🚀 GUIDE COMPLET - INFRASTRUCTURE RÉSEAU SÉCURISÉE POUR AGRITECH BLOCKCHAIN

📋 APERÇU GÉNÉRAL DU PROJET

Ce guide détaillé vous accompagnera dans la mise en place d'une infrastructure réseau complète pour votre projet AgriTech Blockchain, répondant aux exigences académiques du module d'Administration Réseau du professeur Azeddine Khiat tout en posant les bases de votre future solution commerciale.

🎯 PRÉREQUIS TECHNIQUES

Matériel requis

- **PC/Serveur**: 8GB RAM minimum, 4 cœurs CPU, 100GB espace disque
- **VirtualBox/VMware**: Pour héberger les machines virtuelles
- **Réseau**: Connexion internet stable

Logiciels requis

- **Hyperviseur**: VirtualBox 6.1+ ou VMware Workstation 16+
- **Systèmes d'exploitation**: Ubuntu Server 22.04 LTS (3 machines)
- **Outils**: Docker, Docker-Compose, Git, Python 3.9+

📅 PLAN DE TRAVAIL HEBDOMADAIRE

🗓️ SEMAINE 1: CONCEPTION ET PRÉPARATION

Jour 1-2: Préparation de l'environnement

1. Configuration des machines virtuelles

```
# Création de 3 VMs avec les spécifications suivantes:
# VM1: Zone Terrain (1GB RAM, 1 CPU, 20GB disque)
# VM2: Zone Centrale (4GB RAM, 2 CPU, 50GB disque)
# VM3: Zone Clients (2GB RAM, 1 CPU, 30GB disque)
```

2. Installation d'Ubuntu Server 22.04 LTS sur chaque VM

Téléchargez l'ISO: https://ubuntu.com/download/server

3. Configuration réseau initiale des VMs

```
# Sur chaque VM, éditez le fichier netplan:
sudo nano /etc/netplan/00-installer-config.yaml
```

Configuration pour la Zone Terrain (VM1):

```
network:
  version: 2
  ethernets:
    enp0s3:
      dhcp4: no
      addresses: [192.168.10.10/24]
      gateway4: 192.168.10.1
      nameservers:
        addresses: [8.8.8.8, 8.8.4.4]
```

Configuration pour la Zone Centrale (VM2):

```
network:
  version: 2
  ethernets:
    enp0s3:
      dhcp4: no
      addresses: [192.168.20.10/24]
      gateway4: 192.168.20.1
      nameservers:
        addresses: [8.8.8.8, 8.8.4.4]
```

Configuration pour la Zone Clients (VM3):

```
network:
  version: 2
  ethernets:
    enp0s3:
      dhcp4: no
      addresses: [192.168.30.10/24]
      gateway4: 192.168.30.1
      nameservers:
        addresses: [8.8.8.8, 8.8.4.4]
```

4. Appliquer les configurations réseau

```
sudo netplan apply
```

Jour 3-4: Conception de l'architecture réseau détaillée

1. Créez un document d'architecture détaillé

Document à créer: `architecture_reseau.md`

```
# Architecture Réseau AgriTech Blockchain

## Zones réseau
1. **Zone Terrain** (192.168.10.0/24)
   - Capteurs IoT simulés
   - Gateway de terrain

2. **Zone Centrale** (192.168.20.0/24)
   - Serveur MQTT Broker
   - Nœuds Blockchain
   - Serveur de monitoring

3. **Zone Clients** (192.168.30.0/24)
   - Serveur API Gateway
   - Interface Web/Mobile

## Flux de données
- Capteurs → MQTT Broker → Blockchain → API Gateway → Clients
- Flux monitoring: Toutes zones → Serveur monitoring (Grafana/Prometheus)

## Mesures de sécurité
- VPN entre zones
- Pare-feu sur chaque VM
- PKI pour l'authentification des capteurs
- TLS pour communications MQTT
```

