Application Program Interface Guide for Python

Document Version: 2017-06-15

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

Copyright © 2017 Network Development Group, Inc.
www.netdevgroup.com

NETLAB Academy Edition, NETLAB Professional Edition, and NETLAB+ are registered trademarks of Network Development Group, Inc.

VMware is a registered trademark of VMware, Inc. Cisco, IOS, Cisco IOS, Networking Academy, CCNA, and CCNP are registered trademarks of Cisco Systems, Inc. EMC^2 is a registered trademark of EMC Corporation.

Contents

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+*.

Please Note

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.

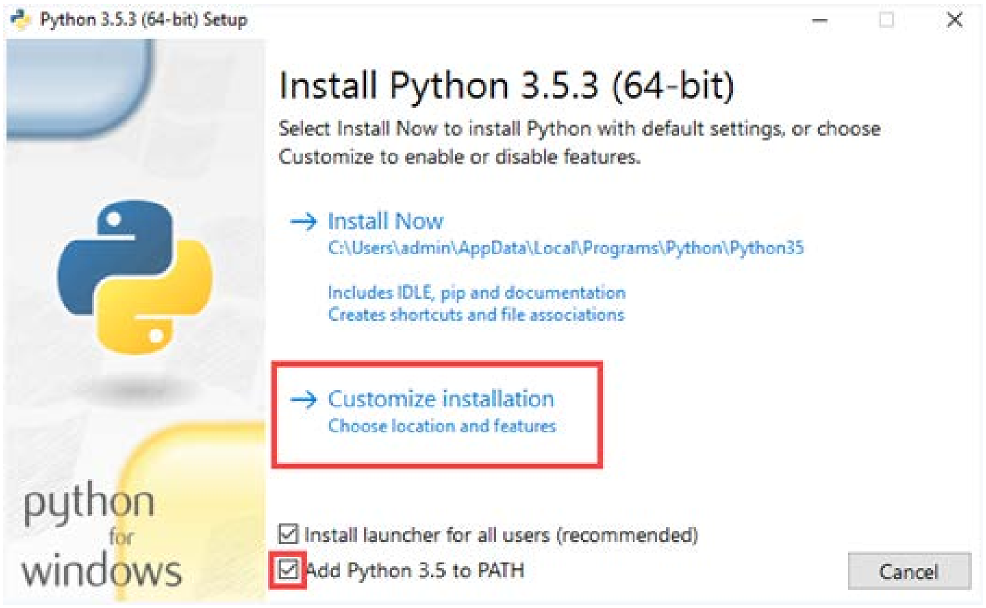
2.1 Installing Python

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**.


Python 3.5.3 (64-bit) Setup

Install Python 3.5.3 (64-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

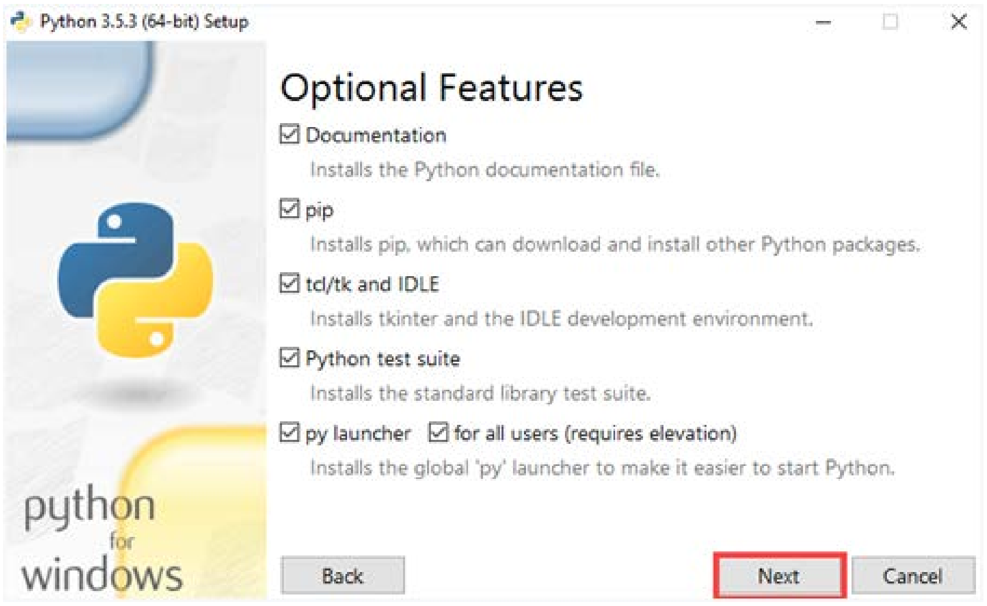
→ Install Now
C:\Users\admin\AppData\Local\Programs\Python\Python35

Includes IDLE, pip and documentation
Creates shortcuts and file associations

→ Customize installation
Choose location and features

☑ Install launcher for all users (recommended)
☑ Add Python 3.5 to PATH

6. In the *Optional Features* screen, ensure all checkboxes are checked and click **Next**.


Python 3.5.3 (64-bit) Setup

Optional Features

☑ Documentation
Installs the Python documentation file.

☑ pip
Installs pip, which can download and install other Python packages.

☑ tcl/tk and IDLE
Installs tkinter and the IDLE development environment.

☑ Python test suite
Installs the standard library test suite.

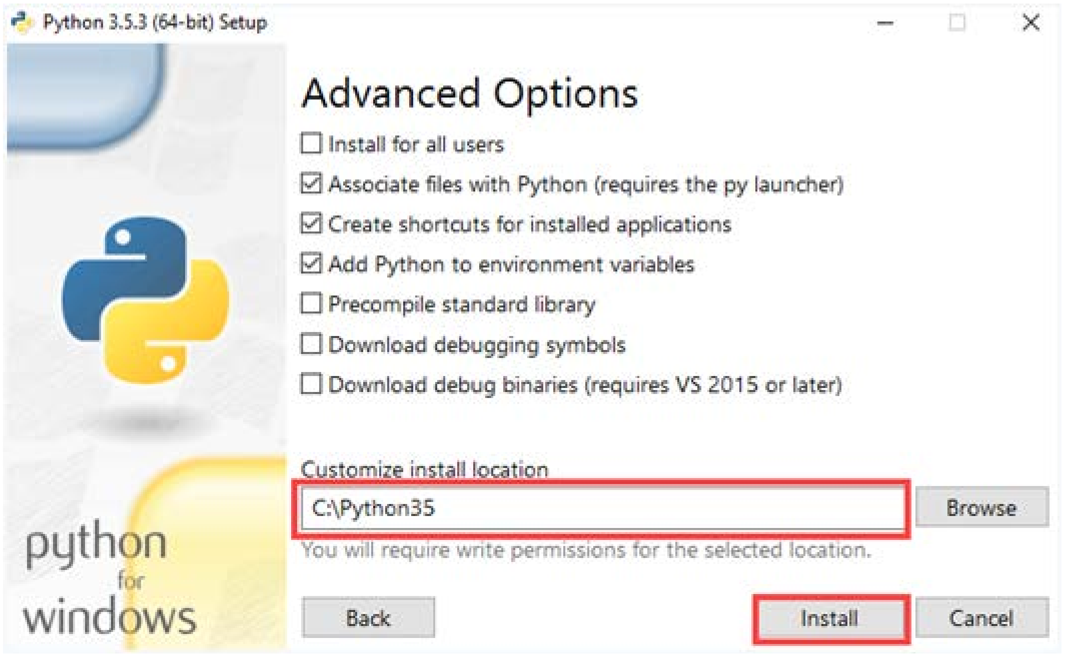
☑ py launcher ☑ for all users (requires elevation)
Installs the global 'py' launcher to make it easier to start Python.

