IP Manager - Complete System Design Document

Version: 2.0
Date: December 08, 2025
Author: Francisco - Son2 Latin Music
Location: Tampa Bay, Florida

Executive Summary

IP Manager is a comprehensive network monitoring and IP address management system built for visualizing and tracking devices across network subnets. The system provides real-time network scanning, historical device tracking, IP reservation capabilities, and visual network management through a modern web interface.

Key Capabilities

- Real-time network scanning with nmap
- 16x16 grid visualization (256 IP addresses)
- Persistent device history tracking with MySQL
- IP address reservation system
- Custom notes/comments per IP
- Multi-network discovery and switching
- Visual status indicators with color coding
- phpMyAdmin database management interface

System Architecture

Deployment Platform

- **Host System:** Ubuntu Server 24.04.3 LTS
- **Network Mode:** Host network (direct access to 192.168.1.x)
- **Deployment Method:** Docker Compose (4 containers)
- **Access Method:** Web browser from any device on network

Container Architecture

```
┌─────────────────────────────────────────────────────┐
│                    Docker Host                      │
│              Ubuntu Server 24.04.3                  │
│                 (Host Network)                      │
├─────────────────────────────────────────────────────┤
│                                                     │
│  ┌──────────────┐  ┌──────────────┐  ┌──────────────┐│
│  │   MySQL      │  │   Backend    │  │   Frontend   ││
│  │  Container   │  │  Container   │  │  Container   ││
│  │              │  │              │  │              ││
│  │  Port 3306   │  │  Port 8000   │  │  Port 3000   ││
│  │              │  │              │  │              ││
│  │  Database    │  │   FastAPI    │  │    React     ││
│  │  Storage     │◄─┤   + nmap     │◄─┤  TypeScript  ││
│  └──────────────┘  │  + Python    │  └──────────────┘│
│                    └──────────────┘                 │
│  ┌──────────────┐                                   │
│  │ phpMyAdmin   │                                   │
│  │  Container   │                                   │
│  │              │                                   │
│  │  Port 8080   │                                   │
│  │              │                                   │
│  │   Web UI     │                                   │
│  └──────────────┘                                   │
│                                                     │
└─────────────────────────────────────────────────────┘
           │
           ├── Network: 192.168.1.x
           │
     ┌─────┴─────┐
     │           │
  Gateway (.1)  Devices (.2-.255)
```

Component Specifications

1. MySQL Database Container

Image: `mysql:8.0`
Container Name: `ipam-mysql`
Port: `3306`
Volume: `mysql-data` (persistent)

Environment Variables:

```yaml
MYSQL_ROOT_PASSWORD: ipmanager_root_2024
MYSQL_DATABASE: ipmanager
MYSQL_USER: ipmanager
MYSQL_PASSWORD: ipmanager_pass_2024
```

Database Schema:

Table: nodes

- Primary node tracking table
- Stores IP address information and status
- Fields:
  - `id` (INT, PRIMARY KEY, AUTO_INCREMENT)
  - `ip_address` (VARCHAR(15), UNIQUE)
  - `subnet` (VARCHAR(15))
  - `last_octet` (INT)
  - `status` (ENUM: 'up', 'down', 'previously_used', 'reserved')
  - `hostname` (VARCHAR(255))
  - `mac_address` (VARCHAR(17))
  - `vendor` (VARCHAR(255))
  - `first_seen` (TIMESTAMP)
  - `last_seen` (TIMESTAMP)
  - `last_scanned` (TIMESTAMP)
  - `times_seen` (INT)
  - `notes` (TEXT) - User custom comments
  - `is_reserved` (BOOLEAN)
  - `reserved_by` (VARCHAR(100))
  - `reserved_at` (TIMESTAMP)

Table: scan_history

- Records of all network scans performed
- Fields:
  - `id` (INT, PRIMARY KEY)
  - `subnet` (VARCHAR(15))
  - `start_ip` (INT)
  - `end_ip` (INT)
  - `total_ips` (INT)
  - `active_ips` (INT)
  - `scan_duration` (FLOAT)
  - `scanned_at` (TIMESTAMP)

Table: ip_reservations

- Detailed reservation tracking
- Fields:
  - `id` (INT, PRIMARY KEY)
  - `ip_address` (VARCHAR(15))
  - `reserved_for` (VARCHAR(255))
  - `description` (TEXT)
  - `reserved_by` (VARCHAR(100))
  - `reserved_at` (TIMESTAMP)
  - `expires_at` (TIMESTAMP, NULL)
  - `is_active` (BOOLEAN)

Table: node_history

- Historical snapshots of node state changes
- Fields:
  - `id` (INT, PRIMARY KEY)
  - `node_id` (INT, FOREIGN KEY → nodes.id)
  - `ip_address` (VARCHAR(15))
  - `status` (ENUM: 'up', 'down')
  - `hostname` (VARCHAR(255))
  - `mac_address` (VARCHAR(17))
  - `vendor` (VARCHAR(255))
  - `recorded_at` (TIMESTAMP)

2. Backend Container

Base Image: `ubuntu:24.04`
Container Name: `ipam-backend`
Port: `8000`
Network Mode: `host` (critical for nmap access)
Privileged: `true` (required for nmap)