2. Mise en place du routage entre les zones

Sur chaque VM, activer le forwarding IP:

```
sudo sysctl -w net.ipv4.ip_forward=1
sudo echo "net.ipv4.ip_forward=1" >> /etc/sysctl.conf
```

Jour 5: Installation des prérequis sur toutes les VMs

1. Mise à jour des systèmes

```
sudo apt update && sudo apt upgrade -y
```

2. Installation des outils communs

```
sudo apt install -y git curl vim net-tools htop iftop tcpdump
```

3. Installation de Docker et Docker-Compose

```
# Docker
sudo apt install -y apt-transport-https ca-certificates curl software-properties-common
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
sudo apt update
sudo apt install -y docker-ce docker-ce-cli containerd.io

# Docker-Compose
sudo curl -L "https://github.com/docker/compose/releases/download/v2.17.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/
sudo chmod +x /usr/local/bin/docker-compose

# Ajouter utilisateur au groupe Docker
sudo usermod -aG docker $USER
newgrp docker
```

SEMAINE 2: INFRASTRUCTURE DE BASE

Jour 1-2: Déploiement du MQTT Broker (Zone Centrale - VM2)

1. Installation de Mosquitto MQTT

```
# Sur VM2 (Zone Centrale)
sudo apt install -y mosquitto mosquitto-clients
```

2. Configuration de Mosquitto avec authentification

```
sudo nano /etc/mosquitto/conf.d/default.conf
```

Ajoutez:

```
listener 1883
allow_anonymous false
password_file /etc/mosquitto/passwd
```

3. Création des utilisateurs MQTT

```
sudo mosquitto_passwd -c /etc/mosquitto/passwd admin
# Entrez le mot de passe lorsque demandé
sudo mosquitto_passwd /etc/mosquitto/passwd sensor1
# Entrez le mot de passe lorsque demandé
```

4. Redémarrage du service Mosquitto

```
sudo systemctl restart mosquitto
```

5. Test du broker MQTT

```
# S'abonner à un topic
mosquitto_sub -h localhost -p 1883 -u admin -P votre_mot_de_passe -t "test/topic"

# Dans un autre terminal, publier un message
mosquitto_pub -h localhost -p 1883 -u admin -P votre_mot_de_passe -t "test/topic" -m "Hello MQTT"
```

Jour 3-4: Configuration des capteurs simulés (Zone Terrain - VM1)

1. Installation des dépendances Python pour les capteurs simulés

```
# Sur VM1 (Zone Terrain)
sudo apt install -y python3-pip
pip3 install paho-mqtt python-dotenv
```

2. Création du script pour simuler les capteurs

```
mkdir -p ~/agritech/sensors
cd ~/agritech/sensors
nano sensor_simulator.py
```

Contenu du fichier:

```
import paho.mqtt.client as mqtt
import time
import random
import json
from datetime import datetime

# Configuration MQTT
MQTT_BROKER = "192.168.20.10"
MQTT_PORT = 1883
MQTT_USER = "sensor1"
MQTT_PASSWORD = "votre_mot_de_passe"
MQTT_TOPIC = "agritech/sensors/data"

# ID du capteur
SENSOR_ID = "SENSOR_001"
SENSOR_LOCATION = "Farm_Souss_Massa"
SENSOR_TYPE = "Environmental"

# Connexion au broker MQTT
client = mqtt.Client()
client.username_pw_set(MQTT_USER, MQTT_PASSWORD)

def connect_mqtt():
    try:
        client.connect(MQTT_BROKER, MQTT_PORT, 60)
        print(f"Connecté au broker MQTT: {MQTT_BROKER}")
        return True
    except Exception as e:
        print(f"Erreur de connexion au broker MQTT: {e}")
        return False

def simulate_sensor_data():
    # Simuler des données environnementales
    temperature = round(random.uniform(18.0, 35.0), 2)
    humidity = round(random.uniform(30.0, 80.0), 2)
    soil_moisture = round(random.uniform(20.0, 60.0), 2)

    timestamp = datetime.now().isoformat()

    # Créer un message JSON
    message = {
        "sensor_id": SENSOR_ID,
        "location": SENSOR_LOCATION,
```

```
        "location": SENSOR_LOCATION,
        "timestamp": timestamp,
        "data": {
            "temperature": temperature,
            "humidity": humidity,
            "soil_moisture": soil_moisture
        }
    }

    return json.dumps(message)

def main():
    if connect_mqtt():
        while True:
            try:
                # Générer données simulées
                payload = simulate_sensor_data()

                # Publier sur MQTT
                result = client.publish(MQTT_TOPIC, payload)

                if result.rc == 0:
                    print(f"Message envoyé: {payload}")
                else:
                    print(f"Échec envoi message, code: {result.rc}")

                # Attendre avant prochain envoi
                time.sleep(30)
            except Exception as e:
                print(f"Erreur: {e}")
                time.sleep(10)

if __name__ == "__main__":
    main()
```

