Documentación Técnica Completa

Proyecto Big Data 2025-2026

Sistema de Control de Accesos NFC

Monitorización, Procesamiento Distribuido
y Visualización de Datos

Equipo Azul

Módulo: Big Data Aplicado

Entorno: Cluster Ambari (CentOS 7)

Febrero 2026

Índice

Parte I

Infraestructura y Conectividad del Cluster

1. Configuración de Conectividad en el Cluster Ambari

1.1. Introducción

Este documento describe el proceso completo para establecer conectividad entre todos los hosts que conforman un cluster de Apache Ambari. El objetivo es garantizar que todos los nodos puedan comunicarse entre sí sin problemas, prerequisito indispensable para la instalación y configuración de los servicios distribuidos.

El cluster está compuesto por 9 hosts con direcciones IP en el rango `172.16.200.x`, cada uno con su propia identidad de red y nombre de host único.

1.2. Topología del Cluster

Tabla 1: Topología del cluster Ambari

Máquina	IP	Hostname	Dominio Completo
Host 1	172.16.200.10	ambari10	ambari10.localdomain
Host 2	172.16.200.11	ambari11	ambari11.localdomain
Host 3	172.16.200.12	ambari12	ambari12.localdomain
Host 4	172.16.200.13	ambari13	ambari13.localdomain
Host 5	172.16.200.15	ambari15	ambari15.localdomain
Host 6	172.16.200.8	ambari8	ambari8.localdomain
Host 7	172.16.200.9	ambari9	ambari9.localdomain
Host 8	172.16.200.5	ambari5	ambari5.localdomain
Host 9	172.16.200.3	ambari3	ambari3.localdomain

1.3. Configuración de Adaptadores de Red

Cada máquina física fue configurada con **dos adaptadores de red**:

1. **Adaptador Puente (Bridge)**: Conecta el cluster de hosts entre sí en la red `172.16.200.x`.
2. **Adaptador NAT**: Proporciona acceso a internet y conectividad externa.

Esta configuración dual permite que los nodos se comuniquen entre ellos manteniendo conexión con el exterior.

1.4. Configuración del Archivo /etc/hosts

El siguiente contenido se replicó de forma idéntica en las 9 máquinas:

Listing 1: Contenido de /etc/hosts

```
172.16.200.11   ambari11  ambari11.localdomain
172.16.200.10   ambari10  ambari10.localdomain
```

```
172.16.200.8    ambari8   ambari8.localdomain
172.16.200.5    ambari5   ambari5.localdomain
172.16.200.12   ambari12  ambari12.localdomain
172.16.200.13   ambari13  ambari13.localdomain
172.16.200.15   ambari15  ambari15.localdomain
172.16.200.9    ambari9   ambari9.localdomain
172.16.200.3    ambari3   ambari3.localdomain
```

2. Instalación de Node-RED en CentOS 7

2.1. Introducción

En la máquina virtual `ambari9` (VM9) se realizó la instalación de Node.js, npm y Node-RED para habilitar un entorno de integración ligera y dashboards complementarios al ecosistema Big Data.

2.2. Problema Inicial: Incompatibilidad

CentOS 7 utiliza `glibc 2.17`, lo que impide instalar Node.js 18, 20 o superiores desde NodeSource.

2.3. Solución: Node.js 16

Node.js 16 es la última versión compatible con `glibc 2.17`.

Listing 2: Instalación de Node.js 16

```
cd /opt
sudo curl -O https://nodejs.org/dist/latest-v16.x/node-v16.20.2-linux-x64.tar.xz
sudo tar -xf node-v16.20.2-linux-x64.tar.xz
sudo mv node-v16.20.2-linux-x64 node16

echo 'export PATH=/opt/node16/bin:$PATH' >> ~/.bashrc
source ~/.bashrc

node -v     # v16.20.2
npm -v      # 8.19.4
```

2.4. Instalación de Node-RED

Listing 3: Instalación de Node-RED 3.1.3

```
mkdir ~/.npm-global
npm config set prefix '~/.npm-global'
echo 'export PATH=$HOME/.npm-global/bin:$PATH' >> ~/.bashrc
source ~/.bashrc

npm install -g --unsafe-perm node-red@3.1.3

# Verificaci n
which node-red
```

Parte II

Apache Kafka y Sistema de Producción de Eventos

3. Configuración de Apache Kafka

3.1. Arquitectura del Sistema

El sistema de control de accesos NFC utiliza Apache Kafka como plataforma central de streaming de eventos. El broker Kafka en `nodo1` procesa eventos generados por el productor `nfc_producer.py` y los distribuye a múltiples consumidores.

