Requisitos de Software e Instalación

Sistema Integral para Proveedores de Internet (ISP)

1. Requisitos del Sistema Operativo

El sistema está diseñado para ejecutarse en un servidor Ubuntu, que ya tienes configurado con Jellyfin y JFA-GO. Recomendamos:

- **Ubuntu Server 20.04 LTS** o superior
- Kernel actualizado
- Servidor SSH configurado
- Firewall (UFW) correctamente configurado

2. Software Base Necesario

A continuación se detallan los componentes de software principales que se necesitarán instalar:

2.1 Docker y Docker Compose

````
Activa la configuración y reinicia Nginx:

```bash
sudo ln -s /etc/nginx/sites-available/isp-system.conf /etc/nginx/sites-enabled/
sudo nginx -t
sudo systemctl restart nginx
````

7. Integración con Jellyfin y JFA-GO Existentes

Dado que ya tienes Jellyfin y JFA-GO instalados en el servidor Ubuntu, necesitamos integrarlos con el nuevo sistema:

7.1 Información Necesaria de Jellyfin

Para integrar el sistema ISP con tu instalación existente de Jellyfin, necesitarás:

1. **URL de la API de Jellyfin**: Generalmente `http://localhost:8096/jellyfin`
2. **Clave API de administrador**: Puede generarse desde la interfaz de Jellyfin (Dashboard > API Keys)
3. **Permisos de acceso**: Asegurarse que la API tiene permisos administrativos

7.2 Configuración para JFA-GO

Dado que JFA-GO no tiene una API oficial, la integración se realizará a través de:
````

1. **Acceso a la base de datos**: Localiza el archivo de base de datos de JFA-GO, generalmente en:

   ```
   /path/to/jfa-go/data/jfa-go.db
   ```

2. **Scripts de Automatización**: Crearemos scripts para:
   - Generar invitaciones automáticas cuando se registra un cliente
   - Sincronizar estado de las cuentas según el estado de pago del cliente
   - Obtener enlaces de invitación para enviarlos automáticamente

8. Estructura de Directorios del Proyecto

```
isp-system/
├── .env                    # Variables de entorno
├── docker-compose.yml      # Configuración Docker
├── backend/                # API y servicios del backend
│   ├── Dockerfile
│   ├── package.json
│   ├── src/
│   │   ├── index.js        # Punto de entrada
│   │   ├── routes/         # Endpoints de API
│   │   ├── controllers/    # Lógica de negocio
│   │   ├── models/         # Modelos de datos
│   │   ├── services/       # Servicios (Mikrotik, Ubiquiti, etc.)
│   │   ├── middleware/     # Middleware (auth, logs, etc.)
│   │   └── utils/          # Utilidades comunes
├── frontend/               # Aplicación Vue.js
│   ├── Dockerfile
│   ├── package.json
│   ├── public/
│   └── src/
│       ├── main.js         # Punto de entrada
│       ├── App.vue         # Componente raíz
│       ├── router/         # Configuración de rutas
│       ├── store/          # Estado global (Vuex)
│       ├── components/     # Componentes reutilizables
│       ├── views/          # Páginas principales
│       ├── assets/         # Recursos estáticos
│       └── utils/          # Utilidades
├── scripts/                # Scripts de automatización
│   ├── backup.sh           # Respaldo de bases de datos
│   ├── install.sh          # Instalación automatizada
│   └── jellyfin-sync.js    # Sincronización con Jellyfin
└── docs/                   # Documentación
    ├── api.md              # Documentación de API
    ├── setup.md            # Guías de instalación
    └── user-manual.md      # Manual de usuario
```

9. Procedimiento de Instalación

A continuación se detalla el proceso completo de instalación del sistema:

9.1 Preparación del Entorno

1. Clonar el repositorio (o crear estructura de directorios)

bash

```
git clone <repository-url> isp-system
cd isp-system
```

2. Copiar y configurar variables de entorno

bash

```
cp .env.example .env
nano .env
```

3. Crear directorios necesarios

bash

```
mkdir -p backend/src/{routes,controllers,models,services,middleware,utils}
mkdir -p frontend/src/{components,views,store,router,assets,utils}
mkdir -p scripts docs
```

9.2 Instalación y Configuración de Backend

1. Inicializar proyecto Node.js

bash

```
cd backend
npm init -y
npm install express mongoose influx redis jsonwebtoken dotenv cors helmet morgan
npm install nodemon --save-dev
```

2. Crear Dockerfile para el backend

bash

```
nano Dockerfile
```

Con el siguiente contenido:

Dockerfile

```
FROM node:18-alpine

WORKDIR /app

COPY package*.json ./

RUN npm install

COPY . .

EXPOSE 3000

CMD ["npm", "start"]
```

9.3 Instalación y Configuración de Frontend

1. Crear proyecto Vue.js

   bash

   ```
   cd ../frontend
   npm init vue@latest
   ```

2. Instalar dependencias

   bash

   ```
   npm install axios vuex vue-router vuetify chart.js leaflet
   ```

3. Crear Dockerfile para el frontend

   bash

   ```
   nano Dockerfile
   ```

   Con el siguiente contenido:

Dockerfile

```
FROM node:18-alpine as build-stage

WORKDIR /app

COPY package*.json ./

RUN npm install

COPY . .

RUN npm run build

FROM nginx:stable-alpine as production-stage

COPY --from=build-stage /app/dist /usr/share/nginx/html

COPY nginx.conf /etc/nginx/conf.d/default.conf

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]
```

9.4 Configuración de Jellyfin y JFA-GO

1. Identificar ubicación de Jellyfin y JFA-GO

bash

```
systemctl status jellyfin
# Anotar rutas de instalación y configuración
```

2. Configurar scripts de integración

bash

```
cd ../scripts
touch jellyfin-sync.js
nano jellyfin-sync.js
# Añadir código de integración con Jellyfin
```

9.5 Lanzamiento del Sistema

1. Construir y lanzar contenedores Docker

bash

```
cd ..
docker-compose build
docker-compose up -d
```

2. Verificar estado de los servicios

bash

```
docker-compose ps
```

3. Inicializar la base de datos

bash

```
docker exec -it isp_api node src/scripts/init-db.js
```

10. Respaldos y Mantenimiento

10.1 Configuración de Respaldos Automáticos

Crear script de respaldo:

bash

```bash
#!/bin/bash
# backup.sh - Script para respaldo de bases de datos ISP
DATE=$(date +%Y-%m-%d_%H-%M-%S)
BACKUP_DIR="/path/to/backups"

# Respaldo PostgreSQL
docker exec isp_postgres pg_dump -U $DB_USER $DB_NAME > $BACKUP_DIR/postgres_$DATE.sql

# Respaldo InfluxDB
docker exec isp_influxdb influx backup -portable $BACKUP_DIR/influxdb_$DATE

# Respaldo Redis (si es necesario)
docker exec isp_redis redis-cli SAVE

# Comprimir respaldos
tar -czf $BACKUP_DIR/backups_$DATE.tar.gz $BACKUP_DIR/postgres_$DATE.sql $BACKUP_DIR/influxdb_$

# Eliminar archivos temporales
rm $BACKUP_DIR/postgres_$DATE.sql
rm -rf $BACKUP_DIR/influxdb_$DATE

# Mantener solo los últimos 7 respaldos
ls -t $BACKUP_DIR/backups_*.tar.gz | tail -n +8 | xargs rm -f
```

Configurar ejecución automática:

bash

```bash
chmod +x scripts/backup.sh
crontab -e

# Añadir la siguiente línea para respaldos diarios a las 2 AM
0 2 * * * /path/to/isp-system/scripts/backup.sh
```

10.2 Actualizaciones del Sistema

Para actualizar el sistema:

bash

```
cd /path/to/isp-system
git pull
docker-compose down
docker-compose build
docker-compose up -d
```

11. Solución de Problemas Comunes

11.1 Verificar Logs de Contenedores

bash

```
docker logs isp_api
docker logs isp_frontend
docker logs isp_postgres
```

11.2 Verificar Conectividad entre Servicios

bash

```
docker exec -it isp_api ping postgres
docker exec -it isp_api ping redis
```

11.3 Reiniciar Servicios Específicos

bash

```
docker-compose restart api
docker-compose restart frontend
```

Este documento cubre los requisitos de software, instalación y configuración del Sistema Integral para ISP. Sigue las instrucciones paso a paso para implementar el sistema en tu servidor Ubuntu existente.
bash

Actualizar repositorios

sudo apt update
sudo apt upgrade -y

Instalar dependencias

sudo apt install -y apt-transport-https ca-certificates curl software-properties-common

Agregar clave GPG oficial de Docker

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

Agregar repositorio de Docker

sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"

Instalar Docker

sudo apt update
sudo apt install -y docker-ce docker-ce-cli containerd.io

Agregar usuario actual al grupo docker

sudo usermod -aG docker $USER

Instalar Docker Compose

sudo curl -L "https://github.com/docker/compose/releases/download/v2.18.1/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose

Verificar instalaciones

docker --version
docker-compose --version

````
### 2.2 Nginx

```bash
# Instalar Nginx
sudo apt install -y nginx

# Iniciar y habilitar Nginx
sudo systemctl start nginx
sudo systemctl enable nginx

# Verificar estado
sudo systemctl status nginx
````

2.3 Certificados SSL (Let's Encrypt)
````

bash

```bash
# Instalar Certbot
sudo apt install -y certbot python3-certbot-nginx

# Obtener certificados (reemplazar con tu dominio)
sudo certbot --nginx -d tudominio.com -d www.tudominio.com
```

3. Componentes del Sistema (Contenedores Docker)

Los siguientes componentes se instalarán mediante Docker Compose:

3.1 Base de Datos

- **PostgreSQL** (Base de datos principal)
- **InfluxDB** (Métricas y monitoreo)
- **Redis** (Caché y sesiones)

3.2 Backend

- **Node.js** con Express (API principal)
- **Microservicios** para integraciones específicas

