22/05/2025

Oubaid HLAIMI

ITIC PARIS

Syllabus de déploiement d’une PKI (DAT)

PROJET D’ARCHITECTURE TECHNIQUE



Table des matières

[Table des figures 3](#_Toc198844775)

[Introduction 4](#_Toc198844776)

[Contexte du projet 4](#_Toc198844777)

[Portée fonctionnelle et technique 4](#_Toc198844778)

[Portée fonctionnelle : 4](#_Toc198844779)

[Portée technique : 4](#_Toc198844780)

[Architecture générale 5](#_Toc198844781)

[Schéma global de l’architecture (physique et logique) 5](#_Toc198844782)

[Description des environnements 6](#_Toc198844783)

[Environnement Sandbox 6](#_Toc198844784)

[Environnement Recette (Staging) 6](#_Toc198844785)

[Environnements futurs et planifiés 7](#_Toc198844786)

[Répartition des rôles (serveurs, clients, pare-feu, etc.) 8](#_Toc198844787)

[Architecture réseau 9](#_Toc198844788)

[Adressage IP et segmentation réseau 10](#_Toc198844789)

[Sécurité réseau (pare-feu, NAT, groupes de sécurité) 10](#_Toc198844790)

[Routage et segmentation 13](#_Toc198844791)

[Sécurité 13](#_Toc198844792)

[Justification de l’utilisation d’une PKI hiérarchique 15](#_Toc198844793)

[Sécurisation de la Root CA (offline) 15](#_Toc198844794)

[Sécurité réseau et contrôle d’accès 15](#_Toc198844795)

[Gestion des clés SSH et accès administrateur 16](#_Toc198844796)

[Déploiement 17](#_Toc198844797)

[Étapes théoriques (concepts PKI) 17](#_Toc198844798)

[Provisionnement de l’infrastructure avec Terraform 17](#_Toc198844799)

[Configuration et déploiement avec Ansible 18](#_Toc198844800)

[Mise en place des containers Docker 19](#_Toc198844801)

[Automatisation avec le script deploy.sh 19](#_Toc198844802)

[Tests de fonctionnement 20](#_Toc198844803)

[Une image contenant texte, capture d’écran, logiciel

Le contenu généré par l’IA peut être incorrect. 23](#_Toc198844804)

[Bibliographie 24](#_Toc198844805)

# Table des figures

[Figure 1: Architecture globale 5](#_Toc198843973)

[Figure 2: Architecture réseau 9](#_Toc198843974)

[Figure 3: Two-Tier CA Hierarchy (TheSSLStore, s.d.) 14](#_Toc198843975)

[Figure 4: Processus PKI 17](#_Toc198843976)

[Figure 5: Exécution de scripts Terraform 18](#_Toc198843977)

[Figure 6: Exécution de scripts Ansible 18](#_Toc198843978)

[Figure 7: Liste des services Docker 19](#_Toc198843979)

[Figure 8: Script deploy 19](#_Toc198843980)

[Figure 9: Verifier le Résolution DNS 20](#_Toc198843981)

[Figure 10: Site web avec le domaine local 20](#_Toc198843982)

[Figure 11: Validation de Certificat SSL 20](#_Toc198843983)

[Figure 12: Logs Root CA 21](#_Toc198843984)

[Figure 13: Logs CA Intermediat 21](#_Toc198843985)

[Figure 14: Logs Web Server 22](#_Toc198843986)

[Figure 15: Logs DNS Server BIN9 22](#_Toc198843987)

[Figure 16: Les Ressources Azure 23](#_Toc198843988)

[Figure 17: Destroy les Ressources Azure 23](#_Toc198843989)

[Figure 18: Les info de compte azure 23](#_Toc198843990)

# Introduction

## Contexte du projet

**PKI**

**(Public Key Infrastructure)**, qui est une architecture cryptographique utilisée pour gérer les clés publiques et privées, assurer la sécurité des échanges, et authentifier les utilisateurs et les services via des certificats numériques. La PKI repose sur une **hiérarchie d'autorités de certification (CA)** et un ensemble de services assurant la gestion des certificats.

Dans ce cadre, le projet utilise des technologies modernes de **conteneurisation** (Docker) et des outils d'**automatisation** pour simplifier et sécuriser le déploiement, la gestion et la maintenance de l'infrastructure PKI. La solution s'appuie sur **Docker** pour containeriser les composants principaux de la PKI, permettant ainsi une gestion isolée et modulaire des différents services. L'infrastructure est automatisée à l'aide de **Terraform**, qui se charge de la création et de la gestion des ressources Azure, et d'**Ansible**, qui est utilisé pour l'automatisation de la configuration des services une fois l'infrastructure déployée.