3.2. Configuración del Broker (nodo1)

Archivo: `/opt/kafka-2.8.1/config/server.properties`

Listing 4: Configuración del broker Kafka

```
# ID  nico  del broker
broker.id=10

# Listeners - Escucha en todas las interfaces
listeners=PLAINTEXT://0.0.0.0:9092

# Advertised listeners - IP anunciada a clientes
advertised.listeners=PLAINTEXT://172.16.200.28:9092

# Conexi n a Zookeeper del cluster
zookeeper.connect=172.16.200.10:2181

# Directorio de almacenamiento de logs
log.dirs=/opt/kafka-2.8.1/kafka-logs

# Retenci n de logs (7 d as)
log.retention.hours=168
```

3.3. Creación del Topic

Listing 5: Creación del topic acceso-centros-nfc

```
cd /opt/kafka-2.8.1

bin/kafka-topics.sh --create \
  --topic acceso-centros-nfc \
  --bootstrap-server 172.16.200.28:9092 \
  --partitions 3 \
  --replication-factor 1
```

4. Productor NFC - Control de Accesos

4.1. Características Técnicas

Tabla 2: Especificaciones del productor NFC

Característica	Valor
Lenguaje	Python 3.x
Broker Kafka	172.16.200.28:9092, 172.16.200.10:9092
Topic	acceso-centros-nfc
Particiones	3
Frecuencia	5 eventos/segundo
Formato	JSON UTF-8
Garantías	acks='all'

4.2. Estructura del Evento NFC

Listing 6: Ejemplo de evento NFC

```
{
  "nfc_id": "NFC-A3F7D9E2C4B1",
  "timestamp": "2026-02-02T08:34:12.456789",
  "estudiante": {
    "nombre": "Mar a  Garc a  L pez",
    "curso": "2   ESO"
  },
  "centro": {
    "nombre": "IES Miguel Servet",
    "codigo": "50008174",
    "maintag": "SEC-MIGUELSERVET",
    "provincia": "Zaragoza"
  },
  "tipo_evento": "ENTRADA",
  "franja_horaria": "ENTRADA_MANANA",
  "punto_acceso": "Entrada Principal",
  "estado": "VALIDADO",
  "temperatura_corporal": 36.5
}
```

4.3. Implementación del Productor

Listing 7: Fragmento del productor NFC

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import json
import time
import random
from datetime import datetime
from kafka import KafkaProducer
from faker import Faker
```

```
10
11  # Configuraci n
12  KAFKA_BROKER = '172.16.200.28:9092'
13  TOPIC_NAME = 'acceso-centros-nfc'
14  EVENTS_PER_SECOND = 5
15
16  fake = Faker('es_ES')
17
18  producer = KafkaProducer(
19      bootstrap_servers=[KAFKA_BROKER],
20      value_serializer=lambda v: json.dumps(v, ensure_ascii=False).
            encode('utf-8'),
21      acks='all'
22  )
23
24  def generar_evento():
25      """Genera un evento de acceso NFC completo"""
26      evento = {
27          'nfc_id': generar_nfc_id(),
28          'timestamp': datetime.now().isoformat(),
29          'estudiante': {
30              'nombre': fake.name(),
31              'curso': random.choice(CURSOS)
32          },
33          'tipo_evento': random.choice(['ENTRADA', 'SALIDA']),
34          'estado': 'VALIDADO' if random.random() > 0.02 else 'RECHAZADO
                ',
35          'temperatura_corporal': round(random.uniform(35.8, 37.2), 1)
36      }
37      return evento
38
39  # Bucle principal
40  while True:
41      evento = generar_evento()
42      producer.send(TOPIC_NAME, value=evento)
43      time.sleep(1 / EVENTS_PER_SECOND)
```

5. Configuración de JMX Prometheus Exporter

5.1. Objetivo

Configurar el servidor Kafka en `nodo1` para exponer métricas operativas mediante JMX Prometheus Exporter, permitiendo que sistemas externos de monitorización puedan consumir información sobre el estado y rendimiento del broker.

5.2. Especificaciones Técnicas

5.3. Configuración del Exporter

Listing 8: kafka-jmx-config.yml

