

# Coding DIY - Build Network Monitoring Web Apps

These labs show you how to use Python, Bottle, HTML, CSS and SQL to build simple solutions.

Things to think about are:

- How should users enter commands into your app?
- How should results be displayed and what information is valuable to users?
- How should data be stored for your app?

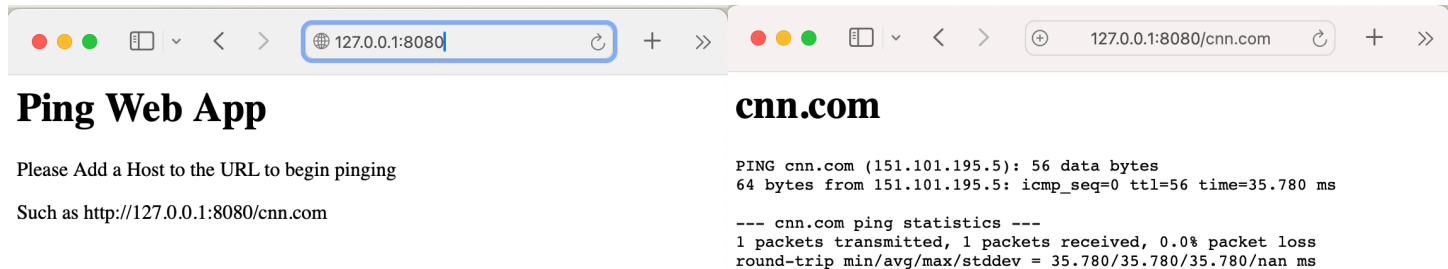
## Setup

```
python3 -m pip install bottle
```

```
python3 -m pip install requests
```

# Ping Single Host

This lab has your app ping a host from a value added in the URL. We use Dynamic Filters in Bottle to take the value and then send the ping command to the OS.



## ping.py

```
from bottle import run, route
import os

@route('/')
def index():
    page = f'''
        <h1>Ping Web App</h1>
        <p>Please Add a Host to the URL to begin pinging</p>
        <p>Such as http://127.0.0.1:8080/cnn.com</p>
    '''
    return page

@route('/:<host>')
def index(host):
    try:
        response = os.popen(f'ping -c 1 {host}').read()
    except:
        response = 'Problem Connecting'
    finally:
        if response == '':
            response = 'Host Could Not Resolve'
    header = '<meta http-equiv="refresh" content="5">'
    page = f''' {header}
        <h1>{host}</h1>
        <pre>{response}</pre>
    '''
    return page

run(host='127.0.0.1', port=8080)
```

# Ping Multiple Hosts

This lab allows you to ping multiple hosts by adding them to the URL separated by commas.

```
127.0.0.1:8080/cnn.com,fox.com,192.168.1.1 X
```

**cnn.com**

```
PING cnn.com (151.101.67.5): 56 data bytes
64 bytes from 151.101.67.5: icmp_seq=0 ttl=56 time=35.943 ms

--- cnn.com ping statistics ---
1 packets transmitted, 1 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 35.943/35.943/35.943/nan ms
```

**fox.com**

```
PING fox.com (23.11.208.173): 56 data bytes
64 bytes from 23.11.208.173: icmp_seq=0 ttl=52 time=49.804 ms

--- fox.com ping statistics ---
1 packets transmitted, 1 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 49.804/49.804/49.804/0.000 ms
```

**192.168.1.1**

```
PING 192.168.1.1 (192.168.1.1): 56 data bytes
64 bytes from 192.168.1.1: icmp_seq=0 ttl=64 time=3.663 ms

--- 192.168.1.1 ping statistics ---
1 packets transmitted, 1 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 3.663/3.663/3.663/nan ms
```