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\`**

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

Advanced Options

- [ ] Install for all users
- [x] Associate files with Python (requires the py launcher)
- [x] Create shortcuts for installed applications
- [x] Add Python to environment variables
- [ ] Precompile standard library
- [ ] Download debugging symbols
- [ ] Download debug binaries (requires VS 2015 or later)

Customize install location

C:\Python35

You will require write permissions for the selected location.

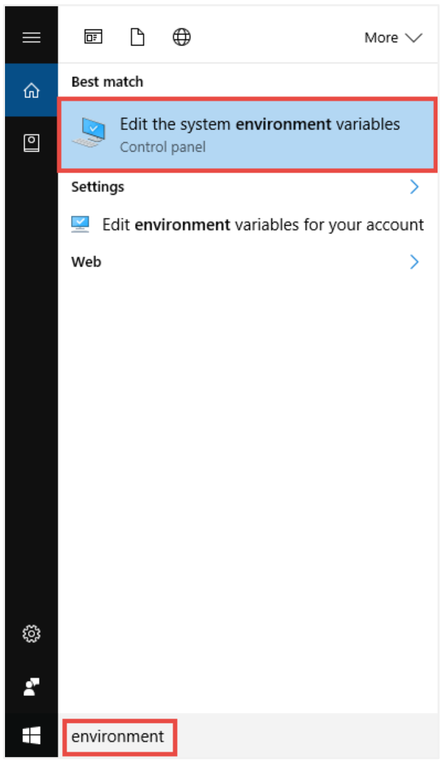
If presented with the *User Account Control* window, select **Yes** to continue.

9. Once the installation process successfully finishes, click **Close**.

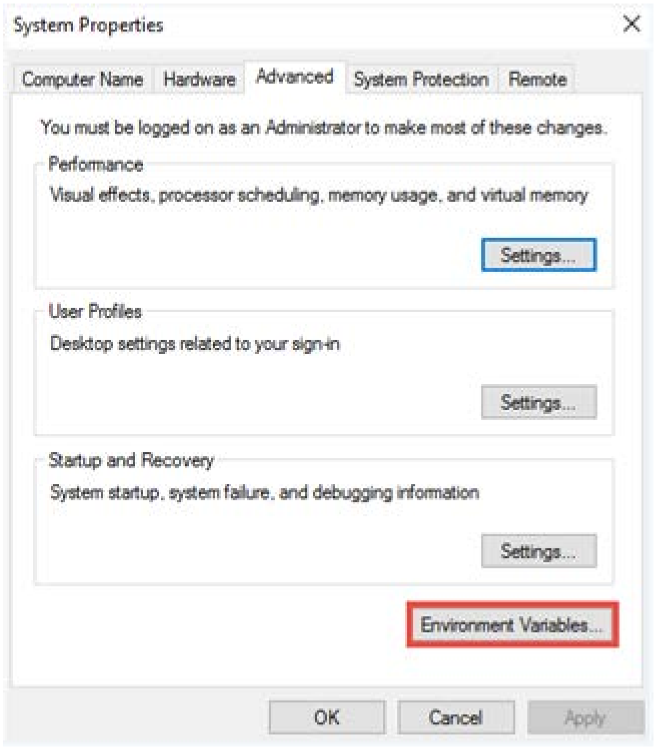
2.2 Verifying Python System Path Variable (Best Practice – Optional)

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.


More
Best match
Edit the system environment variables
Control panel
Settings
Edit environment variables for your account
Web
environment

2. In the *System Properties* window, click on the **Environment Variables** button.


System Properties

Computer Name | Hardware | Advanced | System Protection | Remote

You must be logged on as an Administrator to make most of these changes.

Performance
Visual effects, processor scheduling, memory usage, and virtual memory

User Profiles
Desktop settings related to your sign-in

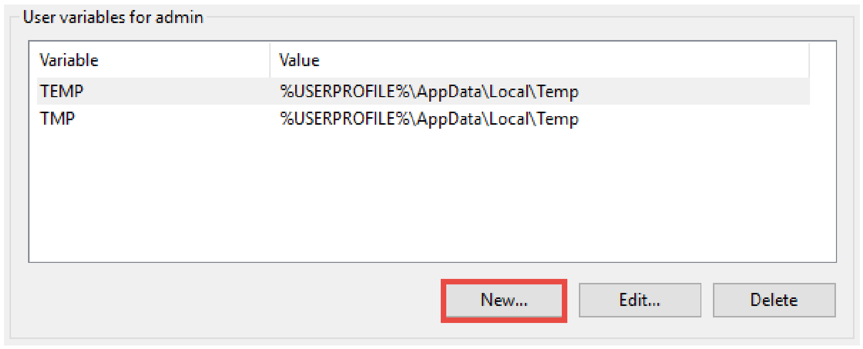
Startup and Recovery
System startup, system failure, and debugging information

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\`**

Please Note

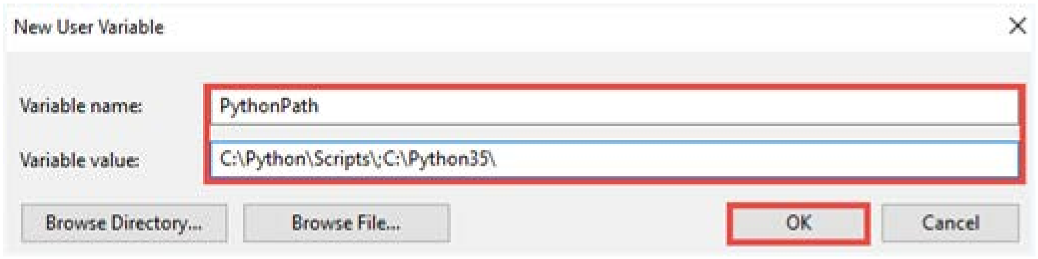
The value for the path depends which directory *Python* has been installed. If following this guide, then the directory should be similar to *C:\Python3X\*.

4. If the path is missing, click on **New** and configure the following:


User variables for admin

Variable	Value
TEMP	%USERPROFILE%\AppData\Local\Temp
TMP	%USERPROFILE%\AppData\Local\Temp

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


New User Variable

Variable name: PythonPath

Variable value: C:\Python\Scripts\;C:\Python35\

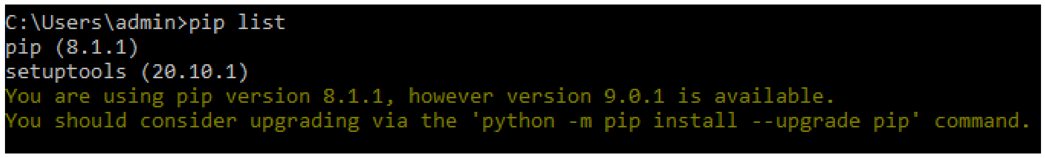
5. Click **OK** to close the *Environment Variables* window.
6. Click **OK** once more to close the *System Properties* window.

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.

```
pip list
```

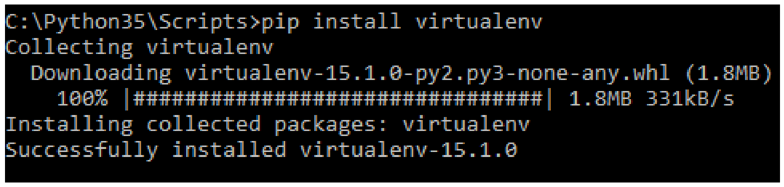


```
C:\Users\admin>pip list
pip (8.1.1)
setuptools (20.10.1)
You are using pip version 8.1.1, however version 9.0.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.
```

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*.

