# Raspberry Pi DNS Monitoring Configuration

Now that you have SSH access, let's transform your Pi into a powerful DNS monitoring system.

## Phase 1: System Preparation

### Step 1: Update and Secure Your Pi

```bash
# SSH into your Pi
ssh pi@192.168.1.100

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

# Install essential tools
sudo apt install -y git python3-pip python3-venv sqlite3 dnsutils tcpdump net-tools htop nano vim

# Check Python version (should be 3.11+)
python3 --version
```

### Step 2: Set Static IP Address

Your Pi needs a consistent IP address for DNS monitoring.

bash

```
# Edit network configuration
sudo nano /etc/dhcpcd.conf

# Add these lines at the end:
interface eth0
static ip_address=192.168.1.100/24
static routers=192.168.1.1
static domain_name_servers=8.8.8.8 8.8.4.4

interface wlan0
static ip_address=192.168.1.100/24
static routers=192.168.1.1
static domain_name_servers=8.8.8.8 8.8.4.4

# Save and exit (Ctrl+X, Y, Enter)

# Restart networking
sudo systemctl restart dhcpcd

# Verify IP address
ip addr show
```

### Step 3: Create Project Structure

bash

```
# Create project directory
mkdir ~/smartguard-monitor
cd ~/smartguard-monitor

# Create directory structure
mkdir -p {src,config,logs,data,tests,docs}

# Initialize git repository
git init
```

## Phase 2: DNS Server Configuration

### Step 4: Install and Configure DNSmasq

bash

```
# Install DNSmasq
sudo apt install -y dnsmasq

# Backup original configuration
sudo cp /etc/dnsmasq.conf /etc/dnsmasq.conf.backup

# Create new configuration
sudo nano /etc/dnsmasq.conf
```

**DNSmasq Configuration (`/etc/dnsmasq.conf`):**

bash

```
# Basic DNS settings
port=53
domain-needed
bogus-priv
no-resolv
no-poll

# Interface binding
interface=eth0
interface=wlan0
bind-interfaces

# Upstream DNS servers
server=8.8.8.8
server=8.8.4.4
server=1.1.1.1

# Cache settings
cache-size=2000
neg-ttl=3600

# Logging
log-queries
log-facility=/var/log/dnsmasq.log

# Local domain
local=/local/
domain=local
expand-hosts

# DHCP range (optional - only if Pi will handle DHCP)
# dhcp-range=192.168.1.50,192.168.1.150,12h
# dhcp-option=option:router,192.168.1.1

# Disable DHCP (we only want DNS)
no-dhcp-interface=eth0
no-dhcp-interface=wlan0
```

## Step 5: Configure DNS Logging

bash

```
# Create log directory
sudo mkdir -p /var/log/smartguard

# Set up log rotation
sudo nano /etc/logrotate.d/smartguard
```

### Log Rotation Configuration:

bash

```
/var/log/dnsmasq.log {
    daily
    rotate 7
    compress
    delaycompress
    missingok
    notifempty
    postrotate
        /bin/kill -HUP `cat /var/run/dnsmasq.pid 2> /dev/null` 2> /dev/null || true
    endscript
}

/var/log/smartguard/*.log {
    daily
    rotate 30
    compress
    delaycompress
    missingok
    notifempty
}
```

## Step 6: Start DNS Services

bash

```
# Enable and start DNSmasq
sudo systemctl enable dnsmasq
sudo systemctl start dnsmasq

# Check status
sudo systemctl status dnsmasq

# Test DNS resolution
nslookup google.com 127.0.0.1
dig @127.0.0.1 facebook.com

# Monitor DNS logs in real-time
sudo tail -f /var/log/dnsmasq.log
```

# Phase 3: Python Environment Setup

## Step 7: Create Python Virtual Environment

bash

```
cd ~/smartguard-monitor

# Create virtual environment
python3 -m venv venv

# Activate virtual environment
source venv/bin/activate

# Upgrade pip
pip install --upgrade pip

# Create requirements file
nano requirements.txt
```

### Requirements.txt:

```
flask==3.0.0
sqlite3
scapy==2.5.0
dnspython==2.4.2
requests==2.31.0
python-dateutil==2.8.2
psutil==5.9.6
watchdog==3.0.0
flask-cors==4.0.0
```

bash

```
# Install Python packages
pip install -r requirements.txt

# Verify installations
python3 -c "import flask, sqlite3, scapy; print('All packages installed successfully')"
```