3. Créer un service systemd pour lancer le simulateur au démarrage

```
sudo nano /etc/systemd/system/sensor-simulator.service
```

Contenu du fichier:

```
[Unit]
Description=AgriTech IoT Sensor Simulator
After=network.target

[Service]
User=ubuntu
WorkingDirectory=/home/ubuntu/agritech/sensors
ExecStart=/usr/bin/python3 /home/ubuntu/agritech/sensors/sensor_simulator.py
Restart=always
RestartSec=10

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

4. Activer et démarrer le service

```
sudo systemctl daemon-reload
sudo systemctl enable sensor-simulator
sudo systemctl start sensor-simulator
```

Jour 5: Configuration du collecteur MQTT (Zone Centrale - VM2)

1. Création du script pour collecter et stocker les données MQTT

```
 # Sur VM2 (Zone Centrale)
mkdir -p ~/agritech/mqtt_collector
cd ~/agritech/mqtt_collector
nano mqtt_collector.py
```

Contenu du fichier:

```
 import paho.mqtt.client as mqtt
import json
import time
import os
from datetime import datetime

# Configuration MQTT
MQTT_BROKER = "localhost"
MQTT_PORT = 1883
MQTT_USER = "admin"
MQTT_PASSWORD = "votre_mot_de_passe"
MQTT_TOPIC = "agritech/sensors/+/data"

# Stockage temporaire pour les données
DATA_DIRECTORY = "./data"
os.makedirs(DATA_DIRECTORY, exist_ok=True)

# Callback pour connexion
def on_connect(client, userdata, flags, rc):
    if rc == 0:
        print("Connecté au broker MQTT")
        client.subscribe(MQTT_TOPIC)
    else:
        print(f"Échec connexion, code: {rc}")

# Callback pour messages reçus
def on_message(client, userdata, msg):
    try:
        print(f"Message reçu sur {msg.topic}: {msg.payload.decode()}")
        data = json.loads(msg.payload.decode())

        # Sauvegarder dans un fichier (simulation avant blockchain)
        sensor_id = data.get("sensor_id", "unknown")
        timestamp = datetime.now().strftime("%Y%m%d%H%M%S")
        filename = f"{DATA_DIRECTORY}/{sensor_id}_{timestamp}.json"

        with open(filename, 'w') as f:
            json.dump(data, f)

        print(f"Données sauvegardées dans {filename}")

        # À ce stade, vous pourriez appeler une fonction pour envoyer les données à la blockchain
        # send_to_blockchain(data)

    except Exception as e:
        print(f"Erreur traitement message: {e}")

client = mqtt.Client()
client.username_pw_set(MQTT_USER, MQTT_PASSWORD)
client.on_connect = on_connect
client.on_message = on_message

def main():
    try:
        client.connect(MQTT_BROKER, MQTT_PORT, 60)
        client.loop_forever()
    except KeyboardInterrupt:
        client.disconnect()
        print("Programme arrêté")
    except Exception as e:
        print(f"Erreur principale: {e}")

if __name__ == "__main__":
    main()
```

2. Créer un service systemd pour le collecteur

```
sudo nano /etc/systemd/system/mqtt-collector.service
```

Contenu du fichier:

```
[Unit]
Description=AgriTech MQTT Data Collector
After=mosquitto.service

[Service]
User=ubuntu
WorkingDirectory=/home/ubuntu/agritech/mqtt_collector
ExecStart=/usr/bin/python3 /home/ubuntu/agritech/mqtt_collector/mqtt_collector.py
Restart=always
RestartSec=10

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

3. Activer et démarrer le service

```
sudo systemctl daemon-reload
sudo systemctl enable mqtt-collector
sudo systemctl start mqtt-collector
```

🔒 SEMAINE 3: SÉCURITÉ ET BLOCKCHAIN

Jour 1-2: Mise en place des pare-feux sur toutes les VMs

1. Configuration UFW sur la VM1 (Zone Terrain)

```
# Sur VM1
sudo apt install -y ufw
sudo ufw default deny incoming
sudo ufw default allow outgoing
sudo ufw allow from 192.168.20.10 to any port 1883 proto tcp comment 'MQTT vers broker'
sudo ufw allow from 192.168.20.10 to any port 22 proto tcp comment 'SSH depuis zone centrale'
sudo ufw enable
```

2. Configuration UFW sur la VM2 (Zone Centrale)

```
# Sur VM2
sudo apt install -y ufw
sudo ufw default deny incoming
sudo ufw default allow outgoing
sudo ufw allow 1883 comment 'MQTT Broker'
sudo ufw allow from 192.168.10.0/24 to any port 1883 proto tcp comment 'MQTT depuis capteurs'
sudo ufw allow from 192.168.30.0/24 to any port 8080 proto tcp comment 'API depuis zone clients'
sudo ufw allow from 192.168.10.0/24 to any port 22 proto tcp comment 'SSH depuis zone terrain'
sudo ufw allow from 192.168.30.0/24 to any port 22 proto tcp comment 'SSH depuis zone clients'
sudo ufw allow 3000 comment 'Grafana'
sudo ufw enable
```

3. Configuration UFW sur la VM3 (Zone Clients)

```
# Sur VM3
sudo apt install -y ufw
sudo ufw default deny incoming
sudo ufw default allow outgoing
sudo ufw allow from 192.168.20.10 to any port 22 proto tcp comment 'SSH depuis zone centrale'
sudo ufw allow 80 comment 'HTTP pour interface web'
sudo ufw allow 8080 comment 'API Gateway'
sudo ufw enable
```

4. Vérification des règles UFW sur chaque machine

```
sudo ufw status verbose
```

Jour 3-5: Déploiement de Hyperledger Fabric (Zone Centrale - VM2)

1. Installation des prérequis pour Hyperledger Fabric

```
# Sur VM2
curl -sSL https://raw.githubusercontent.com/hyperledger/fabric/main/scripts/bootstrap.sh | bash -s
```

2. Configuration du réseau blockchain

```
cd fabric-samples/test-network
./network.sh up createChannel -c agrichannel
```

3. Déploiement d'un smart contract basique pour les données agricoles

```
cd fabric-samples/test-network
./network.sh deployCC -c agrichannel -ccn agridata -ccp ../asset-transfer-basic/chaincode-go -ccl go
```

4. Création d'un script pour envoyer les données MQTT à la blockchain

```
mkdir -p ~/agritech/blockchain_interface
cd ~/agritech/blockchain_interface
nano send_to_blockchain.py
```