Technology Stack:

- Python 3.12
- FastAPI (web framework)
- Uvicorn (ASGI server)
- nmap (network scanning)
- python-nmap (nmap Python wrapper)
- mysql-connector-python (database driver)

Python Dependencies:

```
fastapi==0.104.1
uvicorn[standard]==0.24.0
python-nmap==0.7.1
pydantic==2.5.0
mysql-connector-python==8.2.0
```

System Dependencies:

- nmap (network scanning tool)
- curl (for healthchecks)

API Endpoints:

Core Endpoints:

- `GET /` - API information
- `GET /health` - Health check with DB status
- `POST /api/scan` - Perform network scan
  - Request: `{subnet, start_ip, end_ip}`
  - Response: Full scan results with node details

Node Management:

- `GET /api/node/{ip}` - Get detailed node info with history
- `PUT /api/node/update` - Update node notes
  - Request: `{ip, notes, is_reserved}`

IP Reservation:

- `POST /api/reserve` - Reserve an IP address
  - Request: `{ip, reserved_for, description, reserved_by}`
- `POST /api/release/{ip}` - Release reserved IP

Network Discovery:

- `GET /api/networks/discover` - Discover available network interfaces
  - Returns: List of all reachable subnets

Connection Pooling:

```python
db_config = {
    "host": "localhost",
    "port": 3306,
    "user": "ipmanager",
    "password": "ipmanager_pass_2024",
    "database": "ipmanager",
    "pool_name": "ipmanager_pool",
    "pool_size": 10
}
```

Scanning Process:

1. Receive scan request with subnet range
2. Execute nmap with arguments: `-sn -n -T4`
3. Parse nmap results for:
   - IP addresses responding
   - Hostnames (if available)
   - MAC addresses
   - Vendor information (from MAC OUI)
4. Update database for each IP:
   - Create new node if first time seen
   - Update existing node with new scan data
   - Mark as 'up' if responding
   - Mark as 'previously_used' if was up but now down
5. Record scan in scan_history table
6. Return complete results to frontend

3. Frontend Container

Base Image: `node:18`
Container Name: `ipam-frontend`
Port: `3000`
Network Mode: `host`

Technology Stack:

- React 18
- JavaScript/JSX
- CSS3 with custom animations
- Fetch API for backend communication

Key React Components:

Main App Component:

- State management for:
  - Network scanning status
  - Scan results (256 IPs)
  - Selected IP details
  - Modal visibility states
  - Network discovery data
  - Filter settings

Grid Rendering:

- 16x16 matrix (16 rows × 16 columns = 256 cells)
- Each cell represents one IP address (0-255)
- Dynamic styling based on status
- Click handlers for detailed views

Modal Components:

1. **Details Modal** - Shows full IP information
2. **Reservation Modal** - Form for reserving IPs
3. **Notes Modal** - Text area for custom comments
4. **Network Discovery Modal** - Lists available subnets

Visual Status System:

Status	Color	Icon	Meaning
up	Green (🟢)	●	Device currently online
down	Grey (⚪)	-	Never seen, available
previously_used	Yellow (🟡)	-	Was online, now offline
reserved	Purple (🟣)	🔒	Reserved for specific use
localhost	Blue (💙)	🏠	Host machine (Ubuntu laptop)

Additional Visual Indicators:

- 📝 Notes icon - IP has custom comments
- ● Vendor dot - Device has vendor identification
- Blue dot - IP has associated notes

User Interactions:

1. Select subnet (manual or from network discovery)
2. Click "Start Scan" to scan 0-255
3. View grid with color-coded statuses
4. Click any cell to see details
5. Add notes, reserve IPs, view history
6. Filter by status (All, Active, Available, Previously Used)

4. phpMyAdmin Container

Image: `phpmyadmin:latest`
Container Name: `ipam-phpmyadmin`
Port: `8080`

Purpose:

- Web-based MySQL database management
- Query execution interface
- Data export/import capabilities
- Visual schema browsing

Access:

- URL: `http://<ubuntu-ip>:8080`
- Username: `ipmanager` or `root`
- Password: `ipmanager_pass_2024` or `ipmanager_root_2024`

Network Configuration

Host Network Requirements

Network Mode: `host`
Why: Docker bridge networking on Windows/some Linux systems causes false positives in nmap scans due to NAT gateway behavior. Host network mode gives containers direct access to the physical network.

Network Access:

- Backend can directly scan 192.168.1.0/24 (or any host-reachable subnet)
- Frontend accessible from any device on network
- MySQL accessible for external tools if needed
- phpMyAdmin accessible via web browser

Port Bindings:

```
3000 → React Frontend
8000 → FastAPI Backend
3306 → MySQL Database
8080 → phpMyAdmin
```

Firewall Considerations:

```bash
# Allow access to services (if UFW enabled)
sudo ufw allow 3000/tcp  # Frontend
sudo ufw allow 8000/tcp  # Backend API
sudo ufw allow 8080/tcp  # phpMyAdmin
sudo ufw allow 3306/tcp  # MySQL (optional, for external access)
```

Features and Functionality

1. Real-Time Network Scanning

Technology: nmap with Python wrapper

Scan Parameters:

- `-sn` : Ping scan (no port scan)
- `-n` : No DNS resolution
- `-T4` : Aggressive timing

Information Gathered:

- IP address status (up/down)
- Hostname (when available)
- MAC address
- Vendor identification (via MAC OUI lookup)
- Response time

Scan Performance:

- Full /24 subnet (256 IPs): ~15-30 seconds
- Partial range: Proportional to range size
- Concurrent scanning via nmap's internal threading

2. Visual Grid Interface

16x16 Matrix Design:

- Row 0: IPs .0 - .15
- Row 1: IPs .16 - .31
- ...
- Row 15: IPs .240 - .255

Cell Design:

- Size: 60-65px square
- Border: 2-3px based on status
- Corner indicators for special states
- Hover effect: Scale 1.15x
- Click: Open details modal

Responsive Design:

- Desktop: Full 65px cells
- Tablet: 50px cells
- Mobile: 38px cells
- Grid adapts to screen size

3. Device History Tracking

First Seen:

- Timestamp when device first detected
- Never changes once set
- Useful for device lifecycle tracking

Last Seen:

- Updated whenever device responds to scan
- Shows most recent online time
- Helps identify intermittent devices

Times Seen:

- Counter incremented on each detection
- Indicates frequency of device presence
- Useful for identifying mobile devices vs fixed infrastructure

Status Transitions:

1. Never scanned → 'down' (grey)
2. First detection → 'up' (green)
3. Goes offline → 'previously_used' (yellow)
4. Returns online → 'up' (green)
5. Reserved → 'reserved' (purple) regardless of online status

4. IP Reservation System

Purpose:

- Mark IPs for specific devices/purposes
- Prevent accidental assignment
- Document IP allocation plans

Reservation Process:

1. User clicks IP cell
2. Clicks "Reserve IP" button
3. Fills form:
   - Reserved For (required): Device name/purpose
   - Description (optional): Additional details
   - Reserved By (optional): Person/team name
4. Submits reservation
5. IP status changes to 'reserved' (purple)
6. 🔒 lock icon appears on cell

Reservation Data:

- Stored in both `nodes` table and `ip_reservations` table
- Timestamp recorded
- Can be released at any time
- Survives system restarts

Release Process:

1. Click reserved IP
2. Click "Release IP" button
3. Confirm action
4. Status returns to 'down' or 'previously_used'

5. Custom Comments/Notes

Purpose:

- Document device information
- Add maintenance notes
- Record configuration details
- Store contact information

Features:

- Unlimited text length (TEXT field)
- Preserves line breaks and formatting
- Visible indicator (📝) on cells with notes
- Searchable via phpMyAdmin
- Persistent across scans

Common Use Cases:

```
192.168.1.10 - Web Server
Owner: IT Department
Contact: admin@company.com
SSH Port: 2222
Last Maintenance: 2024-12-01
```

6. Multi-Network Discovery

Auto-Detection:

- Scans all network interfaces on host
- Identifies IP addresses and subnets
- Detects gateway information
- Marks primary network

Network Information:

- Interface name (eth0, wlan0, etc.)
- Host IP address
- Subnet (first 3 octets)
- Network type (Local vs Virtual)
- Gateway IP (for primary network)
- Total addressable IPs

User Interface:

- 🌐 Globe button next to subnet input
- Modal showing all available networks
- One-click network switching
- Primary network highlighted in green

7. Localhost Highlighting

Purpose:

- Visually identify the host machine running the containers
- Quick identification of gateway
- Navigation aid in large networks

Visual Design:

- Blue glowing border
- 🏠 House icon in corner
- Pulsing glow animation (3-second cycle)
- Brighter blue on hover
- Bold number styling

Detection Methods:

1. Gateway detection (usually .1)
2. Match against network discovery results
3. Manual specification (if needed)

8. Statistical Dashboard

Real-Time Metrics:

- Total IPs scanned
- Active devices (currently online)
- Available IPs (never used)
- Previously used (offline devices)
- Reserved IPs
- IPs with custom notes

Visual Presentation:

- Card-based layout
- Icon for each category
- Large numbers for quick scanning
- Hover effects for interactivity

9. Filtering System

Filter Options:

- All (default view)
- Active Only (show green cells)
- Available Only (show grey cells)
- Previously Used (show yellow cells)

Behavior:

- Non-matching cells fade to 20% opacity
- Matching cells remain fully visible
- Stats update to show filtered counts
- Can be combined with search/sort in future versions

Data Flow

Scan Workflow

```
User clicks "Start Scan"
         ↓
Frontend sends POST /api/scan
         ↓
Backend receives request
         ↓
Execute nmap scan on subnet
         ↓
Parse nmap results
         ↓
For each IP (0-255):
  ├─→ Check if node exists in database
  │   ├─→ Yes: Update existing record
  │   │   ├─→ If responding: status = 'up', increment times_seen
  │   │   └─→ If not responding: status = 'previously_used'
  │   └─→ No: Create new node record
  │       └─→ Set appropriate initial status
  ├─→ Record in node_history (if status changed to 'up')
  └─→ Preserve reservation status (don't overwrite)
         ↓
Save scan record to scan_history
         ↓
Return complete results to frontend
         ↓
Frontend updates grid display
         ↓
User sees color-coded results
```