## Step 8: Create Database Schema

bash

```
# Create database initialization script
nano src/setup_database.py
```

### Database Setup Script (`src/setup_database.py`):

python

```python
#!/usr/bin/env python3
import sqlite3
import os
from datetime import datetime

def create_database():
    """Initialize the SmartGuard monitoring database"""

    # Create data directory if it doesn't exist
    os.makedirs('../data', exist_ok=True)

    # Connect to database
    db_path = '../data/smartguard.db'
    conn = sqlite3.connect(db_path)
    cursor = conn.cursor()

    # DNS requests table
    cursor.execute('''
        CREATE TABLE IF NOT EXISTS dns_requests (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            timestamp DATETIME DEFAULT CURRENT_TIMESTAMP,
            client_ip TEXT NOT NULL,
            client_mac TEXT,
            domain TEXT NOT NULL,
            query_type TEXT DEFAULT 'A',
            response_ip TEXT,
            response_time_ms INTEGER,
            status TEXT DEFAULT 'logged',
            classification TEXT,
            confidence_score REAL,
            created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
            INDEX(timestamp),
            INDEX(client_ip),
            INDEX(domain)
        )
    ''')

    # Device information table
    cursor.execute('''
        CREATE TABLE IF NOT EXISTS devices (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            mac_address TEXT UNIQUE NOT NULL,
            ip_address TEXT,
            hostname TEXT,
            device_name TEXT,
            device_type TEXT,
```

```
        first_seen DATETIME DEFAULT CURRENT_TIMESTAMP,
        last_seen DATETIME DEFAULT CURRENT_TIMESTAMP,
        is_monitored BOOLEAN DEFAULT 1,
        notes TEXT
    )
''')

# Traffic statistics table
cursor.execute('''
    CREATE TABLE IF NOT EXISTS traffic_stats (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        timestamp DATETIME DEFAULT CURRENT_TIMESTAMP,
        src_ip TEXT NOT NULL,
        dst_ip TEXT NOT NULL,
        protocol TEXT,
        port INTEGER,
        bytes_sent INTEGER DEFAULT 0,
        packets_count INTEGER DEFAULT 1,
        duration_seconds INTEGER
    )
''')

# System events table
cursor.execute('''
    CREATE TABLE IF NOT EXISTS system_events (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        timestamp DATETIME DEFAULT CURRENT_TIMESTAMP,
        event_type TEXT NOT NULL,
        severity TEXT DEFAULT 'INFO',
        message TEXT NOT NULL,
        details TEXT,
        source_ip TEXT
    )
''')

# Configuration table
cursor.execute('''
    CREATE TABLE IF NOT EXISTS configuration (
        key TEXT PRIMARY KEY,
        value TEXT,
        description TEXT,
        updated_at DATETIME DEFAULT CURRENT_TIMESTAMP
    )
''')

# Insert default configuration
default_config = [
```

```
    ('monitoring_enabled', 'true', 'Enable/disable monitoring'),
    ('log_level', 'INFO', 'Logging level'),
    ('retention_days', '30', 'Data retention period'),
    ('upstream_dns', '8.8.8.8,8.8.4.4', 'Upstream DNS servers'),
    ('web_port', '5000', 'Web dashboard port'),
    ('classification_enabled', 'false', 'AI classification enabled'),
]

cursor.executemany(
    'INSERT OR IGNORE INTO configuration (key, value, description) VALUES (?, ?, ?)',
    default_config
)

# Create views for common queries
cursor.execute('''
    CREATE VIEW IF NOT EXISTS recent_requests AS
    SELECT
        dr.*,
        d.device_name,
        d.device_type
    FROM dns_requests dr
    LEFT JOIN devices d ON dr.client_ip = d.ip_address
    WHERE dr.timestamp > datetime('now', '-24 hours')
    ORDER BY dr.timestamp DESC
''')

cursor.execute('''
    CREATE VIEW IF NOT EXISTS daily_stats AS
    SELECT
        DATE(timestamp) as date,
        COUNT(*) as total_requests,
        COUNT(DISTINCT client_ip) as active_devices,
        COUNT(DISTINCT domain) as unique_domains
    FROM dns_requests
    GROUP BY DATE(timestamp)
    ORDER BY date DESC
''')

# Commit changes
conn.commit()
conn.close()

print(f"✅ Database created successfully at {db_path}")
print("✅ Tables created: dns_requests, devices, traffic_stats, system_events, configuration")
print("✅ Views created: recent_requests, daily_stats")
```

```
if __name__ == '__main__':
    create_database()
```

bash

```
# Run database setup
cd src
python3 setup_database.py
cd ..

# Verify database creation
sqlite3 data/smartguard.db ".tables"
sqlite3 data/smartguard.db ".schema dns_requests"
```

