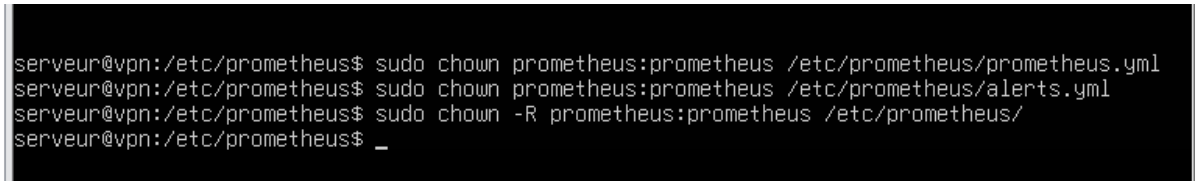


Figure 120: Droits sur les fichiers de configuration

Nous avons utilisé **promtool** pour vérifier la syntaxe des fichiers de configuration, ce qui a confirmé que la configuration Prometheus est valide et sans erreur.

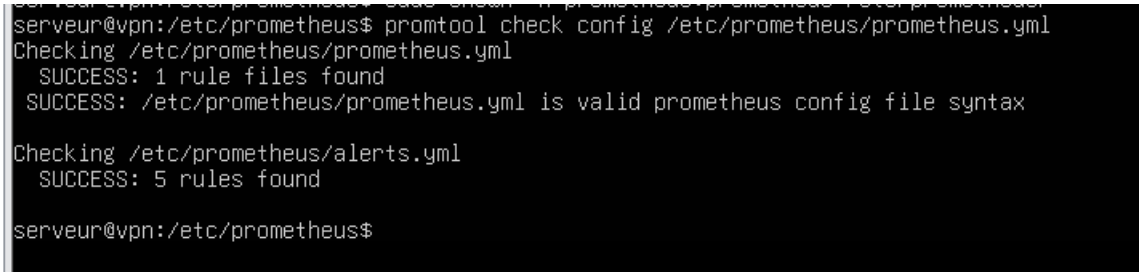


Figure 121: Validation de la configuration

Nous avons vérifié que Prometheus écoute correctement sur le **port 9090**, confirmant que le service est bien démarré et accessible sur le réseau.

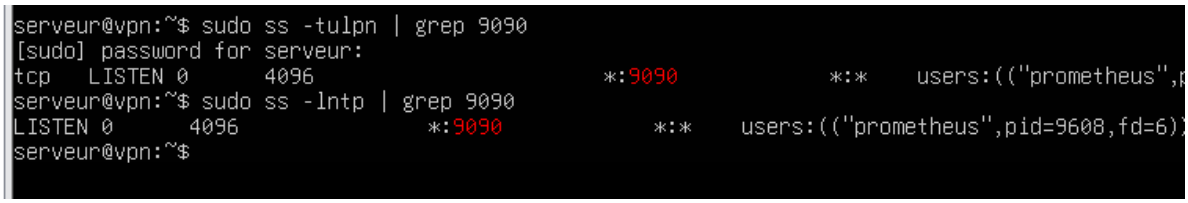


Figure 122: Vérification du port d'écoute

Nous avons accédé à l'interface Web de Prometheus via un navigateur, ce qui valide le bon fonctionnement du service et la disponibilité de l'interface de requêtes.

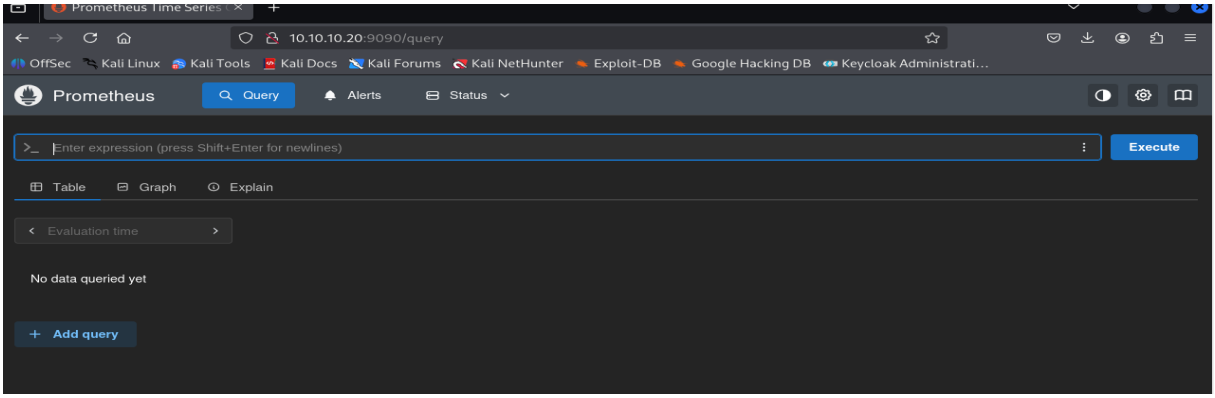


Figure 123: Accès à l'interface Web Prometheus

3.5.3 Déploiement Grafana

Grafana a été déployé afin de fournir une **visualisation claire et intuitive** des données collectées par Prometheus. Il permet de créer des tableaux de bord interactifs affichant en temps réel l'état des serveurs et des services.

À travers Grafana, il est possible de visualiser :

- L'utilisation du CPU et de la mémoire,
- La charge système,
- L'espace disque disponible,
- Le trafic réseau,
- La disponibilité des services.

Ces tableaux de bord facilitent la supervision quotidienne et offrent une vue globale de l'infrastructure, ce qui permet aux administrateurs de réagir rapidement en cas de dégradation des performances.

3.5.4 Création du répertoire des clés APT

Nous commençons par créer le répertoire /etc/apt/keyrings, qui servira à stocker de manière sécurisée les clés GPG des dépôts externes, conformément aux bonnes pratiques recommandées sous Debian/Ubuntu.

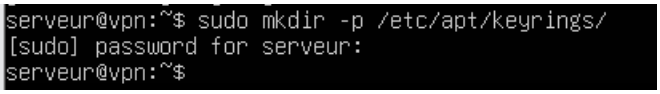


Figure 124: création du répertoire des clés

3.5.3.1 Ajout de la clé GPG de Grafana

Nous téléchargeons ensuite la clé GPG officielle de Grafana depuis le dépôt apt.grafana.com, puis nous la convertissons au format compatible (.gpg) afin de garantir l'authenticité et l'intégrité des paquets Grafana qui seront installés.



Figure 125: ajout de la clé

3.5.3.2 Ajout du dépôt officiel Grafana

Nous ajoutons le dépôt APT officiel de Grafana à la liste des sources du système en l'associant explicitement à la clé GPG précédemment importée, ce qui permet au gestionnaire de paquets d'installer Grafana de manière sécurisée.