Node Detail View Workflow

```
User clicks IP cell
         ↓
Frontend sends GET /api/node/{ip}
         ↓
Backend queries nodes table
         ↓
Backend queries node_history table
         ↓
Return combined data
         ↓
Frontend displays modal with:
  ├─→ Current status
  ├─→ Device information
  ├─→ Timestamps
  ├─→ Custom notes
  └─→ Action buttons
```

Reservation Workflow

```
User clicks "Reserve IP"
        ↓
Frontend shows reservation form
        ↓
User fills in details
        ↓
Frontend sends POST /api/reserve
        ↓
Backend updates nodes table
  ├─→ Set status = 'reserved'
  ├─→ Set is_reserved = TRUE
  ├─→ Store reservation details
  └─→ Timestamp the reservation
        ↓
Backend creates ip_reservations record
        ↓
Return success
        ↓
Frontend refreshes scan
        ↓
Cell displays purple with lock icon
```

User Interface Design

Color Scheme

Primary Colors:

- Primary Blue: `#6366f1` (Indigo)
- Secondary Purple: `#a855f7` (Purple)
- Success Green: `#10b981` (Emerald)
- Warning Yellow: `#f59e0b` (Amber)
- Danger Red: `#ef4444` (Red)

Background:

- Dark: `#0f172a` (Slate)
- Darker: `#020617` (Slate)

Text:

- Primary: `#f8fafc` (Off-white)
- Secondary: `#94a3b8` (Slate gray)

Glassmorphism Effects:

- Background: `rgba(255, 255, 255, 0.05)`
- Border: `rgba(255, 255, 255, 0.1)`
- Backdrop blur: 20px

Typography

Font Family: 'Inter', sans-serif
Monospace: 'Courier New', monospace (for IPs, MACs)

Font Weights:

- Regular: 400
- Medium: 500
- Semibold: 600
- Bold: 700

Animations

Background Glow:

- 3 floating gradient orbs
- 20-second infinite float animation
- Blur: 100px
- Opacity: 0.3

Cell Hover:

- Scale: 1.15x
- Transition: 0.3s cubic-bezier
- Z-index: 10 (above other cells)

Cell Pulse (Active):

- Box-shadow pulse every 2 seconds
- Color: Green rgba with fade

Localhost Glow:

- Blue glow intensity oscillates
- 3-second cycle
- Box-shadow from 15px to 25px

Modal Entry:

- Fade in overlay: 0.3s
- Slide up content: 0.4s
- Spring easing

Button Interactions:

- Hover: translateY(-2px)
- Shadow increase
- 0.3s transition

Layout Structure

```
┌─────────────────────────────────────────┐
│         Header (Logo + Title)           │
├─────────────────────────────────────────┤
│      Control Panel (Glass Card)         │
│ [Subnet Input] [🌐] [Scan Button]       │
├─────────────────────────────────────────┤
│      Stats Dashboard (5 Cards)          │
│ [Active] [Available] [Previous] ...     │
├─────────────────────────────────────────┤
│        Legend Bar (Glass)               │
│ 🟢 Active  ⚪ Available  🟡 Previous     │
├─────────────────────────────────────────┤
│        Filter Bar (Glass)               │
│ [All] [Active] [Available] [Previous]   │
├─────────────────────────────────────────┤
│        Grid Panel (Glass)               │
│                                         │
│  ┌─────────────────────────────┐        │
│  │ 16x16 IP Grid (256 cells)   │        │
│  │                             │        │
│  │ [0]  [1]  [2]  ... [15]     │        │
│  │ [16] [17] [18] ... [31]     │        │
│  │ ...                         │        │
│  │ [240][241][242]...[255]     │        │
│  └─────────────────────────────┘        │
│                                         │
└─────────────────────────────────────────┘
```

Installation and Deployment

Prerequisites

Hardware:

- Ubuntu Server 24.04.3 LTS
- 4GB RAM minimum (8GB recommended)
- 2 CPU cores minimum
- 20GB disk space
- Network interface with access to target subnet

Software:

- Docker Engine
- Docker Compose
- Network connectivity to scan target

Installation Steps

1. Install Docker:

```bash
sudo apt update
sudo apt install -y docker.io docker-compose
sudo usermod -aG docker $USER
sudo systemctl enable docker
sudo systemctl start docker
```

2. Extract IP Manager:

```bash
cd ~
tar -xzf ipmanager-ubuntu.tar.gz
cd ipmanager
```

3. Configure MySQL (Optional - Change Passwords):

```bash
nano docker-compose.yml
# Edit MYSQL_ROOT_PASSWORD and MYSQL_PASSWORD
```

4. Start Services:

```bash
# Start MySQL first (needs time to initialize)
docker compose up -d mysql
sleep 30

# Start all services
docker compose up -d

# Verify all running
docker compose ps
```

5. Access Application:

```bash
# Get Ubuntu laptop IP
hostname -I

# Open in browser from any device
# Frontend: http://<ubuntu-ip>:3000
# API Docs: http://<ubuntu-ip>:8000/docs
# phpMyAdmin: http://<ubuntu-ip>:8080
```