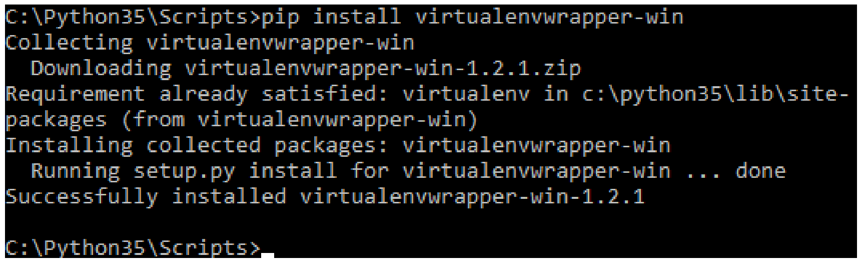
```
pip install virtualenv
```



```
C:\Python35\Scripts>pip install virtualenv
Collecting virtualenv
  Downloading virtualenv-15.1.0-py2.py3-none-any.whl (1.8MB)
    100% |################################| 1.8MB 331kB/s
Installing collected packages: virtualenv
Successfully installed virtualenv-15.1.0
```

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

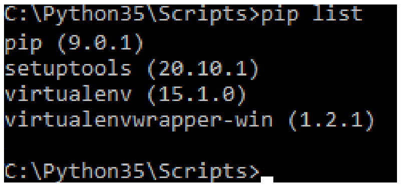
```
pip install virtualenvwrapper-win
```



```
C:\Python35\Scripts>pip install virtualenvwrapper-win
Collecting virtualenvwrapper-win
  Downloading virtualenvwrapper-win-1.2.1.zip
Requirement already satisfied: virtualenv in c:\python35\lib\site-
packages (from virtualenvwrapper-win)
Installing collected packages: virtualenvwrapper-win
  Running setup.py install for virtualenvwrapper-win ... done
Successfully installed virtualenvwrapper-win-1.2.1

C:\Python35\Scripts>_
```

5. Confirm that the package has been installed correctly by issuing the command below using the command prompt.

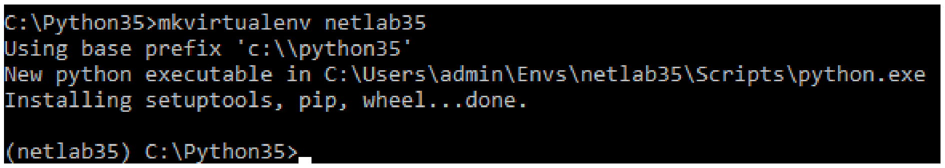
```
pip list
```



```
C:\Python35\Scripts>pip list
pip (9.0.1)
setuptools (20.10.1)
virtualenv (15.1.0)
virtualenvwrapper-win (1.2.1)

C:\Python35\Scripts>
```

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

```
mkvirtualenv netlab35
```



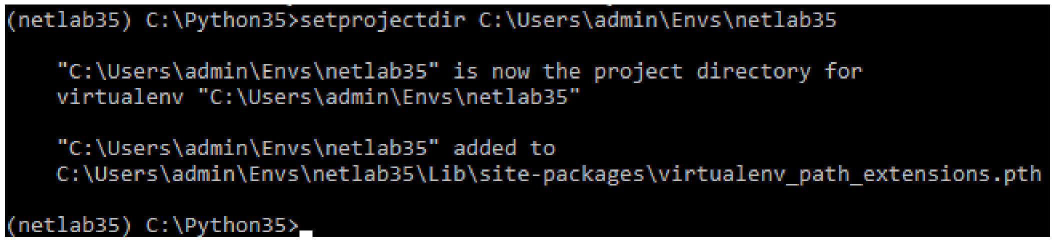
```
C:\Python35>mkvirtualenv netlab35
Using base prefix 'c:\\python35'
New python executable in C:\Users\admin\Envs\netlab35\Scripts\python.exe
Installing setuptools, pip, wheel...done.

(netlab35) C:\Python35>
```

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.

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

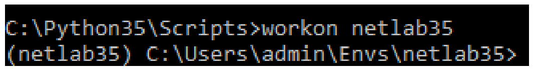


```
(netlab35) C:\Python35>setprojectdir C:\Users\admin\Envs\netlab35

    "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

(netlab35) C:\Python35>
```

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.

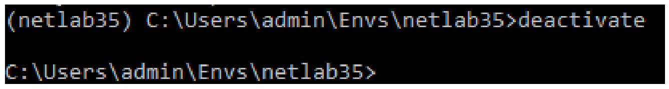
```
workon netlab35
```



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

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

```
deactivate
```

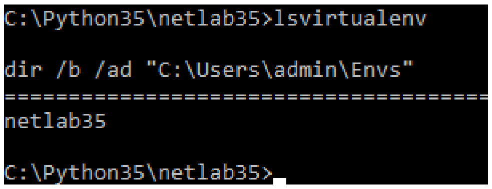


```
(netlab35) C:\Users\admin\Envs\netlab35>deactivate

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.

```
lsvirtualenv
```



```
C:\Python35\netlab35>lsvirtualenv

dir /b /ad "C:\Users\admin\Envs"
=====================================
netlab35

C:\Python35\netlab35>_
```

Verify that the newly created virtual environment exists.

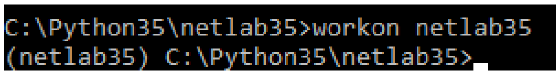
If a virtual environment needs to be deleted, the **`rmvirtualenv`** command can be used followed by the name of the virtual environment.

2.4 Installing iPython (Recommended - Optional)

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

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

```
workon netlab35
```



```
C:\Python35\netlab35>workon netlab35
(netlab35) C:\Python35\netlab35>_
```

2. While engaged in the virtual environment, enter the command below to install *iPython*:

```
pip install ipython
```

3. Verify the installation of *iPython* and its main dependencies by issuing the command below.

```
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.

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.

3.1 Installing the NETLAB+ Client API Packages

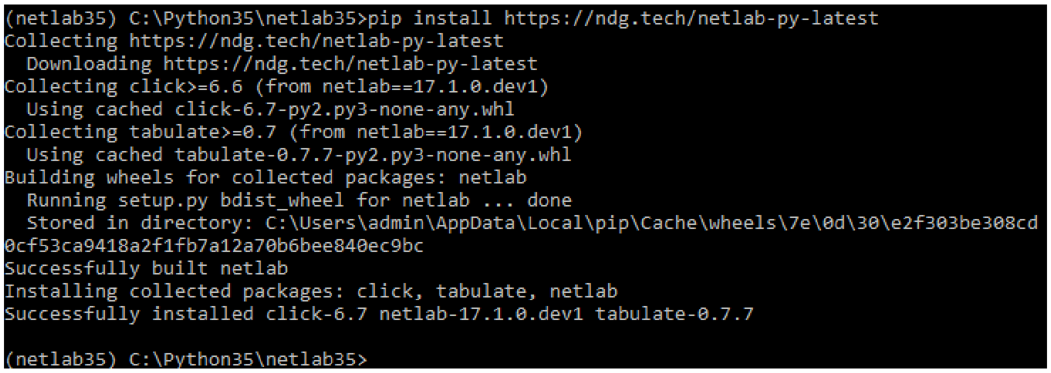
This subsection will provide guidance on how to install the *NETLAB+* client API provided by *NDG*. This package will include all necessary files for the *NETLAB+* API.

