Application Program Interface (API) calls are supported in NETLAB+ VE version 17.1.6 and later.

This guide is to be used along with the NETLAB+ VE Python for SDK documentation. The NETLAB+ API methods described in this guide can be referenced in the SDK documentation, which provides greater detail. To access, visit the following link: https://ndg.tech/netlab-py-docs
# Contents

Introduction ........................................................................................................................ 3

1 Prerequisites ...................................................................................................................... 4
1.1 Pre-installation Configuration ..................................................................................... 4

2 Installing and Preparing Python on Windows ................................................................. 5
2.1 Installing Python ........................................................................................................... 5
2.2 Verifying Python System Path Variable (Best Practice – Optional) ......................... 8
2.3 Installing & Configuring a Virtual Environment (Best Practice - Optional) ............. 11
2.4 Installing iPython (Recommended - Optional) .......................................................... 14

3 Installing and Configuring the NETLAB+ Client API .................................................... 15
3.1 Installing the NETLAB+ Client API Packages ............................................................. 15
3.2 Activate the API in NETLAB+ VE ............................................................................. 16
3.3 Creating an API Key in NETLAB+ VE ....................................................................... 17
3.4 Creating config.json for the Client API Using Windows ............................................ 18
3.4.1 Automated CLI Procedure (Option 1) .................................................................. 18
3.4.1.1 Useful netlab CLI Commands ...................................................................... 22
3.4.2 Manual UI Procedure (Option 2) ........................................................................ 23

4 Issuing API Calls ............................................................................................................ 26
4.1 Using the API to Connect to a NETLAB+ System ...................................................... 26
4.2 Informative NETLAB+ API Methods ....................................................................... 27
4.2.1 Infrastructure API Methods .................................................................................. 28
4.2.2 NETLAB+ API Methods ...................................................................................... 32
4.2.2.1 VM Inventory ............................................................................................... 32
4.2.2.2 Pod Inventory .............................................................................................. 34
4.2.2.3 User & Class Inventory ............................................................................... 36
4.3 NETLAB+ API Methods using Sample Scripts ........................................................... 39
4.3.1 Downloading the Sample Scripts ......................................................................... 40
4.3.2 Creating Empty Master Pods ................................................................................. 41
4.3.2.1 Script Prep .................................................................................................. 41
4.3.2.2 Initiate ........................................................................................................ 42
4.3.3 Bring Pods Online .............................................................................................. 43
4.3.3.1 Script Prep .................................................................................................. 43
4.3.3.2 Initiate ........................................................................................................ 44
4.3.4 Bring Pods Offline ............................................................................................... 45
4.3.4.1 Script Prep .................................................................................................. 45
4.3.4.2 Initiate ........................................................................................................ 47
4.3.5 Remove Pods ....................................................................................................... 48
4.3.5.1 Script Prep .................................................................................................. 48
4.3.5.2 Initiate ........................................................................................................ 47
4.3.6 Cloning Pods ....................................................................................................... 51
4.3.6.1 Script Prep .................................................................................................. 51
4.3.6.2 Initiate ........................................................................................................ 53
4.3.7 Adding Users ....................................................................................................... 54
4.3.7.1 Script Prep .................................................................................................. 54
4.3.7.2 Initiate ........................................................................................................ 55
4.3.8 Removing Users ................................................................................................. 57
4.3.8.1 Script Prep .................................................................................................. 57
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.3.8.2</td>
<td>Initiate</td>
<td>58</td>
</tr>
<tr>
<td>4.3.9</td>
<td>Creating Classes</td>
<td>59</td>
</tr>
<tr>
<td>4.3.9.1</td>
<td>Script Prep</td>
<td>59</td>
</tr>
<tr>
<td>4.3.9.2</td>
<td>Initiate</td>
<td>60</td>
</tr>
<tr>
<td>4.3.10</td>
<td>Removing Classes</td>
<td>61</td>
</tr>
<tr>
<td>4.3.10.1</td>
<td>Script Prep</td>
<td>61</td>
</tr>
<tr>
<td>4.3.10.2</td>
<td>Initiate</td>
<td>62</td>
</tr>
<tr>
<td>Appendix A</td>
<td>Quick Start Guide for Cloning Pods</td>
<td>63</td>
</tr>
<tr>
<td>Appendix A.1</td>
<td>Quick Start Guide for Cloning Pods</td>
<td>63</td>
</tr>
</tbody>
</table>
Introduction

This is the *NETLAB+ Application Program Interface Guide for Python* for the virtual edition of *NETLAB+*.

*NETLAB+* is a remote access solution that allows academic institutions to deliver a hands-on IT training experience with a wide variety of curriculum content options. The training environment that *NETLAB+* provides enables learners to schedule and complete lab exercises for information technology courses. *NETLAB+* is a versatile solution for facilitating IT training in a variety of disciplines including networking, virtualization, storage and cyber security.

*NETLAB+ VE* features the ability to communicate with the system through Application Program Interface (API) calls, allowing customers to create custom automation scripts for many of NETLAB+’s administrative functions, such as automatically adding accounts from a Learning Management System (LMS). This guide provides details on the installation and configuration steps required in order to issue API calls using Python.
1 Prerequisites

This section will help outline what is required before installing/configuring Python and issuing API calls.

1.1 Pre-installation Configuration

Mechanisms that need to be in place:

- **NETLAB+ VE**: The *NETLAB+* needs to be deployed and licensed.
- **Network**: A network needs to be in place where the administrative machine is able to communicate with the *NETLAB+* system.
- **TCP 9000**: Going from outside to inside, this port needs to be opened on the *NETLAB+ VE* system as it provides access for the administrative machine to issue API calls to the *NETLAB+*.

*TCP 9000* is only required for administrators that are connecting remotely to their *NETLAB+ VE* system and not locally on the same network.

For more information regarding connectivity requirements for a *NETLAB+ VE* system, please see the *Firewall Requirements* section of the *NETLAB+ VE Designated Operating Environment Guide*. 
2 Installing and Preparing Python on Windows

This section outlines the steps necessary to install Python on a Windows client. Additional guidance will also be provided on how to install crucial third-party Python software along with the implementation of virtual environments to keep dependencies for separate projects in isolated environments.

2.1 Installing Python

The Python installer is required to install the Python software. This subsection will describe how to download the Python installer from the Python Software Foundation.

1. Using a web browser, preferably on an administrative machine, navigate to http://www.python.org/downloads. This will bring you to the Downloads page supported by Python Foundation.
2. While on the Downloads page, locate Python by its release number and click on its name to navigate to the available downloads section of the Python installer for Windows.

At this time, the NETLAB+ API is only compatible with the following versions of Python: 3.4, 3.5, and 3.6.