Verification

Check Container Status:

```bash
docker compose ps
# Should show 4 containers running
```

Check Backend Health:

```bash
curl http://localhost:8000/health
# Should return: {"status":"healthy","database":"connected"}
```

Check MySQL:

```bash
docker exec -it ipam-mysql mysql -u ipmanager -pipmanager_pass_2024 -e "SHOW DATABASES;"
# Should show 'ipmanager' database
```

Test Frontend:

- Open browser to `http://<ubuntu-ip>:3000`
- Should see IP Manager interface
- Click "Start Scan"
- Should populate grid with colored cells

Usage Guide

Basic Workflow

1. First-Time Setup:

- Access application in browser
- Click 🌐 to see available networks
- Select your primary network
- Run first scan to populate database

2. Daily Usage:

- Open application
- Click "Start Scan" to refresh device status
- Review changes (new devices, offline devices)
- Update notes as needed
- Reserve IPs for new equipment

3. IP Assignment Planning:

- Scan network to see what's used
- Identify available IPs (grey cells)
- Avoid previously-used IPs (yellow) if possible
- Reserve IP before deploying device
- Add notes with device details

4. Device Troubleshooting:

- Search for device by clicking cells
- Check last seen timestamp
- Review history of connections
- Verify MAC address and vendor
- Check custom notes for configuration

Common Tasks

Reserve an IP for New Device:

1. Scan network
2. Find available IP (grey cell)
3. Click cell → Click "Reserve IP"
4. Enter device name, description, your name
5. Submit
6. IP turns purple with lock icon

Add Notes to Existing Device:

1. Click device's IP cell
2. Click "📝 Edit Notes"
3. Type your information
4. Click "Save Notes"
5. Cell shows 📝 indicator

Track Down a Device:

1. Note its MAC address or vendor
2. Scan network
3. Use phpMyAdmin:

sql

```sql
SELECT ip_address, hostname, vendor, last_seen
FROM nodes
WHERE mac_address = 'XX:XX:XX:XX:XX:XX';
```

4. Or click cells until you find matching vendor

Generate Network Report:

1. Open phpMyAdmin (port 8080)

2. Go to nodes table

3. Click "Export"

4. Select CSV format

5. Download

6. Open in Excel for analysis

View Device History:

1. Click IP cell

2. See first seen, last seen, times seen

3. For detailed history, use phpMyAdmin:

```sql
SELECT * FROM node_history
WHERE ip_address = '192.168.1.100'
ORDER BY recorded_at DESC;
```

Advanced Usage

Scan Multiple Networks:

1. Click 🌐 button

2. Select different network

3. Run scan

4. Database tracks all networks separately

5. Switch back and forth as needed

Export Data for Analysis:

```bash
# Connect to MySQL
docker exec -it ipam-mysql mysql -u ipmanager -pipmanager_pass_2024 ipmanager

# Export to CSV
SELECT ip_address, status, vendor, hostname, last_seen, times_seen
INTO OUTFILE '/tmp/network_report.csv'
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
FROM nodes
WHERE subnet = '192.168.1'
ORDER BY CAST(SUBSTRING_INDEX(ip_address, '.', -1) AS UNSIGNED);
```

Automated Scanning (Cron):

```bash
# Add to crontab for hourly scans
0 * * * * curl -X POST http://localhost:8000/api/scan \
  -H "Content-Type: application/json" \
  -d '{"subnet":"192.168.1","start_ip":0,"end_ip":255}' \
  >> /var/log/ipmanager-scans.log 2>&1
```

Database Management

Common Queries

Find All Active Devices:

```sql
SELECT ip_address, hostname, vendor, mac_address, last_seen
FROM nodes
WHERE status = 'up'
ORDER BY ip_address;
```

Find Devices Not Seen Recently:

```sql
SELECT ip_address, hostname, vendor, last_seen,
       TIMESTAMPDIFF(HOUR, last_seen, NOW()) as hours_offline
FROM nodes
WHERE status = 'previously_used'
  AND last_seen < DATE_SUB(NOW(), INTERVAL 7 DAY)
ORDER BY last_seen DESC;
```

Network Usage Statistics:

```sql
SELECT
    status,
    COUNT(*) as count,
    ROUND(COUNT(*) * 100.0 / (SELECT COUNT(*) FROM nodes), 2) as percentage
FROM nodes
WHERE subnet = '192.168.1'
GROUP BY status;
```

Most Frequently Seen Devices:

```sql
SELECT ip_address, hostname, vendor, times_seen, last_seen
FROM nodes
WHERE times_seen > 5
ORDER BY times_seen DESC
LIMIT 20;
```

Reserved IPs Report:

```sql
SELECT n.ip_address, n.notes, n.reserved_by, n.reserved_at,
       r.reserved_for, r.description
FROM nodes n
LEFT JOIN ip_reservations r ON n.ip_address = r.ip_address AND r.is_active = TRUE
WHERE n.status = 'reserved'
ORDER BY n.ip_address;
```

Device History Timeline:

```sql
SELECT h.recorded_at, h.status, h.hostname, h.vendor
FROM node_history h
JOIN nodes n ON h.node_id = n.id
WHERE n.ip_address = '192.168.1.100'
ORDER BY h.recorded_at DESC
LIMIT 50;
```