## ping-multi.py

```
from bottle import run, route
import os

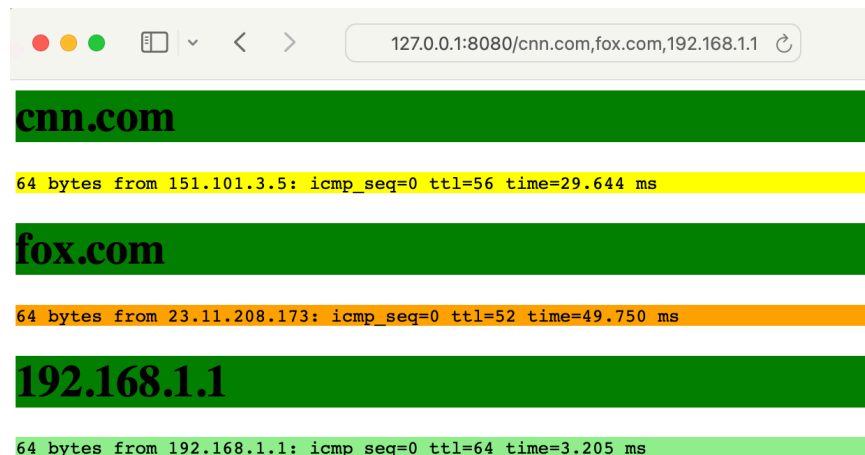
@route('/')
def index():
    page = f'''
        <h1>Ping Web App</h1>
        <p>Please Add a Hosts to the URL to begin pinging</p>
        <p>Such as http://127.0.0.1:8080/
cnn.com,fox.com,192.168.1.1</p>
    '''
    return page

@route('/<host>')
def index(host):
    host_list = host.split(',')
    body = ''
    for value in host_list:
        value = value.strip()
        try:
            response = os.popen(f'ping -c 1 {value}').read()
        except:
            response = 'Problem Connecting'
        finally:
            if response == '':
                response = 'Host Could Not Resolve'
```

```
        body = f'''
                {body}
                <h1>{value}</h1>
                <pre>{response}</pre>
                '''
    header = '<meta http-equiv="refresh" content="5">'
    page = f'''
            {header}
            {body}
            '''
    return page

run(host='127.0.0.1', port=8080)
```

# Ping Multiple Hosts with Color Indicators



This lab allows you to ping multiple hosts and then color code both on whether the host is up, and based off of its latency.

## ping-color.py

```
from bottle import run, route
import os

latency_quality = {'good': 20, 'usable': 40, 'bad': 50}

@route('/')
def index():
    page = f'''
        <h1>Ping Web App</h1>
        <p>Please Add a Hosts to the URL to begin pinging</p>
        <p>Such as http://127.0.0.1:8080/
cnn.com,fox.com,192.168.1.1</p>
    '''
    return page

@route('/<host>')
def index(host):
    command = 'ping -c 1 '
    arg = ' | grep time'
    host_list = host.split(',')
    body = ''
    for host in host_list:
        color = ''
        latency_color = ''
        host = host.strip()
        try:
            response = os.popen(f'{command} {host} {arg}').read()
        except:
```

```

        response = 'Problem Connecting'
    else:
        if 'time' in response:
            color = 'green'
        else:
            color = 'red'
        response_list = response.split(' ')
        for value in response_list:
            if 'time' in value:
                time_list = value.split('=')
                latency = float(time_list[1])
                if latency < latency_quality['good']:
                    latency_color = 'lightgreen'
                elif latency > latency_quality['good'] and latency <=
latency_quality['usable']:
                    latency_color = 'yellow'
                elif latency > latency_quality['usable'] and latency
<= latency_quality['bad']:
                    latency_color = 'orange'
                elif latency > latency_quality['bad']:
                    latency_color = 'red'
                else:
                    latency_color = 'purple'
    finally:
        if response == '':
            response = 'Host Could Not Resolve'

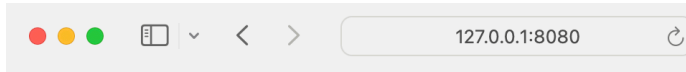
    body = f'''
        {body}
        <h1 style="background-color:{color};">{host}</h1>
        <pre style="background-color:{latency_color}">{response}</
pre>
        ...

    header = '<meta http-equiv="refresh" content="5">'
    page = f'''
        {header}
        {body}
        ...
    return page

run(host='127.0.0.1', port=8080)

```

# Ping App with SQL Configuration

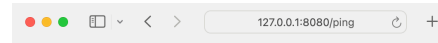


## Ping App

[Home Test](#)

Hosts:

cnn.com  
fox.com  
msnbc.com



## Ping App

[Home Test](#)

**cnn.com**

PING cnn.com (151.101.67.5): 56 data bytes  
64 bytes from 151.101.67.5: icmp\_seq=0 ttl=56 time=32.719 ms  
--- cnn.com ping statistics ---  
1 packets transmitted, 1 packets received, 0.0% packet loss  
round-trip min/avg/max/stddev = 32.719/32.719/32.719/0.000 ms

**fox.com**

PING fox.com (23.11.208.173): 56 data bytes  
64 bytes from 23.11.208.173: icmp\_seq=0 ttl=52 time=49.311 ms  
--- fox.com ping statistics ---  
1 packets transmitted, 1 packets received, 0.0% packet loss  
round-trip min/avg/max/stddev = 49.311/49.311/49.311/nan ms

This lab allows you to store the hosts that you want to ping in a SQLite table.