# Phase 4: DNS Monitoring Implementation

## Step 9: Create DNS Monitor Service

bash

```
nano src/dns_monitor.py
```

### DNS Monitor (`src/dns_monitor.py`):

python

```python
#!/usr/bin/env python3
import socket
import struct
import sqlite3
import threading
import time
import logging
from datetime import datetime
from typing import Optional, Tuple

class DNSMonitor:
    def __init__(self, db_path: str = '../data/smartguard.db'):
        self.db_path = db_path
        self.running = False
        self.socket = None

        # Setup logging
        logging.basicConfig(
            level=logging.INFO,
            format='%(asctime)s - %(levelname)s - %(message)s',
            handlers=[
                logging.FileHandler('../logs/dns_monitor.log'),
                logging.StreamHandler()
            ]
        )
        self.logger = logging.getLogger(__name__)

    def parse_dns_query(self, data: bytes) -> Optional[str]:
        """Parse DNS query packet and extract domain name"""
        try:
            # Skip DNS header (12 bytes)
            if len(data) < 12:
                return None

            domain_parts = []
            i = 12  # Start after header

            while i < len(data):
                # Read length of next label
                if data[i] == 0:  # End of domain name
                    break

                length = data[i]
                if length > 63:  # Invalid label length
                    break
```

```
            # Extract label
            i += 1
            if i + length > len(data):
                break

            label = data[i:i+length].decode('utf-8', errors='ignore')
            domain_parts.append(label)
            i += length

        if domain_parts:
            domain = '.'.join(domain_parts)
            return domain.lower()

    except Exception as e:
        self.logger.error(f"Error parsing DNS query: {e}")

    return None

def log_dns_request(self, client_ip: str, domain: str, query_type: str = 'A'):
    """Log DNS request to database"""
    try:
        conn = sqlite3.connect(self.db_path)
        cursor = conn.cursor()

        cursor.execute('''
            INSERT INTO dns_requests (client_ip, domain, query_type)
            VALUES (?, ?, ?)
        ''', (client_ip, domain, query_type))

        conn.commit()
        conn.close()

        self.logger.info(f"📡 DNS: {client_ip} -> {domain}")

    except Exception as e:
        self.logger.error(f"Database error: {e}")

def forward_dns_request(self, data: bytes, upstream_dns: str = '8.8.8.8') -> Optional[bytes]:
    """Forward DNS request to upstream server"""
    try:
        # Create socket for upstream query
        upstream_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        upstream_socket.settimeout(5)

        # Send to upstream DNS
        upstream_socket.sendto(data, (upstream_dns, 53))
        response, _ = upstream_socket.recvfrom(512)
```

```
            upstream_socket.close()
            return response

        except Exception as e:
            self.logger.error(f"Upstream DNS error: {e}")
            return None

    def handle_dns_request(self, data: bytes, client_addr: Tuple[str, int]) -> Optional[bytes]:
        """Handle incoming DNS request"""
        client_ip = client_addr[0]

        # Parse domain from query
        domain = self.parse_dns_query(data)

        if domain:
            # Log the request
            self.log_dns_request(client_ip, domain)

        # Forward to upstream DNS and return response
        response = self.forward_dns_request(data)
        return response

    def start_monitoring(self, interface: str = '', port: int = 5353):
        """Start DNS monitoring service"""
        try:
            # Create UDP socket
            self.socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
            self.socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
            self.socket.bind((interface, port))

            self.running = True
            self.logger.info(f"🚀 DNS Monitor started on {interface}:{port}")

            while self.running:
                try:
                    # Receive DNS query
                    data, client_addr = self.socket.recvfrom(512)

                    # Handle in separate thread to avoid blocking
                    thread = threading.Thread(
                        target=self._handle_request_thread,
                        args=(data, client_addr)
                    )
                    thread.daemon = True
                    thread.start()
```

```
                except socket.timeout:
                    continue
                except Exception as e:
                    if self.running:
                        self.logger.error(f"Socket error: {e}")

        except Exception as e:
            self.logger.error(f"Failed to start DNS monitor: {e}")
        finally:
            self.stop_monitoring()

    def _handle_request_thread(self, data: bytes, client_addr: Tuple[str, int]):
        """Handle DNS request in separate thread"""
        try:
            response = self.handle_dns_request(data, client_addr)
            if response and self.socket:
                self.socket.sendto(response, client_addr)
        except Exception as e:
            self.logger.error(f"Error handling DNS request: {e}")

    def stop_monitoring(self):
        """Stop DNS monitoring service"""
        self.running = False
        if self.socket:
            self.socket.close()
        self.logger.info("🛑 DNS Monitor stopped")

def main():
    """Main function to run DNS monitor"""
    import os

    # Create logs directory
    os.makedirs('../logs', exist_ok=True)

    # Start DNS monitor
    monitor = DNSMonitor()

    try:
        # Note: Port 5353 for testing (port 53 requires root)
        monitor.start_monitoring(port=5353)
    except KeyboardInterrupt:
        print("\n🛑 Stopping DNS monitor...")
        monitor.stop_monitoring()

if __name__ == '__main__':
    main()
```

