IP Manager - Complete Design & Deployment Documentation

Version: 2.0

Last Updated: December 2024

System Location: Ubuntu Server 192.168.0.199:/home/ubuntu/ipmanager

Executive Summary

IP Manager is a comprehensive network management platform that provides IP address tracking, network scanning, device history, VM provisioning via Proxmox, and integrated performance monitoring. The entire system runs as a containerized application stack on Ubuntu Server using Docker Compose.

Key Capabilities

• **Network Discovery**: Automated scanning and detection of active devices

• **IP Address Management**: Reserve, track, and manage IP allocations

• **Device History**: Complete historical tracking of device connections and changes

• **VM Provisioning**: Direct Proxmox integration for VM creation and deployment

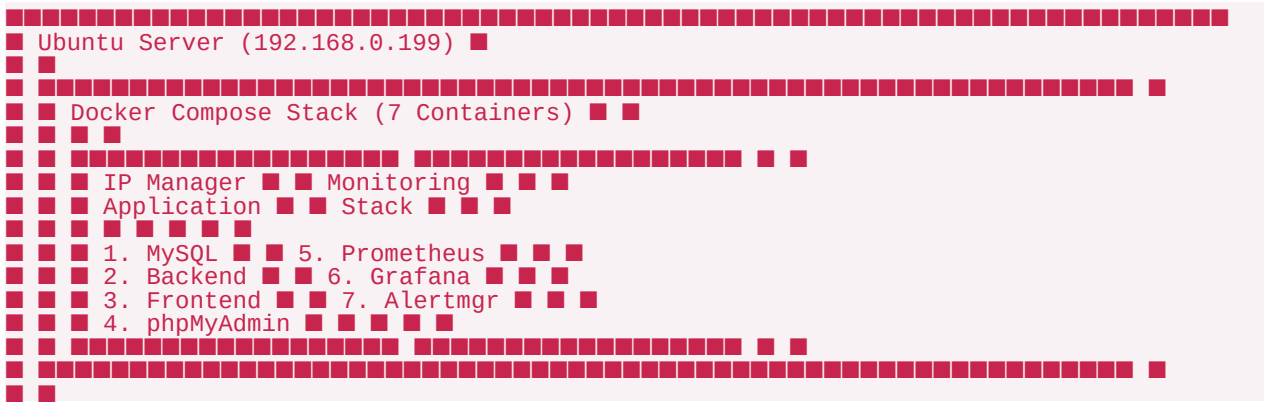
• **Performance Monitoring**: Integrated Prometheus, Grafana, and Alertmanager stack

• **Traffic Testing**: Automated network performance testing with iperf3

• **Modern UI**: Glassmorphism design with responsive interface

Architecture Overview

System Topology



```
■ Network Mode: host (backend for scanning) ■
■ Restart Policy: unless-stopped (auto-recovery) ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■
■ Proxmox API
▼
■■■■■■■■■■■■■■■■■■■■
■ Proxmox Server ■
■ (192.168.0.100) ■
■ ■
■ - VM Creation ■
■ - No Containers ■
■■■■■■■■■■■■■■■■■■■■
```

Container Architecture

All containers run on **Ubuntu Server 192.168.0.199** via a single `docker-compose.yml` file:

Container	Image	Port(s)	Purpose	Network Mode
ipam-mysql	mysql:8.0	3306	Database storage	bridge
ipam-backend	ipmanager-backend	8000	FastAPI REST API	**host**
ipam-frontend	ipmanager-frontend	3000	React UI	bridge
phpmyadmin	phpmyadmin:latest	8080	Database admin	bridge
prometheus	prom/prometheus:latest	9090	Metrics collection	bridge
grafana	grafana/grafana:latest	3001	Dashboards	bridge
alertmanager	prom/alertmanager:latest	9093	Alert handling	bridge

Key Design Decisions:

• **Backend uses `network_mode: host`**: Required for direct network interface access for ICMP scanning and raw socket operations

• **All containers have `restart: unless-stopped`**: Automatic recovery after system reboots or updates

• **Persistent volumes**: All data survives container restarts

Component Details

1. MySQL Database (ipam-mysql)

Purpose: Persistent storage for all application data

Schema:

```
- ip_addresses: Main IP tracking table
- id, ip_address, hostname, mac_address, status,
reserved_by, notes, first_seen, last_seen, created_at, updated_at

- device_history: Historical tracking of all devices
- id, ip_address, hostname, mac_address, status, vendor,
first_seen, last_seen, active

- vm_traffic_tests: Performance test results
- id, source_vm, target_vm, test_type, duration, bandwidth,
```

```
result_data, status, created_at
```

Configuration:

• Root password: ipmanager_root_2024

• Application user: ipmanager / ipmanager_pass_2024

• Port: 3306 (exposed to host for direct access)

• Health check: mysqladmin ping every 10s

• Volume: mysql-data (persistent)

2. FastAPI Backend (ipam-backend)

Purpose: REST API for all application logic

Key Features:

• Network scanning (ICMP ping sweep)

• IP address management (CRUD operations)

• Device history tracking

• Proxmox VM creation integration

• SSH-based traffic testing with iperf3

• Prometheus metrics export

Technology Stack:

