IP Manager - Network Monitoring & Management System

Comprehensive Design & Operations Document

Version: 2.0

Last Updated: December 2024

Author: IP Manager Development Team

Table of Contents

System Overview

Architecture

Features

Component Details

Setup & Installation

VM Template Configuration

Automated Monitoring Setup

Security Configuration

Usage Guide

Debugging & Troubleshooting

Maintenance

API Reference

System Overview

Purpose

IP Manager is a comprehensive network infrastructure monitoring and management system designed for enterprise environments. It provides:

• Real-time network device discovery and tracking

• Proxmox VM provisioning and management

• Network performance testing with iperf3

• Container infrastructure monitoring

• Historical tracking and analytics

Key Capabilities

• **Network Discovery**: Automated scanning of IP ranges with device identification

• **VM Lifecycle Management**: Create, configure, and provision VMs on Proxmox

• **Performance Testing**: Network bandwidth and latency testing between VMs

• **Infrastructure Monitoring**: Real-time container and VM health tracking

• **Automated Setup**: New VMs are automatically configured and ready for testing

Technology Stack

• **Frontend**: React.js with glassmorphism UI

• **Backend**: FastAPI (Python)

• **Database**: MySQL 8.0

• **Virtualization**: Proxmox VE

• **Monitoring**: Prometheus + Grafana + cAdvisor

• **Container Orchestration**: Docker Compose

• **Network Testing**: iperf3 + node_exporter

Architecture

High-Level Architecture

```
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ ■ User
Interface ■ ■ (React Frontend - Port 3000) ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ ■ API
Layer ■ ■ (FastAPI Backend - Port 8000) ■ ■ • Network Scanning • VM Management • Traffic
Testing ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ ■ ■ ■ ■
■ ■ ■ ■ ■■■■■▼■■■■■ ■■■■■▼■■■■■■ ■■■■■▼■■■■■■■ ■■■■■▼■■■■■■■■■■ ■ MySQL ■ ■Proxmox ■ ■
Prometheus■ ■ VMs ■ ■ DB ■ ■ VE ■ ■ +Grafana ■ ■(iperf3 + ■ ■ ■ ■ ■ ■ ■ ■node_export)■
■■■■■■■■■■ ■■■■■■■■■■■ ■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■
```

Container Architecture

```
■■■■■■■■■■■■■■■■■■■■■■ Docker Compose Stack ■■■■■■■■■■■■■■■■■■■■■■■ ■ ■ ■
■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■ ■ ■ ■ ipam-frontend■ ■
ipam-backend ■ ■ ipam-mysql ■ ■ ■ ■ (React) ■ ■ (FastAPI) ■ ■ (MySQL) ■ ■ ■
■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■ ■ ■ ■ ■ ■■■■■■■■■■■■■■■■■■
■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■ ■ ■ ■ prometheus ■ ■ grafana ■ ■ cadvisor ■ ■ ■ ■
(Metrics DB) ■ ■ (Dashboard) ■ ■(Container ■ ■ ■ ■ ■ ■ ■ ■ Monitoring) ■ ■ ■
■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■ ■ ■ ■ ■ ■■■■■■■■■■■■■■■■■■
■■■■■■■■■■■■■■■■■■ ■ ■ ■alertmanager ■ ■ phpmyadmin ■ ■ ■ ■ (Alerts) ■ ■ (DB Admin) ■ ■ ■
■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■ ■ ■ ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
```

Network Flow

```
User → Frontend (3000) → Backend (8000) → MySQL (3306) ↓ Proxmox API (8006) ↓ VM Creation ↓
Auto-add to Prometheus ↓ Metrics Collection ↓ Grafana Visualization
```

Features

1. Network Discovery & Scanning

• **Subnet Scanning**: Scan any /24 subnet for active devices

• **Device Identification**: MAC address lookup, vendor identification, hostname resolution

• **Multi-Network Support**: Automatic detection of available networks

• **Historical Tracking**: Track device uptime, first/last seen timestamps

• **Status Categories**:

• Active (currently online)

• Available (never used)

• Previously Used (offline but seen before)

• Reserved (manually reserved)

2. IP Address Management

• **Reservation System**: Reserve IPs for specific purposes

• **Custom Notes**: Add detailed notes to any IP address

• **Status Management**: Manual status reset, network clearing

• **Visual Grid**: 16x16 grid showing all 256 addresses in a subnet

• **Filtering**: Filter by status (all, active, available, previously used)

3. Proxmox VM Management

• **Automated Provisioning**: Create VMs with one click

• **Template-Based Cloning**: Clone from pre-configured templates

• **Custom Configuration**:

• CPU cores (configurable)

• Memory (configurable)

• Disk size (configurable)

• Static IP assignment

• Gateway and DNS configuration

• **Auto-Start Option**: Start VM immediately after creation

• **Automatic Monitoring Setup**: VMs are automatically added to Prometheus

4. Network Performance Testing

• **iperf3 Integration**: Bandwidth and performance testing

• **Protocol Support**: TCP and UDP testing

• **Configurable Parameters**:

• Test duration (10-300 seconds)

• Bandwidth limits (10M - 10G)

• Parallel streams (1-10)

• Reverse mode (server sends)

• **Real-Time Results**: View test results in Grafana dashboards

• **Historical Data**: All tests stored for analysis

5. Infrastructure Monitoring

• **Container Monitoring**:

• Real-time status table showing all containers

• CPU usage per container

• Memory usage per container

• Network I/O (RX/TX)

• Restart counts

• **VM Monitoring**:

• Node exporter metrics from all VMs

• System-level metrics (CPU, memory, disk, network)

• **Alerting**: Configurable alerts for threshold violations

• **Dashboards**: Professional Grafana dashboards for visualization

6. Automated VM Preparation

Key Innovation: New VMs are automatically ready for traffic testing without manual setup.