## Step 10: Create Web Dashboard

bash

```
nano src/web_dashboard.py
```

## Web Dashboard (`src/web_dashboard.py`):

python

```
#!/usr/bin/env python3
from flask import Flask, render_template, jsonify, request
from flask_cors import CORS
import sqlite3
import json
from datetime import datetime, timedelta

app = Flask(__name__)
CORS(app)

DB_PATH = '../data/smartguard.db'

def get_db_connection():
    """Get database connection"""
    conn = sqlite3.connect(DB_PATH)
    conn.row_factory = sqlite3.Row
    return conn

@app.route('/')
def dashboard():
    """Main dashboard page"""
    return '''
    <!DOCTYPE html>
    <html>
    <head>
        <title>SmartGuard - Network Monitor</title>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <style>
            body { font-family: -apple-system, sans-serif; margin: 0; padding: 20px; background: #f5f5f5; }
            .container { max-width: 1200px; margin: 0 auto; }
            .header { background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
                      color: white; padding: 30px; border-radius: 10px; margin-bottom: 20px; }
            .stats-grid { display: grid; grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));
                          gap: 20px; margin-bottom: 30px; }
            .stat-card { background: white; padding: 20px; border-radius: 8px; box-shadow: 0 2px 10px rgba(0,0,0,0.1); }
            .stat-value { font-size: 2.5em; font-weight: bold; color: #667eea; }
            .stat-label { color: #666; margin-top: 5px; }
            .requests-table { background: white; border-radius: 8px; overflow: hidden; box-shadow: 0 2px 10px rgba(0,0,0,0.1
            table { width: 100%; border-collapse: collapse; }
            th { background: #f8f9fa; padding: 15px; text-align: left; font-weight: 600; }
            td { padding: 12px 15px; border-bottom: 1px solid #eee; }
            .status { display: inline-block; width: 10px; height: 10px; border-radius: 50%; margin-right: 8px; }
            .status.online { background: #4caf50; }
            .refresh-btn { background: #667eea; color: white; border: none; padding: 10px 20px;
                           border-radius: 5px; cursor: pointer; margin-bottom: 20px; }
```

```
    </style>
</head>
<body>
    <div class="container">
        <div class="header">
            <h1>🛡️ SmartGuard Network Monitor</h1>
            <p>Real-time DNS monitoring and analysis</p>
        </div>

        <button class="refresh-btn" onclick="loadData()">🔄 Refresh Data</button>

        <div class="stats-grid">
            <div class="stat-card">
                <div class="stat-value" id="total-requests">-</div>
                <div class="stat-label">Total DNS Requests</div>
            </div>
            <div class="stat-card">
                <div class="stat-value" id="unique-domains">-</div>
                <div class="stat-label">Unique Domains</div>
            </div>
            <div class="stat-card">
                <div class="stat-value" id="active-devices">-</div>
                <div class="stat-label">Active Devices</div>
            </div>
            <div class="stat-card">
                <div class="stat-value" id="monitoring-status">
                    <span class="status online"></span>Online
                </div>
                <div class="stat-label">System Status</div>
            </div>
        </div>

        <div class="requests-table">
            <h3 style="margin: 0; padding: 20px 20px 0;">Recent DNS Requests</h3>
            <table>
                <thead>
                    <tr>
                        <th>Time</th>
                        <th>Device IP</th>
                        <th>Domain</th>
                        <th>Type</th>