```
lowercaseOutputName: true
lowercaseOutputLabelNames: true
rules:
```

Tabla 3: Especificaciones del JMX Exporter

Parámetro	Valor
Servidor	nodo1 (172.16.200.28)
Puerto Métricas	7071
JMX Exporter	0.20.0
Kafka Version	2.8.1
Topic Principal	acceso-centros-nfc

```
  - pattern: kafka.server<type=(.+), name=(.+)><>Value
    name: kafka_server_$1_$2
    type: GAUGE

  - pattern: kafka.network<type=(.+), name=(.+), request=(.+)><>Value
    name: kafka_network_$1_$2
    type: GAUGE
    labels:
      request: "$3"
```

5.4. Modificación del Script de Kafka

Listing 9: Configuración JMX en kafka-server-start.sh

```
# JMX settings
if [ -z "$KAFKA_JMX_OPTS" ]; then
  KAFKA_JMX_OPTS="-javaagent:/opt/jmx_prometheus_javaagent-0.20.0.jar
    =7071:/opt/kafka-jmx-config.yml \
                  -Dcom.sun.management.jmxremote \
                  -Dcom.sun.management.jmxremote.authenticate=false \
                  -Dcom.sun.management.jmxremote.ssl=false \
                  -Dcom.sun.management.jmxremote.port=9999 \
                  -Djava.rmi.server.hostname=172.16.200.28"
fi
export KAFKA_JMX_OPTS
```

Parte III

Bases de Datos y Sistemas de Almacenamiento

6. Configuración de Redis

6.1. Conexión y Autenticación

Redis está configurado con una contraseña. Para interactuar con él, debes autenticarte.

Listing 10: Conexión a Redis

```
redis-cli
127.0.0.1:6379> AUTH tu_contrase a
OK
```

6.2. Verificar Contenido

Listing 11: Listar todas las claves

```
redis-cli -a "tu_contrase a" KEYS "*"
```

6.3. Gestión de Memoria

Listing 12: Configurar política de evicción

```
redis-cli -a "tu_contrase a" CONFIG GET maxmemory-policy
redis-cli -a "tu_contrase a" CONFIG SET maxmemory-policy noeviction
```

7. Configuración de PostgreSQL

7.1. Editar pg_hba.conf

Listing 13: Configuración de acceso

```
sudo vi /var/lib/pgsql/data/pg_hba.conf
```

Añadir las siguientes líneas:

Listing 14: Reglas de acceso en pg_hba.conf

```
# Conexi n desde la IP de la VM
host    migasfree       ambari_user     192.168.1.103/32        md5

# Conexi n desde localhost
host    migasfree       ambari_user     127.0.0.1/32            md5
```

7.2. Creación de Usuario y Base de Datos

Listing 15: Crear usuario y base de datos

```
-- Crear usuario
CREATE USER ambari_user WITH PASSWORD 'password';

-- Crear base de datos
CREATE DATABASE migasfree;

-- Asignar propietario
ALTER DATABASE migasfree OWNER TO ambari_user;

-- Otorgar permisos
GRANT ALL PRIVILEGES ON DATABASE migasfree TO ambari_user;
```

8. Monitoreo de Redis con Prometheus y Grafana

8.1. Arquitectura del Entorno

- Máquina 1 (`nodo1`) — IP: `172.16.200.63` → Servicios: Prometheus, Grafana
- Máquina 2 — IP: `172.16.200.23` → Servicio: Redis

8.2. Instalación de Prometheus

Listing 16: Instalación de Prometheus