## Portée fonctionnelle et technique

### Portée fonctionnelle :

* **Gestion des clés et certificats** : Le système permet de gérer la génération, la signature et l'émission des certificats pour les clients et serveurs, en utilisant une **Root CA** (Certificate Authority) et une **Intermediate CA**.
* **Serveur DNS interne** : Un serveur **DNS** interne est déployé pour gérer les résolutions de noms de domaines internes, facilitant la communication entre les services (par exemple, web.itic.lan).
* **Serveur Web sécurisé** : Un serveur **Web** (Nginx) est déployé pour héberger une application web, sécurisé par des certificats SSL/TLS générés par l'**Intermediate CA**.
* **Automatisation du déploiement** : L'ensemble du système PKI est déployé, configuré et testé automatiquement via **Terraform** et **Ansible**, réduisant ainsi le besoin d'interventions manuelles et permettant une gestion facile à long terme.

### Portée technique :

* **Conteneurisation avec Docker** :
  + Les services **Root CA**, **Intermediate CA**, **DNS Server**, **Web Server**, et **NAT Masquerade** sont tous exécutés dans des **conteneurs Docker** isolés. Cela permet de garantir une grande modularité et une gestion simplifiée des services tout en assurant leur isolation pour des raisons de sécurité.
  + Chaque service communique avec les autres via des **réseaux Docker dédiés**, garantissant une isolation réseau tout en permettant l'interconnexion nécessaire entre les services (par exemple, le Web Server interagit avec l'Intermediate CA pour la gestion des certificats).
* **Provisionnement avec Terraform** :
  + **Terraform** est utilisé pour provisionner l'infrastructure sur **Azure**, incluant la création de la **machine virtuelle (VM)** qui héberge les services. Terraform permet de définir l'infrastructure comme du code, facilitant ainsi sa gestion, sa réutilisation et sa mise à l'échelle.
* **Automatisation de la configuration avec Ansible** :
  + Une fois l'infrastructure en place, **Ansible** prend en charge l'installation et la configuration des différents composants PKI sur la VM. Cela inclut l'installation de **Docker**, le déploiement des **conteneurs Docker**, la configuration des **règles de pare-feu** et la mise en place de la sécurité des services via des certificats.
  + Ansible garantit également que les services sont configurés de manière cohérente et conforme aux spécifications, et qu'ils restent en bon état de fonctionnement.
* **Sécurisation de l’infrastructure** :
  + La solution met en œuvre des pratiques de sécurité modernes, telles que l’utilisation de **certificats SSL/TLS**, l'isolement des services dans des **réseaux Docker privés**, et la gestion des clés via une **PKI hiérarchique**. La **Root CA** est gardée hors ligne pour minimiser les risques, et les services critiques (comme l'Intermediate CA) sont configurés de manière sécurisée pour éviter toute compromission.

Architecture générale

## Schéma global de l’architecture (physique et logique)

A screenshot of a computer

AI-generated content may be incorrect.

Figure 1: Architecture globale

## Description des environnements

Environnement Sandbox

L'**environnement Sandbox** est mon environnement local de développement. Il est configuré sur une **VM Kali** que j'ai déployée à l'aide de **VMware Workstation**. Ce dernier me permet de **développer et de tester** tous les composants du projet PKI (Public Key Infrastructure) en utilisant des outils tels que **Terraform** et **Ansible** pour l'automatisation.

* **VM Kali Linux** : C’est l’environnement principal où je gère et développe tous les fichiers de configuration, les scripts, ainsi que les commandes Terraform et Ansible.
* **Version de sauvegarde** : Toutes mes configurations et travaux sont sauvegardés régulièrement sur **GitHub** afin de garantir la gestion de versions et la collaboration. Le lien vers le dépôt GitHub est disponible ici : [oubaidHL/Syllabus-de-d-ploiement-d-une-PKI-DAT-](https://github.com/oubaidHL/Syllabus-de-d-ploiement-d-une-PKI-DAT-)

Environnement Recette (Staging)

L'**environnement de recette (Staging)** est celui où le système PKI est déployé sur **Azure** à l’aide des scripts **Terraform** et **Ansible**. Cet environnement sert de **bassin de test** avant le déploiement en production.

Dans cet environnement, **Terraform** est utilisé pour créer et gérer l’infrastructure Azure, y compris les **ressources réseau** et la **machine virtuelle (VM)**. L'infrastructure comprend également les configurations nécessaires pour installer **Docker**, ainsi que pour déployer les **conteneurs PKI** via **Ansible**.

Voici un extrait de la configuration **Terraform** utilisée pour déployer l’infrastructure :

terraform {

  required\_providers {

    azurerm = {

      source  = "hashicorp/azurerm"

      version = "~> 4.0"

    }

  }

}

provider "azurerm" {

  features {}

  subscription\_id = var.subscription\_id

  resource\_provider\_registrations = "none"

}

resource "azurerm\_resource\_group" "rg" {

  name     = "dat-cyber-rg"

  location = var.location

}

resource "azurerm\_virtual\_network" "vnet" {

  name                = "dat-cyber-vnet"

  address\_space       = ["10.0.0.0/16"]

  location            = azurerm\_resource\_group.rg.location

  resource\_group\_name = azurerm\_resource\_group.rg.name

}

resource "azurerm\_network\_security\_group" "nsg" {

  name                = "dat-cyber-nsg"

  location            = azurerm\_resource\_group.rg.location

  resource\_group\_name = azurerm\_resource\_group.rg.name

  ...