When a VM is created from template 9001:

■ VM cloned with Ubuntu + cloud-init

■ iperf3 pre-installed and running as service

■ node_exporter pre-installed and exporting metrics

■ Automatic Prometheus target registration

■ Ready for traffic tests immediately

Component Details

Frontend (React)

Port: 3000

Technology: React.js, Glassmorphism UI

Key Files:

• `frontend/src/App.js` - Main application component

• `frontend/src/App.css` - Styling with glassmorphism effects

• `frontend/src/network-styles.css` - Network visualization styles

Features:

• Real-time IP grid visualization

• VM creation modal with form validation

• Traffic test configuration interface

• Network discovery modal

• Status filtering and statistics

Backend (FastAPI)

Port: 8000

Technology: Python 3.12, FastAPI, uvicorn

Key Files:

• `backend/main.py` - Main API application

• `backend/requirements.txt` - Python dependencies

API Endpoints:

• `POST /api/scan` - Network scanning

• `GET /api/node/{ip}` - Get node details with history

• `POST /api/reserve` - Reserve IP address

• `POST /api/release/{ip}` - Release reserved IP

• `PUT /api/node/update` - Update node notes

• `POST /api/network/clear/{subnet}` - Clear network data

• `POST /api/network/reset-status/{subnet}` - Reset node statuses

• `GET /api/networks/discover` - Discover available networks

• `GET /api/proxmox/status` - Check Proxmox connection

• `GET /api/proxmox/templates` - List available VM templates

• `POST /api/proxmox/create-vm` - Create new VM

• `POST /api/traffic/start` - Start traffic test

• `GET /api/traffic/status/{test_id}` - Check test status

• `GET /api/traffic/results/{test_id}` - Get test results

Key Functions:

• `add_prometheus_target(ip_address)` - Auto-register VMs in Prometheus

• `ProxmoxAPI` class - Proxmox VE API wrapper

• Network scanning with nmap

• Database operations with MySQL

Database (MySQL)

Port: 3306

Database: ipmanager

Tables:

• `nodes` - IP address inventory and status

• `scan_history` - Historical scan results

• `reservations` - IP reservations

• `traffic_tests` - Network test results

Schema:

```
CREATE TABLE nodes ( id INT AUTO_INCREMENT PRIMARY KEY, ip_address VARCHAR(15) UNIQUE NOT
NULL, mac_address VARCHAR(17), hostname VARCHAR(255), vendor VARCHAR(255), status VARCHAR(20),
first_seen DATETIME, last_seen DATETIME, times_seen INT DEFAULT 0, is_reserved BOOLEAN DEFAULT
FALSE, reserved_for VARCHAR(255), reserved_by VARCHAR(100), reserved_at DATETIME, notes TEXT
);
```

Prometheus

Port: 9090

Purpose: Metrics collection and time-series database

Scrape Targets:

• cadvisor:8080 - Container metrics

• VMs at port 9100 - Node exporter metrics

Configuration: `monitoring/prometheus/prometheus.yml`

Targets: `monitoring/prometheus/targets/nodes.yml`

Key Metrics:

• `container_cpu_usage_seconds_total` - Container CPU

• `container_memory_usage_bytes` - Container memory

• `container_network_transmit_bytes_total` - Network TX

• `container_network_receive_bytes_total` - Network RX

• `node_*` - VM system metrics

Grafana

Port: 3001

Default Credentials: admin/admin (configurable in .env)

Dashboards:

- **IP Manager - Container Health**: Container monitoring dashboard
- **Network Traffic Tests**: iperf3 test results (if configured)

Features:

- 5-second auto-refresh
- Real-time container status table
- Time-series graphs for CPU, Memory, Network
- Alert visualization

cAdvisor

Port: 8081

Purpose: Container monitoring and metrics collection

Metrics Exposed: `/metrics` endpoint for Prometheus

Access: http://localhost:8081 for web UI

Setup & Installation

Prerequisites

- Ubuntu 22.04+ (host system)
- Docker Engine 24+
- Docker Compose V2
- Proxmox VE 7.0+ (separate server)
- Network connectivity between host and Proxmox
- Sufficient resources: 4GB RAM, 20GB disk

Installation Steps

1. Clone Repository

```
cd ~ git clone <repository-url> ipmanager cd ipmanager
```

2. Configure Environment Variables

```
nano .env
```

Add the following:

```
# MySQL Configuration MYSQL_ROOT_PASSWORD=your_secure_root_password MYSQL_DATABASE=ipmanager
MYSQL_USER=ipmanager MYSQL_PASSWORD=your_secure_mysql_password MYSQL_HOST=127.0.0.1
MYSQL_PORT=3306 # Proxmox Configuration PROXMOX_HOST=192.168.0.100 PROXMOX_PORT=8006
PROXMOX_USER=root@pam PROXMOX_PASSWORD=your_proxmox_password PROXMOX_NODE=proxmox # Grafana
Configuration GF_SECURITY_ADMIN_USER=admin GF_SECURITY_ADMIN_PASSWORD=your_grafana_password
```

Security Note: Set permissions `chmod 600 .env` to protect credentials.