Scan Performance Metrics:

```sql
SELECT
    DATE(scanned_at) as scan_date,
    COUNT(*) as scan_count,
    AVG(scan_duration) as avg_duration,
    AVG(active_ips) as avg_active_devices,
    MAX(active_ips) as max_devices
FROM scan_history
WHERE scanned_at > DATE_SUB(NOW(), INTERVAL 30 DAY)
GROUP BY DATE(scanned_at)
ORDER BY scan_date DESC;
```

Vendor Distribution:

```sql
SELECT
    vendor,
    COUNT(*) as device_count
FROM nodes
WHERE vendor IS NOT NULL
  AND status = 'up'
GROUP BY vendor
ORDER BY device_count DESC;
```

Database Maintenance

Optimize Tables (Monthly):

```sql
OPTIMIZE TABLE nodes;
OPTIMIZE TABLE node_history;
OPTIMIZE TABLE scan_history;
OPTIMIZE TABLE ip_reservations;
```

Archive Old History (Yearly):

```sql
-- Archive node_history older than 1 year
DELETE FROM node_history
WHERE recorded_at < DATE_SUB(NOW(), INTERVAL 1 YEAR);

-- Archive scan_history older than 6 months
DELETE FROM scan_history
WHERE scanned_at < DATE_SUB(NOW(), INTERVAL 6 MONTH);
```

Backup Database:

```bash
# From host
docker exec ipam-mysql mysqldump -u ipmanager -pipmanager_pass_2024 ipmanager > backup-$(date +%

# Restore
docker exec -i ipam-mysql mysql -u ipmanager -pipmanager_pass_2024 ipmanager < backup-20251208.sql
```

Security Considerations

Current Security Posture

Development/Internal Use:

- Default passwords in docker-compose.yml
- No authentication on frontend
- Open network access to all ports
- Root access to phpMyAdmin available

Acceptable For:

- Home lab environments
- Internal corporate networks
- Trusted network segments
- Development/testing

Production Hardening Recommendations

1. Change Default Passwords:

```yaml
environment:
  MYSQL_ROOT_PASSWORD: <strong-random-password>
  MYSQL_PASSWORD: <strong-random-password>
```

2. Implement Authentication:

- Add OAuth2/JWT to FastAPI backend
- Implement login screen on frontend
- Use environment variables for credentials

3. Restrict Network Access:

```bash
# Firewall rules (UFW example)
sudo ufw allow from 192.168.1.0/24 to any port 3000
sudo ufw allow from 192.168.1.0/24 to any port 8000
sudo ufw deny 3306  # Don't expose MySQL externally
sudo ufw deny 8080  # Restrict phpMyAdmin access
```

4. Use HTTPS:

- Set up reverse proxy (nginx)
- Obtain SSL certificates (Let's Encrypt)
- Redirect HTTP to HTTPS

5. Enable phpMyAdmin 2FA:

- Configure in phpMyAdmin settings
- Use Google Authenticator or similar

6. Regular Updates:

```bash
# Update container images
docker compose pull
docker compose up -d

# Update Ubuntu host
sudo apt update && sudo apt upgrade
```

7. Audit Logging:

- Enable MySQL query logging
- Log API access
- Monitor for suspicious activity

8. Backup Strategy:

- Daily automated database backups
- Store backups off-system
- Test restore procedures

Troubleshooting

Common Issues and Solutions

Issue: Blank page on frontend

```
Symptom: Browser shows blank white page
Cause: JavaScript error, missing import, broken component
Solution:
1. Press F12 → Console tab
2. Look for error messages
3. Common fix: Check import statements
4. Restore from backup if needed
```

Issue: "NetworkError when attempting to fetch resource"

```
Symptom: Scan button returns error
Cause: Backend not running or not accessible
Solution:
docker compose logs ipam-backend --tail 50
docker compose restart ipam-backend
curl http://localhost:8000/health
```

Issue: All IPs show as "up" (false positives)

```
Symptom: Every IP address shows green
Cause: Docker bridge networking instead of host mode
Solution:
Verify docker-compose.yml has:
  network_mode: host
Restart containers:
  docker compose down && docker compose up -d
```

Issue: MySQL connection failed

```
Symptom: Backend shows database connection errors
Cause: MySQL not ready or wrong credentials
Solution:
docker compose logs mysql --tail 30
# Wait for "ready for connections" message
# Verify credentials in docker-compose.yml match main.py
docker compose restart ipam-backend
```

Issue: nmap not found

```
Symptom: Backend logs show "No such file or directory: 'nmap'"
Cause: nmap not installed in container
Solution:
docker exec ipam-backend apt update
docker exec ipam-backend apt install -y nmap
docker compose restart ipam-backend
```

Issue: Port already in use

```
Symptom: "port is already allocated" error
Cause: Another service using the port
Solution:
# Find what's using the port
sudo netstat -tulpn | grep 3000
# Stop conflicting service or change port in docker-compose.yml
```

Issue: No networks detected

```
Symptom: Network discovery modal shows "No networks detected"
Cause: 'ip' command not available in container
Solution:
# Backend needs host network access
Verify network_mode: host in docker-compose.yml
Alternative: Hardcode IP in frontend isLocalhostIP function
```