}

resource "azurerm\_linux\_virtual\_machine" "vm" {

  name                  = "dat-cyber-vm"

  resource\_group\_name    = azurerm\_resource\_group.rg.name

  location              = azurerm\_resource\_group.rg.location

  size                  = var.vm\_size

  admin\_username        = var.admin\_username

  network\_interface\_ids = [azurerm\_network\_interface.nic.id]

  ...

}

(OUBAIDHL, s.d.)

Cet environnement est dédié à la validation des scripts de provisionnement d’infrastructure et à la configuration des services. Il permet de vérifier si tout fonctionne comme prévu avant le passage en production.

Environnements futurs et planifiés

Pour le **déploiement en production** dans un contexte réel, je recommande de configurer plusieurs environnements pour gérer les différentes phases du projet. Ces environnements sont :

* **Dev** : Environnement de développement pour les tests unitaires et l'intégration des nouvelles fonctionnalités.
* **Test** : Environnement de recette pour tester les fonctionnalités dans un environnement similaire à la production.
* **Prod** : Environnement de production qui déploie la solution PKI pour l'utilisation réelle par les utilisateurs finaux.

L'architecture de ces environnements peut être gérée avec des **fichiers Terraform** séparés pour chaque environnement (par exemple, dev/main.tf, test/main.tf, prod/main.tf). En outre, l’intégration de l’automatisation CI/CD (via **GitLab CI**, **Jenkins**, ou **GitHub DevOps …**) permettra de gérer ces différents environnements et de mettre à jour les variables de configuration (comme les adresses IP, les ports, etc.).

## Répartition des rôles (serveurs, clients, pare-feu, etc.)

Dans ce projet, les services sont déployés en utilisant **Docker Compose** et sont **modularisés** pour faciliter la gestion et l’extension des différents composants. Voici une description détaillée des rôles des différents services et de la manière dont ils sont connectés dans l’architecture :

1. **Serveur DNS** (dns-server) :
   * Rôle : Fournit la **résolution des noms de domaine** internes comme web.itic.lan. Il est essentiel pour assurer la communication interne entre les différents services PKI.
   * Docker : Exécuté dans un conteneur Docker et exposé sur le port **53 (UDP)**.
2. **Root CA (Certificate Authority)** :
   * Rôle : La **Root CA** est responsable de la signature des certificats de la **Intermediate CA** et est **hors ligne** pour des raisons de sécurité.
   * Docker : Exécuté dans un conteneur Docker. Aucun port n’est exposé publiquement pour la **Root CA**.
3. **Intermediate CA** :
   * Rôle : La **Intermediate CA** signe les certificats pour les clients et serveurs. Elle est **en ligne** et peut être accédée par d'autres services pour la validation des certificats.
   * Docker : Exécuté dans un conteneur Docker et exposé sur le port **9000** (HTTPS).
4. **Web Server** (web-server) :
   * Rôle : Le serveur Web (Nginx) sert l’application sécurisée en utilisant les certificats SSL/TLS fournis par l'**Intermediate CA**.
   * Docker : Exécuté dans un conteneur Docker et exposé sur les ports **80 (HTTP)** et **443 (HTTPS)**.
5. **NAT Masquerade** (nat-masquerade) :
   * Rôle : Ce conteneur est responsable du **masquage de l'adresse IP** et de la **redirection des ports** pour faire passer le trafic entrant vers les services internes.
   * Docker : Exposé sur plusieurs ports, y compris **53 (DNS)**, **80 (HTTP)**, **443 (HTTPS)**, et d'autres ports spécifiques pour le service.

Architecture réseau

Dans cette section, nous allons détailler l'**architecture réseau** de notre solution PKI, en mettant l'accent sur l'adressage IP, la segmentation du réseau, la sécurité du réseau, et le routage entre les différents composants. Ces éléments sont essentiels pour garantir que les services fonctionnent correctement tout en restant sécurisés et isolés les uns des autres.