3. Verify docker-compose.yml

Ensure backend service has:

```
backend: network_mode: host extra_hosts: - "proxmox:${PROXMOX_HOST}" env_file: - .env
```

4. Start Services

```
docker compose up -d
```

5. Verify Services

```
# Check all containers are running docker ps # Expected output: 8 containers running # -
ipam-frontend # - ipam-backend # - ipam-mysql # - prometheus # - grafana # - cadvisor # -
alertmanager # - phpmyadmin
```

6. Access Applications

• **IP Manager UI**: http://localhost:3000

• **Grafana**: http://localhost:3001 (admin/admin)

• **Prometheus**: http://localhost:9090

• **cAdvisor**: http://localhost:8081

• **phpMyAdmin**: http://localhost:8080

7. Initial Configuration

Open IP Manager UI

Click network discovery icon (■)

Select your network

Click "Start Scan"

View discovered devices in the grid

VM Template Configuration

Purpose

Pre-configured VM templates ensure new VMs are immediately ready for:

• Network performance testing

• Metrics collection

• Monitoring integration

Creating the Base Template (Template 9001)

Step 1: Prepare Cloud Image

SSH into your Proxmox server:

```
ssh root@192.168.0.100 cd /var/lib/vz/template/iso wget
https://cloud-images.ubuntu.com/jammy/current/jammy-server-cloudimg-amd64.img
```

Step 2: Create VM from Cloud Image

```
# Create VM 9001 qm create 9001 --name ubuntu-ipmanager-template --memory 2048 --cores 2
--net0 virtio,bridge=vmbr0 # Import cloud image as disk qm importdisk 9001
jammy-server-cloudimg-amd64.img local-lvm # Attach disk qm set 9001 --scsihw virtio-scsi-pci
--scsi0 local-lvm:vm-9001-disk-0 # Add cloud-init drive qm set 9001 --ide2 local-lvm:cloudinit
# Set boot order qm set 9001 --boot c --bootdisk scsi0 # Add serial console qm set 9001
--serial0 socket --vga serial0 # Enable QEMU guest agent qm set 9001 --agent enabled=1 #
Resize disk to 32GB qm resize 9001 scsi0 32G # Set default credentials qm set 9001 --ciuser
root qm set 9001 --cipassword ubuntu # Set network to DHCP (will be overridden per VM) qm set
9001 --ipconfig0 ip=dhcp
```

Step 3: Boot and Configure VM

```
# Start the VM qm start 9001 # Wait 30 seconds for boot sleep 30 # Get the VM's DHCP IP (check
Proxmox UI or use) qm guest exec 9001 -- ip -4 addr show
```

SSH into the VM (replace with actual IP):

```
ssh root@<vm-ip> # Password: ubuntu
```

Step 4: Install Monitoring Tools

Inside the VM:

```
# Update packages apt update # Install required packages apt install -y iperf3
prometheus-node-exporter qemu-guest-agent curl net-tools # Create iperf3 systemd service cat >
/etc/systemd/system/iperf3.service << 'EOF' [Unit] Description=iPerf3 Server
After=network.target [Service] Type=simple ExecStart=/usr/bin/iperf3 -s Restart=always
RestartSec=3 User=root [Install] WantedBy=multi-user.target EOF # Enable services (they'll
auto-start on cloned VMs) systemctl daemon-reload systemctl enable iperf3 systemctl enable
prometheus-node-exporter systemctl enable qemu-guest-agent # Verify services are enabled
systemctl is-enabled iperf3 systemctl is-enabled prometheus-node-exporter systemctl is-enabled
qemu-guest-agent # Clean up to reduce template size apt clean apt autoremove -y # Clear
machine-id so each clone gets unique ID truncate -s 0 /etc/machine-id rm
/var/lib/dbus/machine-id ln -s /etc/machine-id /var/lib/dbus/machine-id # Clear cloud-init so
it runs fresh on clones cloud-init clean --logs --seed # Clear bash history history -c cat
/dev/null > ~/.bash_history # Shutdown shutdown -h now
```

Step 5: Convert to Template

Back on Proxmox host:

```
# Wait for VM to shut down qm wait 9001 # Convert to template qm template 9001 # Verify it's a
template qm list | grep 9001
```

Template Verification

After template creation:

Template 9001 should appear in IP Manager's "Create VM" dropdown

New VMs cloned from this template will have:

- ■ iperf3 service running on port 5201
- ■ node_exporter exporting metrics on port 9100
- ■ qemu-guest-agent for Proxmox integration

Automated Monitoring Setup

How It Works

When a new VM is created through IP Manager:

1. VM Creation Flow

```
User clicks "Create VM" ↓ Frontend sends request to backend ↓ Backend calls Proxmox API to
clone template ↓ Proxmox creates VM with static IP ↓ Backend calls
add_prometheus_target(ip_address) ↓ IP added to prometheus/targets/nodes.yml ↓ Prometheus
reloaded to pick up new target ↓ VM starts and services auto-start ↓ Prometheus begins
scraping VM metrics ↓ Grafana displays VM data
```