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
```

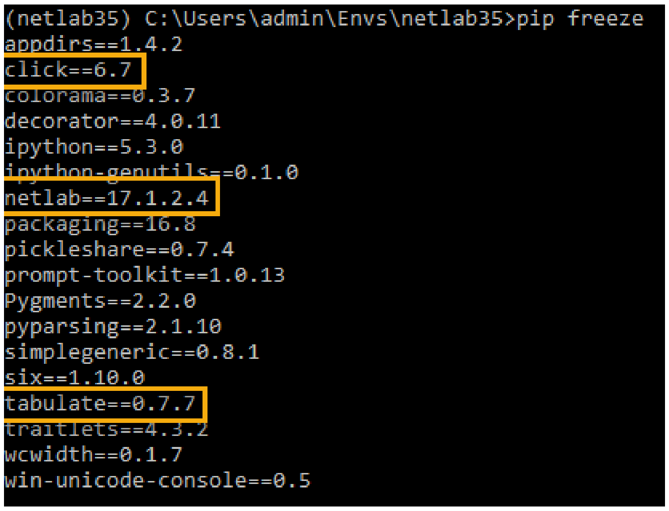


```
(netlab35) C:\Python35\netlab35>pip install https://ndg.tech/netlab-py-latest
Collecting https://ndg.tech/netlab-py-latest
  Downloading https://ndg.tech/netlab-py-latest
Collecting click>=6.6 (from netlab==17.1.0.dev1)
  Using cached click-6.7-py2.py3-none-any.whl
Collecting tabulate>=0.7 (from netlab==17.1.0.dev1)
  Using cached tabulate-0.7.7-py2.py3-none-any.whl
Building wheels for collected packages: netlab
  Running setup.py bdist_wheel for netlab ... done
  Stored in directory: C:\Users\admin\AppData\Local\pip\Cache\wheels\7e\0d\30\e2f303be308cd
0cf53ca9418a2f1fb7a12a70b6bee840ec9bc
Successfully built netlab
Installing collected packages: click, tabulate, netlab
Successfully installed click-6.7 netlab-17.1.0.dev1 tabulate-0.7.7

(netlab35) C:\Python35\netlab35>
```

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.

3. Verify that the **netlab** package has been successfully installed, along with its main dependencies such as *click* and *tabulate*.

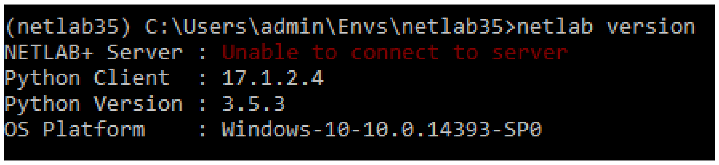
```
pip freeze
```



```
(netlab35) C:\Users\admin\Envs\netlab35>pip freeze
appdirs==1.4.2
click==6.7
colorama==0.3.7
decorator==4.0.11
ipython==5.3.0
ipython-genutils==0.1.0
netlab==17.1.2.4
packaging==16.8
pickleshare==0.7.4
prompt-toolkit==1.0.13
Pygments==2.2.0
pyparsing==2.1.10
simplegeneric==0.8.1
six==1.10.0
tabulate==0.7.7
traitlets==4.3.2
wcwidth==0.1.7
win-unicode-console==0.5
```

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

```
netlab version
```



```
(netlab35) C:\Users\admin\Envs\netlab35>netlab version
NETLAB+ Server : Unable to connect to server
Python Client  : 17.1.2.4
Python Version : 3.5.3
OS Platform    : Windows-10-10.0.14393-SP0
```

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.

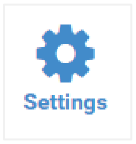
3.2 Activate the API in NETLAB+ VE

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

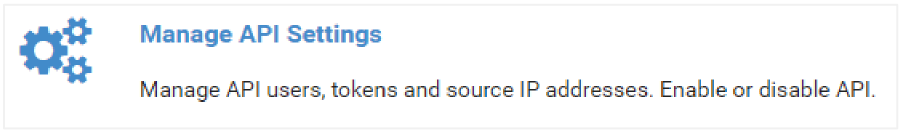
1. Using a compatible web browser, navigate to a desired *NETLAB+ VE* system and login as administrator.

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**.

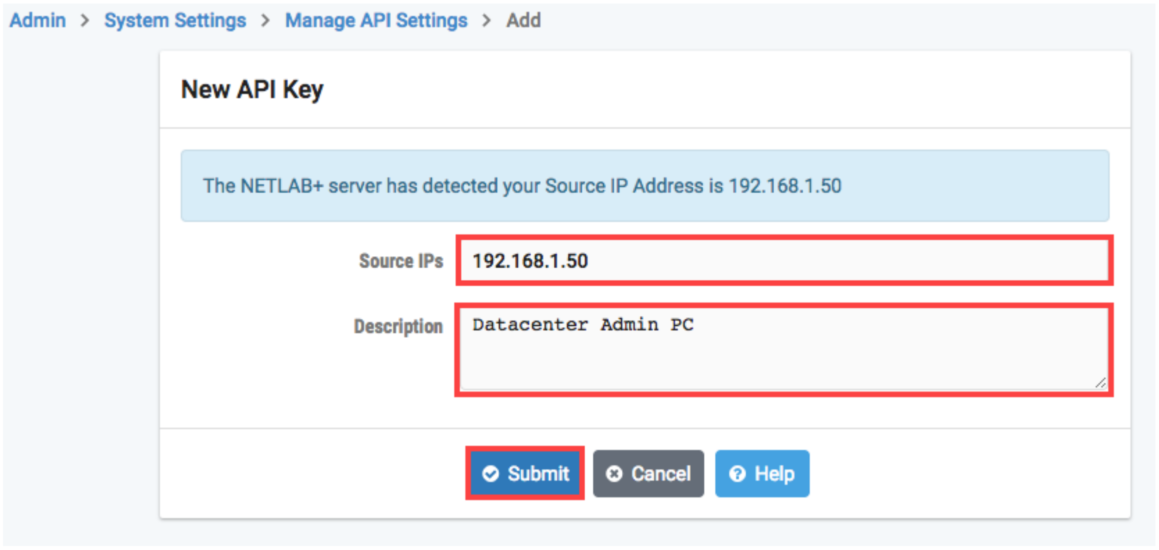

Manage API Settings
Manage API users, tokens and source IP addresses. Enable or disable API.

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.



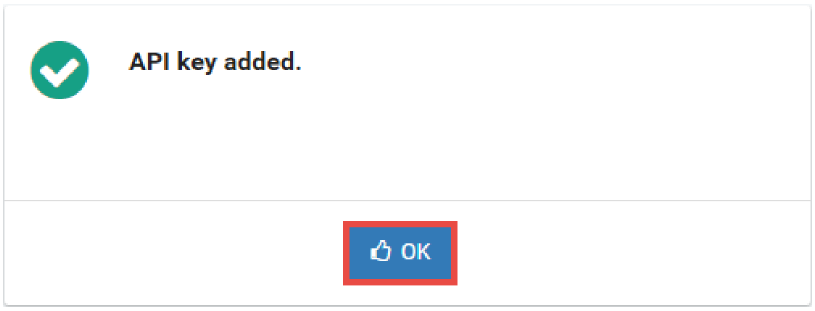
The NETLAB+ server has detected your Source IP Address is 192.168.1.50

Source IPs: 192.168.1.50

Description: Datacenter Admin PC

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.

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

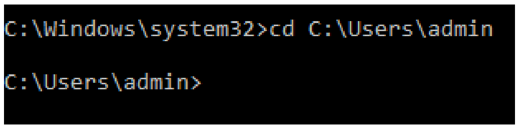
This subsection will provide guidance on how to create and configure a *config.json* file to be used for when connecting to a *NETLAB+ VE* system using API calls. The subsections below describe two ways of accomplishing the same task. The first procedure (Option 1) involves using an automated CLI approach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

3.4.1 Automated CLI Procedure (Option 1)

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>
```

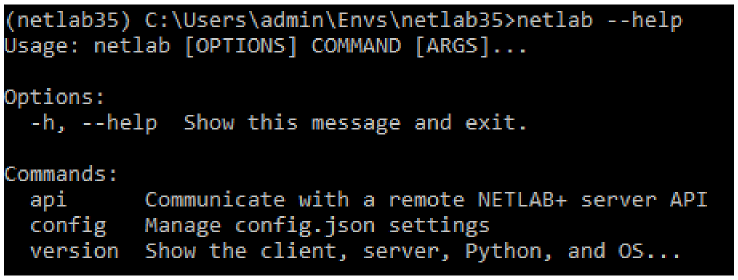


```
C:\Windows\system32>cd C:\Users\admin

C:\Users\admin>
```

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.