• FastAPI (async Python web framework)

• aiomysql (async MySQL driver)

• paramiko (SSH client for remote commands)

• python-nmap (network scanning wrapper for nmap)

• proxmoxer (Proxmox API client)

Network Configuration:

• Uses `network_mode: host` for direct network access

• Can ping any device on 192.168.0.x network

• Can SSH to VMs for traffic testing

• Accesses MySQL on 127.0.0.1:3306

Environment Variables:

```
MYSQL_HOST=127.0.0.1
MYSQL_PORT=3306
MYSQL_USER=ipmanager
MYSQL_PASSWORD=ipmanager_pass_2024
MYSQL_DATABASE=ipmanager
PROXMOX_HOST=192.168.0.100
PROXMOX_USER=root@pam
PROXMOX_PASSWORD=${PROXMOX_PASSWORD}
PROXMOX_NODE=pve
```

API Endpoints:

• `GET /api/scan/{network}` - Scan network for active devices

• `GET /api/ips` - List all IP addresses

• `POST /api/ips` - Create/reserve IP

• `PUT /api/ips/{ip}` - Update IP info

• `DELETE /api/ips/{ip}` - Delete IP

• `GET /api/history` - Get device history

• `POST /api/create-vm` - Create Proxmox VM

• `POST /api/traffic-test` - Run iperf3 test

• `GET /docs` - Interactive API documentation

3. React Frontend (ipam-frontend)

Purpose: User interface for IP Manager

Design System:

• **Glassmorphism UI**: Modern translucent design with backdrop blur

• **Responsive Layout**: Works on desktop, tablet, and mobile

• **Real-time Updates**: Automatic refresh of network data

• **Dark Theme**: Low-eye-strain color scheme

Key Components:

```
App.js
■■■ NetworkScanner - Scan network and display results
■■■ IPList - Show all tracked IPs with filtering
■■■ DeviceHistory - Historical device tracking
■■■ VMCreator - Proxmox VM provisioning form
■■■ TrafficTester - Network performance testing
■■■ MonitoringLinks - Quick access to Grafana/Prometheus
```

Features:

• One-click network scanning

• Visual status indicators (active/inactive/reserved)

• Click-to-copy IP addresses

• Device details modal

• Inline IP reservation

• VM creation wizard

• Traffic test launcher

• Direct links to monitoring dashboards

Technology:

• React 18

• Axios (HTTP client)

• CSS3 (glassmorphism effects)

• Responsive flexbox layout

4. phpMyAdmin (phpmyadmin)

Purpose: Database administration interface

Access: http://192.168.0.199:8080

Credentials: root / ipmanager_root_2024

Use Cases:

• Direct database queries

• Schema modifications

• Data export/import

• Performance monitoring

• Index optimization

5. Prometheus (prometheus)

Purpose: Time-series metrics collection

Scrape Targets:

• Prometheus itself (self-monitoring)

• Node exporters on managed VMs

• iperf3 servers on VMs

Configuration:

```
scrape_interval: 5s
scrape_configs:
- job_name: 'prometheus'
static_configs:
- targets: ['localhost:9090']

- job_name: 'node_exporter'
file_sd_configs:
- files: ['/etc/prometheus/targets/nodes.yml']

- job_name: 'iperf3'
file_sd_configs:
- files: ['/etc/prometheus/targets/iperf.yml']
```

Dynamic Target Discovery:

• `monitoring/prometheus/targets/nodes.yml` - VM node exporters

• `monitoring/prometheus/targets/iperf.yml` - iperf3 servers

Retention: 30 days

Access: http://192.168.0.199:9090

6. Grafana (grafana)

Purpose: Visualization and dashboarding

Features:

• Pre-configured Prometheus datasource

• Dashboard provisioning support

• Live traffic test visualization

• VM performance metrics

• Network bandwidth graphs

Configuration:

• Admin credentials: admin / admin

• Datasource: Prometheus (http://prometheus:9090)

• Provisioning: `/etc/grafana/provisioning`

• Dashboards: `/var/lib/grafana/dashboards`

Access: http://192.168.0.199:3001

7. Alertmanager (alertmanager)

Purpose: Alert routing and notification

Configuration:

```
route:
group_by: ['alertname']
group_wait: 10s
group_interval: 10s
repeat_interval: 1h
receiver: 'web.hook'
```

Access: http://192.168.0.199:9093

Proxmox Integration

VM Creation Workflow

1. **User initiates VM creation** from frontend
2. **Frontend sends request** to backend API
3. **Backend authenticates** with Proxmox API (192.168.0.100)
4. **Backend creates VM** with specified configuration
5. **Backend returns** VM ID and status
6. **Frontend displays** creation confirmation

VM Configuration Options

```
{
"vmid": 100-999, # Unique VM ID
"name": "vm-name", # VM hostname
"cores": 2, # CPU cores
"memory": 2048, # RAM in MB
"disk": 32, # Disk size in GB
"network_bridge": "vmbr0" # Network bridge
}
```

Proxmox Requirements

• Proxmox VE 7.0+

• API access enabled

• User with VM.Allocate privileges

• Network bridge configured

• Storage pool available

Important: Proxmox server (192.168.0.100) runs **NO Docker containers**. It only provides the hypervisor platform and API for VM management.