3. When on the new page, scroll down towards the Files section. Select either the 32-bit or 64-bit installer for Windows, dependent on your local host system.
4. Once the download completes, navigate to the download directory and open the python-x.x.x.exe file.
5. Using the Python Setup wizard, check the box for Add Python to PATH and proceed with the installation by selecting Customize installation.
6. In the *Optional Features* screen, ensure all checkboxes are checked and click *Next*.

![Optional Features screen](image)

Notice that *pip* will be installed as an optional feature. *Pip* is a package management tool for *Python*. This feature allows the installation of additional *Python* packages through the *Python Package Index*.

7. On the *Advanced Options* screen, change the **Customize install location** directory to the following: `C:\Python35\`

![Advanced Options screen](image)

Depending on the version installed, modify the install location appropriately (i.e. *Python 3.4* will result in `C:\Python34\`).
8. Click **Install**.

![Python 3.5.3 (64-bit) Setup](image)

If presented with the *User Account Control* window, select **Yes** to continue.

9. Once the installation process successfully finishes, click **Close**.
2.2  Verifying Python System Path Variable (Best Practice – Optional)

This subsection helps outline the process of verifying and configuring the system path variables for Python. This will provide the operating system a search path that lists the directories for the OS to search for executables.

1. Open the **Start Menu** and type *environment* in the search field. Select the **Edit the system environment variables** option.
2. In the System Properties window, click on the Environment Variables button.

3. In the Environment Variables window, focus on the User variables for admin pane and verify that a Path is present underneath the Variable column with its respective Value being the following: 

   C:\Python35\Scripts;C:\Python35\.

   The value for the path depends which directory Python has been installed. If following this guide, then the directory should be similar to C:\Python3X\.
4. If the path is missing, click on **New** and configure the following:

   ![User variables for admin](image)

   - Variable name: **PythonPath**
   - Variable value: `C:\Python\Scripts\;C:\Python35\`
   - Click **OK**.

5. Click **OK** to close the *Environment Variables* window.
6. Click **OK** once more to close the *System Properties* window.
2.3 Installing & Configuring a Virtual Environment (Best Practice - Optional)

This subsection will provide guidance on how to implement a virtual environment utilizing Python so that dependencies required by different projects can be kept in separated places. For example, one project may require a specific version of a package while another project may require an older version of the same package.

1. Launch the **Windows Command Prompt** as an administrator.
2. Using the command prompt, type the command below to list current packages install for **Python** using **pip** (package manager). Remember that **pip** was selected to be included in the install procedure when **Python** was installed.

   ```bash
   pip list
   ```

   ![Command Prompt Output]

   If a message appears stating that there is an update available, the command `python -m pip install --upgrade pip` can be used to initiate the upgrade process for the package manager.

3. Enter the command below to install the **virtualenv** package using **pip**.

   ```bash
   pip install virtualenv
   ```

   ![Virtualenv Installation Output]

4. Next, install the **virtualenvwrapper** **Python** package to add easy-to-use commands with virtual environments.

   ```bash
   pip install virtualenvwrapper-win
   ```

   ![Virtualenvwrapper Installation Output]
5. Confirm that the package has been installed correctly by issuing the command below using the command prompt.

```
pip list
```

![Image of pip list output]

6. Create a Python virtual environment, specifically for the NETLAB+ VE project.

```
mkvirtualenv netlab35
```

![Image of mkvirtualenv output]

Notice (netlab35) is printed in front of the prompt. This indicates that the netlab35 virtual environment is currently being worked on by the user.

Creating a virtual environment specifically for NETLAB+ VE, while at the same time utilizing a specific Python version will help keep the global site-packages directory clean and manageable. Notice that netlab35 was created to set a reminder that this project is working specifically with Python 3.5.
7. While having the virtual environment activated, set the project directory for the virtual environment by entering the command below.

```bash
setprojectdir C:\Users\admin\Envs\netlab35
```

```
(netlab35) C:\Python35>setprojectdir C:\Users\admin\Envs\netlab35
"C:\Users\admin\Envs\netlab35" is now the project directory for virtualenv "C:\Users\admin\Envs\netlab35"
"C:\Users\admin\Envs\netlab35" added to C:\Users\admin\Envs\netlab35\lib\site-packages\virtualenv_path_extensions.pth
```

The project directory for the virtual environment can be configured to what works best in your situation. However, it is recommended to set the project directory to the user’s home directory as opposed to the Python install directory.

8. Enter the `workon` command to activate the virtual environment and to move into the project directory.

```bash
workon netlab35
```

```
C:\\Python35\\Scripts\\workon netlab35
(netlab35) C:\\Users\\admin\\Envs\\netlab35\\
```

9. To stop working on the current project, issue the `deactivate` command.

```bash
deactivate
```

```
(netlab35) C:\\Users\\admin\\Envs\\netlab35\\deactivate
C:\\Users\\admin\\Envs\\netlab35\\
```

Notice how `netlab35` disappears from the prompt. This usually helps indicate that the project is no longer being worked on.
10. List all the current virtual environments by issuing the command below.

\texttt{lsvirtualenv}

Verify that the newly created virtual environment exists.

\textbf{If a virtual environment needs to be deleted, the \texttt{rmvirtualenv} command can be used followed by the name of the virtual environment.}

\subsection*{2.4 Installing iPython (Recommended - Optional)}

This subsection will provide guidance on how to install \textit{iPython} along with its dependencies for a more enhanced and interactive \textit{Python} shell. This is not necessary but rather an optional package that can be installed. \textit{iPython} helps provide colorful font and helps display information in an easier to read output.

1. Using the command prompt, issue the \texttt{workon} command below to activate the virtual environment for the \textit{NETLAB+} project.

\texttt{workon netlab35}

2. While engaged in the virtual environment, enter the command below to install \textit{iPython}:

\texttt{pip install ipython}

3. Verify the installation of \textit{iPython} and its main dependencies by issuing the command below.

\texttt{pip list}
3 Installing and Configuring the NETLAB+ Client API

This section will provide guidance on how to install the NETLAB+ Python API, how to activate and configure the API, as well as how to setup the configuration of the client to interact with a NETLAB+ VE system.

3.1 Installing the NETLAB+ Client API Packages

This subsection will provide guidance on how to install the NETLAB+ client API provided by NDG. This package will include all necessary files for the NETLAB+ API.

1. Using the command prompt, issue the workon command to activate the virtual environment for the NETLAB+ project.

   workon netlab35

2. Issue a pip command to install the latest Python packages from NDG.

   pip install https://ndg.tech/netlab-py-latest

   Each time you are required to update the NETLAB+ API packages, the pip install command can be used along with the same URL advertised for future updates.