Figure 126: ajout dépôt officiel grafana

3.5.3.3 Installation de Grafana

Une fois le dépôt configuré, nous procédons à l'installation du service Grafana via le gestionnaire de paquets APT, ce qui permet d'installer automatiquement Grafana ainsi que toutes ses dépendances nécessaires.

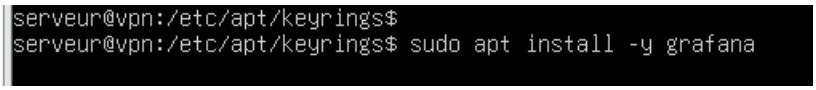


Figure 127: installation grafana

Nous vérifions enfin l'installation en affichant la version de Grafana installée sur le serveur, confirmant ainsi que le service a été correctement déployé et qu'il est prêt à être configuré et démarré.

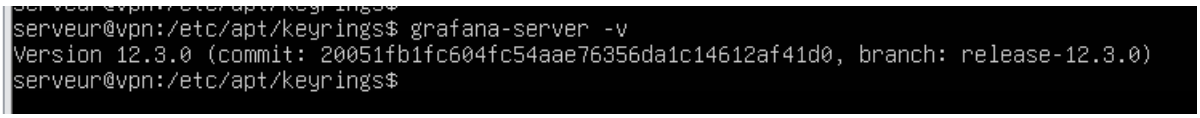


Figure 128 : vérification de l'installation

Nous avons commencé par renforcer la sécurité du serveur de supervision en bloquant l'accès aux ports sensibles utilisés par Prometheus (9090), Node Exporter (9100) et Grafana (3000) via le pare-feu UFW, afin d'éviter toute exposition directe depuis l'extérieur.

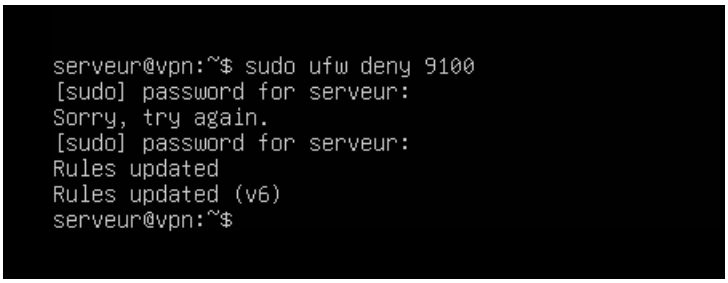


Figure 129: sécurisation des ports

Après avoir bloqué l'accès global au port 3000, nous avons explicitement autorisé uniquement l'adresse IP interne du serveur Pomerium (10.10.10.10) à accéder à Grafana, ce qui garantit que l'interface de supervision ne soit accessible qu'à travers le proxy Zero Trust.

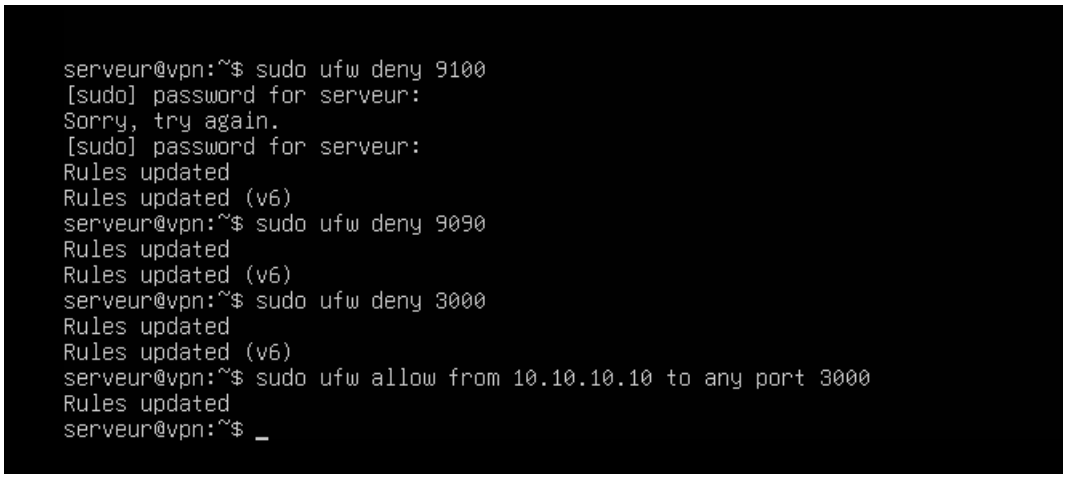


Figure 130: sécurisation des ports

3.5.3.4 Génération du certificat TLS avec mkcert

Nous avons utilisé **mkcert** pour générer un certificat TLS local pour le nom de domaine grafana.internal.ententedigital, afin de sécuriser l'accès HTTPS à l'interface Grafana dans l'environnement interne.

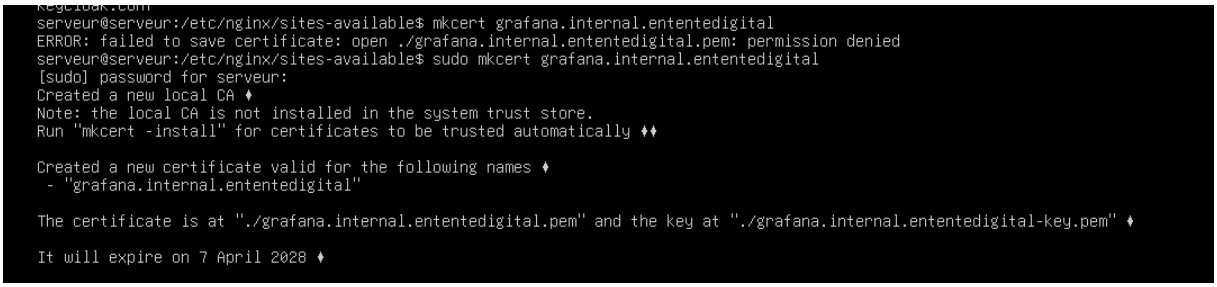


Figure 131:Génération du certificat TLS avec mkcert

Nous avons ensuite copié les fichiers de certificat et de clé privée (.pem) vers le serveur Grafana à l'aide de scp, en établissant une connexion SSH sécurisée entre les deux machines.

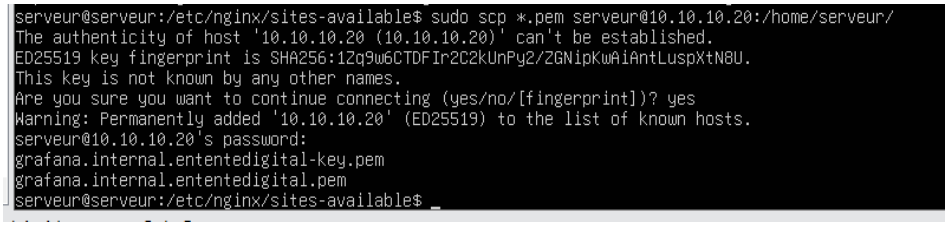


Figure 132: copie des certificats

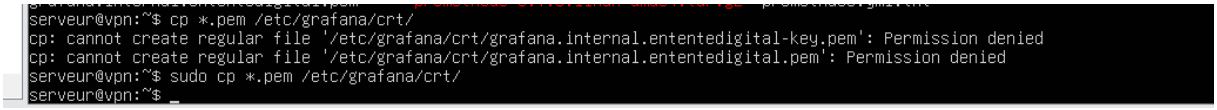


Figure 133: Placement des certificats dans le répertoire Grafana

Les certificats ont été déplacés dans le répertoire dédié /etc/grafana/crt/, afin d'être utilisés directement par le service Grafana pour le chiffrement HTTPS.