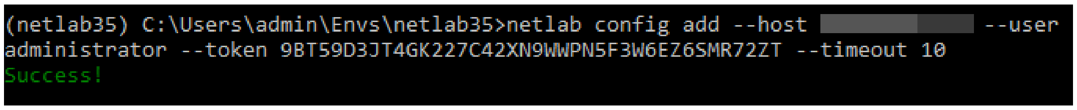
```
netlab --help
```



```
(netlab35) C:\Users\admin\Envs\netlab35>netlab --help
Usage: netlab [OPTIONS] COMMAND [ARGS]...

Options:
  -h, --help  Show this message and exit.

Commands:
  api      Communicate with a remote NETLAB+ server API
  config   Manage config.json settings
  version  Show the client, server, Python, and OS...
```

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.

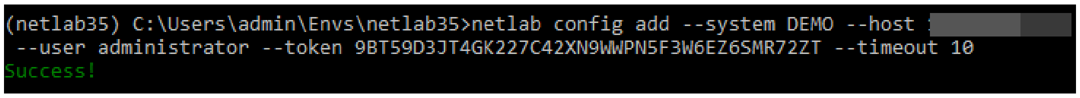
```
netlab config add --host X.X.X.X --user administrator --token
<api_token_value> --timeout 10
```



```
(netlab35) C:\Users\admin\Envs\netlab35>netlab config add --host [redacted] --user
administrator --token 9BT59D3JT4GK227C42XN9WWPN5F3W6EZ6SMR72ZT --timeout 10
Success!
```

…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
```



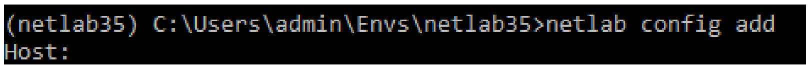
```
(netlab35) C:\Users\admin\Envs\netlab35>netlab config add --system DEMO --host [redacted]
 --user administrator --token 9BT59D3JT4GK227C42XN9WWPN5F3W6EZ6SMR72ZT --timeout 10
Success!
```

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.

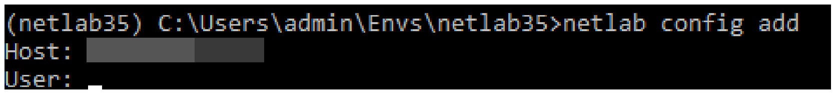
```
netlab config add
```



```
(netlab35) C:\Users\admin\Envs\netlab35>netlab config add
Host:
```

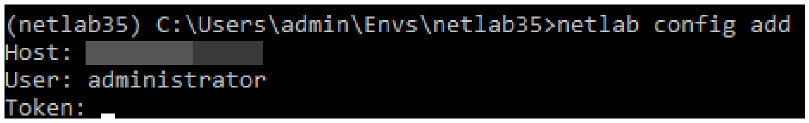
If attempting to add additional *NETLAB+* systems that will not be acting as the default, the following command can be used where *DEMO* can be substituted for another descriptive name: *netlab config add --system DEMO*

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



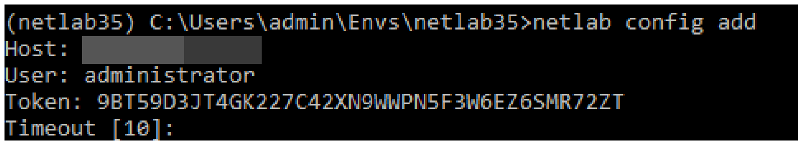
```
(netlab35) C:\Users\admin\Envs\netlab35>netlab config add
Host: [redacted]
User: _
```

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



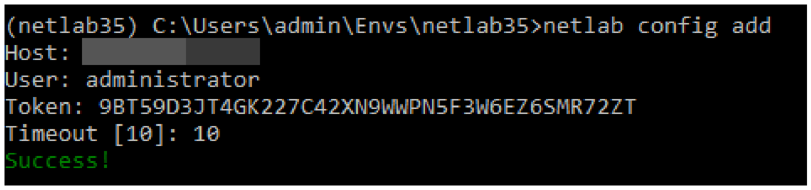
```
(netlab35) C:\Users\admin\Envs\netlab35>netlab config add
Host: [redacted]
User: administrator
Token: _
```

c. Enter the token value.



```
(netlab35) C:\Users\admin\Envs\netlab35>netlab config add
Host: [redacted]
User: administrator
Token: 9BT59D3JT4GK227C42XN9WWPN5F3W6EZ6SMR72ZT
Timeout [10]:
```

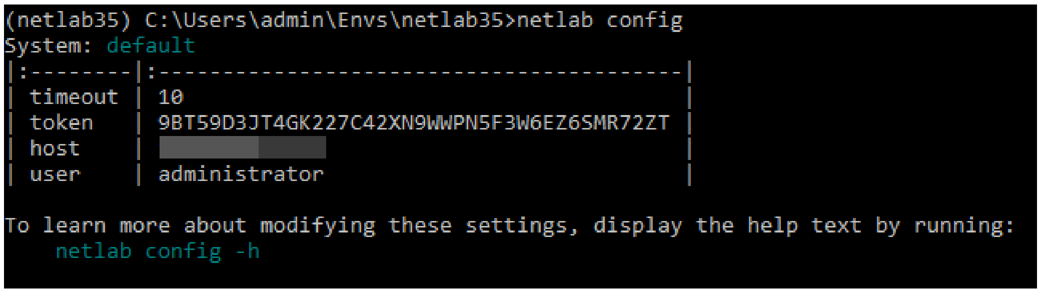
d. Enter a value in seconds for the timeout value (default is 10).



```
(netlab35) C:\Users\admin\Envs\netlab35>netlab config add
Host: [redacted]
User: administrator
Token: 9BT59D3JT4GK227C42XN9WWPN5F3W6EZ6SMR72ZT
Timeout [10]: 10
Success!
```

5. Verify the configurations were entered properly.

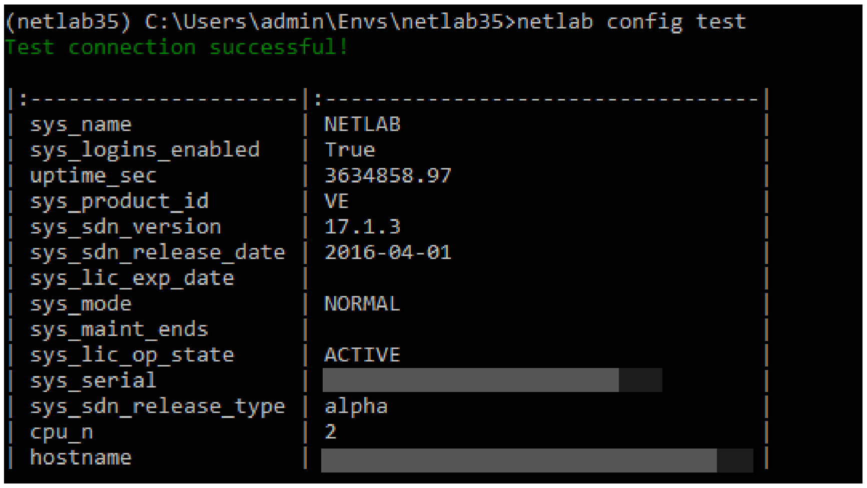
```
netlab config
```



```
(netlab35) C:\Users\admin\Envs\netlab35>netlab config
System: default
|:--------|:-----------------------------------------|
| timeout | 10                                       |
| token   | 9BT59D3JT4GK227C42XN9WWPN5F3W6EZ6SMR72ZT |
| host    | [redacted]                               |
| user    | administrator                            |

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
```



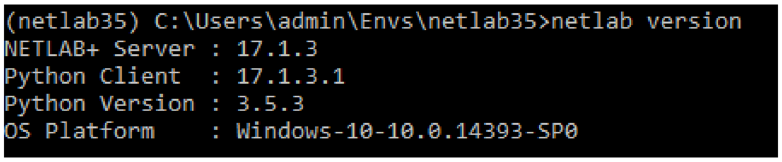
```
(netlab35) C:\Users\admin\Envs\netlab35>netlab config test
Test connection successful!