```
sudo useradd --no-create-home --shell /bin/false prometheus
sudo mkdir /etc/prometheus
sudo mkdir /var/lib/prometheus

cd /tmp
LATEST_VERSION=$(curl -s https://api.github.com/repos/prometheus/
    prometheus/releases/latest | grep '"tag_name":' | cut -d '"' -f 4)
wget https://github.com/prometheus/prometheus/releases/download/
    $LATEST_VERSION/prometheus-$LATEST_VERSION.linux-amd64.tar.gz
tar xvf prometheus-$LATEST_VERSION.linux-amd64.tar.gz

cd prometheus-$LATEST_VERSION.linux-amd64
sudo cp prometheus /usr/local/bin/
sudo cp promtool /usr/local/bin/
```

8.3. Configuración de Prometheus

Listing 17: prometheus.yml

```
global:
  scrape_interval: 15s

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

  - job_name: 'redis'
    static_configs:
      - targets: ['172.16.200.23:9121']
```

8.4. Instalación de Grafana

Listing 18: Instalación de Grafana

```
sudo tee /etc/yum.repos.d/grafana.repo <<EOF
[grafana]
name=grafana
baseurl=https://packages.grafana.com/oss/rpm
repo_gpgcheck=1
enabled=1
```

```
gpgcheck=1
gpgkey=https://packages.grafana.com/gpg.key
EOF

sudo yum install -y grafana
sudo systemctl enable grafana-server
sudo systemctl start grafana-server
```

Acceso: `http://172.16.200.63:3000` (admin/admin)

Parte IV

Procesamiento Distribuido y Análisis

9. Apache Flink

9.1. Arquitectura del Sistema

- **Motor de Procesamiento:** Apache Flink 1.15.4 (Scala 2.12)
- **Entorno de Scripting:** Python 3.6
- **Nodo Maestro:** `ambari13` (172.16.200.13)
- **Base de Datos:** Redis (172.16.200.23)
- **Interfaz de Gestión:** Flink Web UI en puerto 8081

9.2. Instalación de PyFlink

Listing 19: Instalación de PyFlink

```
pip3 install --user --upgrade pip setuptools wheel
pip3 install --user apache-flink==1.15.4
pip3 install --user redis
```

9.3. Configuración de Flink

Archivo: `flink-conf.yaml`

Listing 20: Configuración crítica de Flink

```
rest.address: 0.0.0.0
python.executable: /usr/bin/python3
```

9.4. Implementación del Pipeline

Listing 21: Pipeline de persistencia en Flink

```
from pyflink.table import EnvironmentSettings, TableEnvironment
from pyflink.table.udf import udf
from pyflink.table import DataTypes

@udf(result_type=DataTypes.STRING())
def send_to_redis(id_val):
    import redis
    try:
        r = redis.StrictRedis(
            host='172.16.200.23',
            port=6379,
            db=0,
            password='password'
        )
        r.set(f"sensor_id_{id_val}", "CLUSTER_ACTIVE")
        return f"ID {id_val}: OK"
    except Exception as e:
        return str(e)

# Crear entorno
env_settings = EnvironmentSettings.new_instance() \
    .in_streaming_mode() \
    .build()
t_env = TableEnvironment.create(env_settings)

# Registrar UDF
t_env.create_temporary_function("send_to_redis", send_to_redis)
```

10. Configuración de Zeppelin

10.1. Conectividad JDBC con PostgreSQL

Problema: `ClassNotFoundException` al intentar conectar con PostgreSQL.

Solución:

- Despliegue del conector `postgresql-42.5.0.jar` en `/usr/lib/zeppelin/lib/`
- Modificación del template `spark2-env` en Ambari

Listing 22: Configuración del classpath

```
export SPARK_DIST_CLASSPATH=$(hadoop classpath):/usr/lib/zeppelin/lib/
   postgresql-jdbc.jar
```

10.2. Migración a Python 3

Listing 23: Configuración de Python 3 en Zeppelin

```
# En Zeppelin interpreter settings
zeppelin.pyspark.python=/usr/bin/python3
spark.pyspark.python=/usr/bin/python3
```

10.3. Sincronización de Dependencias

Listing 24: Instalación de librerías en todos los nodos

```
sudo pip3 install --only-binary=:all: pandas matplotlib seaborn Pillow

# Ajustar permisos
sudo chmod -R 755 /usr/lib64/python3.6/site-packages
```

Parte V

Visualización y Dashboards

11. Apache Superset

11.1. Instalación

Listing 25: Instalación de Apache Superset