### ping-sql.py

```
from bottle import run, route, request, post, redirect
import os
import sqlite3

database = ('ping.db')
header = (''
          <h1>Ping App</h1>
          <a href="/">Home<a>
          <a href="/ping">Test</a>
          '')

def db_create():
    conn = sqlite3.connect(database)
    cursor = conn.cursor()
    cursor.execute('CREATE TABLE IF NOT EXISTS host(host_list)')
    conn.commit()
    conn.close()

def db_find_host():
    conn = sqlite3.connect(database)
    cursor = conn.cursor()
    try:
        query = cursor.execute('SELECT * FROM host where rowid = 1')
        query = query.fetchone()
        query = query[0]
    except:
        query = ''
    finally:
        conn.close()

    return query
```

```

@post('/db_insert')
def db_insert():
    hosts = request.forms.get('hosts')
    conn = sqlite3.connect(database)
    cursor = conn.cursor()
    try:
        cursor.execute('UPDATE host set host_list = ? where rowid = 1',
(hosts,))
        if cursor.rowcount == 0:
            cursor.execute('INSERT INTO host(host_list) values(?)',
(hosts,))
    except:
        print('database problem')

    finally:
        conn.commit()
        conn.close()

    redirect('/')

@route('/')
def index():
    host = db_find_host()
    host = host.split('\n')
    host_output = ''
    for item in host:
        item = item.strip()
        host_output = f'{host_output}{item}\n'
    page = f'''
        {header}
        <form action="/db_insert" method="post">
            Hosts:
            <br>
            <textarea name=hosts rows="50" cols="50">{host_output}
</textarea>
            <br>
            <input type="submit">
        </form>
    '''
    return page

@route('/ping')
def ping():
    page = f'''
        <meta http-equiv="refresh" content="5">
        {header}
    '''

```



```
hosts = db_find_host()
print(hosts)
hosts = hosts.split('\n')

for item in hosts:
    item = item.strip()
    command = f'ping -c 1 {item}'
    try:
        response = os.popen(command).read()
    except:
        response = 'Problem'
    page = f'''
        {page}
        <h2>{item}</h2>
        <pre>{response}</pre>
        '''
    return page

db_create()

run(host='127.0.0.1', port=8080)
```

# Ping Sweep Cli Tool

```
192.168.1.1 -- 64 bytes from 192.168.1.1: icmp_seq=0 ttl=64 time=8.481 ms
192.168.1.2 -- 64 bytes from 192.168.1.2: icmp_seq=0 ttl=64 time=10.698 ms
192.168.1.3 -- 64 bytes from 192.168.1.3: icmp_seq=0 ttl=255 time=121.005 ms
192.168.1.4 -- 64 bytes from 192.168.1.4: icmp_seq=0 ttl=32 time=411.041 ms
192.168.1.5 -- 64 bytes from 192.168.1.5: icmp_seq=0 ttl=255 time=107.482 ms
192.168.1.6 -- 64 bytes from 192.168.1.6: icmp_seq=0 ttl=64 time=98.092 ms
```

This lab allows you to do a ping sweep of your local LAN. This is only configured for a Type c Subnet (255.255.255.0)

## ping-sweep.py

```
import os

subnet = '192.168.1.'
command = 'ping -c 1 '
arg = '| grep time'

ip = 1
while ip <= 254:
    response = os.popen(f'{command} {subnet}{ip} {arg}').read()
    print(f'{subnet}{ip} -- {response}')
    ip += 1
```