|:---------------------|:----------------------------------|
| sys_name             | NETLAB                            |
| sys_logins_enabled   | True                              |
| uptime_sec           | 3634858.97                        |
| sys_product_id       | VE                                |
| sys_sdn_version      | 17.1.3                            |
| sys_sdn_release_date | 2016-04-01                        |
| sys_lic_exp_date     |                                   |
| sys_mode             | NORMAL                            |
| sys_maint_ends       |                                   |
| sys_lic_op_state     | ACTIVE                            |
| sys_serial           | [redacted]                        |
| sys_sdn_release_type | alpha                             |
| cpu_n                | 2                                 |
| hostname             | [redacted]                        |
```

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.

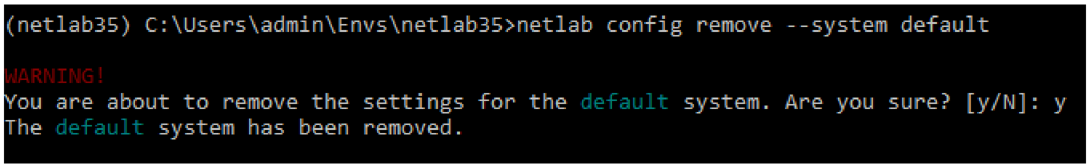
```
netlab version
```



```
(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.
```

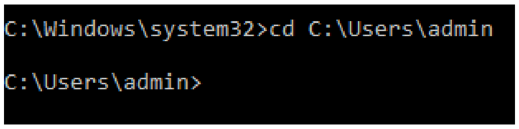
A confirmation will appear, enter **y** followed by pressing the **Enter** key to confirm.

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>
```

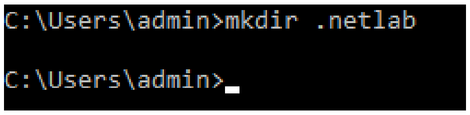


```
C:\Windows\system32>cd C:\Users\admin

C:\Users\admin>
```

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
```



```
C:\Users\admin>mkdir .netlab

C:\Users\admin>_
```

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

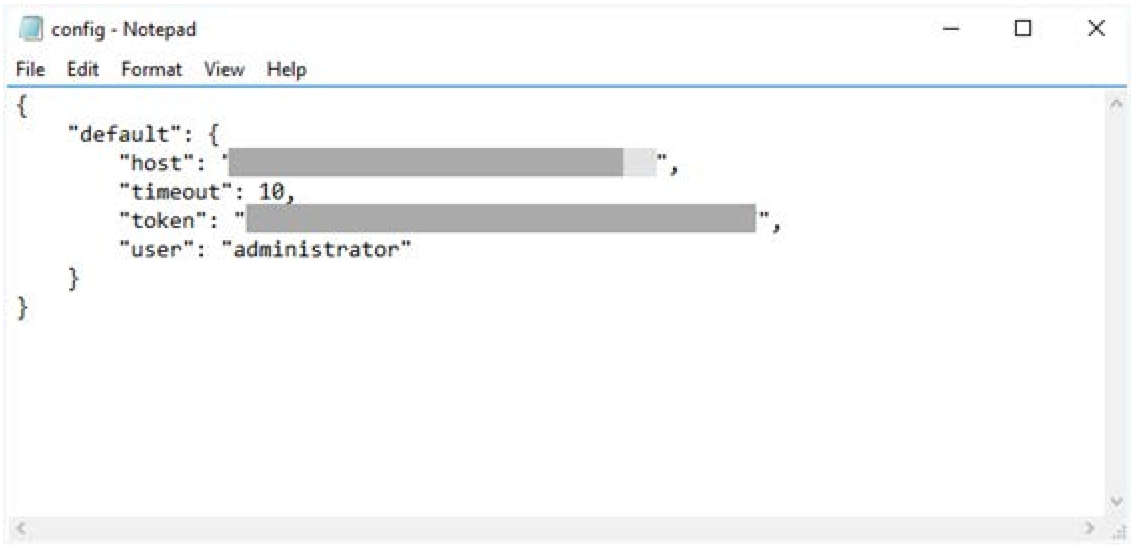
3. Launch **Notepad**.

A different text editor can be used.

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

```
{
        "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.

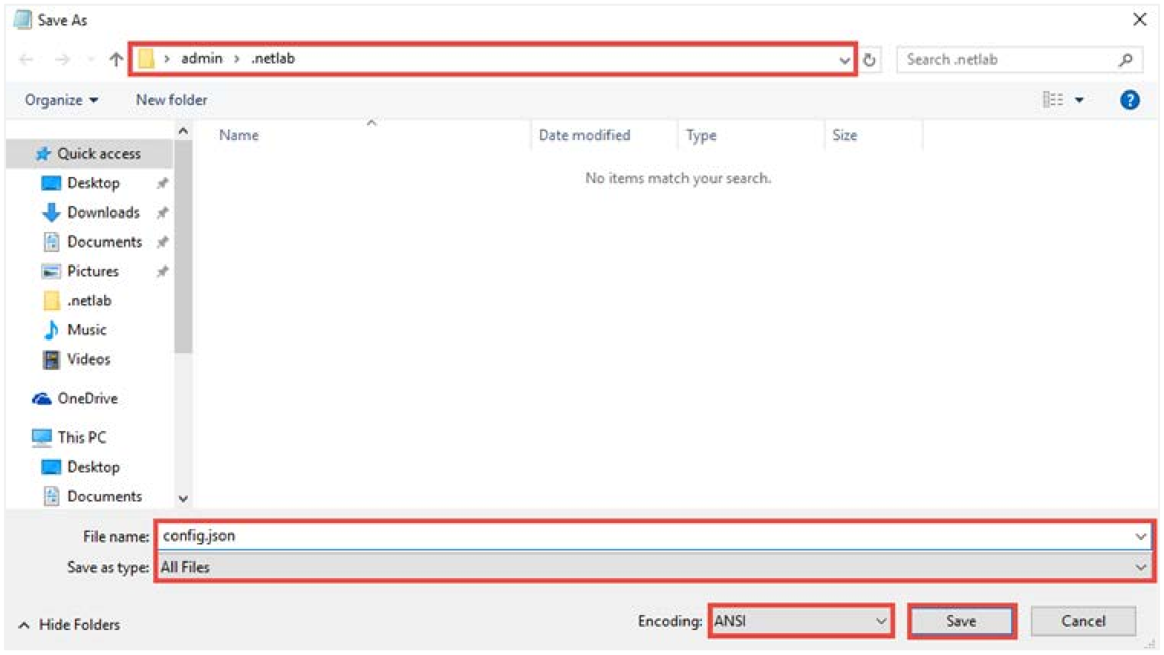


```
{
    "default": {
        "host": "[redacted]",
        "timeout": 10,
        "token": "[redacted]",
        "user": "administrator"
    }
}
```

It is recommended to first validate the *JSON* configuration by going to a *JSON* validator website such as jsonlint.com. Simply copy/paste the entire configuration into the validator and run the script. If it comes back with "*Valid JSON*", then proceed by copying it from the validator script and pasting it into *Notepad*.

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.