Nous avons ajusté les droits d'accès aux fichiers de certificats (chmod 600) pour garantir que seuls les processus autorisés puissent lire la clé privée, conformément aux bonnes pratiques de sécurité.



Figure 134:Correction des permissions sur les fichiers de certificats

3.5.3.5 Fichier de configuration grafana

```
#################################### Server ####################################
[server]
# Protocol (http, https, h2, socket)
protocol = http

# Minimum TLS version allowed. By default, this value is empty. Accepted values are: TLS1.2, TLS1.3. If nothing is set TLS1.2 would be taken
;min_tls_version = ""

# The ip address to bind to, empty will bind to all interfaces
http_addr = 10.10.10.10

# The http port to use
http_port = 3000

# The public facing domain name used to access grafana from a browser
domain = grafana.internal.ententedigital

# Redirect to correct domain if host header does not match domain
# Prevents DNS rebinding attacks
;enforce_domain = false

# The full public facing url you use in browser, used for redirects and emails
# If you use reverse proxy and sub path specify full url (with sub path)
root_url = https://grafana.internal.ententedigital/

# Serve Grafana from subpath specified in `root_url` setting. By default it is set to `false` for compatibility reasons.
serve_from_sub_path = false

# Log web requests
;router_logging = false

# the path relative working path
;static_root_path = public

# enable gzip
enable_gzip = true

# https certs & key file
;cert_file = /etc/grafana/crt/grafana.crt
;cert_key = /etc/grafana/crt/grafana.key

# optional password to be used to decrypt key file
;cert_pass =
```

Figure 135: fichier de configuration grafana

Nous avons mis à jour le fichier de configuration de Grafana afin d'adapter son fonctionnement à une exposition sécurisée derrière le proxy Pomerium. Grafana est configuré pour écouter sur le **port 3000** (http_port = 3000) ce port est utilisé uniquement par le serveur Pomerium et sur 'adresse IP interne **10.10.10.10** au lieu de **0.0.0.0**, ce qui permet de restreindre l'accès au service et d'éviter toute exposition inutile sur d'autres interfaces réseau. L'accès à Grafana se fait ainsi exclusivement via le proxy Pomerium, conformément au principe Zero Trust.

Le **nom de domaine interne** grafana.internal.ententedigital a été défini comme domaine d'accès officiel, ce qui garantit que Grafana est utilisé uniquement via l'URL prévue et non par adresse IP. Le paramètre root_url a également été renseigné afin d'assurer le bon fonctionnement des redirections et des liens lorsque Grafana est utilisé derrière un reverse proxy.

La configuration HTTPS native de Grafana est volontairement laissée inactive, car le **chiffrement TLS est pris en charge par Pomerium**, ce qui respecte le principe Zero Trust et évite toute double terminaison TLS inutile.

3.5.3.6 Configuration cote keycloak

Nous avons créé et vérifié le **groupe monitoring** dans Keycloak afin de gérer spécifiquement l'accès aux outils de supervision, notamment **Grafana**, en s'appuyant sur le proxy **Pomerium** pour appliquer un contrôle d'accès basé sur l'appartenance à ce groupe.

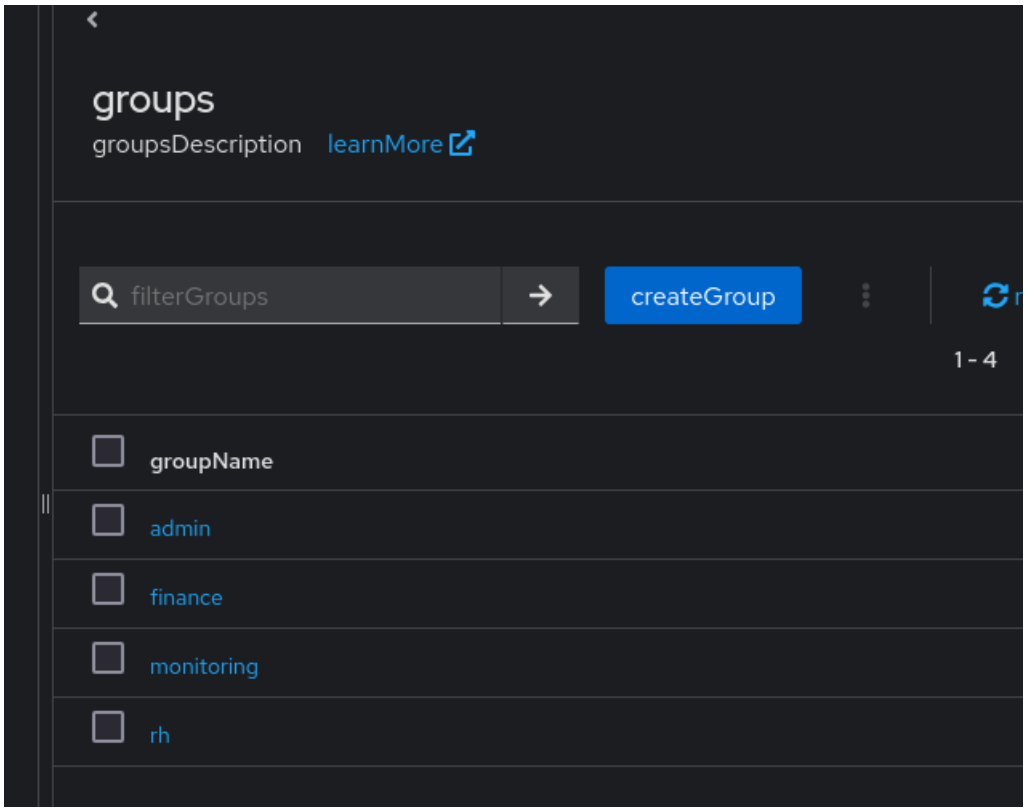


Figure 136: création group pour grafana

3.5.3.7 Simulation

Après authentification réussie, nous accédons au tableau de bord Grafana *Node Exporter Full* pour le serveur **pomerium-server**, ce qui confirme que Prometheus collecte correctement les métriques du serveur Pomerium et que Grafana les affiche en temps réel (CPU, mémoire, disque, uptime).