Traffic Testing System

Architecture

```
IP Manager (192.168.0.199)
│
├──→ SSH to Source VM (192.168.0.32)
│    └──→ Launch iperf3 client → Target VM (192.168.0.33)
│
├──→ Prometheus scrapes iperf3 metrics
└──→ Grafana visualizes in real-time
```

Testing Process

1. **User selects source and target VMs**

2. **Configures test parameters:**

• Protocol: TCP or UDP

• Duration: 1-300 seconds

• Bandwidth limit: 10M-1G

3. **Backend executes:**

```
ssh ubuntu@source-vm "iperf3 -c target-vm -t 60 -b 100M -J"
```

4. **Results stored** in database

5. **Metrics exported** to Prometheus

6. **Live graphs** available in Grafana

VM Preparation

Each VM being tested must have:

```
# Node exporter for system metrics
sudo systemctl enable --now node_exporter # Port 9100

# iperf3 server for traffic tests
sudo systemctl enable --now iperf3-server # Port 5201

# SSH password authentication
PasswordAuthentication yes # In /etc/ssh/sshd_config
```

Test Results

Results include:

- Throughput (Mbits/sec)
- Packet loss (%)
- Jitter (ms)
- Retransmissions
- CPU utilization
- Test duration
- Timestamp

Directory Structure

```
/home/ubuntu/ipmanager/
├── docker-compose.yml # Main orchestration file
├── .env # Environment variables (Proxmox password)
│
├── backend/
│ ├── Dockerfile # Backend container build
│ ├── main.py # FastAPI application
│ ├── requirements.txt # Python dependencies
│ └── /app # Mounted in container
│
├── frontend/
│ ├── Dockerfile # Frontend container build
│ ├── package.json # Node dependencies
│ ├── src/
│ │ ├── App.js # Main React component
│ │ └── App.css # Glassmorphism styles
│ └── /app # Mounted in container
│
├── mysql/
│ └── init.sql # Database initialization
│
├── monitoring/
│ ├── prometheus/
│ │ ├── prometheus.yml # Prometheus config
│ │ ├── targets/
│ │ ├── nodes.yml # VM node exporters
│ │ └── iperf.yml # iperf3 targets
│ │
│ ├── grafana/
│ │ ├── provisioning/
│ │ │ ├── datasources/
│ │ │ │ └── prometheus.yml # Auto-configure Prometheus
│ │ │ └── dashboards/
│ │ │ └── dashboards.yml # Dashboard config
│ │ └── dashboards/ # Custom dashboards
│ │
│ └── alertmanager/
│ └── config.yml # Alert routing
│
├── volumes/ (Docker managed)
├── mysql-data/ # Database persistence
├── prometheus-data/ # Metrics storage
├── grafana-data/ # Dashboard configs
└── alertmanager-data/ # Alert state
```

Deployment Guide

Prerequisites

• Ubuntu Server 20.04+ at 192.168.0.199

• Docker and Docker Compose installed

• Network access to 192.168.0.x subnet

• Proxmox server accessible at 192.168.0.100

Installation Steps

1. Prepare System

```
# Update system
sudo apt update && sudo apt upgrade -y

# Install Docker
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh
sudo usermod -aG docker $USER

# Install Docker Compose
sudo apt install docker-compose-plugin -y

# Verify installation
docker --version
docker compose version
```

2. Clone/Setup Project

```
# Navigate to home directory
cd /home/ubuntu

# Create project directory (if not exists)
mkdir -p ipmanager
cd ipmanager

# Copy all project files to this location
# (backend/, frontend/, mysql/, monitoring/, docker-compose.yml)
```

3. Configure Environment

```
# Create .env file for sensitive credentials
cat > .env << 'EOF'
PROXMOX_PASSWORD=your_proxmox_password_here
EOF

# Secure the file
chmod 600 .env
```

4. Create Monitoring Structure

```
# Create all monitoring directories
mkdir -p monitoring/{prometheus/targets,grafana/{provisioning/{datasources,dashboards},
dashboards},alertmanager}

# Create Prometheus config
cat > monitoring/prometheus/prometheus.yml << 'YAML'
global:
scrape_interval: 5s
evaluation_interval: 5s

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

```
- job_name: 'node_exporter'
file_sd_configs:
- files: ['/etc/prometheus/targets/nodes.yml']

- job_name: 'iperf3'
file_sd_configs:
- files: ['/etc/prometheus/targets/iperf.yml']
YAML

# Create initial target files
echo '[]' > monitoring/prometheus/targets/nodes.yml
echo '[]' > monitoring/prometheus/targets/iperf.yml

# Create Grafana datasource
cat > monitoring/grafana/provisioning/datasources/prometheus.yml << 'YAML'
apiVersion: 1
datasources:
- name: Prometheus
type: prometheus
access: proxy
url: http://prometheus:9090
isDefault: true
editable: true
YAML