# ARP Dump and MAC Address Vendor Lookup

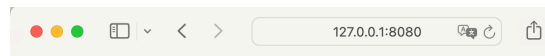
```
[ '192.168.1.1', '10:c6b:d8c:ed', 'Not in Database']  
[ '192.168.1.2', '28:80:88:47:c1:8', 'NETGEAR']  
[ '192.168.1.3', 'fc:b9:7e:1a:3b:22', 'Not Found']  
[ '192.168.1.4', 'f0:18:98:24:d0:bc', 'Apple, Inc.']  
[ '192.168.1.5', 'fc:b9:7e:33:44:7e', 'GE Appliances']  
[ '192.168.1.6', 'd4:90:9c:d4:c5:68', 'Not Found']  
[ '192.168.1.7', '9c:76:13:59:af:dc', 'Not Found']  
[ '192.168.1.9', '9c:76:13:59:b7:d5', 'Ring LLC']  
[ '192.168.1.10', '28:80:88:47:c0:18', 'NETGEAR']  
[ '192.168.1.11', 'ac:c9:6:17:4b:86', 'Not Found']  
[ '192.168.1.13', '7c:bb:8a:b4:d6:e5', 'Not Found']  
[ '192.168.1.14', '16:8b:bc:ba:7b:39', 'Not in Database']  
[ '192.168.1.15', '5c:e9:1e:86:a7:2c', 'Apple, Inc.']  
[ '192.168.1.16', '68:27:19:cb:65:91', 'Not Found']  
[ '192.168.1.17', 'a8:51:ab:98:8e:52', 'Apple, Inc.']  
[ '192.168.1.21', 'c2:fd:24:0:55:aa', 'Not in Database']  
[ '192.168.1.24', '50:26:ef:6d:ff:27', 'Not Found']  
[ '192.168.1.25', '70:66:2a:1e:cf:ff', 'Sony Interactive Entertainment Inc.']  
[ '192.168.1.26', '4c:fc:aa:8d:2:fc', 'Tesla, Inc.']  
[ '192.168.1.255', 'ff:ff:ff:ff:ff:ff', 'Not Found']  
[ '224.0.0.251', '1:0:5e:0:0:fb', 'Not in Database']  
[ '239.255.255.250', '1:0:5e:7f:ff:fa', 'Not in Database']
```

This lab dumps your ARP table and then matches the MAC Addresses to vendors using a REST API

## arp.py

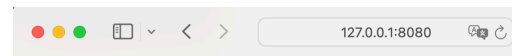
```
import os  
import requests  
  
command = 'arp -a'  
  
response = os.popen(command).readlines()  
#print(response)  
  
for x in response:  
    #print(x)  
    if ':' in x:  
        #print(x)  
        record = [0,0,0]  
        x = x.split(' ')  
        for y in x:  
            if '.' in y:  
                y = y.replace('(', '').replace(')', '')  
                #print(y)  
                record[0] = y  
            elif ':' in y:  
                #print(y)  
                record[1] = y  
        try:  
            response_mac = requests.get(f'https://api.maclookup.app/v2/  
macs/{record[1]}').json()  
            record[2] = response_mac['company']  
            if response_mac['company'] == '':  
                record[2] = 'Not in Database'  
        except:  
            record[2] = 'Not Found'  
  
    print(record)
```

# ARP Device Discovery with SQL Bckend



## Latest Response

192.168.1.1	10:c:6b:d:8c:ed	
192.168.1.2	28:80:88:47:c1:8	NETGEAR
192.168.1.3	fc:b9:7e:1a:3b:22	GE Appliances
192.168.1.4	f0:18:98:24:d0:bc	Apple, Inc.
192.168.1.5	fc:b9:7e:33:44:7e	GE Appliances
192.168.1.6	d4:90:9c:d4:c5:68	Apple, Inc.
192.168.1.7	9c:76:13:59:af:dc	Ring LLC
192.168.1.9	9c:76:13:59:b7:d5	
192.168.1.10	28:80:88:47:c0:18	NETGEAR
192.168.1.11	ac:c9:6:17:4b:86	
192.168.1.13	7c:bb:8a:b4:d6:e5	
192.168.1.14	16:8b:bc:ba:7b:39	
192.168.1.15	5c:e9:1e:86:a7:2c	Apple, Inc.
192.168.1.16	68:27:19:cb:65:91	Microchip Technology Inc.
192.168.1.17	a8:51:ab:98:8e:52	Apple, Inc.
192.168.1.21	c2:fd:24:0:55:aa	
192.168.1.24	50:26:ef:6d:ff:27	
192.168.1.25	70:66:2a:1e:cf:ff	Sony Interactive Entertainment Inc.
192.168.1.26	4c:fc:aa:8d:2:fc	Tesla, Inc.
192.168.1.255	ff:ff:ff:ff:ff:ff	
224.0.0.251	1:0:5e:0:0:fb	
239.255.255.250	1:0:5e:7f:ff:fa	