                        <th>Status</th>
                    </tr>
                </thead>
                <tbody id="requests-table">
                    <tr><td colspan="5" style="text-align: center; color: #666;">Loading...</td></tr>
                </tbody>
```

```
        </table>
    </div>
</div>

<script>
    function loadData() {
        // Load statistics
        fetch('/api/stats')
            .then(response => response.json())
            .then(data => {
                document.getElementById('total-requests').textContent = data.total_requests || 0;
                document.getElementById('unique-domains').textContent = data.unique_domains || 0;
                document.getElementById('active-devices').textContent = data.active_devices || 0;
            })
            .catch(error => console.error('Error loading stats:', error));

        // Load recent requests
        fetch('/api/recent-requests')
            .then(response => response.json())
            .then(data => {
                const tbody = document.getElementById('requests-table');
                tbody.innerHTML = '';

                if (data.length === 0) {
                    tbody.innerHTML = '<tr><td colspan="5" style="text-align: center; color: #666;">No requests yet</td
                    return;
                }

                data.forEach(request => {
                    const row = tbody.insertRow();
                    row.innerHTML = `
                        <td>${new Date(request.timestamp).toLocaleString()}</td>
                        <td>${request.client_ip}</td>
                        <td>${request.domain}</td>
                        <td>${request.query_type}</td>
                        <td><span class="status online"></span>Logged</td>
                    `;
                });
            })
            .catch(error => console.error('Error loading requests:', error));
    }

    // Load data on page load
    loadData();

    // Auto-refresh every 10 seconds
    setInterval(loadData, 10000);
```

```
        </script>
    </body>
    </html>
    '''

@app.route('/api/stats')
def api_stats():
    """Get system statistics"""
    try:
        conn = get_db_connection()

        # Total requests today
        total_requests = conn.execute(
            "SELECT COUNT(*) FROM dns_requests WHERE DATE(timestamp) = DATE('now')"
        ).fetchone()[0]

        # Unique domains today
        unique_domains = conn.execute(
            "SELECT COUNT(DISTINCT domain) FROM dns_requests WHERE DATE(timestamp) = DATE('now')"
        ).fetchone()[0]

        # Active devices today
        active_devices = conn.execute(
            "SELECT COUNT(DISTINCT client_ip) FROM dns_requests WHERE DATE(timestamp) = DATE('now')"
        ).fetchone()[0]

        conn.close()

        return jsonify({
            'total_requests': total_requests,
            'unique_domains': unique_domains,
            'active_devices': active_devices,
            'status': 'online'
        })

    except Exception as e:
        return jsonify({'error': str(e)}), 500

@app.route('/api/recent-requests')
def api_recent_requests():
    """Get recent DNS requests"""
    try:
        limit = request.args.get('limit', 50)

        conn = get_db_connection()
        requests = conn.execute(
            '''SELECT timestamp, client_ip, domain, query_type, status
```

```
            FROM dns_requests
            ORDER BY timestamp DESC
            LIMIT ?''',
            (limit,)
        ).fetchall()
        conn.close()

        return jsonify([dict(row) for row in requests])

    except Exception as e:
        return jsonify({'error': str(e)}), 500

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=True)
```