2. Backend Auto-Registration Code

Located in `backend/main.py`:

```
def add_prometheus_target(ip_address: str): """Add a new VM to Prometheus targets"""
targets_file = Path("/app/monitoring/prometheus/targets/nodes.yml") try: # Read existing
targets if targets_file.exists(): with open(targets_file, 'r') as f: data = yaml.safe_load(f)
or [] else: data = [] # Ensure proper structure if not data: data = [{ 'targets': [], 'labels':
{ 'job': 'node_exporter', 'environment': 'production' } }] # Add new target if not present
new_target = f"{ip_address}:9100" if new_target not in data[0]['targets']:
data[0]['targets'].append(new_target) data[0]['targets'].sort() # Keep sorted # Write back to
file with open(targets_file, 'w') as f: yaml.dump(data, f, default_flow_style=False,
sort_keys=False) # Reload Prometheus try: requests.post('http://localhost:9090/-/reload',
timeout=5) print(f"✓ Added {ip_address} to Prometheus targets and reloaded") except Exception
as e: print(f"■ Added target but couldn't reload Prometheus: {e}") return True else:
print(f"Target {ip_address} already exists in Prometheus") return False except Exception as e:
print(f"Failed to add Prometheus target: {e}") return False # Called after VM creation
@app.post("/api/proxmox/create-vm") async def create_proxmox_vm(request: ProxmoxVMRequest): #
... VM creation code ... if response.ok: # ... database update ... # ADD TO PROMETHEUS TARGETS
add_prometheus_target(request.ip_address) return { "success": True, "vmid": vmid, "vm_name":
request.vm_name, "ip_address": request.ip_address }
```

3. Verification Process

After creating a VM, verify the automation worked:

```
# 1. Check VM was added to Prometheus targets cat
~/ipmanager/monitoring/prometheus/targets/nodes.yml # Should show your new VM IP # - targets:
# - 192.168.0.XX:9100 # 2. Check Prometheus is scraping the VM curl -s
http://localhost:9090/api/v1/targets | grep "192.168.0.XX" # Should show: "health":"up" # 3.
SSH into VM and verify services ssh root@192.168.0.XX # Check iperf3 systemctl status iperf3
ss -tlnp | grep 5201 # Check node_exporter systemctl status prometheus-node-exporter curl
http://localhost:9100/metrics | head # 4. Verify in Grafana # Open http://localhost:3001 #
Check if VM appears in graphs
```

Troubleshooting Automation

If VMs aren't automatically added to monitoring:

Check 1: Backend has access to targets file

```
docker exec ipam-backend ls -la /app/monitoring/prometheus/targets/
```

Check 2: Volume mount is correct

```
grep -A5 "backend:" ~/ipmanager/docker-compose.yml | grep volumes
```

Should show:

```
volumes: - ./monitoring/prometheus/targets:/app/monitoring/prometheus/targets
```

Check 3: PyYAML is installed

```
docker exec ipam-backend python3 -c "import yaml; print('OK')"
```

Check 4: Prometheus reload endpoint is accessible

```
docker exec ipam-backend curl -X POST http://localhost:9090/-/reload
```

Security Configuration

1. Environment Variables (.env)

Location: `~/ipmanager/.env`

Permissions: `chmod 600 .env`

Gitignore: Ensure `.env` is in `.gitignore`

Never commit:

- Passwords
- API keys
- Database credentials
- Proxmox credentials

2. Network Security

- Backend uses `network_mode: host` for Proxmox access
- Containers communicate via Docker network
- Expose only necessary ports to host

Exposed Ports:

- 3000 - Frontend (can restrict to localhost)

• 3001 - Grafana (can restrict to localhost)

• 8000 - Backend API (can restrict to localhost)

• 8080 - phpMyAdmin (should restrict to localhost)

• 9090 - Prometheus (should restrict to localhost)

Production Recommendation: Use nginx reverse proxy with SSL

3. Database Security

• Strong MySQL root password

• Separate MySQL user for application

• Database accessible only from Docker network

• Regular backups recommended

4. Proxmox Security

• Use dedicated Proxmox user (not root@pam)

• Create API token instead of password

• Limit permissions to VM management only

Creating Proxmox API Token:

```
# On Proxmox pveum user add ipmanager@pve pveum aclmod / -user ipmanager@pve -role PVEVMAdmin
pveum user token add ipmanager@pve ipmanager-token
```

Update `.env`:

```
PROXMOX_USER=ipmanager@pve PROXMOX_TOKEN_NAME=ipmanager-token
PROXMOX_TOKEN_VALUE=<generated-token>
```

5. Container Security

• Run containers as non-root where possible

• Use official images only

• Regular image updates

• Scan for vulnerabilities

```
# Update all images docker compose pull docker compose up -d
```

Usage Guide

Creating a VM

Access IP Manager: http://localhost:3000

Scan Network: Click ■ icon, select network, start scan

Select Available IP: Click on green (available) cell

Click "Create Proxmox VM"

Configure VM:

• **VM Name**: Enter descriptive name

• **Template**: Select template 9001 (pre-configured)

• **CPU Cores**: 2-4 recommended

• **Memory**: 2048MB minimum

• **Disk Size**: 32GB recommended

• **Gateway**: Usually .1 of your subnet

• **DNS**: 8.8.8.8 or your DNS server

• **Start VM**: Check to auto-start

Click "Create VM"

Wait: VM creation takes 30-60 seconds

Verify:

• VM appears in Proxmox UI

• IP marked as reserved in IP Manager

• VM appears in Grafana after 1-2 minutes

Running Traffic Tests

Ensure Source VM has iperf3: Check "Check Monitoring" button

Click "Traffic Test" on source VM

Select Target VM: Choose from dropdown

Configure Test:

• **Protocol**: TCP or UDP

• **Duration**: 60 seconds recommended

• **Bandwidth**: Start with 100M

• **Parallel Streams**: 1 for basic, 4-8 for stress test

Start Test

Monitor:

• Real-time status in UI

• Detailed graphs in Grafana

• Results saved to database

Monitoring Containers

Access Grafana: http://localhost:3001

Login: admin/admin (or configured password)

Open Dashboard: "IP Manager - Container Health"

View Table: Top panel shows all containers with status

Analyze Graphs:

• CPU usage trends

• Memory consumption

• Network traffic patterns

• Restart counts

Debugging & Troubleshooting

Container Issues

Problem: Container not showing in status table

Diagnosis:

```
# 1. Check container is actually running docker ps | grep <container-name> # 2. Check if
cAdvisor sees it curl -s http://localhost:8081/metrics | grep "name=\"<container-name>\"" # 3.
Check Prometheus is scraping cAdvisor curl -s http://localhost:9090/api/v1/targets | grep
cadvisor # Should show: "health":"up"
```

Solutions:

• **Container not running**: `docker compose up -d`

- **cAdvisor not seeing it**: `docker compose restart cadvisor`
- **Prometheus not scraping**: Check prometheus.yml has cadvisor target

Problem: cAdvisor shows down in Prometheus

Diagnosis:

```
# Check cAdvisor IP docker inspect cadvisor | grep '"IPAddress"' | tail -1 # Check what IP
Prometheus is using grep cadvisor ~/ipmanager/monitoring/prometheus/prometheus.yml
```

Solution: IP addresses may change after restarts

```
# Get current IP CADVISOR_IP=$(docker inspect cadvisor | grep '"IPAddress"' | tail -1 | awk
-F'"' '{print $4}') # Update prometheus.yml nano
~/ipmanager/monitoring/prometheus/prometheus.yml # Change to: # - targets:
['<CADVISOR_IP>:8080'] # OR better: use container name # - targets: ['cadvisor:8080'] # Reload
Prometheus curl -X POST http://localhost:9090/-/reload
```

Permanent Fix: Use container name instead of IP

```
# Add to docker-compose.yml under prometheus service links: - cadvisor # Change prometheus.yml
to use name - targets: ['cadvisor:8080']
```

Problem: Grafana shows "No Data"

Diagnosis:

```
# 1. Check Grafana can reach Prometheus docker exec grafana curl -s
http://prometheus:9090/api/v1/query?query=up # 2. Check datasource UID curl -s -u admin:admin
http://localhost:3001/api/datasources | python3 -c "import sys, json; [print(f\"{d['name']}:
uid={d['uid']}\") for d in json.load(sys.stdin)]" # Should show: Prometheus: uid=prometheus
```

Solution: Datasource UID mismatch

```
# Recreate datasource with correct UID cat >
~/ipmanager/monitoring/grafana/provisioning/datasources/prometheus.yml << 'EOF' apiVersion: 1
deleteDatasources: - name: Prometheus orgId: 1 datasources: - name: Prometheus type:
prometheus access: proxy url: http://prometheus:9090 isDefault: true uid: prometheus editable:
true jsonData: timeInterval: 5s EOF # Restart Grafana docker compose restart grafana
```

VM Monitoring Issues

Problem: VM not appearing in Prometheus targets

Diagnosis:

```
# 1. Check if VM was added to targets file cat
~/ipmanager/monitoring/prometheus/targets/nodes.yml | grep <vm-ip> # 2. Check Prometheus
targets curl -s http://localhost:9090/api/v1/targets | grep <vm-ip> # 3. Check backend logs
docker logs ipam-backend | grep "Added.*to Prometheus"
```

Solutions:

A. VM not in targets file - Manual add:

```
nano ~/ipmanager/monitoring/prometheus/targets/nodes.yml # Add under targets: - targets: -
192.168.0.XX:9100 # Your VM IP labels: job: node_exporter environment: production # Reload
Prometheus curl -X POST http://localhost:9090/-/reload
```

B. Backend can't write to targets file - Fix permissions:

```
# Check volume mount docker exec ipam-backend ls -la /app/monitoring/prometheus/targets/ # Fix
if needed chmod 755 ~/ipmanager/monitoring/prometheus/targets/ chmod 644
~/ipmanager/monitoring/prometheus/targets/nodes.yml
```

C. PyYAML not installed:

```
docker exec ipam-backend pip list | grep yaml # If not found, add to requirements.txt echo
"pyyaml" >> ~/ipmanager/backend/requirements.txt docker compose build backend docker compose
up -d backend
```

Problem: VM shows as "down" in Prometheus

Diagnosis:

```
# 1. Check if VM is accessible ping -c 2 <vm-ip> # 2. Check if node_exporter is running on VM
ssh root@<vm-ip> "systemctl status prometheus-node-exporter" # 3. Test metrics endpoint curl
http://<vm-ip>:9100/metrics | head
```

Solutions:

A. VM not responding - Check VM in Proxmox:

```
# On Proxmox qm status <vmid> qm start <vmid> # If stopped
```

B. node_exporter not running - Start service:

```
ssh root@<vm-ip> systemctl start prometheus-node-exporter systemctl enable
prometheus-node-exporter
```