3.3 Frontend

- **Vue.js** (Aplicación web principal)
- **Nginx** (Servidor de contenido estático)

4. Archivo Docker Compose

Aquí está el archivo `docker-compose.yml` básico para el sistema:

yaml

```yaml
version: '3.8'

services:
  # Base de datos principal
  postgres:
    image: postgres:14
    container_name: isp_postgres
    volumes:
      - postgres_data:/var/lib/postgresql/data
    environment:
      POSTGRES_PASSWORD: ${DB_PASSWORD}
      POSTGRES_USER: ${DB_USER}
      POSTGRES_DB: ${DB_NAME}
    ports:
      - "5432:5432"
    restart: unless-stopped
    networks:
      - isp_network

  # Base de datos para métricas
  influxdb:
    image: influxdb:2.6
    container_name: isp_influxdb
    volumes:
      - influxdb_data:/var/lib/influxdb2
    environment:
      - DOCKER_INFLUXDB_INIT_MODE=setup
      - DOCKER_INFLUXDB_INIT_USERNAME=${INFLUXDB_USER}
      - DOCKER_INFLUXDB_INIT_PASSWORD=${INFLUXDB_PASSWORD}
      - DOCKER_INFLUXDB_INIT_ORG=${INFLUXDB_ORG}
      - DOCKER_INFLUXDB_INIT_BUCKET=${INFLUXDB_BUCKET}
    ports:
      - "8086:8086"
    restart: unless-stopped
    networks:
      - isp_network

  # Sistema de caché
  redis:
    image: redis:7-alpine
    container_name: isp_redis
    volumes:
      - redis_data:/data
    ports:
      - "6379:6379"
    restart: unless-stopped
```

```yaml
    networks:
      - isp_network

  # Backend API
  api:
    build:
      context: ./backend
      dockerfile: Dockerfile
    container_name: isp_api
    volumes:
      - ./backend:/app
      - /app/node_modules
    environment:
      - NODE_ENV=production
      - DB_HOST=postgres
      - DB_USER=${DB_USER}
      - DB_PASSWORD=${DB_PASSWORD}
      - DB_NAME=${DB_NAME}
      - REDIS_HOST=redis
      - INFLUX_URL=http://influxdb:8086
      - JWT_SECRET=${JWT_SECRET}
    ports:
      - "3000:3000"
    depends_on:
      - postgres
      - redis
      - influxdb
    restart: unless-stopped
    networks:
      - isp_network

  # Frontend
  frontend:
    build:
      context: ./frontend
      dockerfile: Dockerfile
    container_name: isp_frontend
    volumes:
      - ./frontend:/app
      - /app/node_modules
    ports:
      - "8080:80"
    depends_on:
      - api
    restart: unless-stopped
    networks:
      - isp_network
```

```
networks:
  isp_network:
    driver: bridge

volumes:
  postgres_data:
  infludb_data:
  redis_data:
```

5. Variables de Entorno (`.env`)

Crea un archivo `.env` en la misma ubicación que el docker-compose.yml:

```
# Base de datos PostgreSQL
DB_USER=isp_user
DB_PASSWORD=securepassword123
DB_NAME=isp_db

# InfluxDB
INFLUXDB_USER=influx_user
INFLUXDB_PASSWORD=influxpassword123
INFLUXDB_ORG=isp_org
INFLUXDB_BUCKET=network_metrics

# Seguridad
JWT_SECRET=yoursupersecretkey123456

# API Keys (reemplazar con tus claves reales)
MIKROTIK_API_KEY=your_mikrotik_api_key
UBIQUITI_API_KEY=your_ubiquiti_api_key
```

6. Configuración de Nginx como Proxy Inverso

Crea un archivo de configuración para Nginx:

```bash
sudo nano /etc/nginx/sites-available/isp-system.conf
```

Con el siguiente contenido:

nginx

```
server {
    listen 80;
    server_name tudominio.com www.tudominio.com;

    # Redireccionar HTTP a HTTPS
    location / {
        return 301 https://$host$request_uri;
    }
}

server {
    listen 443 ssl;
    server_name tudominio.com www.tudominio.com;

    # Configuración SSL (Certbot lo configurará)
    ssl_certificate /etc/letsencrypt/live/tudominio.com/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/tudominio.com/privkey.pem;
    include /etc/letsencrypt/options-ssl-nginx.conf;
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

    # Frontend
    location / {
        proxy_pass http://localhost:8080;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    # Backend API
    location /api {
        proxy_pass http://localhost:3000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    # Jellyfin (asumiendo que está ejecutándose en el puerto 8096)
    location /jellyfin {
        proxy_pass http://localhost:8096;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
```

```
        # Configuración WebSocket para Jellyfin
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
    }

    # JFA-GO (asumiendo que está ejecutándose en el puerto 8056)
    location /jfa-go {
        proxy_pass http://localhost:8056;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    # Límites y configuraciones de seguridad
    client_max_body_size 100M;

    # Caché para archivos estáticos
    location ~* \.(jpg|jpeg|png|gif|ico|css|js)$ {
        expires 7d;
        add_header Cache-Control "public, no-transform";
    }
}
```