Issue: Permission denied on scan

```
Symptom: nmap fails with permission errors
Cause: Container not running in privileged mode
Solution:
Verify docker-compose.yml has:
  privileged: true
Restart: docker compose up -d
```

Issue: Slow scans

```
Symptom: Scan takes 2+ minutes
Cause: Network congestion or host performance
Solution:
# Reduce scan range
POST /api/scan with start_ip: 1, end_ip: 50
# Or adjust nmap timing (in main.py)
Change -T4 to -T3 (more polite) or -T5 (faster but aggressive)
```

Log Locations

Container Logs:

```bash
docker compose logs ipam-backend
docker compose logs ipam-frontend
docker compose logs mysql
docker compose logs phpmyadmin
```

Application Logs:

- Backend: stdout/stderr (view with docker logs)
- Frontend: Browser console (F12)
- MySQL: Inside container at `/var/log/mysql/`
- Scan errors: Written to backend stdout

Debugging Tips:

```bash
# Live tail all logs
docker compose logs -f

# Check container health
docker compose ps

# Enter container for debugging
docker exec -it ipam-backend bash
docker exec -it ipam-mysql mysql -u root -p

# Test network connectivity from container
docker exec ipam-backend ping 192.168.1.1
docker exec ipam-backend nmap -sn 192.168.1.1
```

Performance Optimization

Scan Performance

Current Performance:

- Full /24 subnet (256 IPs): 15-30 seconds
- Influenced by: network latency, active devices, nmap timing

Optimization Options:

1. Parallel Scanning (Already Implemented):

- nmap handles concurrency internally
- `-T4` timing profile balances speed and reliability

2. Reduce Scan Range:

```javascript
// Frontend: Scan only likely range
POST /api/scan
{
  "subnet": "192.168.1",
  "start_ip": 10,
  "end_ip": 100
}
```

3. Scheduled Background Scans:

```bash
# Cron job for automated scanning
*/15 * * * * curl -X POST http://localhost:8000/api/scan \
  -H "Content-Type: application/json" \
  -d '{"subnet":"192.168.1","start_ip":0,"end_ip":255}'
```

4. Database Indexing (Already Implemented):

```sql
-- Indexes on nodes table
INDEX idx_ip (ip_address)
INDEX idx_subnet (subnet)
INDEX idx_status (status)
INDEX idx_last_seen (last_seen)
```

Frontend Performance

Grid Rendering:

- 256 cells × React components = potential slowdown
- Mitigation: CSS transforms instead of re-renders
- Filter hiding uses opacity, not removal

Optimization Techniques:

```javascript
// Use React.memo for cell components (future enhancement)
const GridCell = React.memo(({ ip, status, ... }) => {
  // Cell rendering
});

// Virtualization for very large grids (future)
// Only render visible cells
```

Database Performance

Connection Pooling:

```python
# Already implemented
pool_size = 10
# Reuses connections instead of creating new ones
```

Query Optimization:

- Use indexes for all WHERE clauses
- Limit result sets with LIMIT
- Use prepared statements (automatic with mysql-connector)

Archive Old Data:

- Move old node_history to archive table
- Keep scan_history for 6 months max
- Keeps active tables small and fast

Future Enhancements

Planned Features

1. Port Scanning

```python
# Add to scan arguments
nm.scan(hosts=ip_range, arguments='-sn -p 22,80,443,3389')
# Display open ports on detail modal
```

2. Device Categorization

```sql
-- Add to nodes table
ALTER TABLE nodes ADD COLUMN device_type VARCHAR(50);
-- Categories: Server, Workstation, Printer, IoT, Mobile, etc.
```

3. Alerting System

```python
# Email/Slack alerts for:
- New devices detected
- Devices offline > X hours
- IP conflicts
- Unauthorized devices
```

4. Network Topology Visualization

```javascript
// Graph view showing:
- Gateway at center
- Devices as nodes
- Connections as edges
- Visual clustering by vendor/type
```

5. DHCP Integration

```python
# Read DHCP leases file
- Compare scanned vs. leased IPs
- Identify rogue devices
- Match hostnames from DHCP
```

6. Historical Graphs

```javascript
// Charts showing:
- Device count over time
- Uptime patterns
- Network usage trends
- Vendor distribution changes
```

7. Export/Import

```python
# API endpoints for:
POST /api/export/csv
POST /api/export/json
POST /api/import/csv
- Bulk operations on nodes
```

8. Mobile App

```
React Native version:
- Same functionality
- Mobile-optimized grid
- Push notifications
- QR code scanning for device info
```

9. Advanced Search

```javascript
// Search by:
- IP range
- Vendor
- Hostname pattern
- Last seen date range
- Custom note content
```

10. Role-Based Access Control

```python
# User roles:
- Admin: Full access
- Operator: Scan and view
- Viewer: Read-only
# Implement with JWT tokens
```

Architectural Improvements

1. Microservices Split

```
- Scanning service (dedicated)
- API service (FastAPI)
- Database service (MySQL)
- Alert service (new)
- Reporting service (new)
```