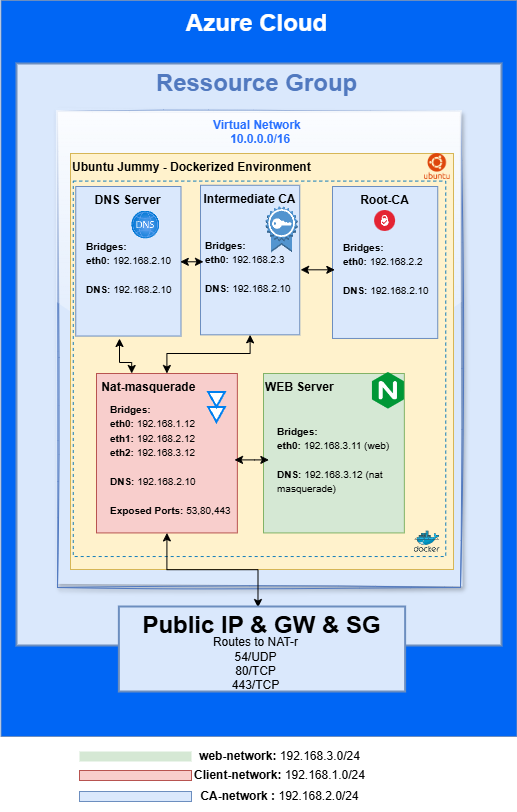


Figure 2: Architecture réseau

## Adressage IP et segmentation réseau

L'**adressage IP** et la **segmentation** du réseau sont utilisés pour garantir une communication correcte tout en isolant les services afin de renforcer la sécurité.

* **ca-network** : Ce réseau est dédié aux services **PKI** internes (Root CA, Intermediate CA, DNS). Ces services sont isolés des autres services exposés.
  + **Plage d'IP** : 192.168.2.0/24
  + **Exemple d'IP** :
    - dns-server : 192.168.2.10
    - root-ca : 192.168.2.2
    - intermediate-ca : 192.168.2.3
* **web-network** : Ce réseau est utilisé pour exposer le **serveur Web** (nginx), qui est accessible depuis l'extérieur.
  + **Plage d'IP** : 192.168.3.0/24
  + **Exemple d'IP** : web-server : 192.168.3.11
* **client-network** : Réseau externe pour gérer le trafic entrant et sortant via le **NAT Masquerade**.
  + **Plage d'IP** : 192.168.1.0/24
  + **Exemple d'IP** : nat-masquerade : 192.168.1.12

**Réseaux Docker dans docker-compose.yml :**

networks:

  client-network:

    driver: bridge

    ipam:

      config:

        - subnet: 192.168.1.0/24

  ca-network:

    driver: bridge

    ipam:

      config:

        - subnet: 192.168.2.0/24

  web-network:

    driver: bridge

    ipam:

      config:

        - subnet: 192.168.3.0/24

(OUBAIDHL, s.d.)

## Sécurité réseau (pare-feu, NAT, groupes de sécurité)

**Pare-feu et règles de sécurité Azure :**

Les services qui sont exposés à l'extérieur, comme le **Web Server** et le **DNS Server**, sont protégés par des **règles de pare-feu Azure** (NSG - Network Security Groups). Ces règles définissent les ports autorisés pour chaque service.

1. **Ports exposés** :
   * **Port 53 (UDP)** : pour les requêtes DNS.
   * **Port 80 (HTTP)** et **Port 443 (HTTPS)** : pour le serveur web.
   * **Ports 9000 et 9001** : pour des services spécifiques (comme la gestion des certificats).
2. **Règles de sécurité** dans **Terraform** pour gérer ces accès :

resource "azurerm\_network\_security\_group" "nsg" {

  name                = "dat-cyber-nsg"

  location            = azurerm\_resource\_group.rg.location

  resource\_group\_name = azurerm\_resource\_group.rg.name

  security\_rule {

    name                       = "Allow-DNS-UDP"

    priority                   = 100

    direction                  = "Inbound"

    access                     = "Allow"

    protocol                   = "Udp"

    source\_port\_range          = "\*"

    destination\_port\_range     = "53"

    source\_address\_prefix      = "\*"

    destination\_address\_prefix = "\*"

  }

  security\_rule {

    name                       = "Allow-HTTP"

    priority                   = 110

    direction                  = "Inbound"

    access                     = "Allow"

    protocol                   = "Tcp"

    source\_port\_range          = "\*"

    destination\_port\_range     = "80"

    source\_address\_prefix      = "\*"

    destination\_address\_prefix = "\*"

  }

  security\_rule {

    name                       = "Allow-HTTPS"

    priority                   = 120

    direction                  = "Inbound"

    access                     = "Allow"

    protocol                   = "Tcp"

    source\_port\_range          = "\*"

    destination\_port\_range     = "443"

    source\_address\_prefix      = "\*"

    destination\_address\_prefix = "\*"

  }

  security\_rule {

    name                       = "Allow-SSH"

    priority                   = 150

    direction                  = "Inbound"

    access                     = "Allow"

    protocol                   = "Tcp"

    source\_port\_range          = "\*"

    destination\_port\_range     = "22"

    source\_address\_prefix      = "\*"

    destination\_address\_prefix = "\*"

  }

}

(OUBAIDHL, s.d.)

1. **NAT Masquerade** : Ce conteneur est responsable de **rediriger le trafic** des ports ouverts sur la machine hôte (port 53, 80, 443) vers les conteneurs appropriés, tout en assurant la gestion des règles NAT via **iptables**. Il sert de **passerelle** entre l'extérieur et les services internes.