For additional information on managing API users, tokens and source IP addresses, please see the Manage API Settings section of the NETLAB+ VE Administrator Guide.
3. Verify that the **netlab** package has been successfully installed, along with its main dependencies such as *click* and *tabulate*.

```
pip freeze
```

![Image of pip freeze output]

4. Verify the functionality of the new **netlab** command along with additional information about the versioning.

```
netlab version
```

![Image of netlab version output]

Do not be concerned about the given output for **NETLAB+ Server** at this time. A connection hasn’t been initiated with a **NETLAB+** system at this point and so it can be ignored.

3.2 Activate the API in **NETLAB+ VE**

This subsection will provide guidance on how to activate the API feature using the **NETLAB+ VE** administrative web interface.

1. Using a compatible web browser, navigate to a desired **NETLAB+ VE** system and login as administrator.

Information on compatible web browsers can be found in the *Supported Clients* section of the **NETLAB+ VE Designated Operating Environment Guide**.
2. Once logged in, click on the **Settings** icon located on the homepage.

3. On the **System Settings** page, click on **Manage API Settings**.

4. On the **Manage API Settings** page, click on the **Enable API** button.

### 3.3 Creating an API Key in NETLAB+ VE

This subsection will provide guidance on how to create an API key in the **NETLAB+ VE** interface as well as how to generate a token used for making API calls.

1. Once the API feature is enabled, on the **Manage API Settings**, click on the **Add API Key** button.
2. In the **New API Key** pane, enter an IP address into the **Source IPs** field. A description can be entered to help distinguish the holder of the API key.

*API calls can only be made based from the IP addresses that are inputted into the **Source IPs** field, otherwise the connection will fail.*
3. Click the **Submit** button.

4. Once submitted successfully, click the **OK** button to continue.

![API key added.](image)

5. Notice on the **View API Key** page, a token is presented. Copy this token as it will be required when continuing to the next subsection.

3.4 **Creating config.json for the Client API Using Windows**

This subsection will provide guidance on how to create and configure a `config.json` file to be used for when connecting to a NETLAB+ VE system using API calls. The subsections below describe two ways of accomplishing the same task. The first procedure (Option 1) involves using an automated CLI approach and the second procedure (Option 2) involves a manual UI approach.

3.4.1 **Automated CLI Procedure (Option 1)**

![Note](image)

This subsection describes one of two options for creating and configuring a `config.json` file. You may choose the automated procedure by following the steps below or follow the manual procedure described in Section 3.4.2.

1. Using the command prompt with administrative rights, enter the command below to navigate to the current local user account directory.

   ```
   cd C:\Users\<current_user>
   ```

   The directory `C:\Users\<current_user>` should be a user with administrative privileges. The `<current_user>` field should be replaced with your local system username. In this example, the user is named `admin`. 
2. Enter the `netlab` command below to view available options.

```
netlab --help
```

3. The command for adding a `NETLAB+` system can be inputted in one line. Entering the command below will add the specified system to a `config.json` file using the `API` token generated in the previous subsection. It is recommended to first start by configuring the default `NETLAB+` system.

```
netlab config add --host X.X.X.X --user administrator --token <api_token_value> --timeout 10
```

...Or, if interested in specifying a name for the `NETLAB+ API` configuration, the command below would be used.

```
netlab config add --system DEMO --host X.X.X.X --user administrator --token <api_token_value> --timeout 10
```

Here is a breakdown for each option used.

- **add**: This adds a new system with all required settings.
  - **--system**: Specify a desired name for a particular `NETLAB+` system.
  - **--host**: Specify a `NETLAB+` system either by an IPv4 address or FQDN.
  - **--user**: Specify the user account.
  - **--token**: Specify the token value.
  - **--timeout**: Specify the amount of time in seconds of no response from the socket before it will close.
4. Another option for adding a NETLAB+ system to the local `config.json` file can be done interactively as shown in the steps below. To begin, enter the command below to add a default NETLAB+ system.

```
netlab config add
```

If attempting to add additional NETLAB+ systems that will not be acting as the default, the following command can be used where `DEMO` can be substituted for another descriptive name: `netlab config add --system DEMO`

a. Continue the process by specifying a NETLAB+ system either by an IPv4 address of FQDN followed by pressing the `Enter` key.

```
(netlab35) C:\Users\admin\Envs\netlab35>netlab config add
Host: [IP/hostname]
User: 
```

b. Next, enter the username of the user who will have API access.

```
(netlab35) C:\Users\admin\Envs\netlab35>netlab config add
Host: [IP/hostname]
User: administrator
Token: 
```

c. Enter the token value.

```
(netlab35) C:\Users\admin\Envs\netlab35>netlab config add
Host: [IP/hostname]
User: administrator
Token: 9bTS9D3GT4GK2x5C42XN9WPNFSF3W6E26SMR72ZT
Timeout [10]: 
```

d. Enter a value in seconds for the timeout value (default is 10).

```
(netlab35) C:\Users\admin\Envs\netlab35>netlab config add
Host: [IP/hostname]
User: administrator
Token: 9bTS9D3GT4GK2x5C42XN9WPNFSF3W6E26SMR72ZT
Timeout [10]: 10
SUCCESS!
```
5. Verify the configurations were entered properly.

```
netlab config
```

![Image of netlab config output]

To learn more about modifying these settings, display the help text by running:

```
netlab config -h
```

6. Test the configuration by initiating the command below.

```
netlab config test
```

![Image of netlab config test output]

If the connection test fails and the configuration appears correct, it is advised to verify that your Source IP address is correctly entered in the API settings in the NETLAB+ administrative dashboard, reference Section 3.3.
3.4.1.1 Useful netlab CLI Commands

1. The command below can be used to verify current installed versions.

   netlab version

```
(netlab35) C:\Users\admin\Envs\netlab35>netlab version
NETLAB+ Server : 17.1.3
Python Client   : 17.1.3.1
Python Version  : 3.5.3
OS Platform     : Windows-10-10.0.14393-SP0
```

2. To remove a NETLAB+ API configuration, enter the command below with the system name specified.

   netlab config remove --system default

```
(netlab35) C:\Users\admin\Envs\netlab35>netlab config remove --system default
WARNING!
You are about to remove the settings for the default system. Are you sure? [y/N]: y
The default system has been removed.
```

A confirmation will appear, enter y followed by pressing the Enter key to confirm.
3.4.2 Manual UI Procedure (Option 2)

This subsection describes one of two options for creating and configuring a `config.json` file. You may choose the manual procedure by following the steps below or follow the automated procedure described in Section 3.4.1.

1. Using the command prompt with administrative rights, enter the command below to navigate to the current local user account directory.

   ```
cd C:\Users\<current_user>
   ```

   The directory `C:\Users\<current_user>` should be a user with administrative privileges. The `<current_user>` field should be replaced with your local system user. In this example, the user is named `admin`.