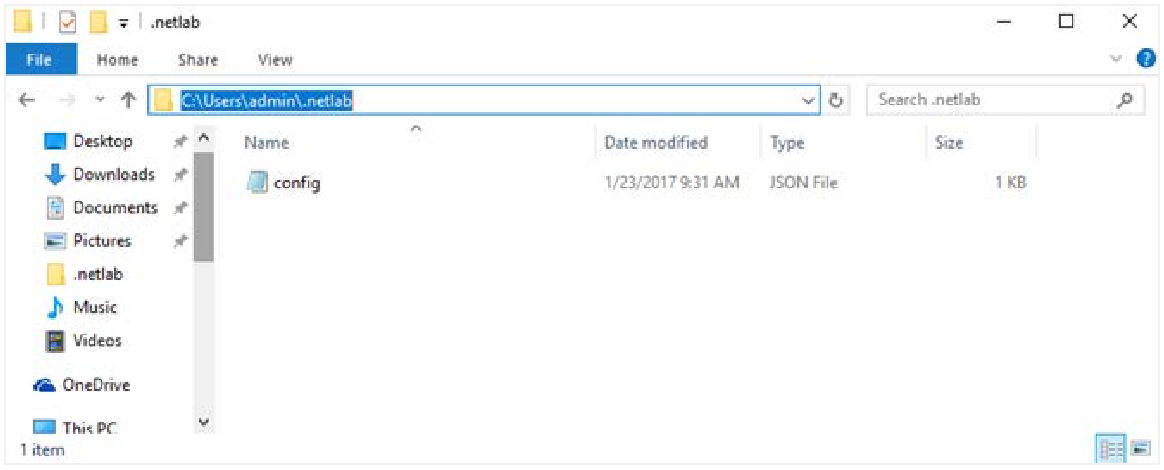
7. Type **`config.json`** in the *File name* field.
8. Select **All Files** as the *Save as type*.
9. Select **ANSI** as the *encoding* type and click **Save**.



No items match your search.

File name: config.json
Save as type: All Files
Encoding: ANSI

10. Confirm that the file successfully saved with *JSON* appearing under the *Type* column.



C:\Users\admin\.netlab

Name	Date modified	Type	Size
config	1/23/2017 9:31 AM	JSON File	1 KB

4 Issuing API Calls

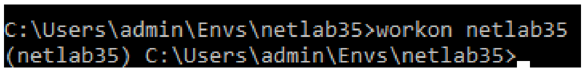
This section will help provide guidance on how to initiate the *API* call with a *NETLAB+ VE* system as well as how to run Python scripts against the system to automate tedious tasks.

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
```

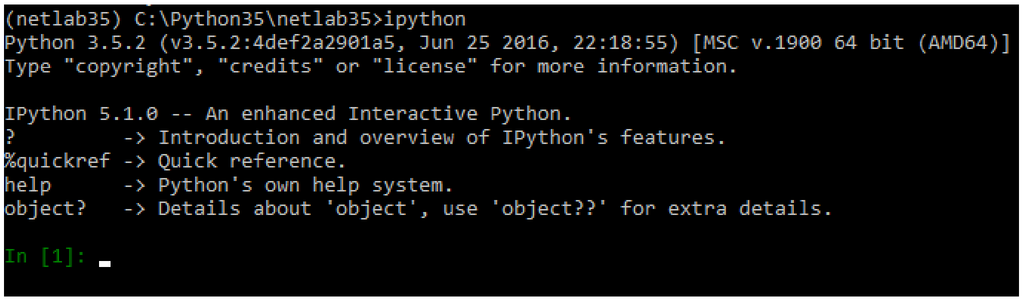


```
C:\Users\admin\Envs\netlab35>workon netlab35
(netlab35) C:\Users\admin\Envs\netlab35>
```

The *netlab35* can be replaced with a user specific version dependent on the versioning of the project that is being worked on.

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

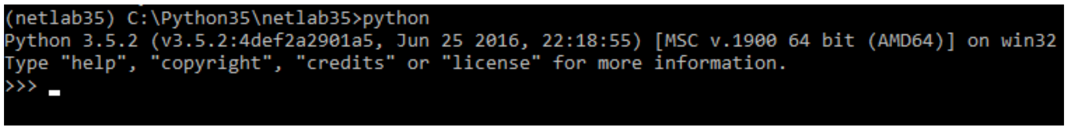
```
ipython
```



```
(netlab35) C:\Python35\netlab35>ipython
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:18:55) [MSC v.1900 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 5.1.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]:
```

To launch regular **Python**, issue the command below.

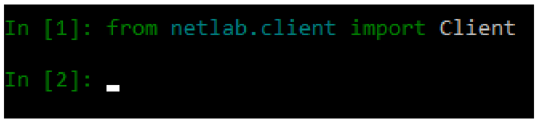
```
python
```



```
(netlab35) C:\Python35\netlab35>python
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:18:55) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

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.

```
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

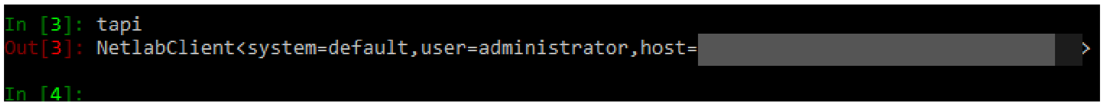
5. Instantiate the *Client* and assign it to a variable named *tapi*. A different variable name can be used here such as *client = Client()* or *api = Client()*. This will load the default configuration and attempt to connect to the remote API.

```
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.

```
tapi
```



```
In [3]: tapi
Out[3]: NetlabClient<system=default,user=administrator,host=[redacted]>

In [4]:
```

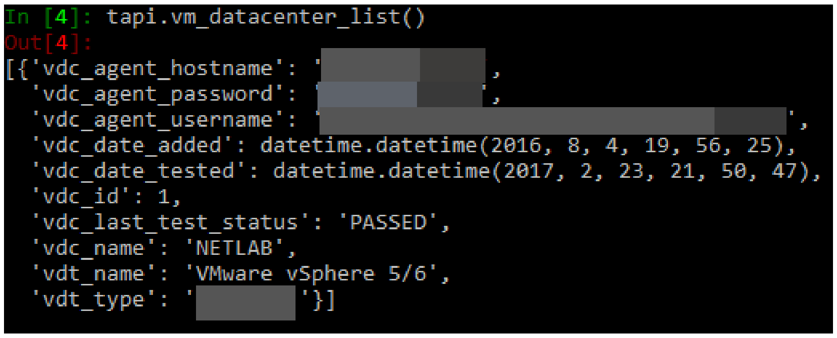
4.2 Informative NETLAB+ API Methods

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.

```
tapi.vm_datacenter_list()
```

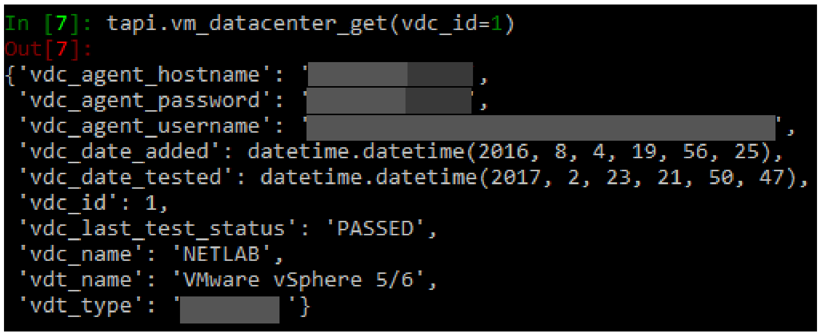


```
In [4]: tapi.vm_datacenter_list()
Out[4]:
[{'vdc_agent_hostname': '[redacted]',
  'vdc_agent_password': '[redacted]',
  'vdc_agent_username': '[redacted]',
  'vdc_date_added': datetime.datetime(2016, 8, 4, 19, 56, 25),
  'vdc_date_tested': datetime.datetime(2017, 2, 23, 21, 50, 47),
  'vdc_id': 1,
  'vdc_last_test_status': 'PASSED',
  'vdc_name': 'NETLAB',
  'vdt_name': 'VMware vSphere 5/6',
  'vdt_type': '[redacted]'}]
```

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.