**iptables** pour redirection des ports dans nat-masquerade :

# Port forwarding rules

iptables -t nat -A PREROUTING -i eth0 -p udp --dport 53 -j DNAT --to-destination 192.168.2.10:53

iptables -t nat -A PREROUTING -i eth1 -p udp --dport 53 -j DNAT --to-destination 192.168.2.10:53

iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 9000 -j DNAT --to-destination 192.168.2.2:9000

iptables -t nat -A PREROUTING -i eth1 -p tcp --dport 9000 -j DNAT --to-destination 192.168.2.2:9000

iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 9001 -j DNAT --to-destination 192.168.2.3:9000

iptables -t nat -A PREROUTING -i eth1 -p tcp --dport 9001 -j DNAT --to-destination 192.168.2.3:9000

iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 80 -j DNAT --to-destination 192.168.3.11:80

iptables -t nat -A PREROUTING -i eth1 -p tcp --dport 80 -j DNAT --to-destination 192.168.3.11:80

iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 443 -j DNAT --to-destination 192.168.3.11:443

iptables -t nat -A PREROUTING -i eth1 -p tcp --dport 443 -j DNAT --to-destination 192.168.3.11:443

(OUBAIDHL, s.d.)

## Routage et segmentation

Le **routage** dans cette architecture est assuré par **NAT Masquerade**, qui effectue la redirection des ports entre les différents réseaux Docker. Cela permet aux services internes d’être **isolés** tout en permettant un accès contrôlé aux services exposés.

**Flux Réseau entre les Composants :**

1. **Trafic interne** : Le trafic entre les services dans le **réseau ca-network** (comme root-ca, intermediate-ca, dns-server) ne traverse pas le NAT et reste localisé.
2. **Trafic externe** : Les utilisateurs externes accèdent au **serveur Web** via les ports **80** et **443**, tandis que les requêtes DNS sont envoyées au **serveur DNS** via le port **53**.
3. **Redirection de trafic** : Le conteneur **NAT Masquerade** gère le **forwarding de ports** pour que les services internes soient accessibles depuis l’extérieur, sans exposer directement les autres services internes comme **Root CA** et **Intermediate CA**.

🡪 L'architecture réseau est configurée de manière à isoler les services sensibles tout en permettant une **communication sécurisée** entre eux. Le **NAT Masquerade** joue un rôle clé dans la gestion du **routage** et de la **redirection des ports**, permettant ainsi un accès contrôlé et sécurisé aux services externes (DNS et Web). Les règles de sécurité Azure et les configurations Docker garantissent que chaque service est correctement isolé et que l'accès est limité aux ports nécessaires.

Sécurité

La sécurité dans une infrastructure PKI (Public Key Infrastructure) est primordiale pour garantir l'intégrité et la confidentialité des communications. Dans ce projet, la sécurité a été abordée sous plusieurs aspects, y compris la structure de la PKI hiérarchique, la protection de la Root CA, la sécurisation des réseaux et du contrôle d’accès, ainsi que la gestion des clés SSH et des accès administrateurs.

L'usage d'une **PKI hiérarchique** permet d'améliorer la sécurité en structurant l'autorité de certification (CA) en plusieurs niveaux. Dans ce projet, la **Root CA** est **hors ligne**, ce qui minimise le risque de compromission. L'**Intermediate CA**, quant à elle, est en ligne et responsable de l’émission des certificats pour les clients et serveurs.