2. Create a new folder named `.netlab` by entering the command below.

   ```
mkdir .netlab
   ```

   There is period in front of the word “netlab”.

3. Launch **Notepad**.

![Notepad icon]

A different text editor can be used.

4. Using **Notepad**, type the following **JSON** configuration:

```json
{
    "default": {
        "host": "ip_address",
        "user": "user_name",
        "token": "token_value",
        "ssl": true
    }
}
```

   a. **Host**: Type the IPv4 address or FQDN of the **NETLAB+** system.
   b. **User**: Type the username found in the **NETLAB+** database; most cases will be “administrator”.
   c. **Token**: Enter the token generated from **Section 3.3**.
   d. **SSL**: Leave enabled.

![Notepad window]

5. When ready to save, select **File > Save As**.
6. In the **Save As** window, navigate to the `.netlab` folder found in the `C:\Users\current_user\` directory. Set this as the save path.
7. Type config.json in the File name field.
8. Select All Files as the Save as type.
9. Select ANSI as the encoding type and click Save.

10. Confirm that the file successfully saved with JSON appearing under the Type column.
4 Issuing API Calls

This section will help provide guidance on how to initiate the API call with a NETLAB+ VE system as well as how to run Python scripts against the system to automate tedious tasks.

4.1 Using the API to Connect to a NETLAB+ System

This subsection will provide guidance on how to initiate a connection with a NETLAB+ system utilizing Python.

1. Launch the command prompt as administrator.
2. Using the command prompt, enter the command below to initiate the netlab35 virtual environment.

```
workon netlab35
```

3. Once the virtual environment is activated, issue the command below to launch iPython (if installed).

```
ipython
```

To launch regular Python, issue the command below.

```
python
```
4. Get started by importing the `Client` class from the `netlab.client` module. This will always be the first step when initiating a connection to the API.

```python
from netlab.client import Client
```

In [1]: from netlab.client import Client
In [2]:

More information on the `netlab.client` module can be found in the `NETLAB+ VE Python SDK` documentation. To access, visit the following link: [https://ndg.tech/netlab-py-docs](https://ndg.tech/netlab-py-docs)

5. Instantiate the `Client` and assign it to a variable named `tapi`. A different variable name can be used here such as `client = Client()` or `api = Client()`. This will load the default configuration and attempt to connect to the remote API.

```python
tapi = Client()
```

When `Client()` is specified with empty parentheses, this means that it will automatically pull the “`default`” set configuration from the `config.json` file. If, for instance, an additional configuration was set in the same file with the name of `dev1`, then to load `dev1` configs, the client would be loaded with `tapi = Client('dev1')`. The system name is a string and must be enclosed with single or double quotes.

6. Issue the command below to verify which client information was pulled from the `config.json` configuration file. Since empty parentheses was used for this example, it should pull the default configuration.

```python
tapi
```

```
NetlabClient<system=default,user=administrator,host=------------------>
```

4.2 Informative NETLAB+ API Methods

This subsection will provide guidance on helpful API methods that can request useful information from a NETLAB+ system, its datacenter and hosts. It is assumed that an active Python API session is already established from the previous subsection.
4.2.1 Infrastructure API Methods

1. Using an active Python API session, the method below will retrieve a list of all associated datacenters with a NETLAB+ system along with datacenter information.

   ```python
   tapi.vm_datacenter_list()
   ```

   ![Python code output]

   The `vdc_id` is the identifier assigned to a datacenter that has been established with a particular NETLAB+ system. This value can be used with other methods utilizing “get”.

2. To retrieve datacenter information for a specific datacenter, pass the `vdc_id` from the previous step into the method below.

   ```python
   tapi.vm_datacenter_get(vdc_id=1)
   ```

   ![Python code output]

   The `vdc_id` was valid and so passing it to the `vm_datacenter_get` method resulted in a successful API call. If the `vdc_id` does not exist, an exception will be thrown. For reference regarding exceptions, visit the following for more information:
   [https://netlab-py.s3.amazonaws.com/docs/exceptions.html](https://netlab-py.s3.amazonaws.com/docs/exceptions.html)
3. Another way to retrieve the \texttt{vdc\_id} property is by calling the \texttt{vm\_datacenter\_find} method below.

\begin{verbatim}
tapi.vm_datacenter_find(vdc_name='NETLAB')
\end{verbatim}

```
In [9]: tapi.vm_datacenter_find(vdc_name='NETLAB')
Out[9]: 1
```

The name of the datacenter must be known to retrieve the \texttt{vdc\_id} using this method.

4. More information can be pulled about a \texttt{NETLAB+} system by calling the method below.

\begin{verbatim}
tapi.system_status_get()
\end{verbatim}

```
In [14]: tapi.system_status_get()
Out[14]:
{'hostname': 'ndg-\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
To list all hosts associated with a NETLAB+ system and their respective host information, the method below can be used:

```python
tapi.vm_host_list()
```

```
In [15]: tapi.vm_host_list()
Out[15]:
[[{'vh_bios_date': datetime.datetime(2013, 5, 27, 0, 0), 'vh_bios_version': '1.2.1', 'vh_com_path': 'OUTSIDE', 'vh_cpu_cores': 12, 'vh_cpu_count': 2, 'vh_cpu_mhz': 2200, 'vh_cpu_model': 'Intel(R) Xeon(R) CPU E5-2630 0 @ 2.30GHz', 'vh_cpu_threads': 24, 'vh_date_added': datetime.datetime(2016, 8, 4, 20, 1, 7), 'vh_date_modified': None, 'vh_date_tested': None, 'vh_id': 1, 'vh_inside_ipv4_addr': None, 'vh_inside_vswitch_0': None, 'vh_last_test_status': None, 'vh_memory_mb': 262094, 'vh_name': 'server', 'vh_online': True, 'vh_os_build': '3073146', 'vh_os_description': 'VMware ESXi 6.0.0 build-3073146', 'vh_os_name': 'VMware ESXi', 'vh_os_type': 'vmx-x86', 'vh_os_vendor': 'VMware, Inc.', 'vh_os_version': '6.0.0', 'vh_outside_ipv4_addr': '192.168.56.1', 'vh_pra_enabled': False, 'vh_pra_max_cpu': None, 'vh_pra_max_mem_mb': None, 'vh_pra_max_vm': None, 'vh_sys_model': 'PowerEdge C6220', 'vh_sys_service_tag': '93324289', 'vh_sys_vendor': 'Dell Inc.', 'vh_uuid': '00000000-0000-0000-0000-000000000000'}]]
```
6. To list information pertaining to only a specified host, pass the `vh_id` to the method below.