```
su -
yum install -y gcc gcc-c++ libffi-devel python-devel python-pip \
python-wheel openssl-devel cyrus-sasl-devel openldap-devel

pip install --upgrade pip setuptools
pip install apache-superset
```

11.2. Configuración Inicial

Listing 26: Inicialización de Superset

```
export FLASK_APP=superset
superset db upgrade
superset fab create-admin
superset init
```

11.3. Instalación del Driver PostgreSQL

Listing 27: Instalación de psycopg2-binary

```
pip3 install psycopg2-binary
```

11.4. Iniciar Superset

Listing 28: Ejecutar Superset

```
export FLASK_APP=superset
superset run -h 0.0.0.0 -p 9089 --with-threads
```

Acceso: `http://172.16.200.30:9089`

11.5. Conexión a PostgreSQL

SQLAlchemy URI:

`postgresql://ambari:password@172.16.200.43:5432/migasfree`

11.6. Dashboard Panel Migasfree

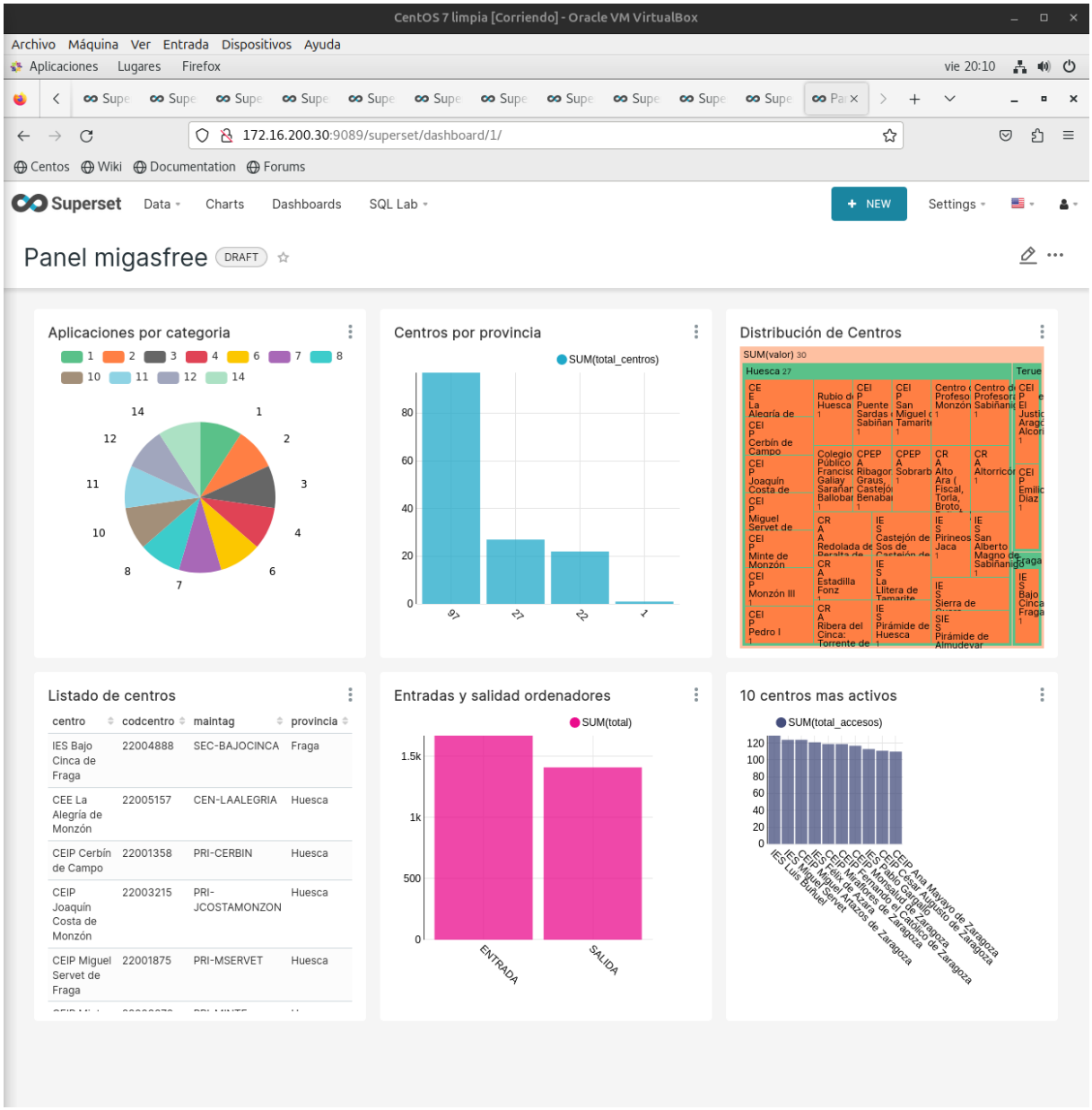

Figura 1: Dashboard Panel Migasfree - Vista completa del sistema de gestión

11.6.1. Componentes del Dashboard

1. **Aplicaciones por categoría** (Pie Chart)
   - Distribución de 414 aplicaciones en 14 categorías

- Identifica predominancia de software

2. **Centros por provincia** (Gráfico de barras)
   - Visualización de concentración geográfica
   - Provincia líder: ∼100 centros
3. **Distribución de Centros** (TreeMap)
   - 30 centros distribuidos por zonas
   - Provincias: Huesca (27), Teruel, Zaragoza
4. **Listado de centros** (Tabla)
   - Información detallada: nombre, código, maintag, provincia
   - Consulta rápida de datos específicos
5. **Entradas y salida ordenadores** (Comparativo)
   - Volumen: ∼1.5k entradas y salidas
   - Monitoreo de flujo de equipamiento
6. **10 centros más activos** (Ranking)
   - Rango: 80-120 accesos por centro
   - Identificación de centros con mayor uso

11.6.2. Métricas Clave

Tabla 4: Métricas principales del dashboard

Métrica	Valor	Descripción
Total Aplicaciones	414	Software en inventario
Total Centros	30	Centros registrados
Provincias	4	Cobertura geográfica
Flujo Equipos	∼1.5k	Movimiento entrada/salida
Categorías Software	14	Clasificaciones disponibles

11.6.3. Casos de Uso

- **Gestión de Inventario:** Supervisar distribución de software y hardware
- **Planificación Geográfica:** Identificar zonas con mayor/menor densidad
- **Monitoreo de Actividad:** Detectar centros con alto uso del sistema
- **Auditoría de Equipos:** Rastrear entradas y salidas
- **Análisis de Software:** Evaluar categorías más utilizadas