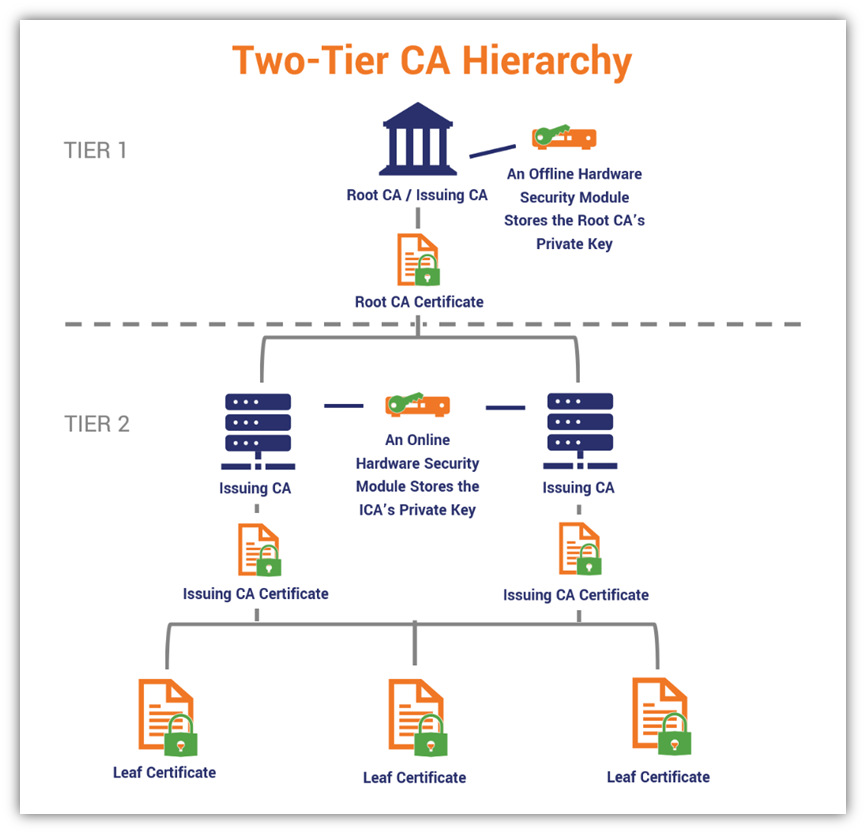


Figure : Two-Tier CA Hierarchy (TheSSLStore, s.d.)

* **Root CA (offline)** : La **Root CA** est conçue pour être **hors ligne** afin de la protéger contre les attaques directes. Elle est utilisée uniquement pour signer les **Intermediate CAs** et n'est pas directement accessible depuis l'extérieur.
* **Intermediate CA (online)** : L'**Intermediate CA** est utilisée pour émettre des certificats pour les serveurs ou les utilisateurs. Elle repose sur la confiance de la **Root CA** pour garantir la validité de ses certificats.
* **Avantages de la hiérarchie** :
  + **Isolation des clés privées** : La **Root CA** ne sera jamais exposée aux attaques directes. Cela protège les clés privées utilisées pour la signature des certificats.
  + **Séparation des rôles** : La **Root CA** reste statique, tandis que l'**Intermediate CA** peut être plus dynamique et peut être utilisée pour signer des certificats pour des services spécifiques.

## Justification de l’utilisation d’une PKI hiérarchique

L'usage d'une **PKI hiérarchique** permet d'améliorer la sécurité en structurant l'autorité de certification (CA) en plusieurs niveaux. Dans ce projet, la **Root CA** est **hors ligne**, ce qui minimise le risque de compromission. L'**Intermediate CA**, quant à elle, est en ligne et responsable de l’émission des certificats pour les clients et serveurs.

* **Root CA (offline)** : La **Root CA** est conçue pour être **hors ligne** afin de la protéger contre les attaques directes. Elle est utilisée uniquement pour signer les **Intermediate CAs** et n'est pas directement accessible depuis l'extérieur.
* **Intermediate CA (online)** : L'**Intermediate CA** est utilisée pour émettre des certificats pour les serveurs ou les utilisateurs. Elle repose sur la confiance de la **Root CA** pour garantir la validité de ses certificats.
* **Avantages de la hiérarchie** :
  + **Isolation des clés privées** : La **Root CA** ne sera jamais exposée aux attaques directes. Cela protège les clés privées utilisées pour la signature des certificats.
  + **Séparation des rôles** : La **Root CA** reste statique, tandis que l'**Intermediate CA** peut être plus dynamique et peut être utilisée pour signer des certificats pour des services spécifiques.

## Sécurisation de la Root CA (offline)

La **Root CA** joue un rôle crucial dans la sécurité globale de l'infrastructure PKI. Dans ce projet, la **Root CA** est **offline**, ce qui signifie qu'elle n'est **pas connectée à Internet** et ne peut être directement compromise. Cela augmente considérablement la sécurité du système.

* **Protocole de signature sécurisé** : La **Root CA** est utilisée uniquement pour signer des **certificats d'autorités intermédiaires** et des certificats pour des services internes spécifiques. Toute utilisation d’une **Root CA** nécessite une procédure manuelle et sécurisée pour garantir que le processus de signature ne soit pas exposé.
* **Stockage des clés privées** : Les clés privées de la **Root CA** sont stockées dans un environnement très sécurisé, avec des mesures physiques (p. ex., dans un **HSM - Hardware Security Module**) et des pratiques rigoureuses de contrôle d’accès.
* **Protection physique et logicielle** : La **Root CA** est protégée dans un environnement **isolé**, et son utilisation nécessite une authentification forte et une surveillance continue.

## Sécurité réseau et contrôle d’accès

La sécurité réseau et le contrôle d’accès sont des éléments essentiels pour éviter les accès non autorisés et protéger les données circulantes entre les services internes.