Contenu du fichier:

```
import json
import os
import sys
import time
import subprocess
from watchdog.observers import Observer
from watchdog.events import FileSystemEventHandler

# Chemin vers les données MQTT stockées
DATA_DIRECTORY = "../mqtt_collector/data"

# Configuration blockchain
NETWORK_DIR = "/home/ubuntu/fabric-samples/test-network"
CHANNEL_NAME = "agrichannel"
CHAINCODE_NAME = "agridata"

class DataFileHandler(FileSystemEventHandler):
    def on_created(self, event):
        if not event.is_directory and event.src_path.endswith('.json'):
            print(f"Nouveau fichier détecté: {event.src_path}")
            self.process_data_file(event.src_path)

    def process_data_file(self, file_path):
        try:
            # Lire le fichier JSON
            with open(file_path, 'r') as f:
```

```
                data = json.load(f)

            # Préparer les données pour la blockchain
            sensor_id = data.get("sensor_id")
            timestamp = data.get("timestamp")
            temperature = data.get("data", {}).get("temperature")
            humidity = data.get("data", {}).get("humidity")
            soil_moisture = data.get("data", {}).get("soil_moisture")

            # Envoyer à la blockchain via peer chaincode invoke
            asset_id = f"{sensor_id}_{timestamp.replace(':', '-')}"

            # Commande pour invoquer le chaincode
            cmd = [
                "bash", "-c",
                f"cd {NETWORK_DIR} && "
                f"export PATH=$PATH:{NETWORK_DIR}/../bin && "
                f"export FABRIC_CFG_PATH={NETWORK_DIR}/../config && "
                f"export CORE_PEER_TLS_ENABLED=true && "
                f"export CORE_PEER_LOCALMSPID='Org1MSP' && "
                f"export CORE_PEER_TLS_ROOTCERT_FILE={NETWORK_DIR}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1
                f"export CORE_PEER_MSPCONFIGPATH={NETWORK_DIR}/organizations/peerOrganizations/org1.example.com/users/Admin@org1.exa
                f"export CORE_PEER_ADDRESS=localhost:7051 && "
                f"peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile {NETWORK_D

            print("Exécution de la commande blockchain...")
            result = subprocess.run(cmd, capture_output=True, text=True)

            if result.returncode == 0:
                print(f"Données envoyées à la blockchain avec succès: {asset_id}")
                # Déplacer ou supprimer le fichier traité
                os.remove(file_path)
            else:
                print(f"Erreur lors de l'envoi à la blockchain: {result.stderr}")

        except Exception as e:
            print(f"Erreur traitement fichier {file_path}: {e}")

def main():
    # Vérifier si le répertoire existe
    if not os.path.exists(DATA_DIRECTORY):
        print(f"Le répertoire {DATA_DIRECTORY} n'existe pas. Création...")
        os.makedirs(DATA_DIRECTORY)

    # Configuration du watchdog
    event_handler = DataFileHandler()
    observer = Observer()
    observer.schedule(event_handler, DATA_DIRECTORY, recursive=False)
    observer.start()

    print(f"Surveillance des fichiers dans {DATA_DIRECTORY}...")

    try:
        while True:
            time.sleep(1)
    except KeyboardInterrupt:
        observer.stop()
        print("Programme arrêté")

    observer.join()

if __name__ == "__main__":
    main()
```

5. Installer les dépendances et créer un service

```
pip3 install watchdog

sudo nano /etc/systemd/system/blockchain-interface.service
```

Contenu du fichier:

```
[Unit]
Description=AgriTech Blockchain Interface
After=network.target

[Service]
User=ubuntu
WorkingDirectory=/home/ubuntu/agritech/blockchain_interface
ExecStart=/usr/bin/python3 /home/ubuntu/agritech/blockchain_interface/send_to_blockchain.py
Restart=always
RestartSec=10

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

6. Activer et démarrer le service

```
sudo systemctl daemon-reload
sudo systemctl enable blockchain-interface
sudo systemctl start blockchain-interface
```

□ SEMAINE 4: MONITORING ET FINALISATION

Jour 1-2: Installation de Grafana et Prometheus (Zone Centrale - VM2)

1. Déploiement de Prometheus avec Docker

```
# Sur VM2
mkdir -p ~/monitoring/prometheus
cd ~/monitoring/prometheus

# Créer la configuration Prometheus
nano prometheus.yml
```

Contenu du fichier:

```
global:
  scrape_interval: 15s
  evaluation_interval: 15s

scrape_configs:
  - job_name: 'prometheus'
    static_configs:
      - targets: ['localhost:9090']

  - job_name: 'node_exporter'
    static_configs:
      - targets: ['node_exporter:9100']
```

2. Déploiement avec Docker Compose

