# ****IST105 - Introduction to Programming****

## ****Assignment #8****

### **Networking Automation with Python: IPv6 Addressing, DHCP Configuration, and AWS Deployment**

### **Important Note:**

Assignments that do not fulfill all the requirements or contain errors preventing successful execution on the AWS EC2 instance with a load balancer will receive a grade of zero. Ensure your repository is complete, functional, and publicly accessible before submission. Thoroughly test your application before finalizing.

## ****Prerequisites:****

* **Understanding of IPv4, IPv6, and DHCP (DHCPv4/DHCPv6)**.
* Basic knowledge of **Python networking libraries**.
* Familiarity with **Git and GitHub** for version control.
* Access to **AWS Academy** for deploying applications.

## ****Objective:****

In this assignment, students will develop a **Python-based network automation tool** that simulates **IPv6 address allocation** and **DHCPv4/DHCPv6 configuration**.

The system will:

* **Generate and validate IPv6 addresses**.
* **Simulate a DHCP server** for **IPv4 or IPv6** address leasing.
* **Deploy the application on two AWS EC2 instances**.
* **Use Git and GitHub** for version control with **three branches** (main, development, feature1).

## ****Assignment Tasks****

### **Develop the Python Network Configuration Tool**

#### **(a) Create the User Input Form (**form.php**)**

* Write a **PHP script** that allows users to:
  + Enter **client device MAC addresses** to request an **IP address**.
  + Choose **DHCPv4 or DHCPv6** as the allocation method.
  + Submit the request via **POST**.

#### **(b) Develop the Python Script (network\_config.py)**

Create a Python script that simulates a DHCP server by dynamically assigning IPv4 and IPv6 addresses based on user input. The script should perform the following tasks:

1. **Receive User Input:**
   * Accepts input via form.php or URL parameters.
   * Inputs include **MAC address** and **DHCP version selection (DHCPv4 or DHCPv6)**.
2. **Generate IPv6 Address (if DHCPv6 is selected):**
   * Uses the **EUI-64 format** to generate an IPv6 address based on the MAC address.
   * Assigns the address from a predefined IPv6 subnet (e.g., 2001:db8::/64).
3. **Simulate a DHCP Server:**
   * If **DHCPv4 is selected**, assigns a **dynamic IPv4 address** from a predefined subnet (e.g., 192.168.1.0/24).
   * If **DHCPv6 is selected**, assigns a **dynamic IPv6 address** from a configured IPv6 subnet.
   * Manages assigned IPs using a **lease database (Python dictionary)** to prevent duplicate allocations.
4. **Validate Network Configuration Before Assignment:**
   * Ensures the **MAC address format** is valid.
   * Checks if an **IP is already assigned** to the given MAC address (reuse existing lease).
   * Ensures that available IPs exist in the selected subnet; otherwise, returns an error.
5. **Output Assigned IP Address in JSON Format:**
   * Returns a structured response containing:
     + Assigned **IPv4 or IPv6 address**
     + Corresponding **MAC address**
     + **Lease duration** (for DHCPv4)
     + **Subnet details**

#### **(c) Create the PHP Processing Script (**process.php**)**

* Write a PHP script that:
  + Receives input from form.php.
  + Calls network\_config.py to process the IP address request.
  + Displays the **assigned IP address and lease expiration**.

**Example Input/Output:**

User Input (via form or URL parameters):

**MAC Address: 00:1A:2B:3C:4D:5E**

**DHCP Version: DHCPv6**

Output (Assigned IP and Lease Info):

**mac\_address: "00:1A:2B:3C:4D:5E",**

**assigned\_ipv6: "2001:db8::1a2b:3c4d:5e01",**

**lease\_time: "3600 seconds"**

User Input (via form or URL parameters):

**MAC Address: 00:1A:2B:3C:4D:5E**

**DHCP Version: DHCPv4**

Output (Assigned IP and Lease Info):

**mac\_address: "00:1A:2B:3C:4D:5E",**

**assigned\_ipv6: "192.168.1.10",**

**lease\_time: "3600 seconds"**

B:3C:4D:5E DHCP Version: DHCPv4

1. **Create two EC2 Instance:**
   * Launch two new EC2 instance using the Amazon Linux AMI.
   * Configure security groups to allow HTTP (port 80) and SSH (port 22) access.
   * Note the public IP address of your instance.

**Install Required Software:**

* 1. Install Apache web server, PHP, and Python.
  2. Start and enable the Apache service.

**3. GitHub Repository and Branching:**

* **Create a GitHub Repository:**

Create a new public GitHub repository and push your local repository to GitHub.

* **Create Branches:**

Set up the following branches in your GitHub repository:

* + **main** - This branch should contain the final, tested version of the application.
  + **development** - For integrating changes and testing before merging into the main branch.
  + **feature1** - For implementing the bitwise operations feature.

**4. Deploy to EC2 Instance:**

* **Clone the GitHub Repository to EC2:**
  + Connect to your EC2 instance via SSH:

**ssh -i <your-key-pair.pem> ec2-user@<your-public-ip>**

* + Use git clone to retrieve your GitHub repository on the EC2 instance:

**git clone <your-github-repository-url>**

* + Change directory into your project folder (replace <repository-name> with the actual name of your GitHub repository):

**cd <repository-name>**

* **File Permissions:**

Ensure that the data\_management.py script has the correct permissions to be executed:

**sudo chmod +x /var/www/html/network\_config.py**

Test the script locally on each instance:

**python3 network\_config.py**

* **Start Apache Server:**

Start or restart the Apache server to ensure it’s running your updated code:

**sudo service httpd start**

**# or**

**sudo service httpd restart**

**5. Set Up a Load Balancer:**

* **Navigate to the EC2 Dashboard:** Access the AWS Management Console and open the EC2 service.
* **Create an Application Load Balancer:** Go to the Load Balancers section, and click Create Load Balancer. Select the Application Load Balancer type.
* **Add EC2 Instances to a Target Group:** During the configuration, create or select a target group. Add both previously created EC2 instances to the target group to ensure they are part of the load balancer's traffic distribution.
* **Configure Listener for HTTP Traffic:** Set up a listener to route HTTP traffic on port 80 to the target instances in the target group.

**6. Testing and Verification:**

* Test the script using the public DNS name of the load balancer.
* Verify that the load balancer distributes traffic evenly across both instances by running tests multiple times.

**Submission Requirements:**

* **GitHub Repository Link:**

Submit the public link to your GitHub repository, which should include the branches main, development, and feature1.

**[Insert your URL here]**

* **Screenshot of EC2 Compute Instance 1 - Web Application:**

Include a screenshot of the deployed web application showing the public IP address in the URL.

**[Insert your screenshot here]**

* **Screenshot of EC2 Compute Instance 2 - Web Application:**
* Include a screenshot of the deployed web application showing the public IP address in the URL.

**[Insert your screenshot here]**

* **Load Balancer Verification:**
* Include a screenshot showing the load balancer’s DNS name and successful traffic routing.

**[Insert your screenshot here]**

**Important Notes:**

* **Functionality:** Make sure your application is fully functional on the EC2 instance and accessible via the public IP address.
* **Testing:** Verify all features thoroughly before submission.

**Tips:**

* **Git Commands Recap:** Use git branch, git checkout, git merge, and git push commands effectively to manage branches.
* **Testing Before Merging:** Always test changes on the feature1 and development branches before merging to main.