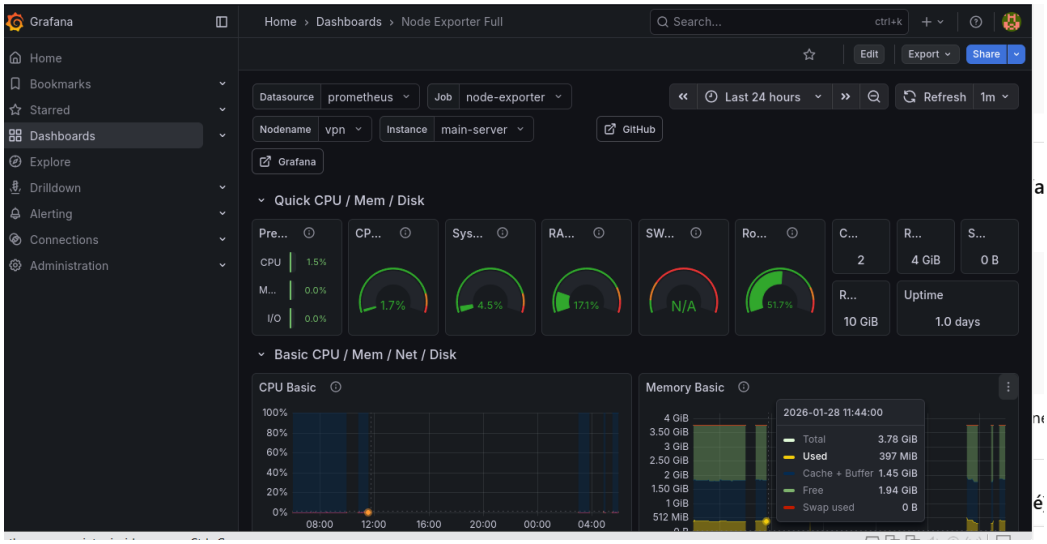


Figure 137: supervision proxy

Nous visualisons ensuite les métriques du serveur local à travers le même dashboard, ce qui démontre que le **Node Exporter** est bien déployé sur plusieurs machines et que Prometheus centralise les données de supervision sans exposition directe des serveurs.

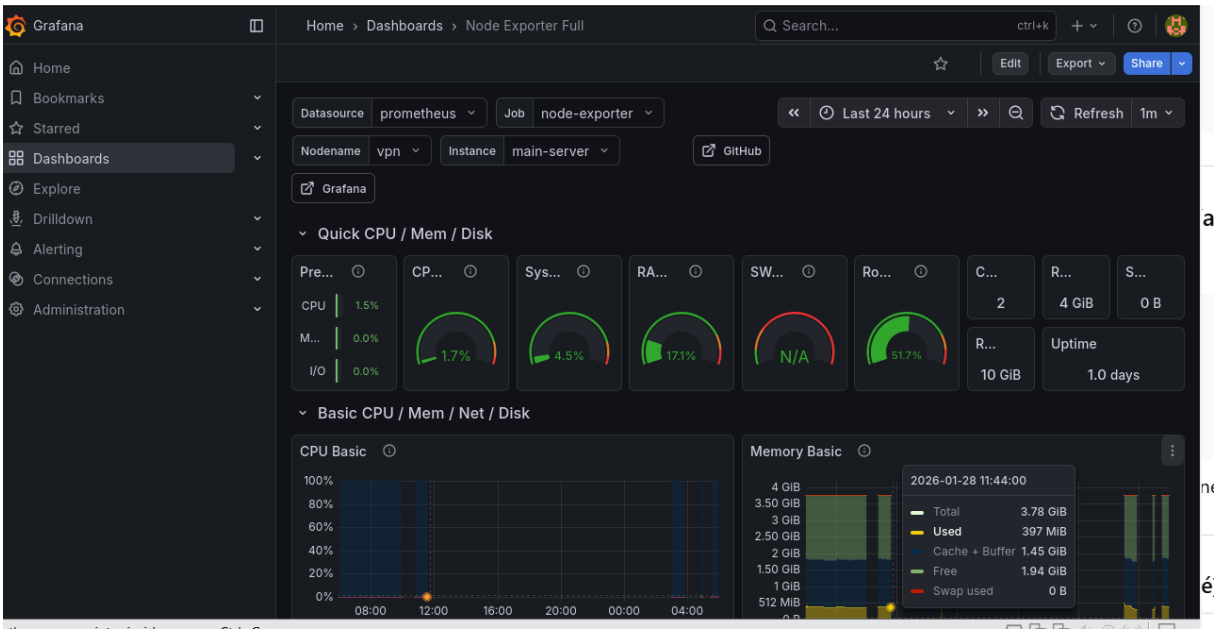


Figure 138: supervision serveur local

Enfin, nous avons configuré un **panel Grafana** basé sur la métrique **ALERTS** provenant de Prometheus, afin de visualiser en temps réel l'état des alertes (par exemple *HighCPUUsage*) et leur statut (*pending*, *firing*, etc.), ce qui permet à l'équipe **monitoring** de suivre rapidement les incidents et l'état de santé des serveurs supervisés.