```
nano docker-compose.yml
```

Contenu du fichier:

```
version: '3'

services:
  prometheus:
    image: prom/prometheus
    container_name: prometheus
    ports:
      - 9090:9090
    volumes:
      - ./prometheus.yml:/etc/prometheus/prometheus.yml
    restart: always
    networks:
      - monitoring

  node_exporter:
    image: prom/node-exporter
    container_name: node_exporter
    ports:
      - 9100:9100
    restart: always
    networks:
      - monitoring

  grafana:
    image: grafana/grafana
    container_name: grafana
    ports:
      - 3000:3000
    environment:
      - GF_SECURITY_ADMIN_USER=admin
      - GF_SECURITY_ADMIN_PASSWORD=admin_password
    volumes:
      - grafana-storage:/var/lib/grafana
    restart: always
    depends_on:
      - prometheus
    networks:
      - monitoring

networks:
  monitoring:

volumes:
  grafana-storage:
```

3. Lancement de la stack de monitoring

```
docker-compose up -d
```

4. Vérification

```
docker ps
```

5. Configuration de Grafana

- Accédez à Grafana via http://192.168.20.10:3000
- Connectez-vous avec admin/admin_password
- Ajoutez Prometheus comme source de données (http://prometheus:9090)
- Importez des dashboards pour le monitoring système

Jour 3-4: Configuration de l'API Gateway (Zone Clients - VM3)

1. Installation de Node.js

```
# Sur VM3
curl -sL https://deb.nodesource.com/setup_16.x | sudo -E bash -
sudo apt install -y nodejs
```

2. Création de l'API Gateway

```
mkdir -p ~/agritech/api_gateway
cd ~/agritech/api_gateway
npm init -y
npm install express cors helmet morgan dotenv paho-mqtt
```

3. Création du fichier serveur

```
nano server.js
```

Contenu du fichier:

```
const express = require('express');
const cors = require('cors');
const helmet = require('helmet');
const morgan = require('morgan');
const mqtt = require('mqtt');
require('dotenv').config();

// Configuration
const PORT = process.env.PORT || 8080;
const app = express();

// Middleware
app.use(cors());
app.use(helmet());
app.use(express.json());
app.use(morgan('combined'));

// MQTT Client
const mqttClient = mqtt.connect('mqtt://192.168.20.10:1883', {
  username: 'admin',
  password: 'votre_mot_de_passe'
});

// Stocker les dernières données reçues
const latestSensorData = {};

mqttClient.on('connect', () => {
  console.log('Connecté au broker MQTT');
  mqttClient.subscribe('agritech/sensors/+/data', (err) => {
    if (!err) {
      console.log('Abonné aux topics des capteurs');
    }
  });
});

mqttClient.on('message', (topic, message) => {
  try {
    const data = JSON.parse(message.toString());
    const sensorId = data.sensor_id;

    // Stocker les données les plus récentes
    latestSensorData[sensorId] = data;
    console.log(`Données reçues pour le capteur ${sensorId}`);
  } catch (error) {
    console.error('Erreur traitement message MQTT:', error);
  }
});
```

```
// Routes API
app.get('/api/sensors', (req, res) => {
  res.json(Object.values(latestSensorData));
});

app.get('/api/sensors/:id', (req, res) => {
  const sensorId = req.params.id;
  if (latestSensorData[sensorId]) {
    res.json(latestSensorData[sensorId]);
  } else {
    res.status(404).json({ error: 'Capteur non trouvé' });
  }
});

// Route pour la santé de l'API
app.get('/health', (req, res) => {
  res.json({ status: 'ok', timestamp: new Date().toISOString() });
});

// Démarrer le serveur
app.listen(PORT, () => {
  console.log(`API Gateway démarré sur le port ${PORT}`);
});
```

4. Création du service systemd

```
sudo nano /etc/systemd/system/api-gateway.service
```

Contenu du fichier:

```
[Unit]
Description=AgriTech API Gateway
After=network.target

[Service]
User=ubuntu
WorkingDirectory=/home/ubuntu/agritech/api_gateway
ExecStart=/usr/bin/node /home/ubuntu/agritech/api_gateway/server.js
Restart=always
RestartSec=10

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

5. Activer et démarrer le service

```
sudo systemctl daemon-reload
sudo systemctl enable api-gateway
sudo systemctl start api-gateway
```

Jour 5: Tests et documentation finale

1. Test de bout en bout

```
# Test du flux complet
# 1. Vérifier que le simulateur de capteurs envoie des données
sudo systemctl status sensor-simulator

# 2. Vérifier que les données sont reçues par le collecteur MQTT
sudo systemctl status mqtt-collector
tail /var/log/syslog | grep mqtt-collector

# 3. Vérifier que les données sont envoyées à la blockchain
sudo systemctl status blockchain-interface
tail /var/log/syslog | grep blockchain-interface

# 4. Vérifier que l'API Gateway reçoit les données
curl http://192.168.30.10:8080/api/sensors
```

2. Finalisation de la documentation

Créez un document final comprenant:

- Schéma de l'architecture réseau
- Configuration des pare-feux
- Flux de données complet
- Procédures de maintenance

SÉCURISATION COMPLÈTE

Configuration de Fail2Ban sur toutes les VMs

```
# Installer Fail2Ban
sudo apt install -y fail2ban

# Configurer Fail2Ban pour SSH
sudo nano /etc/fail2ban/jail.local
```

Contenu du fichier:

```
[sshd]
enabled = true
port = ssh
filter = sshd
logpath = /var/log/auth.log
maxretry = 3
bantime = 3600
```

```
# Redémarrer Fail2Ban
sudo systemctl restart fail2ban
```

Configuration TLS pour MQTT (Zone Centrale - VM2)

```
# Générer un certificat auto-signé
sudo mkdir -p /etc/mosquitto/certs
sudo openssl req -new -x509 -days 365 -nodes -out /etc/mosquitto/certs/mosquitto.crt -keyout /etc/mosquitto/certs/mosquitto.key

# Configurer Mosquitto pour utiliser TLS
sudo nano /etc/mosquitto/conf.d/default.conf
```

Contenu du fichier:

```
listener 8883
certfile /etc/mosquitto/certs/mosquitto.crt
keyfile /etc/mosquitto/certs/mosquitto.key
allow_anonymous false
password_file /etc/mosquitto/passwd
```

```
# Redémarrer Mosquitto
sudo systemctl restart mosquitto
```

ÉVALUATION DU PROJET

Critères d'évaluation

1. **Architecture réseau** :
    - Segmentation correcte en zones
    - Mise en place de pare-feux entre zones
    - Routage fonctionnel
2. **Flux de données** :
    - Capteurs → MQTT → Blockchain → API
    - Monitoring fonctionnel
3. **Sécurité** :
    - Authentification MQTT
    - Pare-feux correctement configurés
    - TLS pour communications sensibles
4. **Documentation** :
    - Schémas clairs
    - Procédures complètes
    - Plan d'adressage IP

PRÉPARATION POUR LA PRÉSENTATION

1. **Préparer des démonstrations en direct** :
    - Simuler un capteur envoyant des données
    - Montrer les données dans le broker MQTT
    - Visualiser les données dans Grafana
    - Interroger l'API pour récupérer les données
2. **Préparer des slides** :
    - Architecture globale
    - Flux de données
    - Mesures de sécurité mises en place
    - Défis rencontrés et solutions apportées
3. **Préparer une démonstration de récupération après incident** :
    - Arrêt d'un service
    - Procédure de récupération
    - Reprise du flux de données

Ce guide complet vous permettra de mettre en place une infrastructure réseau professionnelle pour votre projet AgriTech Blockchain, tout en respectant les exigences académiques du module d'Administration Réseau.

En suivant ces étapes, vous disposerez d'une base solide pour développer votre solution commerciale, avec une architecture sécurisée et évolutive qui pourra être adaptée à vos besoins futurs.