```python
tapi.vm_host_get(vh_id=1)
```

The `vh_id` was identified in the previous method where a list was generated for all hosts.
4.2.2 NETLAB+ API Methods

Several API methods that can automate tedious administrative tasks on a NETLAB+ system are described in the subsections below.

4.2.2.1 VM Inventory

1. Using an active Python API session, the method below will retrieve a list of all VMs in the Virtual Machine Inventory in a NETLAB+ system along with their respective information.

   ```python
tapi.vm_inventory_list()
   ```

Take notice of the `vm_id` property and its respective value. The value is unique when trying to identify VMs in the inventory and can be used with other methods utilizing “get”.
2. To retrieve VM information for a particular virtual machine, pass the `vm_id` parameter along with a specified value into the method below.

```python
tapi.vm_inventory_get(vm_id=163)
```

The `vm_id` was a known virtual machine and so passing it to the `vm_inventory_get` function resulted in a successful API call.
4.2.2.2 Pod Inventory

1. List all currently installed pods on a NETLAB+ system by calling the method below.

   `tapi.pod_list()`

   ![Image of tapi.pod_list() output]

2. List all the different pod types by calling the method below.

   `tapi.pod_types_list()`

   ![Image of tapi.pod_types_list() output]
3. To retrieve a list of all VMs that have a snapshot, write the script below followed by pressing the Enter key after each line. On the last line, press the Enter key twice to run the script.

```python
for vm in tapi.vm_inventory_list():
    if not vm['vm_snapshot']:
        print(vm['vm_name'])
```

Notice a list of VMs that have a snapshot in place is outputted onto the screen.
4.2.2.3 User & Class Inventory

Issuing the class_* and user_* methods, as described in this subsection, can result in increased overhead on the server side. When using these methods to query data, there are certain extended properties present that can result in higher compute to return the data to the user. It is recommended to use these methods cautiously.

1. List all classes configured on a NETLAB+ system by calling the method below.

   ```python
tapi.class_list()
   ```

   ![Screenshot](image)

   The property `cls_name` holds the name of the class.
2. Grab information about a specific class by passing the value of the `cls_id` property in the method below.

```python
  tapi.class_get(cls_id=1)
```

![Code output]

3. Retrieve a roster list from a specific class by passing the `cls_id` value in the method below.

```python
  tapi.class_roster_list(cls_id=1)
```

![Code output]
4. To retrieve a list of all accounts on a NETLAB+ system, call the method below.

```python
tapi.user_account_list()
```

![Image of user account list output]

Each user is provided a unique `acc_id` value.

5. Pass the value of the `acc_id` property to the method below to retrieve information on a specific user.

```python
tapi.user_account_get(acc_id=110803)
```

![Image of user account get output]

6. The method below may be used to reset a user's password.

```python
tapi.user_account_password_set(acc_id=110803, new_password='netlab123')
```

![Image of password reset output]

Passing the `acc_id` and `new_password` parameters will help accomplish a password reset.
7. List all communities on a given NETLAB+ system by calling the method below.

```python
tapi.user_community_list()
```

8. To retrieve information on a specific community, the method below can be called with the `com_id` parameter specified.

```python
tapi.user_community_get(com_id=1)
```

### 4.3 NETLAB+ API Methods using Sample Scripts

This subsection will provide guidance on helpful API methods using sample scripts that can automate tedious administrative tasks on a NETLAB+ system.

1. Launch a command prompt as administrator and enter the command below to navigate to the virtual environment.

```bash
workon netlab35
```

The `netlab35` can be replaced with a user specified project name dependent on the versioning of the project that is being worked on.
4.3.1 Downloading the Sample Scripts

1. Go to the URL provided below to download the sample scripts from NDG.

   https://ndg.tech/netlab-py-samples

2. Once downloaded, extract the netlab-samples-X.X.X archive into the project directory that was configured in Section 2.3 of this guide.

3. For easier accessibility, move all file contents from the samples subdirectory into the root directory of the specified project directory.
4.3.2 Creating Empty Master Pods

This subsection will provide guidance on how to automate the task of creating empty master pods that can later be filled with their necessary VMs.

4.3.2.1 Script Prep

1. Navigate to where the sample.py file is saved and edit the file using a text editor.
2. Change the client system configuration to the appropriate value. In this example, the default config is used.

3. Scroll down to the pod_add method and uncomment any lines that you wish to use, depending on the type(s) of master pods.

   The first two lines have been uncommented and so two master pods will be created. The first master pod will be called RHSA7_GM and will have a pod ID of 1000 as shown in the pod_id object. The second pod will be called NDG_EH_GM and will have a pod ID of 2000. Both pod_id and pod_name can be customizable.
4. Scroll all the way to the bottom and add a line that includes `pod_add()` . This will tell the script to initiate the uncommented lines from the `pod_add` method.

5. Once finished editing the `sample.py` file, save the changes.

### 4.3.2.2 Initiate

1. Using the command prompt with administrative access, make sure to be activated in a virtual environment, reference Section 4.3. Call the `sample.py` file to launch with the Python program by entering the command below.

   ```
   ipython sample.py
   ```

   ...Or, regular python can be used to launch the `sample.py` script as shown below.

   ```
   python sample.py
   ```

   Wait for the job to complete. Once the prompt comes back, the job is then finished running.

2. If the prompt comes back with no errors, verify the work was done in the NETLAB+ UI.

<table>
<thead>
<tr>
<th>Pod ID</th>
<th>Type</th>
<th>Pod Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>![redhat] SysAdmin</td>
<td>RHSA7_GM</td>
</tr>
<tr>
<td>2000</td>
<td>![NDG] Security Ethical Hacking</td>
<td>NDG_EH_GM</td>
</tr>
</tbody>
</table>
4.3.3 Bring Pods Online

This subsection will provide guidance on how to automate the task of putting pods online.

4.3.3.1 Script Prep

1. Navigate to where the sample.py file is saved and edit the file using a text editor.
2. Change the client system configuration to the appropriate value. In this example, the default config is used.
3. Scroll down to the pod_online method and take notice of the instructions.

Notice two master pods have been created with the pod names specified and their respective pod IDs.

It is recommended to comment back the uncommented lines or any additional lines added to the sample.py file before reusing the script again for a different task. It may be better to save the script as a new file when making changes so that there is always an original available to work from.

This value would reflect the specified --system value that was used when configuring the config.json file from Section 3.4.
4. Scroll all the way to the bottom and add a line that includes `pod_online()` and configure it to specify a single pod ID like shown below. This will make a call to the `pod_online` method with the specified values.

```python
# pod_list()
# pod_add()
# pod_rng = range(1001, 1005)
# pod_offline(range(13000, 13002))
pod_online(4411)
```

...Or, to specify a range, edit the script as shown below.