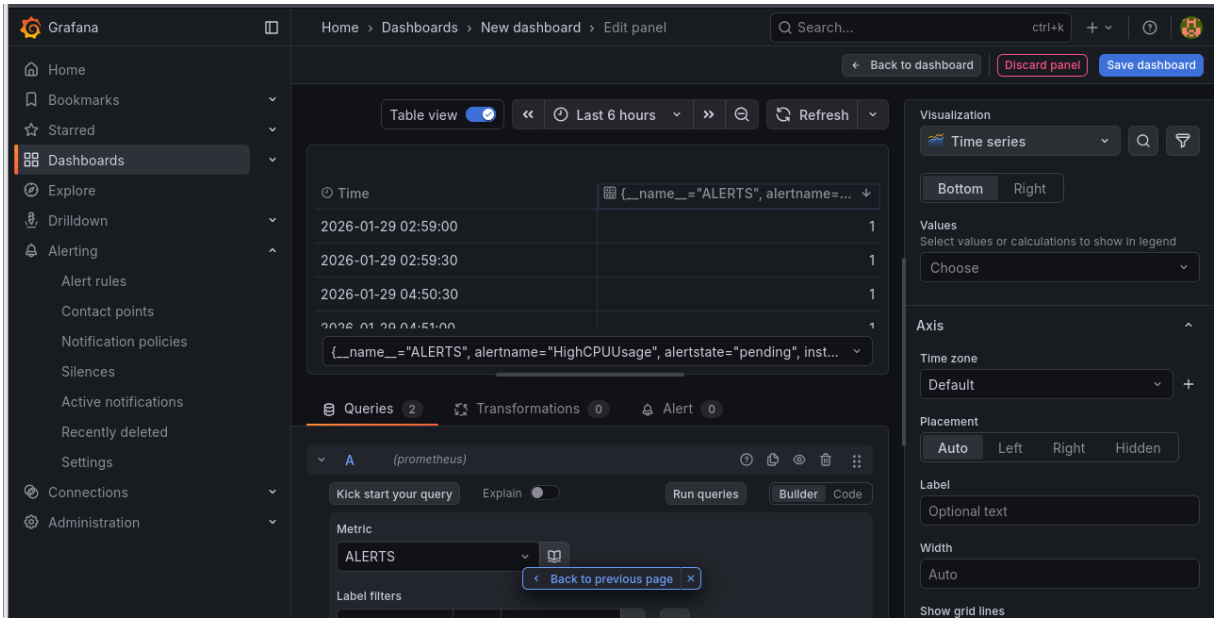


Figure 139: alerte

La mise en place de la supervision basée sur **Prometheus**, **Node Exporter** et **Grafana** permet d'assurer une visibilité complète, centralisée et en temps réel de l'infrastructure. Les métriques système sont collectées de manière fiable sur les serveurs, analysées par Prometheus et restituées de façon claire et exploitable via Grafana, avec des tableaux de bord et des alertes adaptées.

3.6 Mise en œuvre du cloudflared

La mise en place de Cloudflared a pour objectif d'exposer une application de manière sécurisée vers l'extérieur, sans ouvrir de ports entrants sur le pare-feu et sans exposer directement l'adresse IP du serveur.

Dans un premier temps, l'outil Cloudflared est installé sur le serveur hébergeant l'application publique. L'installation permet d'obtenir le binaire nécessaire à la création d'un tunnel sécurisé entre le serveur local et l'infrastructure Cloudflare.

Une fois l'installation terminée, une authentification est effectuée auprès de Cloudflare à l'aide de la commande dédiée. Cette étape permet d'associer le serveur local au compte Cloudflare de l'organisation et d'autoriser la création de tunnels sécurisés.

Ensuite, un **tunnel Cloudflared** est créé. Ce tunnel établit une connexion sortante chiffrée depuis le serveur local vers Cloudflare, éliminant ainsi le besoin d'exposer le serveur via une adresse IP publique ou des règles NAT.

Le tunnel est ensuite démarré en mode service afin de garantir sa disponibilité permanente. À partir de ce moment, toute requête provenant d'Internet vers le nom de domaine public transite par l'infrastructure Cloudflare avant d'être redirigée vers l'application locale via le tunnel sécurisé.

Enfin, des tests d'accès sont réalisés afin de vérifier que l'application est accessible depuis l'extérieur, tout en restant protégée contre les attaques réseau classiques, telles que le scan de ports ou l'exposition directe des services.

3.6.1 Installation de cloudfared

Transfert du paquet Cloudflared vers le serveur



Figure 140:copie du cloudflared vers le serveur

Apres le copie vers le serveur dédié , on va procéder à l'installation de Cloudflared sur le serveur à l'aide de la commande dpkg -i. Le système décompresse et configure le paquet, ce qui rend le service Cloudflared opérationnel et disponible pour la création de tunnels sécurisés

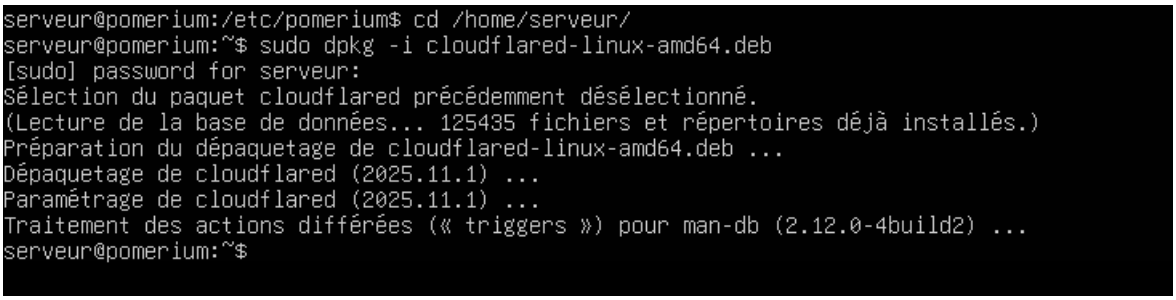


Figure 141: installation du cloudflare

À l'exécution de la commande cloudflared tunnel login, qui génère une URL d'authentification. Cette étape permet de lier le serveur au compte Cloudflare de l'organisation afin de l'autoriser à créer et gérer des tunnels Cloudflare.



Figure 142: authentification cloudflare



Figure 143: autorisation tunnel

Le capture ci-dessus montre l'interface Cloudflare demandant l'autorisation d'ajouter un tunnel pour le domaine ententedigital.com. La sélection et la validation du domaine confirment que Cloudflare est autorisé à servir de point d'origine sécurisé pour ce domaine via Cloudflare Tunnel.

Nous avons ensuite autorisé la création du tunnel Cloudflare en sélectionnant le domaine ententedigital.com depuis l'interface Cloudflare. Cette action permet à Cloudflare de devenir le point d'entrée public et d'établir un lien sécurisé entre Internet et notre infrastructure interne.

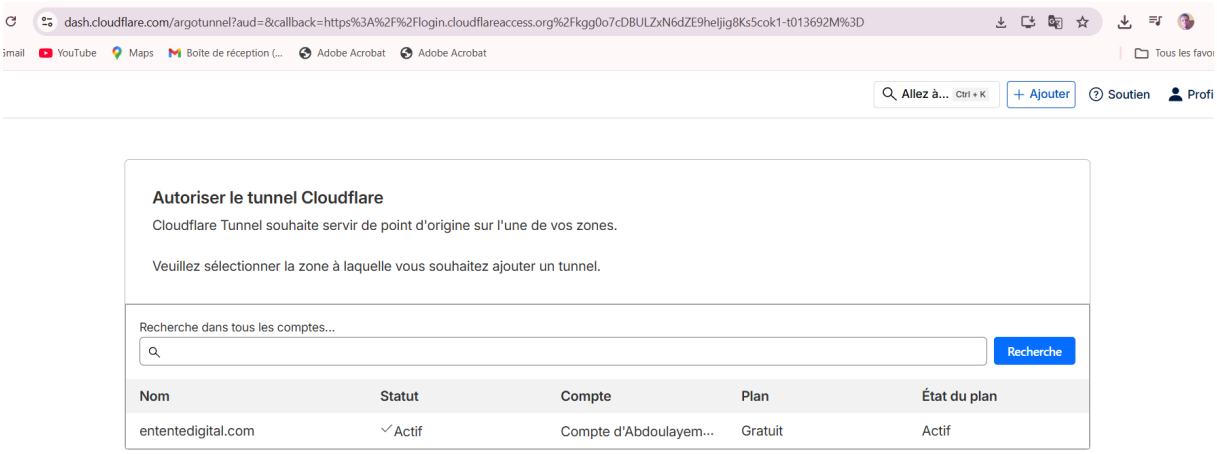


Figure 144: autorisation du tunnel

Nous avons transféré le certificat d'origine généré par Cloudflared vers le serveur interne à l'aide d'un transfert sécurisé. Ce certificat est nécessaire pour authentifier le serveur auprès de Cloudflared et garantir que seules les machines autorisées peuvent établir un tunnel.