## Database Records

10:c:6b:d:8c:ed	
fc:b9:7e:1a:3b:22	GE Appliances
f0:18:98:24:d0:bc	Apple, Inc.
fc:b9:7e:33:44:7e	GE Appliances
d4:90:9c:d4:c5:68	Apple, Inc.
9c:76:13:59:af:dc	Ring LLC
9c:76:13:59:b7:d5	Ring LLC
16:8b:bc:ba:7b:39	
5c:e9:1e:86:a7:2c	Apple, Inc.
a8:51:ab:98:8e:52	Apple, Inc.
c2:fd:24:0:55:aa	
ff:ff:ff:ff:ff:ff	
1:0:5e:0:0:fb	
1:0:5e:7f:ff:fa	
28:80:88:47:c1:8	NETGEAR
28:80:88:47:c0:18	NETGEAR
ac:c9:6:17:4b:86	
7c:bb:8a:b4:d6:e5	
68:27:19:cb:65:91	Microchip Technology Inc.
50:26:ef:6d:ff:27	
70:66:2a:1e:cf:ff	Sony Interactive Entertainment Inc.
4c:fc:aa:8d:2:fc	Tesla, Inc.

This lab allows you to dump your ARP table, find vendors for MAC Addresses, and then save those values to a SQLite table. The MAC Address API provides sporadic results so this allows you to update records as new information is available from the API. This is an extremely simple version of machine learning.

## arp-web.py

```
from bottle import run, route
import os
import requests
import sqlite3

command = 'arp -a'
database = 'arp.db'

def db_create():
    conn = sqlite3.connect(database)
    cursor = conn.cursor()
    cursor.execute('CREATE TABLE IF NOT EXISTS arp(mac,vendor)')
    conn.commit()
    conn.close()

def db_find(mac):
    conn = sqlite3.connect(database)
    cursor = conn.cursor()
    try:
        query = cursor.execute('SELECT * FROM arp where mac = ?',(mac,))
        query = query.fetchone()
        print(query)
```

```

except:
    query = ''
finally:
    conn.close()

return query

def db_find_all():
    conn = sqlite3.connect(database)
    cursor = conn.cursor()
    try:
        query = cursor.execute('SELECT * FROM arp')
        query = query.fetchall()
    except:
        query = ''
    finally:
        conn.close()

    return query

def db_insert(mac, vendor):
    conn = sqlite3.connect(database)
    cursor = conn.cursor()
    try:
        cursor.execute('UPDATE arp SET vendor = ? where mac = ?',(vendor,
mac,))
        if cursor.rowcount == 0:
            cursor.execute('INSERT INTO arp(mac,vendor) values(?,?)',(mac,
vendor,))
    except:
        print('database problem')
    finally:
        conn.commit()
        conn.close()

@route('/')
def index():
    page='<table>'
    response = os.popen(command).readlines()

    for line in response:
        if ':' in line:
            record={'ip':'','mac':'','vendor':''} #Create a Dictionary
for Host Values
            line = line.split(' ')
            for value in line:
                if '.' in value:
                    value = value.replace('(', '').replace(')','')

```

```

        record['ip'] = value
    elif ':' in value:
        record['mac'] = value
    try:
        response_mac = requests.get(f'https://api.maclookup.app/v2/mac/{record['mac']}').json()
        record['vendor'] = response_mac['company']
        if response_mac['company'] == '':
            record['vendor'] = ''
    except:
        record['vendor'] = ''

    page = f'''
        {page}
        <tr>
            <td>{record['ip']}</td>
            <td>{record['mac']}</td>
            <td>{record['vendor']}</td>
        </tr>
    '''

    response_query = db_find(record['mac'])
    if response_query == None or response_query[1] == '':
        db_insert(record['mac'], record['vendor'])

    page = f'{page}</table>'

    response_all = db_find_all()
    table_db = '<table>'
    for record in response_all:
        table_db = f'''
            {table_db}
            <tr>
                <td>{record[0]}</td>
                <td>{record[1]}</td>
            </tr>
        '''

    table_db = f'{table_db} </table>'

    page = f'''
        <h1>Latest Response</h1>
        {page}
        <h1>Database Records</h1>
        {table_db}
    '''

    return(page)

db_create()

```

```
run(host='127.0.0.1', port=8080)
```