```python
# pod_list()
# pod_add()
# pod_rng = range(1001, 1005)
# pod_offline(range(13000, 13002))
pod_online(range(4411, 4414))
```

5. Once finished editing the `sample.py` file, save it.

### 4.3.3.2 Initiate

1. Using the command prompt with administrative access, make sure to be activated in a virtual environment, reference Section 4.3. Call the `sample.py` file to launch with the `Python` program by entering the command below.

```
ipython sample.py
```

```
(netlab35) C:\Python35\netlab35>ipython sample.py
Brought Online: 4411  OK
Brought Online: 4412  OK
Brought Online: 4413  OK
```

STOP Wait for the job to complete. Once the prompt comes back, the job is then finished running.
2. Verify the work was done in the NETLAB+ UI.

![Image of NETLAB+ UI](image)

> It is recommended to comment back the uncommented lines or any additional lines added to the sample.py file before reusing the script again for a different task. It may be better to save the script as a new file when making changes so that there is always an original available to work from.

### 4.3.4 Bring Pods Offline

This subsection will provide guidance on how to automate the task of putting pods offline.

#### 4.3.4.1 Script Prep

1. Navigate to where the `sample.py` file is saved and edit the file using a text editor.
2. Change the client system configuration to the appropriate value. In this example, the default config is used.

```python
46
47
48
topi = Client(system='default')
```

This value would reflect the specified `--system` value that was used when configuring the `config.json` file from Section 4.3.
3. Scroll down to the `pod_offline` method and take notice of the instructions.

```python
307 def pod_offline(pid_range):
308     """Takes pod or pods offline.
309     This function accepts either an integer for the pod_id or a range object. You can specify a range of pods that you wish to loop through or an individual pod id.
310     pod_offline(1001) or pod_offline(range(1001, 1011))
311     Uses the pod_state_changed() method in the NETLAB+ API.
312     """
313     if isinstance(pid_range, range):
314         for pid in pid_range:
315             output = make_pod_state_change(pid, pod_id, state="OFFLINE")
316             print(f"Went offline: {pid} {output}")
317     elif isinstance(pid_range, int):
318         output = make_pod_state_change(pid, pod_id, state="OFFLINE")
319         print(f"Went offline: {pid} {output}")
```

The `pod_offline` method can be called upon using two different functions, either by a single pod ID or a range of pod IDs.

4. Scroll all the way to the bottom and add a line that includes `pod_offline()` and configure it to specify a single pod ID like shown below. This function will make a call to the `pod_offline` method with the specified values.

```python
308 # pod_list()
309 # pod_add()
310 # pod_rng = range(1001, 1005)
311 pod_offline(4411)
312 # pod_online(4411)
```

...Or, to specify a range, edit the script as shown below.

```python
308 # pod_list()
309 # pod_add()
310 # pod_rng = range(1001, 1005)
311 pod_offline(range(4411, 4414))
312 # pod_online(4411)
```

5. Once finished editing the `sample.py` file, save it.
4.3.4.2 Initiate

1. Using the command prompt with administrative access, make sure to be activated in a virtual environment, reference Section 4.3. Call the sample.py file to launch with the Python program by entering the command below.

```
ipython sample.py
```

Wait for the job to complete. Once the prompt comes back, the job is then finished running.

2. Verify the work was done in the NETLAB+ UI.

It is recommended to comment back the uncommented lines or any additional lines added to the sample.py file before reusing the script again for a different task. It may be better to save the script as a new file when making changes so that there is always an original available to work from.
4.3.5 Remove Pods

This subsection will provide guidance on how to automate the task of removing pods.

4.3.5.1 Script Prep

1. Navigate to where the `sample.py` file is saved and edit the file using a text editor.
2. Change the client system configuration to the appropriate value. In this example, the default config is used.

   ```python
   # The system will need to be changed to the name given to your system in the .netlab/config.json file.
   tapi = Client(system='default')
   ```

   This value would reflect the specified `--system` value that was used when configuring the `config.json` file from Section 3.4.

3. Scroll down to the `pod_remove` method and take notice of the instructions.

   ```python
   def pod_remove(pidrange, remove_vms="DISK"):
       ""
       Removes an existing pod from the NETLAB+ pod inventory.
       ""
       # First, set or ensure the pod is set to offline
       result = tapi.pod_state_change(pid=pidrange, state="OFFLINE")
       offline_time = datetime.datetime.now()
       print("Pod Offline: " + str(pidrange) + \"\t\" + result + \"\t\" + str(offline_time))
       # Now we can remove the pod.
       result = tapi.pod_remove_task(pid=pidrange, remove_vms=remove_vms)
       removed_time = datetime.datetime.now()
       print("Pod Removed: " + str(pidrange) + \"\t\" + result + \"\t\" + str(removed_time))
   
   # Check to see if the pidrange variable is range or int type
   if not isinstance(pidrange, int):
       # First, set or ensure the pod is set to offline
       result = tapi.pod_state_change(pid=pidrange, state="OFFLINE")
       offline_time = datetime.datetime.now()
       print("Pod Offline: " + str(pidrange) + \"\t\" + result + \"\t\" + str(offline_time))
       # Now we can remove the pod.
       result = tapi.pod_remove_task(pid=pidrange, remove_vms=remove_vms)
       removed_time = datetime.datetime.now()
       print("Pod Removed: " + str(pidrange) + \"\t\" + result + \"\t\" + str(removed_time))
   ```

The `pod_remove` method can be called upon using three different functions, either by a single pod ID or a range of pod IDs as well as the type of deletion.
4. Scroll all the way to the bottom and add a line that includes `pod_remove()` and configure it to specify which pods require removal. In the example below, a single pod ID and `DISK` as the type of deletion is being scripted. This function will make a call to the `pod_remove` method with the specified values.

```python
pod_remove(4413, remove_vms="DISK")
```

...Or, to specify a range, edit the script as shown below.

```python
pod_remove(range(4413, 4416), remove_vms="DISK")
```

Here is a breakdown of the various `remove_vms` properties that can be passed:

- `remove_vms="NONE"`: Do not delete any VMs (they will remain in the NETLAB+ inventory)
- `remove_vms="LOCAL"`: Remove VMs from NETLAB+ inventory only (VMs remains in datacenter)
- `remove_vms="DATACENTER"`: Remove VMs from NETLAB+ inventory and datacenter (VM files not deleted from disk)
- `remove_vms="DISK"`: Remove VMs from NETLAB+ inventory, datacenter, AND delete unshared VM files from disk

5. Once finished editing the `sample.py` file, save it.
4.3.5.2 Initiate

1. Using the command prompt with administrative access, make sure to be activated in a virtual environment, reference Section 4.3. Call the sample.py file to launch with the Python program by entering the command below.

   ```
   ipython sample.py
   ```

   ![Command Prompt Image]

   Wait for the job to complete. Once the prompt comes back, the job is then finished running.

2. Verify the work was done in the NETLAB+ UI.

   ![Warning Image]

   It is recommended to comment back the uncommented lines or any additional lines added to the sample.py file before reusing the script again for a different task. It may be better to save the script as a new file when making changes so that there is always an original available to work from.
4.3.6 Cloning Pods

This subsection will provide guidance on how to automate the task of cloning pods.

4.3.6.1 Script Prep

1. Navigate to where the sample.py file is saved and edit the file using a text editor.
2. Change the client system configuration to the appropriate value. In this example, the default config is used.