2. Message Queue

```
- Add RabbitMQ or Redis
- Async scan processing
- Better scalability
```

3. Caching Layer

```
- Redis for frequently accessed data
- Reduce database load
- Faster response times
```

4. High Availability

```
- Multiple backend instances
- Load balancer (nginx)
- MySQL replication
- Failover capability
```

Technical Specifications

System Requirements

Minimum:

- CPU: 2 cores @ 2.0 GHz
- RAM: 4 GB
- Disk: 20 GB SSD
- Network: 100 Mbps

Recommended:

- CPU: 4 cores @ 2.5 GHz
- RAM: 8 GB
- Disk: 50 GB SSD
- Network: 1 Gbps

Optimal:

- CPU: 6+ cores @ 3.0+ GHz
- RAM: 16 GB
- Disk: 100 GB NVMe SSD
- Network: 10 Gbps

Scalability Limits

Current Architecture:

- Max subnet size: /16 (65,536 IPs)
- Realistic limit: /24 (256 IPs) per scan
- Concurrent scans: 1 (sequential only)
- Database: ~1 million nodes before optimization needed
- Frontend: 256 cells rendered efficiently

Performance Scaling:

Network Size	Scan Time	DB Growth/Month
/24 (256)	15-30 sec	~5 MB
/23 (512)	30-60 sec	~10 MB
/22 (1024)	1-2 min	~20 MB
/16 (65536)	30-60 min	~500 MB

Technology Versions

Container Images:

- mysql:8.0
- ubuntu:24.04
- node:18
- phpmyadmin:latest

Python Packages:

- fastapi==0.104.1
- uvicorn==0.24.0
- python-nmap==0.7.1
- pydantic==2.5.0
- mysql-connector-python==8.2.0

JavaScript Libraries:

- react@18.x
- Native Fetch API (no axios needed)

System Tools:

- nmap 7.94+
- Docker 24.0+
- Docker Compose 2.0+

Maintenance Procedures

Daily Tasks

- None required (system is self-maintaining)
- Optional: Review scan results for anomalies

Weekly Tasks

- Review new devices detected
- Update notes for unknown devices
- Check for devices offline > 7 days

Monthly Tasks

- Backup database
- Review and clean old scan history
- Optimize database tables
- Update container images
- Review reserved IPs for accuracy

Quarterly Tasks

- Full security audit
- Performance review
- Archive old data (>6 months)
- Update documentation

Annual Tasks

- Major version updates
- Hardware capacity planning
- Disaster recovery test
- Security penetration test

Backup Strategy

Automated Daily Backup:

bash

```bash
#!/bin/bash
# /usr/local/bin/ipmanager-backup.sh

BACKUP_DIR="/backups/ipmanager"
DATE=$(date +%Y%m%d)

# Create backup directory
mkdir -p $BACKUP_DIR

# Backup MySQL database
docker exec ipam-mysql mysqldump -u ipmanager -pipmanager_pass_2024 \
  ipmanager > $BACKUP_DIR/ipmanager-$DATE.sql

# Compress
gzip $BACKUP_DIR/ipmanager-$DATE.sql

# Remove backups older than 30 days
find $BACKUP_DIR -name "*.sql.gz" -mtime +30 -delete

echo "Backup completed: ipmanager-$DATE.sql.gz"
```

Add to crontab:

bash

```bash
# Run daily at 2 AM
0 2 * * * /usr/local/bin/ipmanager-backup.sh >> /var/log/ipmanager-backup.log 2>&1
```

License and Credits

System: IP Manager v2.0
Developer: Francisco - Son2 Latin Music
Location: Tampa Bay, Florida
Deployment: December 8, 2025

Technology Credits:

- FastAPI - Modern Python web framework
- React - JavaScript UI library
- MySQL - Relational database
- nmap - Network scanning utility
- Docker - Containerization platform
- Ubuntu - Linux operating system

Open Source Components:

- python-nmap: GPLv3
- FastAPI: MIT License
- React: MIT License
- MySQL: GPL
- phpMyAdmin: GPL

Support and Documentation

Primary Documentation:

- This design document
- API documentation: http://localhost:8000/docs
- Code comments in source files

External Resources:

- FastAPI: https://fastapi.tiangolo.com/
- React: https://react.dev/
- nmap: https://nmap.org/book/
- MySQL: https://dev.mysql.com/doc/

Internal Support:

- Review container logs for errors
- Check phpMyAdmin for data verification
- Test API endpoints at /docs interface

Conclusion

IP Manager provides a comprehensive, visual, and persistent solution for network IP address management. The system successfully combines real-time scanning with historical tracking, reservation capabilities, and an intuitive user interface.

Key Achievements: ✅ Visual 16x16 grid representation of network
✅ Persistent device history in MySQL database
✅ IP reservation and custom notes system
✅ Multi-network discovery and switching
✅ Professional modern dark-themed UI
✅ Complete database management via phpMyAdmin
✅ Docker-based deployment for portability

Production Ready For:

- Home network management
- Small business networks
- Lab environments
- Network documentation
- DHCP planning
- Device inventory

The system has been successfully deployed on Ubuntu Server 24.04.3 and is operational as of December 8, 2025.

Document Version: 2.0

Last Updated: December 08, 2025

Status: Production

Next Review: March 08, 2026