C. Port 9100 blocked - Check firewall:

```
ssh root@<vm-ip> ufw status ufw allow 9100/tcp # If firewall is active
```

D. node_exporter not installed - Install:

```
ssh root@<vm-ip> apt update apt install -y prometheus-node-exporter systemctl enable --now
prometheus-node-exporter
```

Traffic Test Issues

Problem: Traffic test fails with "iperf3: error"

Diagnosis:

```
# 1. SSH into source VM ssh root@<source-vm-ip> # 2. Check if iperf3 is installed which iperf3
iperf3 --version # 3. Check if target iperf3 server is running ssh root@<target-vm-ip> "ss
-tlnp | grep 5201" # 4. Test manually iperf3 -c <target-vm-ip> -t 10
```

Solutions:

A. iperf3 not installed:

```
ssh root@<vm-ip> apt update apt install -y iperf3
```

B. iperf3 server not running on target:

```
ssh root@<target-vm-ip> # Create service cat > /etc/systemd/system/iperf3.service << 'EOF'
[Unit] Description=iPerf3 Server After=network.target [Service] Type=simple
ExecStart=/usr/bin/iperf3 -s Restart=always RestartSec=3 User=root [Install]
WantedBy=multi-user.target EOF systemctl daemon-reload systemctl enable --now iperf3 systemctl
status iperf3
```

C. Network connectivity issue:

```
# From source VM, test connectivity ping <target-vm-ip> telnet <target-vm-ip> 5201
```

D. Port 5201 blocked:

```
ssh root@<target-vm-ip> ufw allow 5201/tcp ufw allow 5201/udp
```

Problem: Test runs but no results in Grafana

Diagnosis:

```
# 1. Check test was recorded in database docker exec ipam-mysql mysql -u ipmanager
-pipmanager_pass_2024 ipmanager -e "SELECT * FROM traffic_tests ORDER BY id DESC LIMIT 5;" #
2. Check Prometheus has iperf metrics curl -s
http://localhost:9090/api/v1/query?query=iperf3_sent_bytes # 3. Check Grafana dashboard exists
curl -s -u admin:admin http://localhost:3001/api/dashboards/uid/traffic-tests
```

Solution: Create traffic test dashboard or check existing queries match your metric names.

Proxmox Connection Issues

Problem: "Proxmox is not connected"

Diagnosis:

```
# 1. Test Proxmox API accessibility curl -k https://192.168.0.100:8006/api2/json/version # 2.
Check environment variables docker exec ipam-backend env | grep PROXMOX # 3. Check backend can
resolve Proxmox hostname docker exec ipam-backend ping -c 2 proxmox # 4. Check backend logs
docker logs ipam-backend | grep -i proxmox
```

Solutions:

A. Proxmox not accessible - Network issue:

```
# From host ping 192.168.0.100 curl -k https://192.168.0.100:8006
```

B. Wrong credentials:

```
# Test credentials manually curl -k -d "username=root@pam&password=YOUR_PASSWORD" \
https://192.168.0.100:8006/api2/json/access/ticket # If fails, update .env with correct
credentials nano ~/ipmanager/.env docker compose restart backend
```

C. Hostname resolution fails:

```
# Check extra_hosts in docker-compose.yml grep -A2 "extra_hosts:"
~/ipmanager/docker-compose.yml # Should show: # extra_hosts: # - "proxmox:${PROXMOX_HOST}" #
Verify .env has PROXMOX_HOST grep PROXMOX_HOST ~/ipmanager/.env
```

D. SSL certificate verification failing:

Backend should have `verify_ssl=False` in ProxmoxAPI calls. Check:

```
docker exec ipam-backend grep -n "verify_ssl" /app/main.py
```

Should show `verify_ssl=False` in ProxmoxAPI initialization.

Network Scanning Issues

Problem: Scan returns no results

Diagnosis:

```
# 1. Check nmap is installed in backend docker exec ipam-backend which nmap # 2. Test scan
manually docker exec ipam-backend nmap -sn 192.168.0.0/24 # 3. Check backend logs docker logs
ipam-backend | tail -50
```

Solutions:

A. nmap not installed:

```
# Add to backend Dockerfile RUN apt-get update && apt-get install -y nmap # Rebuild docker
compose build backend docker compose up -d backend
```

B. Network not accessible from container:

```
# Backend uses host network, so should work # Verify network_mode in docker-compose.yml grep
-A5 "backend:" ~/ipmanager/docker-compose.yml | grep network_mode # Should show: network_mode:
host
```

C. Insufficient permissions:

```
# Add NET_ADMIN capability if needed # In docker-compose.yml under backend: cap_add: -
NET_ADMIN
```

Database Issues

Problem: "Database connection failed"

Diagnosis:

```
# 1. Check MySQL container is running docker ps | grep ipam-mysql # 2. Check MySQL is healthy
docker inspect ipam-mysql | grep -A5 Health # 3. Test connection from backend docker exec
ipam-backend mysql -h 127.0.0.1 -u ipmanager -pipmanager_pass_2024 -e "SELECT 1" # 4. Check
backend logs docker logs ipam-backend | grep -i database
```

Solutions:

A. MySQL container not running:

```
docker compose up -d ipam-mysql docker logs ipam-mysql
```

B. Wrong credentials:

```
# Verify .env file cat ~/ipmanager/.env | grep MYSQL # Reset MySQL password if needed docker
exec ipam-mysql mysql -u root -p"$MYSQL_ROOT_PASSWORD" -e \ "ALTER USER 'ipmanager'@'%'
IDENTIFIED BY 'new_password';" # Update .env and restart docker compose restart backend
```

C. Database not initialized:

```
# Check if tables exist docker exec ipam-mysql mysql -u ipmanager -p ipmanager -e "SHOW
TABLES;" # If empty, run init script docker exec -i ipam-mysql mysql -u ipmanager -p ipmanager
< ~/ipmanager/mysql/init.sql
```

General Debugging Commands

Check All Container Status

```
docker ps -a docker compose ps
```

View Container Logs

```
# All logs docker compose logs # Specific container docker logs ipam-backend --tail 100 #
Follow logs docker logs -f ipam-backend # With timestamps docker logs -t ipam-backend
```

Restart All Services

```
docker compose restart
```

Rebuild After Code Changes

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

Check Resource Usage

```
docker stats # Or in Grafana container monitoring dashboard
```

Access Container Shell

```
docker exec -it ipam-backend /bin/bash docker exec -it ipam-mysql /bin/bash
```

Check Network Connectivity

```
# From backend to other services docker exec ipam-backend ping prometheus docker exec
ipam-backend ping ipam-mysql docker exec ipam-backend curl http://prometheus:9090
```

Verify File Mounts

```
# Check if volumes are mounted correctly docker inspect ipam-backend | grep -A20 Mounts
```

Maintenance

Regular Tasks

Daily

- Check container status: `docker ps`
- Monitor Grafana dashboards for anomalies
- Review failed traffic tests

Weekly

- Review container resource usage
- Check for container restarts: See "Container Restarts" panel in Grafana
- Verify backup jobs completed

Monthly

- Update Docker images: `docker compose pull && docker compose up -d`

- Review and clean old test data from database
- Check disk space: `df -h`
- Rotate logs if needed

Backup Procedures

Database Backup

```
# Backup MySQL database docker exec ipam-mysql mysqldump -u root -p"$MYSQL_ROOT_PASSWORD"
ipmanager > backup-$(date +%Y%m%d).sql # Restore from backup docker exec -i ipam-mysql mysql
-u root -p"$MYSQL_ROOT_PASSWORD" ipmanager < backup-20241219.sql
```

Configuration Backup

```
# Backup entire configuration tar -czf ipmanager-config-$(date +%Y%m%d).tar.gz \
~/ipmanager/.env \ ~/ipmanager/docker-compose.yml \ ~/ipmanager/monitoring/prometheus/ \
~/ipmanager/monitoring/grafana/
```

Restore Configuration

```
tar -xzf ipmanager-config-20241219.tar.gz -C ~/ docker compose restart
```

Updating the System

Update Application Code

```
cd ~/ipmanager git pull docker compose build docker compose up -d
```

Update Dependencies

```
# Backend docker compose build --no-cache backend docker compose up -d backend # Frontend
docker compose build --no-cache frontend docker compose up -d frontend
```

Update Base Images

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

Log Rotation

Configure Log Rotation

```
# Create logrotate config sudo tee /etc/logrotate.d/docker-containers << 'EOF'
/var/lib/docker/containers/*/*.log { rotate 7 daily compress missingok delaycompress
copytruncate } EOF
```

Manual Log Cleanup

```
# Clean old logs docker system prune -a --filter "until=720h" # Remove all stopped containers
docker container prune
```

API Reference

Network Scanning

POST /api/scan

Scan a network subnet for active devices.

Request Body:

```
{ "subnet": "192.168.0", "start_ip": 0, "end_ip": 255 }
```

Response:

```
{ "results": [ { "ip": "192.168.0.1", "status": "up", "hostname": "gateway", "mac_address":
"00:11:22:33:44:55", "vendor": "Cisco", "first_seen": "2024-12-01T10:00:00", "last_seen":
"2024-12-19T14:30:00", "times_seen": 150, "notes": "Main gateway" } ] }
```

VM Management

GET /api/proxmox/status

Check Proxmox connection status.

Response:

```
{ "connected": true, "host": "192.168.0.100", "node": "proxmox", "version": "9.0.3" }
```

GET /api/proxmox/templates

List available VM templates.

Response:

```
{ "templates": [ { "vmid": 9000, "name": "ubuntu-22-template", "status": "template" }, {
"vmid": 9001, "name": "ubuntu-ipmanager-template", "status": "template" } ] }
```

POST /api/proxmox/create-vm

Create a new VM from template.

Request Body:

```
{ "ip_address": "192.168.0.50", "vm_name": "test-vm-01", "cores": 2, "memory": 2048,
"disk_size": 32, "template_id": 9001, "start_vm": true, "gateway": "192.168.0.1",
"nameserver": "8.8.8.8", "bridge": "vmbr0" }
```

Response:

```
{ "success": true, "vmid": 105, "vm_name": "test-vm-01", "ip_address": "192.168.0.50",
"message": "VM test-vm-01 created successfully with ID 105" }
```

Traffic Testing

POST /api/traffic/start

Start an iperf3 traffic test between two VMs.