```
# The system will need to be changed to the name given to your system in the .netlab/config.json
tmp1 = Client(system='default')
```

This value would reflect the specified --system value that was used when configuring the config.json file from Section 3.4.

3. Scroll down to the pod_clone method and take notice of the instructions.

```
def pod_clone(src_pod_id=None, clone_pod_id=None, clone_pod_name=None, pc_clone_spect=None):
    """
    Clones an existing pod.
    The pod_clone task() available via the API is one of the most complex, requested and used of the methods available in the API. There are many variables that can be changed to allow cloning of pods to changing hardware configurations as well as various pod configurations.
    Required information includes:
    source_pod_id: the numerical pod identifier of the pod you wish to clone, normally this is the Master.
    clone_pod_id: the numerical pod id that you will assign to the pod.
    clone_pod_name: the name given to the cloned instance of the pod.
    """
    pc_clone_spect = [["clone_datastore": "datastore1", "clone vh_name": "192.168.1.100"]]
    for pid in clone_pod_id:
        print(datetime.datetime.now())
        output = tmp1.pod_clone_task(source_pod_id=src_pod_id, clone_pod_id=pid, clone_pod_name=clone_name+str(pid),
        pc_clone_spect=pc_clone_spect)
        print("Cloned Pod: " + str(pid) + ",clone name=" + str(output))
```

The pod_clone method can be configured to use many variables.
4. Scroll all the way towards the bottom of the script and take notice of the `lplus_base` method. This will be used an example to copy from. Create a similar method for any desired pod type that you wish to clone. For this example, **RHSA7** pod type will be used.

```python
def rhsa7():
    # specify the pod id of the master pod.
    pod_master = 4411
    # pod_rng specifies a linear range of pod id's that you wish to manipulate.
    pod_rng = range(4413, 4416)
    # Set a prefix for pod identification.
    # A good prefix will include identifiers for course/lab set, datastore location and host. In the sample
    # below we also supply a % at the end as the pod id will be attached to it, please see pod_clone().
    pod_prefix = '%HBS1
    
    # pod_clone(src_psid=pod_master, clone_pid_rng=pod_rng, clone_pname=pod_prefix,
    # pc_clone spec=\"\"clone_datastore\": \"\"clone_vh_name\": \"172.30.0.81\"\")]
    # Uncomment the following if you wish to bring pods online. You can change the pod_online() function to pod_offline() to take the pods offline.
    # for p in pod_range:
    #    pod_offline(p)
    # pod_online()
    # Uncomment the following line to remove the pods in the pod id range (pod_rng) passed to the function.
    # pod_remove(pod_rng)
```

Here is a breakdown of the various properties used:

- **pod_master**: Identify the pod ID of the master pod to be cloned from.
- **pod_rng**: Configure the range for the number of cloned pods.
- **pod_prefix**: Configure the starting name convention for each cloned pod.
- **clone_datastore**: Identify the name of the datastore to create the cloned pod VMs to.
- **clone_vh_name**: Identify the host that the cloned pods will be running on.

5. Scroll all the way to the bottom and add a line that calls out the new method for the pod cloning. In this example, the **RHSA7** pod type is defined for pod cloning and so the `rhsa7()` method is added.

```python
# pod_list()
# pod_add()
# pod_rng = range(1001, 1005)
# pod_offline(range(4411, 4414))
# pod_online(4411)
# pod_remove(range(4413, 4416), remove_vms="DISK")
rhsa7()
```

6. Once finished editing the `sample.py` file, save it.
4.3.6.2 Initiate

1. Using the command prompt with administrative access, make sure to be activated in a virtual environment, reference Section 4.3. Call the `sample.py` file to launch with the Python program by entering the command below.

```
ipython sample.py
```

Wait for the job to complete. Once the prompt comes back, the job is then finished running.

2. Verify the work was done in the NETLAB+ UI.

The script automatically puts the cloned pods in an Online state.

It is recommended to comment back the uncommented lines or any additional lines added to the `sample.py` file before reusing the script again for a different task. It may be better to save the script as a new file when making changes so that there is always an original available to work from.
4.3.7 Adding Users

This subsection will provide guidance on how to automate the task of adding users.

4.3.7.1 Script Prep

1. Navigate to where the sample.py file is saved and edit the file using a text editor.
2. Change the client system configuration to the appropriate value. In this example, the default config is used.

```python
# The system will need to be changed to the name given to your system in the .netlab/config.jsons
tmpl = Client(system='default')
```

This value would reflect the specified --system value that was used when configuring the config.json file from Section 3.4.

3. Scroll down to the `user_add` method and take notice of the instructions.

```python
def user_add(com_id, acc_user_id, acc_password, acc_full_name, acc_type="S", cls_id=None):
    ""
    Add a user to your system.
    
    You can add other parameters to the user_account_add() method.
    
    Other parameters that may be of interest:
    - cls_id : Class ID will allow you to add the user to a class when creating the user.
    - tz_id : Time Zone ID allows you to set the set the time zone for the user upon creation.
    - acc_type: Account Type, "S" for student and "I" for instructor accounts.
    
    user_add(com_id, acc_user_id, acc_password, acc_full_name, acc_type="S",
             cls_id, tz_id, acc_display_name, acc_sort_name, acc_email)
    ""

tmpl.user_account_add(com_id=com_id, acc_user_id=acc_user_id, acc_password=acc_password,
                      acc_full_name=acc_full_name, acc_type=acc_type, cls_id=cls_id)
    return
```
4. Scroll all the way to the bottom and add a line that includes `user_add()` and configure it to add a new user account. An example is shown below for adding a student named **John Doe** to be added to the **default** community under the **NDG Training** (cls_id=6809) class with a temporary password of **demo123**. An additional student is added in the same task named **Sally Doe**.