Parte VI

Conclusiones y Próximos Pasos

12. Logros Alcanzados

Pipeline de datos en tiempo real completamente operativo

Cluster Kafka distribuido con alta disponibilidad

Productor NFC robusto generando eventos realistas

Integración con múltiples sistemas de procesamiento (Flink, Spark)

Monitorización completa con Prometheus y Grafana

Visualización avanzada con Apache Superset

Almacenamiento distribuido en Redis y PostgreSQL

Dashboard interactivo con Node-RED

Arquitectura escalable y mantenible

13. Arquitectura Final del Sistema

El sistema integra los siguientes componentes en un flujo completo de datos:

1. **Generación:** Productor NFC simula eventos de acceso en centros educativos
2. **Ingesta:** Apache Kafka gestiona el streaming con 3 particiones
3. **Procesamiento:** Apache Flink y Spark procesan datos en tiempo real
4. **Almacenamiento:** Redis (caché) y PostgreSQL (persistencia)
5. **Monitorización:** Prometheus + Grafana + JMX Exporter
6. **Visualización:** Apache Superset + Node-RED dashboards
7. **Análisis:** Apache Zeppelin con Python 3 y librerías ML

14. Lecciones Aprendidas

14.1. Técnicas

- Importancia de la configuración de red en sistemas distribuidos
- Gestión de offsets en Kafka para garantizar no perder mensajes
- Sincronización de dependencias en entornos multi-nodo
- Compatibilidad de versiones en ecosistemas legacy (CentOS 7)

14.2. Operacionales

- Uso de Docker para simplificar despliegues
- Debugging de conectividad entre componentes distribuidos
- Generación de datos sintéticos realistas para pruebas
- Configuración de firewalls en arquitecturas distribuidas

15. Próximos Pasos

15.1. Mejoras Técnicas

1. Implementar modelos de Machine Learning para análisis predictivo
2. Escalar a múltiples centros educativos simultáneamente
3. Optimizar el rendimiento del cluster para mayor throughput
4. Implementar alertas automatizadas basadas en métricas
5. Añadir autenticación y seguridad en todos los servicios

15.2. Expansión Funcional

1. Desarrollar API REST para consultas en tiempo real
2. Integrar con sistemas de control de acceso físico reales
3. Implementar análisis de patrones de comportamiento
4. Crear reportes automáticos personalizados
5. Añadir soporte para múltiples tipos de sensores

16. Resumen de Tecnologías Implementadas

A. Comandos de Referencia Rápida

A.1. Kafka

Listing 29: Comandos útiles de Kafka