# Create dashboard provisioning
cat > monitoring/grafana/provisioning/dashboards/dashboards.yml << 'YAML'
apiVersion: 1
providers:
- name: 'Default'
orgId: 1
folder: ''
type: file
disableDeletion: false
updateIntervalSeconds: 10
allowUiUpdates: true
options:
path: /var/lib/grafana/dashboards
YAML

# Create Alertmanager config
cat > monitoring/alertmanager/config.yml << 'YAML'
global:
resolve_timeout: 5m

route:
group_by: ['alertname']
group_wait: 10s
group_interval: 10s
repeat_interval: 1h
receiver: 'web.hook'

receivers:
- name: 'web.hook'
webhook_configs:
- url: 'http://127.0.0.1:5001/'
YAML
```

5. Deploy Stack

```
# Build and start all containers
docker compose up -d

# Watch startup logs
docker compose logs -f

# Verify all containers are running
docker ps
```

Expected output:

```
CONTAINER ID IMAGE PORTS NAMES
xxxxxxxxx ipmanager-frontend 0.0.0.0:3000->3000/tcp ipam-frontend
xxxxxxxxx ipmanager-backend (host mode) ipam-backend
xxxxxxxxx mysql:8.0 0.0.0.0:3306->3306/tcp ipam-mysql
xxxxxxxxx phpmyadmin:latest 0.0.0.0:8080->80/tcp phpmyadmin
xxxxxxxxx prom/prometheus:latest 0.0.0.0:9090->9090/tcp prometheus
xxxxxxxxx grafana/grafana:latest 0.0.0.0:3001->3000/tcp grafana
xxxxxxxxx prom/alertmanager:latest 0.0.0.0:9093->9093/tcp alertmanager
```

6. Verify Services

```
# Test each service endpoint
curl http://localhost:3000 # Frontend
curl http://localhost:8000/docs # Backend API docs
curl http://localhost:8080 # phpMyAdmin
curl http://localhost:9090 # Prometheus
curl http://localhost:3001 # Grafana
curl http://localhost:9093 # Alertmanager

# Test database connection
docker exec -it ipam-mysql mysql -uipmanager -pipmanager_pass_2024 -e "SHOW DATABASES;"
```

7. Configure Firewall (Optional)

```
# Allow access from local network only
sudo ufw allow from 192.168.0.0/24 to any port 3000
sudo ufw allow from 192.168.0.0/24 to any port 8000
sudo ufw allow from 192.168.0.0/24 to any port 8080
sudo ufw allow from 192.168.0.0/24 to any port 9090
sudo ufw allow from 192.168.0.0/24 to any port 3001
sudo ufw allow from 192.168.0.0/24 to any port 9093
```

Access Information

Web Interfaces

All services accessible from any device on 192.168.0.x network:

Service	URL	Credentials	Purpose
IP Manager	http://192.168.0.199:3000	None	Main application
API Docs	http://192.168.0.199:8000/docs	None	Interactive API
phpMyAdmin	http://192.168.0.199:8080	root / ipmanager_root_2024	Database admin
Prometheus	http://192.168.0.199:9090	None	Metrics query
Grafana	http://192.168.0.199:3001	admin / admin	Dashboards
Alertmanager	http://192.168.0.199:9093	None	Alert management

Container Access

```
# SSH to Ubuntu server
ssh ubuntu@192.168.0.199

# Access container shell
docker exec -it ipam-backend bash
docker exec -it ipam-frontend bash
```

```
docker exec -it ipam-mysql bash

# View container logs
docker logs ipam-backend -f
docker logs ipam-frontend -f
docker logs prometheus -f
```

Operations Guide

Starting/Stopping Services

```
# Navigate to project directory
cd /home/ubuntu/ipmanager

# Stop all containers
docker compose down

# Start all containers
docker compose up -d

# Restart specific container
docker compose restart backend
docker compose restart frontend

# View status
docker compose ps
```

Viewing Logs

```
# All containers
docker compose logs -f

# Specific container (last 100 lines)
docker logs ipam-backend --tail 100 -f

# Search logs
docker logs ipam-backend 2>&1 | grep "ERROR"
```

Updating Containers

```
# Rebuild and restart specific service
docker compose up -d --build backend

# Rebuild all services
docker compose build
docker compose up -d
```

Database Backup

```
# Backup database
docker exec ipam-mysql mysqldump -uipmanager -pipmanager_pass_2024 ipmanager >
backup-$(date +%Y%m%d).sql

# Restore database
docker exec -i ipam-mysql mysql -uipmanager -pipmanager_pass_2024 ipmanager <
backup.sql
```

Adding VMs to Monitoring

```
# Edit Prometheus targets
nano monitoring/prometheus/targets/nodes.yml

# Add VM targets
- targets:
- '192.168.0.32:9100'
- '192.168.0.33:9100'
labels:
job: 'node_exporter'
environment: 'production'

# Reload Prometheus (no restart needed)
curl -X POST http://localhost:9090/-/reload

# Verify targets
curl http://localhost:9090/api/v1/targets | jq
```

Troubleshooting

Containers Won't Start

```
# Check Docker daemon
sudo systemctl status docker

# View detailed container status
docker compose ps -a

# Check specific container logs
docker logs ipam-backend --tail 50

# Remove and recreate containers
docker compose down
docker compose up -d --force-recreate
```

Network Scanning Not Working

Symptoms: Scan returns no results even though devices are active

Causes:

1. Backend not using host network mode

2. Firewall blocking ICMP

3. Backend container crashed

Solutions:

```
# Verify backend is in host mode
docker inspect ipam-backend | grep -i network

# Should show: "NetworkMode": "host"

# Test ICMP from backend
docker exec ipam-backend ping -c 3 192.168.0.1

# Check backend logs
docker logs ipam-backend --tail 100 | grep -i scan

# Restart backend
docker compose restart backend
```

Database Connection Errors

Symptoms: Backend can't connect to MySQL

Solutions:

```
# Verify MySQL is running and healthy
docker ps | grep mysql
docker exec ipam-mysql mysqladmin ping -h localhost

# Test connection from backend
docker exec ipam-backend nc -zv 127.0.0.1 3306

# Check MySQL logs
docker logs ipam-mysql --tail 50

# Restart MySQL
docker compose restart mysql
```

Prometheus Not Scraping Targets

Symptoms: Targets show as "DOWN" in Prometheus

Solutions:

```
# Check target configuration
cat monitoring/prometheus/targets/nodes.yml

# Test connectivity from Prometheus container
docker exec prometheus wget -O- http://192.168.0.32:9100/metrics

# Verify VM has node_exporter running
ssh ubuntu@192.168.0.32 "systemctl status node_exporter"

# Check Prometheus logs
docker logs prometheus --tail 50
```

System Reboot Issues

Problem: Containers don't start after server reboot

Solution:

```
# Verify Docker starts on boot
sudo systemctl is-enabled docker
sudo systemctl enable docker

# Manually start containers after reboot
cd /home/ubuntu/ipmanager
docker compose up -d

# Or if that doesn't work:
docker compose down
docker compose up -d
```

Note: All containers have `restart: unless-stopped` policy, so they should auto-start after reboot if Docker daemon starts correctly.

Security Considerations

Network Security

• Backend requires host network mode for scanning

- Expose only to trusted 192.168.0.x network
- Use firewall rules to restrict external access
- Consider VPN for remote administration

Credential Management

```
# Secure .env file
chmod 600 .env
chown ubuntu:ubuntu .env

# Rotate passwords regularly
# Update docker-compose.yml environment variables
# Restart affected containers
```

SSH Key Management

For traffic testing, backend needs SSH access to VMs:

```
# Generate SSH key for backend
ssh-keygen -t ed25519 -f ~/.ssh/ipmanager_key

# Add public key to managed VMs
ssh-copy-id -i ~/.ssh/ipmanager_key ubuntu@192.168.0.32

# Configure backend to use key
# (Update backend/main.py with key_filename parameter)
```

Database Security

- MySQL only accepts connections from localhost (backend)
- Use strong passwords (already configured)
- Regular backups with encryption
- Limit phpMyAdmin access to admin workstations

Performance Optimization

Database Tuning

```
-- Add indexes for frequent queries
CREATE INDEX idx_ip_status ON ip_addresses(status);
CREATE INDEX idx_device_history_active ON device_history(active, last_seen);
CREATE INDEX idx_traffic_tests_date ON vm_traffic_tests(created_at);
```

Prometheus Retention

```
# Adjust in monitoring/prometheus/prometheus.yml
storage:
tsdb:
retention.time: 30d # Reduce to 7d for less storage
retention.size: 10GB # Add size limit
```

Container Resource Limits

```
# Add to docker-compose.yml for each service
deploy:
resources:
limits:
cpus: '1.0'
memory: 1G
reservations:
cpus: '0.5'
memory: 512M
```

Backup and Recovery

Full System Backup

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

BACKUP_DIR="/backup/ipmanager/$(date +%Y%m%d)"
mkdir -p $BACKUP_DIR

# Backup database
docker exec ipam-mysql mysqldump -uipmanager -pipmanager_pass_2024 --all-databases >
$BACKUP_DIR/database.sql

# Backup Prometheus data
docker exec prometheus tar czf - /prometheus > $BACKUP_DIR/prometheus-data.tar.gz

# Backup Grafana dashboards
docker exec grafana tar czf - /var/lib/grafana > $BACKUP_DIR/grafana-data.tar.gz

# Backup configuration files
tar czf $BACKUP_DIR/configs.tar.gz /home/ubuntu/ipmanager

echo "Backup completed: $BACKUP_DIR"
```

Recovery Procedure

```
# Stop containers
cd /home/ubuntu/ipmanager
docker compose down

# Restore database
docker exec -i ipam-mysql mysql -uroot -pipmanager_root_2024 < backup/database.sql

# Restore configurations
tar xzf backup/configs.tar.gz -C /

# Restart containers
docker compose up -d
```

Future Enhancements

Planned Features

1. **DHCP Integration**

• Automatic IP reservation from DHCP

• Lease tracking and renewal

• Static IP assignment

2. **VLAN Management**

• Multi-VLAN support

• Per-VLAN scanning

• VLAN-aware VM creation

3. **Advanced Monitoring**

• Custom Grafana dashboards

• Automated alerting rules

• Anomaly detection

4. **API Authentication**

• JWT token authentication

• Role-based access control (RBAC)