```
user_add(1, 'jdoe', 'demo123', 'John Doe', 'S', 6809)
user_add(1, 'sdoe', 'demo123', 'Sally Doe', 'S', 6809)
```

Here is a breakdown of the various `user_add` properties that can be passed:

- **com_id**: The community ID in which to add the user to, typically 1 is the **default** community.
- **acc_user_id**: Assign a username to an account.
- **acc_password**: Assign a password for the account.
- **acc_full_name**: Assign a full name for the account.
- **acc_type**: Assign (S) for student or (I) for instructor account type.
- **cls_id**: Assign account to a specified class using an ID value. This value can be obtained from **Section 4.2.2.3**.
- **tz_id**: Assign a time zone ID for an account.

5. Once finished editing the **sample.py** file, save it.

**4.3.7.2 Initiate**

1. Using the command prompt with administrative access, make sure to be activated in a virtual environment, reference **Section 4.3**. Call the **sample.py** file to launch with the **Python** program by entering the command below.

```
ipython sample.py
```

Wait for the job to complete. Once the prompt comes back, the job is then finished running.
2. Verify the work was done in the NETLAB+ UI.

It is recommended to comment back the uncommented lines or any additional lines added to the sample.py file before reusing the script again for a different task. It may be better to save the script as a new file when making changes so that there is always an original available to work from.
4.3.8 Removing Users

This subsection will provide guidance on how to automate the task of removing users.

4.3.8.1 Script Prep

1. Navigate to where the sample.py file is saved and edit the file using a text editor.
2. Change the client system configuration to the appropriate value. In this example, the default config is used.

```python
# The system will need to be changed to the name given to your system in the .netlab/config.jsons
tmpl = Client(system='default')
```

This value would reflect the specified --system value that was used when configuring the config.json file from Section 3.4.

3. Scroll down to the user_del method and take notice of the instructions.

```python
def user_del(search_params=''):  ""
    Delete a user from the system.
    user_del('student1')
    del_list = user_list(search_param=search_param)
    for x in del_list:
        tmpl.user_account.remove(acc_id=x)
    print('acc_id' + str(x) + 'deleted')
    return
```

4. Scroll all the way to the bottom and add a line that includes `user_del()` and configure it to remove a user account. In this example, the user `jdoe` is removed from the system.

```python
# pod_list()  # pod_add()  # pod_remove = range(1001, 1005)
# pod_offline(range(4411, 4414))  # pod_online(4411)
# pod_remove(range(4413, 4416), remove_vms="DISK")
# rhost()  # user_add('jdoe', 'demo123', 'John Doe', 'S', 6809)  user_del('jdoe')
```

5. Once finished editing the sample.py file, save it.
### 4.3.8.2 Initiate

1. Using the command prompt with administrative access, make sure to be activated in a virtual environment, reference Section 4.3. Call the `sample.py` file to launch with the Python program by entering the command below.

   ```
   ipython sample.py
   ```

   ![Command Prompt Output]

   Wait for the job to complete. Once the prompt comes back, the job is then finished running.

2. Verify the work was done in the NETLAB+ UI.

   ![NETLAB+ UI Output]

   It is recommended to comment back the uncommented lines or any additional lines added to the `sample.py` file before reusing the script again for a different task. It may be better to save the script as a new file when making changes so that there is always an original available to work from.
4.3.9  Creating Classes

This subsection will provide guidance on how to automate the task of creating classes.

4.3.9.1  Script Prep

1. Navigate to where the sample.py file is saved and edit the file using a text editor.
2. Change the client system configuration to the appropriate value. In this example, the default config is used.
3. Scroll down to the class_add method and take notice of the instructions.
4. Scroll all the way to the bottom and add a line that includes class_add() and configure it to add a new class. This example creates two classes: Class 1 and Class 2, and adds them to the default community (com_id=1).
5. Once finished editing the sample.py file, save it.
4.3.9.2 Initiate

1. Using the command prompt with administrative access, make sure to be activated in a virtual environment, reference Section 4.3. Call the `sample.py` file to launch with the Python program by entering the command below.

   ```
   ipython sample.py
   ```

   ![Wait for the job to complete. Once the prompt comes back, the job is then finished running.]

   The new class IDs are shown in the output.

2. Verify the work was done in the NETLAB+ UI.

   ![It is recommended to comment back the uncommented lines or any additional lines added to the sample.py file before reusing the script again for a different task. It may be better to save the script as a new file when making changes so that there is always an original available to work from.]

   ![The new class IDs are shown in the output.]

   ![Wait for the job to complete. Once the prompt comes back, the job is then finished running.]

   ![It is recommended to comment back the uncommented lines or any additional lines added to the sample.py file before reusing the script again for a different task. It may be better to save the script as a new file when making changes so that there is always an original available to work from.]

4.3.10 Removing Classes

This subsection will provide guidance on how to automate the task of removing classes.

4.3.10.1 Script Prep

1. Navigate to where the sample.py file is saved and edit the file using a text editor.
2. Change the client system configuration to the appropriate value. In this example, the default config is used.
3. Scroll down to the class_remove method and take notice of the instructions.
4. Scroll all the way to the bottom and add a line that includes class_remove() and configure it to remove a class. This example removes Class 1 from the system and retains the user accounts attached to the class.

Here is a breakdown of the various class_remove properties that can be passed:

- cls_id: Assign account to a specified class using an ID value. This value can be obtained from Section 4.2.2.3.
- delete_students: Set either True (delete attached student accounts) or False (do no delete attached student accounts)

5. Once finished editing the sample.py file, save it.
4.3.10.2 Initiate

1. Using the command prompt with administrative access, make sure to be activated in a virtual environment, reference Section 4.3. Call the sample.py file to launch with the Python program by entering the command below.

   ipython sample.py

   (netlab35) C:\Python35\netlab35>ipython sample.py
   OK
   (netlab35) C:\Python35\netlab35>

   Stop Wait for the job to complete. Once the prompt comes back, the job is then finished running.

2. Verify the work was done in the NETLAB+ UI.

   ![Community: default](default) ![Search](search)

<table>
<thead>
<tr>
<th>Name</th>
<th>Leads</th>
<th>Enrolled</th>
<th>End Date</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class 1</td>
<td>None</td>
<td>0</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>NDG Training</td>
<td>None</td>
<td>1</td>
<td>None</td>
<td></td>
</tr>
</tbody>
</table>

   Warning It is recommended to comment back the uncommented lines or any additional lines added to the sample.py file before reusing the script again for a different task. It may be better to save the script as a new file when making changes so that there is always an original available to work from.
Appendix A

Appendix A.1  Quick Start Guide for Cloning Pods

This section will help outline the processes required, at minimum, to get started on cloning pods as the end goal of a NETLAB+ administrator. To get started, follow the steps provided below to accomplish the cloning task.

1. Install and configure Python, go through Section 2.1.
2. Install the NETLAB+ client API packages, see Section 3.1.
3. Activate the API feature in NETLAB+, see Section 3.2.
4. Create an API key, see Section 3.3.
5. Create and configure a config.json file for the specified NETLAB+, see Section 3.4.
6. Download sample scripts, see Section 4.3.1.
7. Modify the sample scripts to clone specific pods, see Section 4.3.6.