```
# Listar topics
bin/kafka-topics.sh --list --bootstrap-server localhost:9092

# Describir topic
bin/kafka-topics.sh --describe --topic acceso-centros-nfc \
  --bootstrap-server localhost:9092

# Consumer de consola
bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 \
  --topic acceso-centros-nfc --from-beginning
```

Tabla 5: Stack tecnológico completo del proyecto

Categoría	Tecnología	Versión
Messaging	Apache Kafka	2.8.1
Coordinación	Apache Zookeeper	3.4.6
Stream Processing	Apache Flink	1.15.4
Batch Processing	Apache Spark	3.1.2
Caché	Redis	Latest
Base de Datos	PostgreSQL	13.x
Monitorización	Prometheus + Grafana	Latest
Visualización	Apache Superset	Latest
Notebooks	Apache Zeppelin	Latest
Automatización	Node-RED	3.1.3
Lenguajes	Python 3, Scala 2.12	3.6+
SO	CentOS	7
Gestión Cluster	Apache Ambari	Latest

```

# Producer de consola
bin/kafka-console-producer.sh --broker-list localhost:9092 \
  --topic acceso-centros-nfc
```

A.2. Redis

Listing 30: Comandos útiles de Redis

```
# Conectar con autenticaci n
redis-cli -a "password"

# Listar todas las claves
KEYS "*"

# Ver valor de una clave
GET sensor_id_1

# Configurar pol tica de evicci n
CONFIG SET maxmemory-policy noeviction
```

A.3. PostgreSQL

Listing 31: Comandos útiles de PostgreSQL

```
# Conectar a base de datos
psql -U ambari_user -d migasfree

# Listar tablas
\dt

# Verificar contenido
SELECT COUNT(*) FROM server_computer;

# Salir de psql
```

```
\q
```

B. Resolución de Problemas Comunes

Tabla 6: Troubleshooting guide

Problema	Causa	Solución
Kafka no conecta	Firewall bloqueando puerto 9092	`firewall-cmd -add-port=9092/tcp`
JMX métricas no disponibles	Puerto 7071 no abierto	`firewall-cmd -add-port=7071/tcp`
Flink slots no disponibles	Procesos zombie Python	Reiniciar cluster Flink
Zeppelin Python 2.7	Configuración incorrecta	Actualizar `PYSPARK_PYTHON`
PostgreSQL conexión rechazada	`pg_hba.conf` mal configurado	Añadir regla de acceso y reiniciar
Redis sin autenticar	Falta contraseña	`AUTH password`
Superset puerto en uso	Proceso previo activo	`lsof -i :9089` y kill

C. Referencias y Documentación

C.1. Documentación Oficial

- Apache Kafka: https://kafka.apache.org/documentation/
- Apache Flink: https://flink.apache.org/
- Apache Spark: https://spark.apache.org/docs/latest/
- Node-RED: https://nodered.org/docs/
- Prometheus: https://prometheus.io/docs/
- Grafana: https://grafana.com/docs/
- Apache Superset: https://superset.apache.org/docs/
- Redis: https://redis.io/documentation
- PostgreSQL: https://www.postgresql.org/docs/

C.2. Repositorios GitHub

- JMX Exporter: https://github.com/prometheus/jmx_exporter
- Redis Exporter: https://github.com/oliver006/redis_exporter
- PyFlink: https://github.com/apache/flink