• API rate limiting

5. **Automated Testing**

• Scheduled network scans

• Periodic traffic tests

• Health check automation

6. **Mobile App**

• iOS/Android native apps

• Push notifications

• Offline mode support

Technical Specifications

System Requirements

Ubuntu Server:

• OS: Ubuntu 20.04 LTS or newer

• CPU: 4+ cores recommended

• RAM: 8GB minimum, 16GB recommended

• Disk: 50GB+ SSD

• Network: 1Gbps NIC

Managed VMs:

• OS: Ubuntu 20.04+ or compatible Linux

• RAM: 512MB minimum

• Packages: node_exporter, iperf3, openssh-server

Software Versions

Component	Version	Notes
Docker	24.0+	Container runtime
Docker Compose	2.20+	Orchestration
MySQL	8.0	Database
Python	3.9+	Backend runtime
Node.js	18+	Frontend build
Prometheus	Latest	Metrics
Grafana	Latest	Visualization

Network Requirements

• Subnet: 192.168.0.0/24 (configurable)

• ICMP: Enabled for scanning

• Ports: 3000, 8000, 8080, 9090, 3001, 9093

• Proxmox: API access on port 8006

Support and Maintenance

Routine Maintenance

Daily:

• Monitor container health: `docker ps`

• Review error logs: `docker compose logs | grep ERROR`

Weekly:

• Database backup

• Check disk usage: `df -h`

• Review Grafana dashboards

Monthly:

• Update Docker images: `docker compose pull`

• Rotate logs: `docker system prune -f`

• Security updates: `apt update && apt upgrade`

Getting Help

Log Collection:

```
# Collect all logs for debugging
docker compose logs > ipmanager-logs.txt
docker ps -a >> ipmanager-logs.txt
docker compose ps >> ipmanager-logs.txt
```

System Information:

```
# System details
uname -a
docker --version
docker compose version
free -h
df -h
```

Changelog

Version 2.0 (Current)

New Features:

- ■ Unified deployment on single Ubuntu Server
- ■ All 7 containers managed by single docker-compose.yml
- ■ Auto-restart policy (unless-stopped) for all containers
- ■ Backend uses host network mode for direct scanning
- ■ Integrated Proxmox VM creation
- ■ Glassmorphism UI redesign
- ■ Complete monitoring stack (Prometheus + Grafana + Alertmanager)
- ■ Traffic testing with iperf3 integration
- ■ Device history tracking
- ■ Persistent volumes for all data

Bug Fixes:

- ■ Fixed network scanning reliability
- ■ Resolved container restart issues after system reboot
- ■ Corrected database connection handling
- ■ Fixed CORS issues between frontend and backend
- ■ Improved error handling throughout application

Infrastructure Changes:

- ■ Moved from Windows to Ubuntu Server
- ■ Consolidated from multi-server to single-server deployment
- ■ Standardized on Docker Compose orchestration
- ■ Implemented health checks for all critical services
- ■ Added persistent volume management

Version 1.0 (Legacy)

• Basic IP scanning on Windows
• Simple IP tracking
• Manual database management
• No containerization

Appendix A: docker-compose.yml

See `/home/ubuntu/ipmanager/docker-compose.yml` for complete configuration.

Key sections:

• Services definition (7 containers)
• Network configuration (bridge + host)
• Volume mappings (persistent data)
• Environment variables (credentials)
• Health checks (MySQL)
• Restart policies (unless-stopped)

Appendix B: API Reference

Scan Network

```
GET /api/scan/{network}
```

Example: `GET /api/scan/192.168.0.0/24`

Response:

```
{
"active_hosts": [
{
"ip": "192.168.0.32",
"hostname": "vm-test-01",
"mac": "00:11:22:33:44:55",
"status": "active",
"vendor": "Proxmox"
}
],
"scan_time": "2024-12-14T10:30:00Z",
"total_scanned": 254,
"active_count": 12
}
```

Create VM

```
POST /api/create-vm
Content-Type: application/json

{
"vmid": 101,
"name": "new-vm",
```

```
"cores": 2,
"memory": 2048,
"disk": 32
}
```

Response:

```
{
"success": true,
"vmid": 101,
"status": "running",
"created_at": "2024-12-14T10:35:00Z"
}
```

Run Traffic Test

```
POST /api/traffic-test
Content-Type: application/json

{
"source_vm": "192.168.0.32",
"target_vm": "192.168.0.33",
"test_type": "tcp",
"duration": 60,
"bandwidth": "100M"
}
```

Response:

```
{
"test_id": 42,
"status": "completed",
"throughput_mbps": 95.2,
"packet_loss_pct": 0.0,
"jitter_ms": 0.5,
"test_duration": 60
}
```

Appendix C: Monitoring Queries

Prometheus Queries

Network throughput:

```
rate(node_network_receive_bytes_total{device="eth0"}[5m])
```

CPU usage:

```
100 - (avg by (instance) (rate(node_cpu_seconds_total{mode="idle"}[5m])) * 100)
```

Memory usage:

```
100 * (1 - ((node_memory_MemAvailable_bytes / node_memory_MemTotal_bytes)))
```

Active iperf3 connections:

```
iperf3_active_connections
```

Appendix D: Network Scanning Implementation with nmap

Overview

The IP Manager backend uses **nmap** (Network Mapper) for network discovery and device enumeration. This provides reliable, fast, and detailed information about active devices on the network.