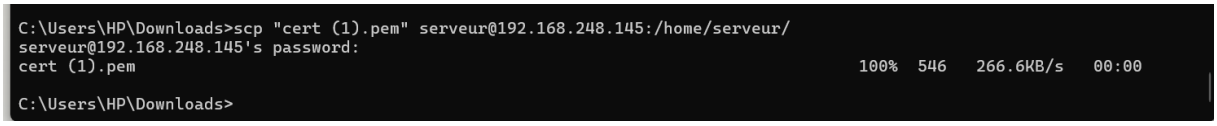


Figure 145: copie du certificat vers le serveur

Nous avons ensuite placé le certificat dans le répertoire. cloudflared du serveur afin que le service Cloudflared puisse l'utiliser automatiquement lors de son démarrage, sans nécessiter une authentification manuelle à chaque connexion.



On a créé un tunnel Cloudflare dédié, nommé ententedigital-tunnel, ce qui a permis de générer un identifiant unique ainsi qu'un fichier de credentials. Ce tunnel assure une connexion sortante chiffrée entre le serveur interne et Cloudflare, sans exposition directe d'adresse IP publique ni ouverture de ports entrants.

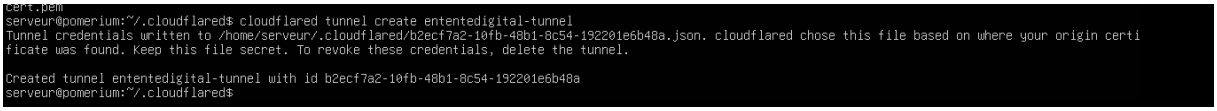


Figure 146: création tunnel

Nous avons créé un tunnel Cloudflare dédié, nommé **ententedigital-tunnel**, ce qui a permis de générer un identifiant unique ainsi qu'un fichier de credentials. Ce tunnel assure une connexion sortante chiffrée entre le serveur interne et Cloudflare, sans exposition directe d'adresse IP publique ni ouverture de ports entrants.

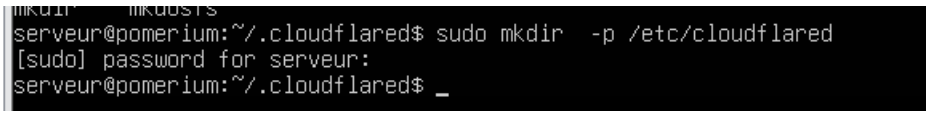


Figure 147: création répertoire cloudflare

3.6.2 Fichier de configuration cloudflared

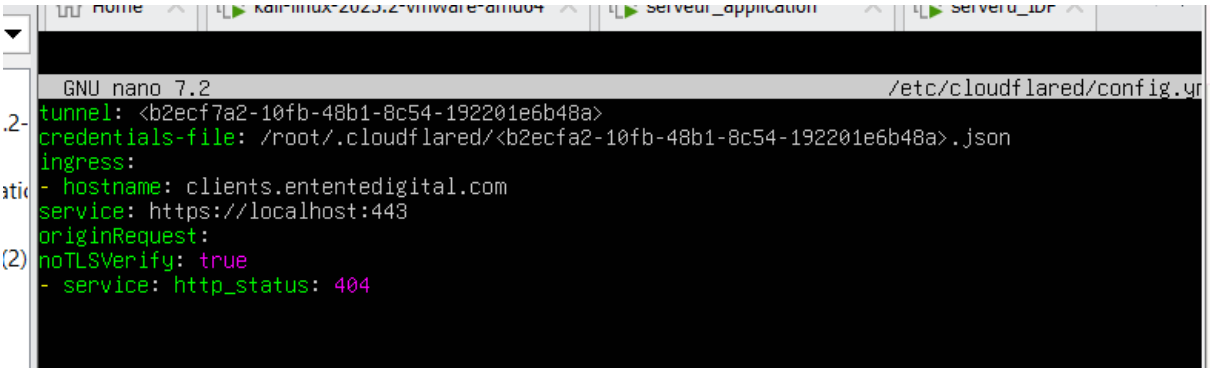


Figure 148: fichier de configuration cloudflared

Dans le fichier **config.yml de Cloudflared**, nous avons défini l'ensemble des paramètres nécessaires au fonctionnement du tunnel et à l'exposition sécurisée du service interne.

Tout d'abord, la directive **tunnel** permet d'indiquer l'identifiant unique du tunnel Cloudflare. Cet identifiant correspond au tunnel que nous avons créé précédemment et il sert à lier ce fichier de configuration à ce tunnel précis.

Ensuite, le champ **credentials-file** spécifie le chemin vers le fichier JSON contenant les informations d'authentification du tunnel. Ce fichier est généré automatiquement lors de la création du tunnel et permet à Cloudflared de s'authentifier auprès de l'infrastructure Cloudflare de manière sécurisée, sans nécessiter d'identifiants manuels.

La section **ingress** définit les règles de routage entre les noms de domaine publics et les services internes. Elle indique comment les requêtes entrantes provenant de Cloudflare doivent être traitées et vers quelle application locale elles doivent être redirigées.

Dans cette section, le paramètre **hostname** correspond au nom de domaine public *clients.ententedigital.com*. Il s'agit du point d'entrée exposé sur Internet, accessible via les serveurs Cloudflare, sans révéler l'adresse IP publique du serveur interne.

Le champ **service** précise l'adresse du service interne à atteindre. Ici, nous avons configuré une redirection vers **https://localhost:443**, ce qui signifie que Cloudflared transmet les requêtes reçues depuis Cloudflare vers l'application locale exécutée en HTTPS sur le port 443.

L'option **originRequest**, et plus précisément **noTLSVerify: true**, permet d'indiquer à Cloudflared de ne pas vérifier le certificat TLS présenté par le service interne. Ce choix est

justifié par l'utilisation d'un certificat interne ou auto-signé, tout en conservant un chiffrement TLS entre Cloudflare et le client final.

Enfin, la dernière règle **service: http_status:404** agit comme une règle de secours. Elle garantit que toute requête ne correspondant à aucun hostname défini dans la section ingress reçoit une réponse 404, empêchant ainsi l'accès involontaire à des services non autorisés.

Nous avons créé la route DNS du tunnel à l'aide de la commande **cloudflared tunnel route dns**, ce qui a permis d'associer automatiquement le sous-domaine *clients.ententedigital.com* au tunnel Cloudflare via un enregistrement CNAME géré par Cloudflare.