* **Contrôle d’accès via Azure** : Des **règles de sécurité Azure** (NSG) sont appliquées pour garantir que seules les connexions nécessaires puissent accéder aux ressources de l'infrastructure. Par exemple, seul le **port 22 (SSH)** est autorisé pour l'accès administrateur à la **VM Azure**.
  + **Redirection de ports limitée** : Les ports comme **53 (DNS)**, **80 (HTTP)**, et **443 (HTTPS)** sont redirigés via le conteneur **NAT Masquerade** pour contrôler et filtrer les connexions entrantes.
  + **Utilisation d'IP privées** : Les services internes sont configurés avec des **adresses IP privées**, garantissant qu'ils ne soient pas accessibles depuis l'extérieur sans passer par des points de contrôle comme le **NAT Masquerade**.
* **Réseaux Docker isolés** : Chaque service fonctionne dans un **réseau Docker** dédié et isolé. Par exemple, le **Root CA** et l'**Intermediate CA** sont isolés dans le réseau **ca-network**, tandis que le serveur web est situé dans un autre réseau (par exemple, **web-network**). Cette **segmentation du réseau** empêche la communication non autorisée entre les services.
* **Firewall (iptables)** : Des règles **iptables** sont appliquées pour gérer les connexions entre les conteneurs, assurant que le **trafic réseau** est redirigé de manière sécurisée.

## Gestion des clés SSH et accès administrateur

La gestion des **clés SSH** et le contrôle d'accès des administrateurs sont des éléments essentiels pour sécuriser l'accès à la **machine virtuelle (VM)** Azure et à l'infrastructure en général.

* **Accès à la VM via clé SSH** : L'accès à la **VM Azure** est **restreint** et nécessite une **clé SSH privée**. Cette clé est stockée localement sur l’appareil de l'administrateur (moi), ce qui garantit que l’accès à la VM est uniquement possible pour l'administrateur désigné.
  + **Pas d'accès par mot de passe** : L’accès via mot de passe est **désactivé** afin de minimiser les risques d'attaque par force brute.
  + **Clé privée sécurisée** : La clé privée est protégée par des **permissions strictes** sur le système de fichiers (par exemple, chmod 600 ~/.ssh/id\_rsa).
* **Terraform et Ansible** : L’accès à l’infrastructure est également sécurisé par **Terraform** et **Ansible**. Ces outils utilisent des variables pour gérer les configurations (comme le **chemin de la clé publique SSH** dans **Terraform**), et l'accès est contrôlé via des scripts automatisés, réduisant ainsi le besoin d'interventions manuelles.

🡪 **Résumé des mesures de sécurité**

* **PKI Hiérarchique** : Utilisation de **Root CA (offline)** et **Intermediate CA** pour garantir une sécurité maximale des certificats et des signatures.
* **Root CA (offline)** : L'isolement de la **Root CA** garantit une meilleure sécurité contre les attaques.
* **Sécurité réseau** : Utilisation de **NSG Azure**, **réseaux Docker isolés**, et **règles de pare-feu** pour limiter l'accès aux services et ressources critiques.
* **Contrôle d’accès** : Accès administrateur strictement contrôlé via des **clés SSH** privées stockées localement, avec des permissions adaptées.

Déploiement

## Étapes théoriques (concepts PKI)

Cette partie rappelle les fondamentaux de la PKI :

* Architecture hiérarchique avec **Root CA** hors ligne et **Intermediate CA** en ligne.
* Processus d’émission et de signature des certificats numériques.
* Rôle du serveur DNS pour la résolution interne des noms de domaine.
* Importance de la sécurisation des clés privées et des flux.