## Phase 5: Testing and Verification

### Step 11: Test the System

```
bash

# Create test script
nano test_system.py
```

#### Test Script:

python

```python
#!/usr/bin/env python3
import subprocess
import time
import requests
import sqlite3

def test_database():
    """Test database connectivity"""
    try:
        conn = sqlite3.connect('../data/smartguard.db')
        cursor = conn.cursor()
        cursor.execute("SELECT COUNT(*) FROM dns_requests")
        count = cursor.fetchone()[0]
        conn.close()
        print(f"✅ Database: {count} DNS requests logged")
        return True
    except Exception as e:
        print(f"❌ Database error: {e}")
        return False

def test_web_dashboard():
    """Test web dashboard"""
    try:
        response = requests.get('http://localhost:5000', timeout=5)
        if response.status_code == 200:
            print("✅ Web dashboard: Accessible")
            return True
        else:
            print(f"❌ Web dashboard: HTTP {response.status_code}")
            return False
    except Exception as e:
        print(f"❌ Web dashboard error: {e}")
        return False

def test_dns_resolution():
    """Test DNS resolution"""
    try:
        result = subprocess.run(['nslookup', 'google.com', '127.0.0.1'],
                                capture_output=True, text=True, timeout=10)
        if result.returncode == 0:
            print("✅ DNS resolution: Working")
            return True
        else:
            print(f"❌ DNS resolution failed: {result.stderr}")
            return False
    except Exception as e:
```

```
        print(f"❌ DNS resolution error: {e}")
        return False

def main():
    print("🧪 Testing SmartGuard System")
    print("=" * 40)

    tests = [
        ("Database", test_database),
        ("Web Dashboard", test_web_dashboard),
        ("DNS Resolution", test_dns_resolution)
    ]

    passed = 0
    for name, test_func in tests:
        print(f"\nTesting {name}...")
        if test_func():
            passed += 1
        time.sleep(1)

    print("\n" + "=" * 40)
    print(f"Tests completed: {passed}/{len(tests)} passed")

    if passed == len(tests):
        print("🎉 All tests passed! System is ready.")
    else:
        print("⚠️  Some tests failed. Check configuration.")

if __name__ == '__main__':
    main()
```

## Step 12: Start All Services

```bash
# Terminal 1: Start web dashboard
cd ~/smartguard-monitor/src
source ../venv/bin/activate
python3 web_dashboard.py

# Terminal 2: Start DNS monitor (in another SSH session)
cd ~/smartguard-monitor/src
source ../venv/bin/activate
sudo python3 dns_monitor.py  # Requires sudo for port 53

# Terminal 3: Run tests (in another SSH session)
cd ~/smartguard-monitor
python3 test_system.py
```

### Step 13: Configure Router DNS

Now configure your router to use the Pi as DNS server:

1. **Access Router Admin**: Usually http://192.168.1.1
2. **Find DHCP/DNS Settings**: Look for "LAN", "DHCP", or "DNS" settings
3. **Set Primary DNS**: 192.168.1.100 (your Pi's IP)
4. **Set Secondary DNS**: 8.8.8.8 (backup)
5. **Save and Reboot Router**

### Step 14: Verify End-to-End Functionality

```bash
# From any device on network, test DNS
nslookup facebook.com

# Should show:
# Server: 192.168.1.100
# Address: 192.168.1.100#53

# Check Pi logs
sudo tail -f /var/log/dnsmasq.log

# Check SmartGuard database
sqlite3 ~/smartguard-monitor/data/smartguard.db
SELECT * FROM dns_requests ORDER BY timestamp DESC LIMIT 10;
```

## Phase 6: Monitoring and Maintenance

### Step 15: Create Startup Scripts

bash

```
# Create systemd service for DNS monitor
sudo nano /etc/systemd/system/smartguard-dns.service
```

### Systemd Service:

ini

```
[Unit]
Description=SmartGuard DNS Monitor
After=network.target

[Service]
Type=simple
User=pi
WorkingDirectory=/home/pi/smartguard-monitor/src
Environment=PATH=/home/pi/smartguard-monitor/venv/bin
ExecStart=/home/pi/smartguard-monitor/venv/bin/python3 dns_monitor.py
Restart=always
RestartSec=10

[Install]
WantedBy=multi-user.target
```

bash

```
# Enable and start service
sudo systemctl enable smartguard-dns
sudo systemctl start smartguard-dns
sudo systemctl status smartguard-dns
```

## Success Verification

You should now have:

- ✅ Pi with static IP (192.168.1.100)
- ✅ DNSmasq logging all DNS queries
- ✅ Python DNS monitor capturing requests to database
- ✅ Web dashboard showing real-time activity
- ✅ Router configured to use Pi as DNS server
- ✅ All network devices' DNS requests being monitored

**Next Steps**: Week 3-4 will add AI classification to analyze the domains being requested!

Access your dashboard at: http://192.168.1.100:5000