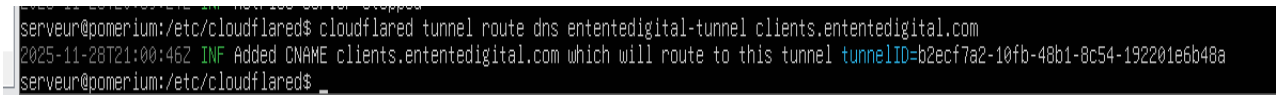


Figure 149: dns route

3.6.3 Configuration Nginx

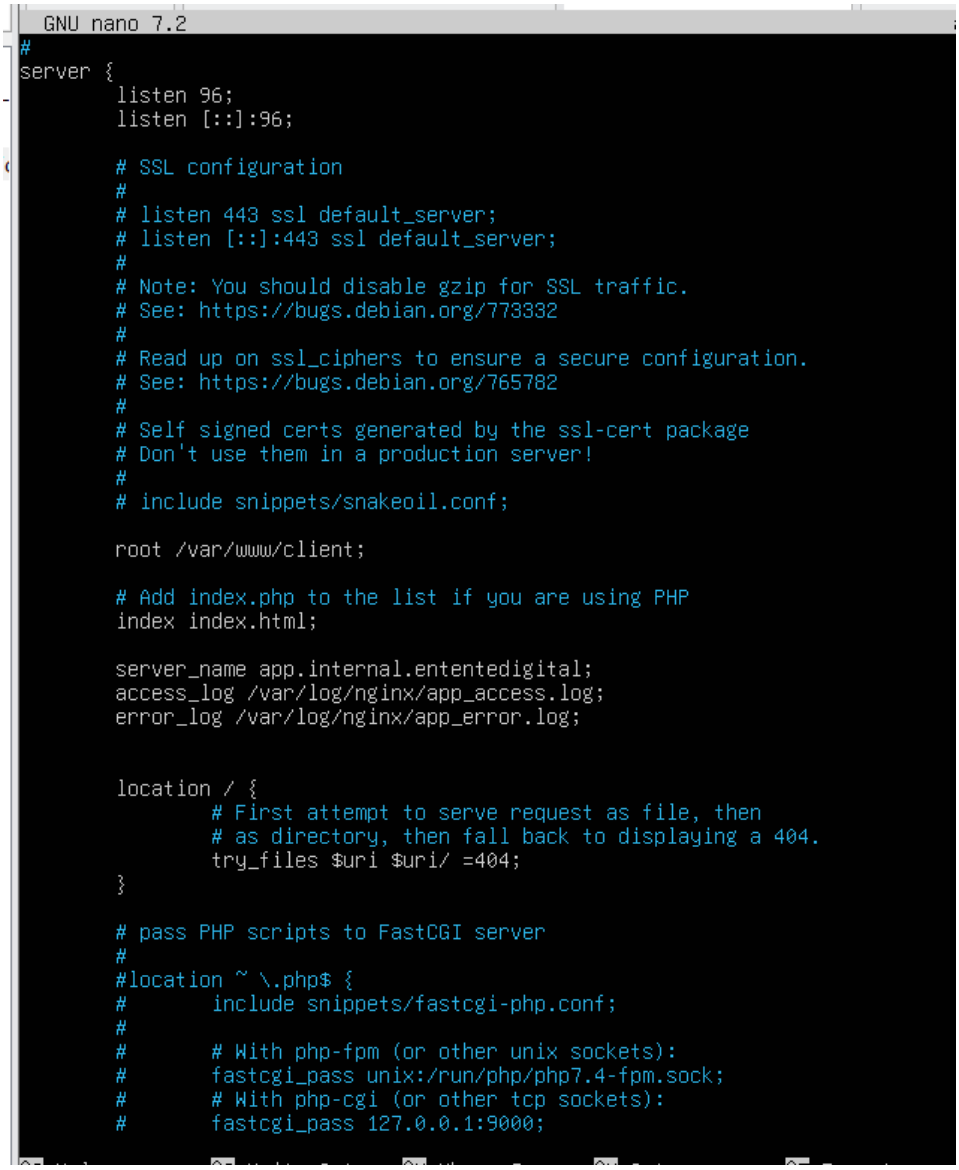


Figure 150: fichier de configuration nginx

La configuration définit les ports sur lesquels Nginx écoute les connexions. La directive *listen 96;* indique que Nginx écoute sur le port 96 pour IPv4, tandis que *listen [::]:96;* active l'écoute pour IPv6. Ce port non standard permet de séparer cette application des autres services.

Les lignes concernant le SSL sont commentées car Cloudflared gère la terminaison SSL/TLS. Les utilisateurs se connectent en HTTPS à Cloudflare, qui communique ensuite avec votre serveur via un tunnel chiffré. Nginx reçoit donc du trafic HTTP simple sur le port 96.

La directive *root /var/www/client;* définit le répertoire racine des fichiers web. La directive *index index.html;* spécifie le fichier par défaut à servir.

La directive **server_name** indique le nom de domaine interne associé à notre application public. Elle permet à Nginx d'identifier les requêtes destinées à ce service précis, notamment lorsque plusieurs applications sont hébergées sur le même serveur.

Les directives **access_log** et **error_log** définissent les fichiers de journalisation. Elles permettent de tracer les accès à l'application ainsi que les erreurs éventuelles, ce qui est essentiel pour le suivi, le diagnostic et la supervision du service.

Le bloc **location /** précise la manière dont Nginx traite les requêtes entrantes. L'instruction **try_files** indique que Nginx tente d'abord de servir la ressource demandée sous forme de fichier ou de répertoire, puis retourne une erreur 404 si la ressource n'existe pas. Cela empêche l'exposition de chemins non valides ou sensibles.

Les sections relatives au traitement des scripts PHP sont volontairement commentées. Elles montrent que le serveur est configuré pour servir une application statique ou front-end, sans exécution de scripts côté serveur, ce qui réduit la surface d'attaque et simplifie l'architecture.

3.6.4 Simulation

Une fois toutes les configurations finalisées, nous procédons à une phase de simulation en accédant à l'URL publique https://clients.ententedigital.com depuis un navigateur, afin de vérifier que le nom de domaine est correctement résolu et que l'application est bien accessible via l'infrastructure Cloudflared.