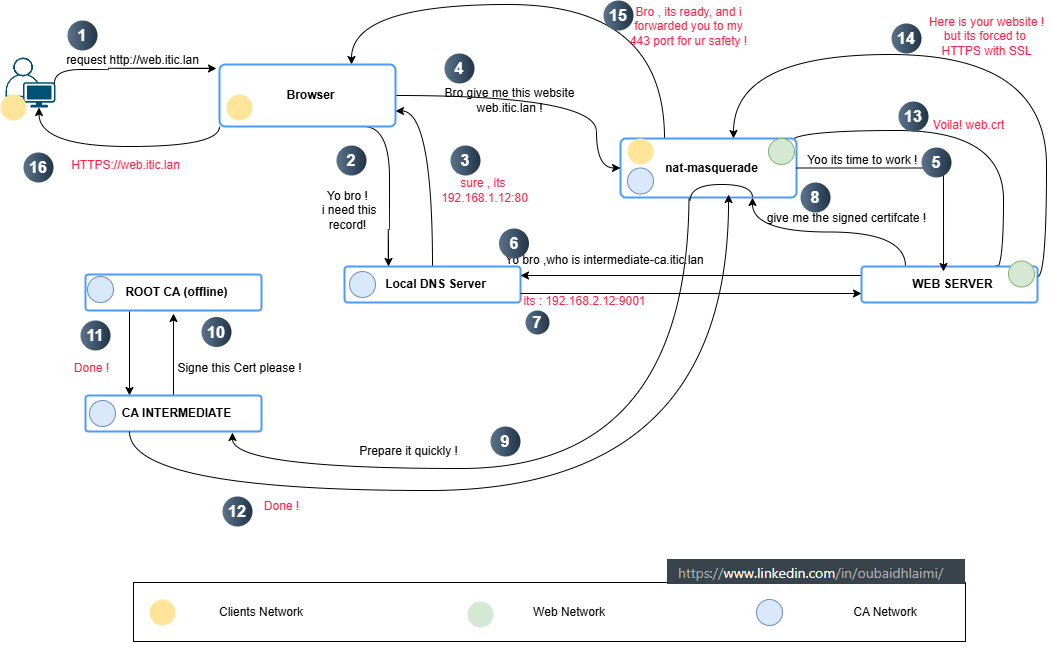


Figure : Processus PKI

## Provisionnement de l’infrastructure avec Terraform

* Déploiement de la VM Azure et du réseau.
* Création du groupe de ressources, réseau virtuel, sous-réseaux, interface réseau, règles NSG.
* Variables configurables : emplacement, taille VM, clés SSH.
* Résultat attendu : VM accessible avec IP publique.

Une image contenant texte, capture d’écran

Le contenu généré par l’IA peut être incorrect.

Figure 5: Exécution de scripts Terraform

## Configuration et déploiement avec Ansible

* Gestion dynamique de l’inventaire Ansible avec l’IP de la VM.
* Automatisation de l’installation de Docker et les autres outils sur la VM.
* Déploiement des conteneurs PKI via playbooks et rôles Ansible.
* Vérification des états de services post-déploiement.

Une image contenant texte, capture d’écran

Le contenu généré par l’IA peut être incorrect.

Figure 6: Exécution de scripts Ansible

## Mise en place des containers Docker

Conteneurs déployés : Root CA, Intermediate CA, DNS Server, Web Server, NAT Masquerade.  
Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Le contenu généré par l’IA peut être incorrect.

Figure 7: Liste des services Docker

## Automatisation avec le script deploy.sh

* Script bash orchestrant la génération de clés SSH, le provisionnement avec Terraform et le déploiement avec Ansible.
* Gestion automatique de l’attente de l’IP publique de la VM.
* Simplification du déploiement en une seule commande.

Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.

Figure 8: Script deploy

## Tests de fonctionnement

* Résolution DNS interne via nslookup ou dig sur web.itic.lan.

Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.

Figure 9: Verifier le Résolution DNS

* Accès HTTPS au serveur web avec vérification du certificat.

Une image contenant capture d’écran, logiciel, Logiciel multimédia, Logiciel de graphisme

Le contenu généré par l’IA peut être incorrect.

Figure 10: Site web avec le domaine local

* Validation des certificats via openssl et visualisation dans navigateur.

Une image contenant texte, logiciel, Icône d’ordinateur, Page web

Le contenu généré par l’IA peut être incorrect.

Figure 11: Validation de Certificat SSL

* Vérification des logs et de l’état des conteneurs.

Une image contenant texte, capture d’écran, logiciel, Icône d’ordinateur

Le contenu généré par l’IA peut être incorrect.

Figure : Logs Root CA

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Le contenu généré par l’IA peut être incorrect.

Figure : Logs CA Intermediat

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Le contenu généré par l’IA peut être incorrect.

Figure : Logs Web Server

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Le contenu généré par l’IA peut être incorrect.

Figure 15: Logs DNS Server BIN9

Une image contenant texte, logiciel, Icône d’ordinateur, Page web

Le contenu généré par l’IA peut être incorrect.

Figure : Les Ressources Azure

Une image contenant texte, capture d’écran, logiciel

Le contenu généré par l’IA peut être incorrect.

Figure 17: Destroy les Ressources Azure

Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.

Figure 18: Les info de compte azure

# Bibliographie

OUBAIDHL. (s.d.). *projet*. Récupéré sur https://github.com/oubaidHL/Syllabus-de-d-ploiement-d-une-PKI-DAT-

TheSSLStore. (s.d.). *ssl*. Récupéré sur https://www.thesslstore.com/blog/pki-architecture-fundamentals-of-designing-a-private-pki-system/

Ressources utilisées  
Toutes les ressources utilisées dans ce projet (codes, diagrammes, documentation) sont disponibles dans le dépôt GitLab suivant :

📂 **Dépôt GitLab** : <https://github.com/oubaidHL/Syllabus-de-d-ploiement-d-une-PKI-DAT->

Vous y trouverez :

* 📄 **Le rapport complet** au format Microsoft Word (Rapport\_PKI.docx)
* 📊 **Les diagrammes** au format PNG et le format **Draw.io**
* 💻 **Les scripts et codes sources** utilisés pour le déploiement de la PKI

🔹 **Vous êtes libre de :**

* **Mettre à jour** les ressources pour les améliorer.
* **Les utiliser à des fins personnelles ou académiques** (merci de citer la source).

💡 **Si ce projet vous a été utile, n'hésitez pas à :**

* ⭐ **Laisser une star** sur le dépôt GitHub pour soutenir le travail.
* 📢 **Recommander mon profil LinkedIn** : [linkedin.com/in/oubaidhlaimi](https://www.linkedin.com/in/oubaidhlaimi/) – vos retours sont précieux !