Why nmap?

Advantages:

• **Mature and battle-tested**: Industry-standard network scanning tool used worldwide

• **Fast scanning**: Optimized algorithms for quick network discovery

• **Rich device information**: Hostname, MAC address, vendor identification

• **Low false positives**: Reliable detection of active hosts

• **Cross-platform**: Works consistently across different operating systems

• **No elevated privileges needed**: Can perform host discovery without root access

Comparison to alternatives:

• **vs. ICMP ping**: nmap uses multiple techniques (ARP, ICMP, TCP) for better detection

• **vs. scapy**: nmap is faster and more reliable for large-scale scanning

• **vs. custom socket programming**: nmap handles edge cases and optimizations automatically

Implementation Details

Python Library:

```
import nmap
nm = nmap.PortScanner()
```

The backend uses **python-nmap**, which is a Python wrapper around the nmap command-line tool.

Dockerfile Installation:

```
RUN apt-get update && apt-get install -y \
nmap \
...
```

The nmap binary is installed in the backend container during build time.

Scan Configuration:

```
nm.scan(hosts=ip_range, arguments='-sn -n -T4')
```

Nmap Arguments Explained

Argument	Purpose	Impact
`-sn`	Ping scan (no port scan)	Fast host discovery only, doesn't scan ports
`-n`	No DNS resolution	Speeds up scanning by skipping reverse DNS lookups
`-T4`	Aggressive timing	Faster scans (level 4 of 0-5), safe for most LANs

Additional timing details:

• `-T0` (Paranoid): Extremely slow, evades IDS

- `-T1` (Sneaky): Very slow
- `-T2` (Polite): Slows to reduce bandwidth
- `-T3` (Normal): Default balanced timing
- `-T4` (Aggressive): **Our choice** - Fast for reliable LANs
- `-T5` (Insane): Extremely fast, may miss hosts

Scan Process Flow

```
User clicks "Scan Network"
↓
Frontend → Backend API: GET /api/scan/192.168.0.0/24
↓
Backend invokes nmap
↓
nmap scans 192.168.0.1-254
- Sends ARP requests (for local network)
- Sends ICMP echo requests
- Checks for responses
↓
nmap returns results
- Active hosts with IP
- MAC addresses
- Hostnames (if available)
- Vendor info (from MAC OUI)
↓
Backend processes results
- Updates MySQL database
- Tracks device history
- Marks previously-seen devices
↓
Backend returns JSON to frontend
↓
Frontend displays results in UI
```

Data Extraction

From nmap scan results:

```
# Check if host is up
is_up = ip in active_ips and nm[ip].state() == "up"

# Get hostname
if 'hostnames' in host_info and host_info['hostnames']:
hostname = host_info['hostnames'][0].get('name')

# Get MAC address
if 'addresses' in host_info:
mac_address = host_info['addresses'].get('mac')

# Get vendor (from MAC OUI database)
if mac_address and 'vendor' in host_info:
vendor = host_info['vendor'].get(mac_address)
```

MAC OUI (Organizationally Unique Identifier):

• First 3 bytes of MAC address identify manufacturer

• nmap includes built-in OUI database

• Examples: `00:0C:29` = VMware, `08:00:27` = VirtualBox, `DC:A6:32` = Raspberry Pi

Network Mode: host

Critical requirement:

```
backend:
network_mode: host
```

Why host mode is required:

1. **Direct network interface access**: nmap needs to send raw packets on the network
2. **ARP scanning**: On local networks, nmap uses ARP for faster, more reliable detection
3. **No NAT/bridge overhead**: Direct access to 192.168.0.x network without Docker networking layer
4. **Accurate source addressing**: Responses come back directly to the backend

Alternative approaches (not used):

• Bridge mode with `--privileged`: Security risk, unnecessary

• Host networking tools in bridge: Complex, unreliable

• Pure Python scanning: Slower, less reliable

Performance Characteristics

Scan speed:

• **192.168.0.1-254** (254 IPs): 2-5 seconds typical

• **Single IP**: <100ms

• **Subnet /24**: 2-10 seconds depending on active hosts

Resource usage:

• CPU: Minimal (mostly I/O bound)

• Memory: ~20MB per scan

• Network: Burst of packets, then idle

Factors affecting speed:

• Network latency

• Number of active devices

• Response time of devices

• Network congestion

Database Integration

After each scan:

```
# For each IP in range
for ip in range(start, end):
if ip is active:
# Create or update node
update_or_create_node(conn, ip, status='up',
hostname=hostname, mac=mac, vendor=vendor)
# Record in history
record_history(conn, ip, status='up')
else:
if was_previously_active:
# Mark as previously_used
update_node(conn, ip, status='previously_used')
```

Database behavior:

• **New active device**: Create record with `status='up'`

• **Known device online**: Update `last_seen`, increment `times_seen`

• **Known device offline**: Change `status='previously_used'`

• **Never-seen inactive IP**: No database record created

Security Considerations

Network scanning ethics:

• ■■ Only scan networks you own or have permission to scan

• ■■ Scanning unauthorized networks may be illegal

• ■ IP Manager is designed for internal network management

Container security:

• Backend runs in Docker with limited privileges

• No `--privileged` flag needed

• `network_mode: host` is the only special permission

• MySQL credentials are environment variables, not hardcoded

Firewall considerations:

• nmap generates legitimate network traffic

• May trigger IDS/IPS alerts if configured

• Local LANs typically don't block ARP or ICMP

Troubleshooting nmap Scans

No devices found:

```
# Test nmap manually in backend container
docker exec -it ipam-backend nmap -sn 192.168.0.1-10

# Check network mode
docker inspect ipam-backend | grep NetworkMode
# Should show: "NetworkMode": "host"

# Test basic connectivity
docker exec -it ipam-backend ping -c 3 192.168.0.1
```

Incomplete results:

```
# Increase verbosity for debugging
nm.scan(hosts=ip_range, arguments='-sn -n -T4 -v')

# Check backend logs
docker logs ipam-backend | grep -i nmap
```

Permission issues:

```
# Verify nmap is installed
docker exec ipam-backend which nmap
docker exec ipam-backend nmap --version

# Check for network connectivity
docker exec ipam-backend ip addr show
```

Alternative Scan Methods (Not Implemented)

Why we don't use these:

1. **Pure ICMP ping (subprocess)**:

• ■ Only detects ICMP-responsive devices

• ■ No MAC/vendor info

• ■ Slower than nmap

2. **Scapy (Python packet manipulation)**:

• ■ Requires root privileges

• ■ More complex code

• ■ Slower than nmap

• ■ More flexible for custom protocols

3. **ARP-scan**:

• ■ Only works on local subnet

• ■ Limited vendor database

• ■ Very fast for local networks

• ■ Lightweight

4. **Masscan (ultra-fast port scanner)**:

• ■ Overkill for host discovery

• ■ Can overwhelm networks

• ■ Excellent for large-scale port scanning

Why nmap is the best choice for IP Manager:

• Perfect balance of speed, reliability, and features

• Rich device information (hostname, MAC, vendor)

• Doesn't require root/sudo

• Battle-tested and maintained

• Works consistently in Docker containers

Example Scan Output

Backend console output:

```
============================================================
Scanning 192.168.0.0-255 with nmap...
============================================================
Nmap found 12 responding hosts
UP: 192.168.0.1 (NETGEAR)
UP: 192.168.0.10 (Apple)
UP: 192.168.0.32 (Proxmox)
UP: 192.168.0.33 (Proxmox)
UP: 192.168.0.50 (Dell Inc.)
UP: 192.168.0.100 (Proxmox VE)
UP: 192.168.0.199 (Ubuntu)
...
Scan completed in 3.2 seconds
Database updated: 12 active, 5 previously_used
```

JSON Response to frontend:

```
{
"subnet": "192.168.0",
```

```
"total_ips": 254,
"active_ips": 12,
"inactive_ips": 242,
"previously_used_ips": 5,
"scan_time": 3.24,
"results": [
{
"ip": "192.168.0.1",
"status": "up",
"hostname": "gateway",
"mac_address": "A4:91:B1:XX:XX:XX",
"vendor": "NETGEAR",
"last_scanned": "2024-12-14T15:30:00Z",
"first_seen": "2024-12-01T10:00:00Z",
"times_seen": 145
}
]
}
```


Future Enhancements

Potential improvements:

1. **Parallel scanning**: Scan multiple subnets simultaneously
2. **Service detection**: Add `-sV` flag for version detection
3. **OS detection**: Add `-O` for operating system fingerprinting
4. **Custom timing**: Allow users to choose scan speed
5. **Scheduled scans**: Automatic periodic network discovery
6. **Diff reports**: Highlight new/missing devices since last scan

References

Official Documentation:

• nmap documentation: https://nmap.org/book/man.html

• python-nmap library: https://pypi.org/project/python-nmap/

• nmap timing options: https://nmap.org/book/man-performance.html

Backend Implementation:

• File: `/home/ubuntu/ipmanager/backend/main.py`

• Function: `async def scan_ip_range(subnet: str, start_ip: int, end_ip: int)`

• Dockerfile: `/home/ubuntu/ipmanager/backend/Dockerfile` (line 10: nmap installation)

Document Information

Author: Francisco

Organization: Internal Infrastructure

Last Revision: December 14, 2024

Document Version: 2.0

Status: Production

Related Documentation:

• Docker Compose Reference: docker-compose.yml

• API Documentation: http://192.168.0.199:8000/docs

• Proxmox API: https://pve.proxmox.com/wiki/Proxmox_VE_API

Revision History:

• v2.0 (2024-12-14): Complete rewrite for unified Ubuntu deployment

• v1.5 (2024-11): Added monitoring stack integration

• v1.0 (2024-10): Initial Windows-based design

END OF DOCUMENT