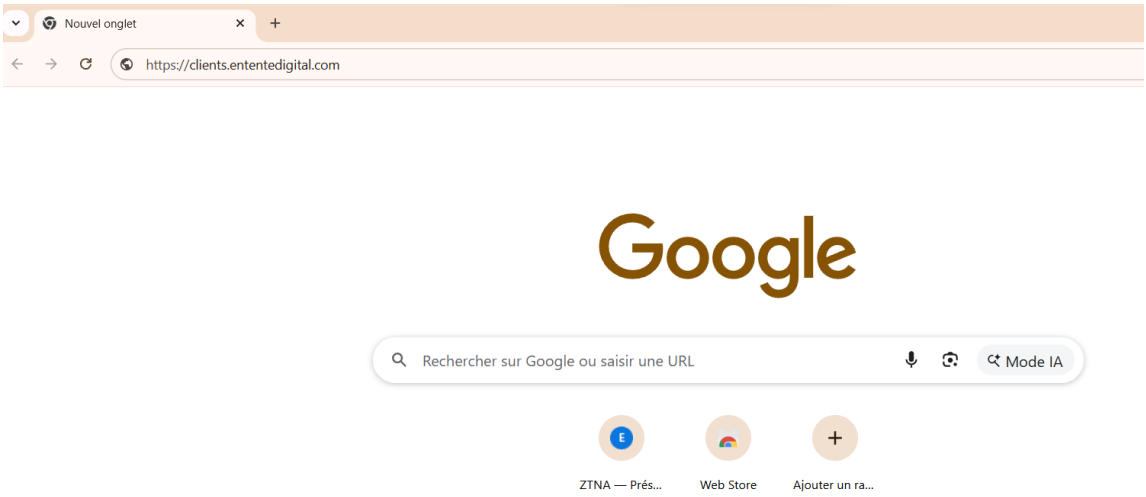


Figure 151: connexion à l'app publique



Figure 152: page d'accueil d'application

Après la résolution correcte du nom de domaine, la page de l'application s'affiche avec succès, ce qui confirme que le trafic est bien acheminé à travers le tunnel Cloudflared jusqu'au service interne et que l'application est accessible de manière sécurisée, sans exposition directe de l'infrastructure interne.

Cloudflared joue un rôle clé en servant de point d'entrée sécurisé vers les applications internes : il assure la résolution DNS, protège l'accès public et établit un tunnel chiffré vers l'infrastructure interne, permettant d'exposer les services sans adresse IP publique tout en renforçant la sécurité et la disponibilité.

Conclusion

Ce chapitre d'implémentation a permis de concrétiser l'architecture définie en amont à travers le déploiement effectif des différentes briques techniques. L'intégration de **Cloudflared Tunnel**, **Pomerium** et **Keycloak** a permis de mettre en place un accès sécurisé de type **Zero Trust**, sans exposition d'IP publique, avec une authentification centralisée et un contrôle d'accès fin basé sur les groupes et les rôles.

Par ailleurs, l'implémentation de la **supervision** avec **Prometheus**, **Node Exporter** et **Grafana** a assuré le suivi en temps réel de l'état et des performances des serveurs et des applications.

Les métriques collectées, les tableaux de bord et les règles d'alerting constituent un outil essentiel pour la détection proactive des incidents et le maintien de la disponibilité des services.

Ainsi, cette phase d'implémentation démontre la faisabilité et l'efficacité de la solution proposée, en combinant **sécurité**, **contrôle d'accès** et **observabilité**, tout en respectant le principe du moindre privilège et les bonnes pratiques des architectures modernes.

Conclusion générale

Au terme de ce travail, les trois chapitres ont permis de présenter, concevoir et mettre en œuvre une solution complète de Zero Trust Network Access (ZTNA) répondant aux exigences actuelles de sécurité des systèmes d'information. Le premier chapitre a posé les bases théoriques du modèle Zero Trust, en mettant en évidence ses principes fondamentaux, ses avantages par rapport aux solutions traditionnelles comme les VPN, ainsi que les enjeux liés à la gestion des identités, des accès et de la sécurité applicative.

Le deuxième chapitre a été consacré à la conception de l'architecture de la solution. Il a permis de décrire une architecture sécurisée et cohérente, combinant des composants essentiels tels que Keycloak pour l'authentification et la gestion des identités, Pomerium comme reverse proxy Zero Trust, NGINX pour le routage applicatif, ainsi que Cloudflared pour l'exposition sécurisée des applications publiques. Cette architecture met en œuvre le principe du moindre privilège et garantit que chaque accès est strictement contrôlé, authentifié et autorisé.

Enfin, le troisième chapitre a porté sur l'implémentation pratique de la solution. Il a détaillé l'installation, la configuration et l'intégration des différents services, ainsi que la mise en place d'une supervision complète basée sur Prometheus, Node Exporter et Grafana. Les tests réalisés ont permis de valider le bon fonctionnement des accès aux applications internes et publiques, tout en démontrant la robustesse du modèle Zero Trust. Les incidents rencontrés, notamment liés à la configuration TLS et au reverse proxy, ont également permis de mettre en évidence l'importance d'une gestion rigoureuse des certificats et des services critiques.

En conclusion, ce projet démontre que la mise en place d'une solution ZTNA est non seulement réalisable, mais également pertinente pour renforcer la sécurité des accès applicatifs. L'approche adoptée offre une meilleure visibilité, un contrôle d'accès fin et une réduction significative de la surface d'attaque, tout en restant évolutive et adaptée aux environnements modernes. Cette solution constitue ainsi une base solide pour une sécurisation durable et scalable des systèmes d'information.

Bibliographie

1. **Kim, D. & Solomon, M. G.** (2016). *Cybersecurity and Cyberwar: What Everyone Needs to Know*. Oxford University Press.
2. **Stallings, W.** (2017). *Network Security Essentials: Applications and Standards*. Pearson Education.
3. **NIST** (2020). *Special Publication 800-207 – Zero Trust Architecture*. National Institute of Standards and Technology.
4. **NIST** (2006). *Special Publication 800-92 – Guide to Computer Security Log Management*. National Institute of Standards and Technology.
5. **ISO/IEC** (2013). *ISO/IEC 27001 – Information Security Management Systems*. International Organization for Standardization.
6. **ENISA** (2022). *Threat Landscape Report*. European Union Agency for Cybersecurity.
7. **ENISA** (2021). *Zero Trust Adoption Challenges*. European Union Agency for Cybersecurity.
8. **Gartner** (2021). *ZTNA Market Guide*. Gartner Research.
9. **Gartner** (2022). *Security and Risk Management Trends*. Gartner Research.

Webographie

1. **Forrester Research** (2010). *The Zero Trust Model*.
2. **Google** (2014). *BeyondCorp: A New Approach to Enterprise Security*.
3. **Microsoft** (2018). *Microsoft Security Insights*.
4. **Mozilla Developer Network (MDN)** (2023). *Web Security Documentation*.
5. **OWASP** (2023). *Authentication Cheat Sheet*.
6. **RFC 6749** (2012). *The OAuth 2.0 Authorization Framework*.
7. **OpenID Foundation** (2014). *OpenID Connect Core Specification*.

Documentation technique et sources officielles

1. **Cloudflare** – *Cloudflare Tunnel Documentation*.
2. **Cloudflare** – *Cloudflare Security Overview*.
3. **Prometheus** – *Prometheus Documentation*.
4. **Prometheus** – *Prometheus Overview*.
5. **Grafana Labs** – *Grafana Documentation*.
6. **Grafana Labs** – *Grafana Architecture Overview*.
7. **Grafana Labs** – *Grafana Security Documentation*.