Request Body:

```
{ "source_ip": "192.168.0.50", "target_ip": "192.168.0.51", "protocol": "tcp", "duration": 60,
"bandwidth": "100M", "parallel": 1, "reverse": false }
```

Response:

```
{ "test_id": "test-abc123", "status": "running", "source": "192.168.0.50", "target":
"192.168.0.51", "duration": 60 }
```

GET /api/traffic/status/{test_id}

Check test status.

Response:

```
{ "test_id": "test-abc123", "status": "completed", "progress": 100 }
```

GET /api/traffic/results/{test_id}

Get test results.

Response:

```
{ "test_id": "test-abc123", "status": "completed", "source": "192.168.0.50", "target":
"192.168.0.51", "protocol": "tcp", "bandwidth_mbps": 942.5, "bytes_transferred": 7140000000,
"retransmits": 12, "jitter_ms": null, "lost_percent": null }
```

Appendix

Port Reference

Port	Service	Purpose
3000	Frontend	Web UI
3001	Grafana	Monitoring dashboards
3306	MySQL	Database
5201	iperf3	Traffic testing (VMs)
8000	Backend	API server
8006	Proxmox	Virtualization API
8080	phpMyAdmin	Database admin
8081	cAdvisor	Container monitoring
9090	Prometheus	Metrics collection
9093	Alertmanager	Alert routing
9100	node_exporter	VM metrics (VMs)

File Structure

```
ipmanager/ ■■■ .env # Environment variables (DO NOT COMMIT) ■■■ .env.example # Template for
.env ■■■ docker-compose.yml # Container orchestration ■■■ .gitignore # Git ignore rules ■
■■■ backend/ # FastAPI backend ■ ■■■ main.py # Main application ■ ■■■ requirements.txt #
Python dependencies ■ ■■■ Dockerfile # Backend container image ■ ■■■ frontend/ # React
frontend ■ ■■■ src/ ■ ■ ■■■ App.js # Main React component ■ ■ ■■■ App.css # Styles ■ ■ ■■■
network-styles.css # Network grid styles ■ ■■■ package.json # Node dependencies ■ ■■■
Dockerfile # Frontend container image ■ ■■■ nginx.conf # Nginx configuration ■ ■■■
monitoring/ # Monitoring configuration ■ ■■■ prometheus/ ■ ■ ■■■ prometheus.yml # Prometheus
config ■ ■ ■■■ targets/ ■ ■ ■■■ nodes.yml # VM targets (auto-updated) ■ ■ ■ ■■■ grafana/ ■
■ ■■■ provisioning/ ■ ■ ■ ■■■ datasources/ ■ ■ ■ ■ ■■■ prometheus.yml # Datasource config
■ ■ ■ ■■■ dashboards/ ■ ■ ■ ■■■ dashboard.yml # Dashboard provisioning ■ ■ ■■■ dashboards/
■ ■ ■■■ container-monitoring.json # Container dashboard ■ ■ ■ ■■■ alertmanager/ ■ ■■■
config.yml # Alert routing ■ ■■■ mysql/ ■ ■■■ init.sql # Database initialization ■ ■■■
scripts/ # Utility scripts ■■■ prepare-vm.sh # VM preparation script
```

Environment Variables Reference

```
# MySQL MYSQL_ROOT_PASSWORD= # MySQL root password MYSQL_DATABASE= # Database name (ipmanager)
MYSQL_USER= # Application MySQL user MYSQL_PASSWORD= # Application MySQL password MYSQL_HOST=
# MySQL host (127.0.0.1) MYSQL_PORT= # MySQL port (3306) # Proxmox PROXMOX_HOST= # Proxmox
server IP PROXMOX_PORT= # Proxmox API port (8006) PROXMOX_USER= # Proxmox user (root@pam)
PROXMOX_PASSWORD= # Proxmox password PROXMOX_NODE= # Proxmox node name (proxmox) # Grafana
GF_SECURITY_ADMIN_USER= # Grafana admin username GF_SECURITY_ADMIN_PASSWORD= # Grafana admin
password
```

Common Commands Quick Reference

```
# Start all services docker compose up -d # Stop all services docker compose down # Restart
specific service docker compose restart <service-name> # View logs docker compose logs -f
<service-name> # Rebuild after code changes docker compose build docker compose up -d # Check
container status docker compose ps # Access container shell docker exec -it <container-name>
/bin/bash # Backup database docker exec ipam-mysql mysqldump -u root -p ipmanager > backup.sql
# Check Prometheus targets curl http://localhost:9090/api/v1/targets # Reload Prometheus
config curl -X POST http://localhost:9090/-/reload # Test VM connectivity ping <vm-ip> ssh
root@<vm-ip> # Check VM services ssh root@<vm-ip> "systemctl status iperf3
prometheus-node-exporter"
```

Document Version History

Version	Date	Changes
1.0	2024-12-01	Initial documentation
2.0	2024-12-19	Added VM template section, automated monitoring, comprehensive debugging guide

Support & Contributing

Getting Help

Check this documentation first

Review the debugging section

Check container logs: `docker compose logs`

Review GitHub issues (if applicable)

Reporting Issues

Include:

• Docker version: `docker --version`

• Docker Compose version: `docker compose version`

• OS version: `lsb_release -a`

• Container status: `docker compose ps`

• Relevant logs: `docker compose logs`

• Steps to reproduce

Contributing

Fork the repository

Create a feature branch

Make your changes

Test thoroughly

Submit a pull request with detailed description

End of Document