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

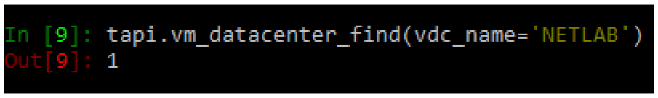


```
In [7]: tapi.vm_datacenter_get(vdc_id=1)
Out[7]:
{'vdc_agent_hostname': '[redacted]',
 'vdc_agent_password': '[redacted]',
 'vdc_agent_username': '[redacted]',
 'vdc_date_added': datetime.datetime(2016, 8, 4, 19, 56, 25),
 'vdc_date_tested': datetime.datetime(2017, 2, 23, 21, 50, 47),
 'vdc_id': 1,
 'vdc_last_test_status': 'PASSED',
 'vdc_name': 'NETLAB',
 'vdt_name': 'VMware vSphere 5/6',
 'vdt_type': '[redacted]'}
```

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

3. Another way to retrieve the *vdc_id* property is by calling the *vm_datacenter_find* method below.

```
tapi.vm_datacenter_find(vdc_name='NETLAB')
```

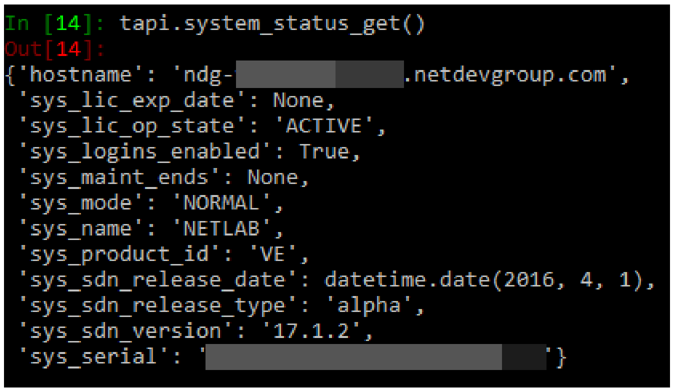


```
In [9]: tapi.vm_datacenter_find(vdc_name='NETLAB')
Out[9]: 1
```

The name of the datacenter must be known to retrieve the *vdc_id* using this method.

4. More information can be pulled about a *NETLAB+* system by calling the method below.

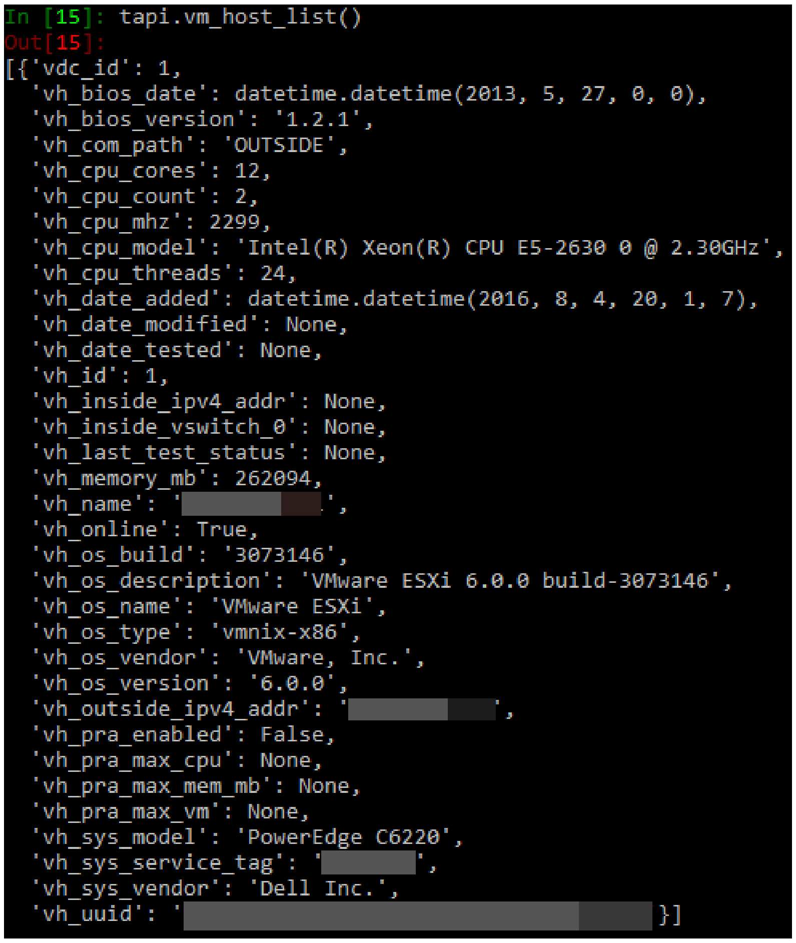
```
tapi.system_status_get()
```



```
In [14]: tapi.system_status_get()
Out[14]:
{'hostname': 'ndg-[redacted].netdevgroup.com',
 'sys_lic_exp_date': None,
 'sys_lic_op_state': 'ACTIVE',
 'sys_logins_enabled': True,
 'sys_maint_ends': None,
 'sys_mode': 'NORMAL',
 'sys_name': 'NETLAB',
 'sys_product_id': 'VE',
 'sys_sdn_release_date': datetime.date(2016, 4, 1),
 'sys_sdn_release_type': 'alpha',
 'sys_sdn_version': '17.1.2',
 'sys_serial': '[redacted]'}
```

5. To list all hosts associated with a *NETLAB+* system and their respective host information, the method below can be used.

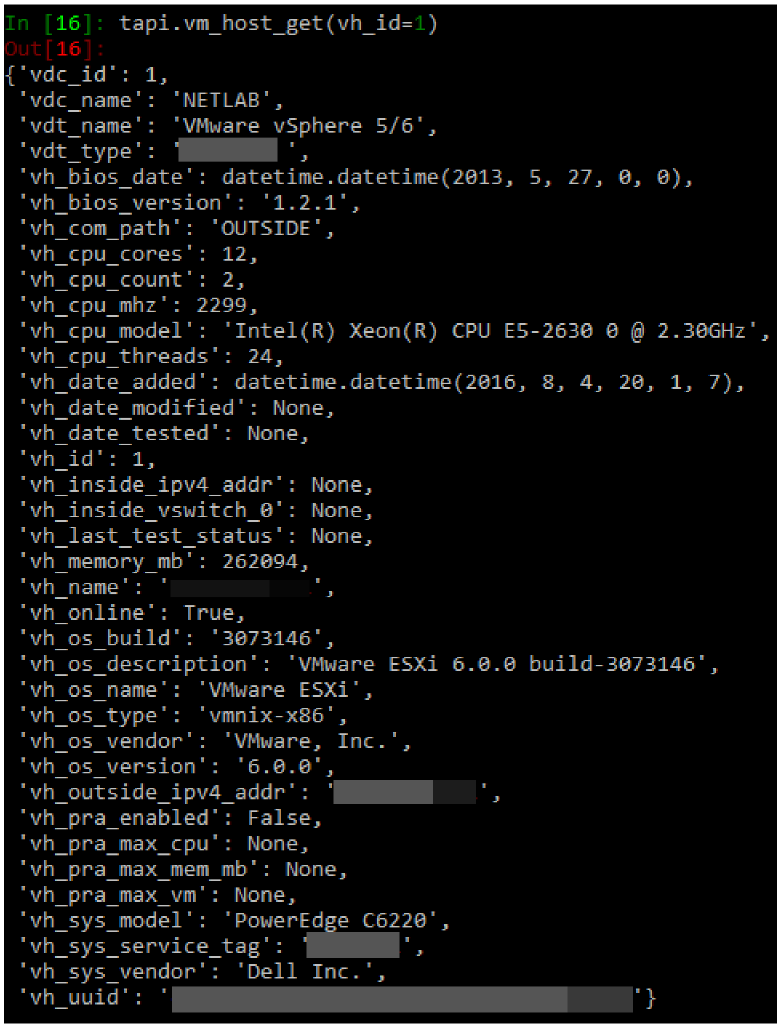
```
tapi.vm_host_list()
```



```
In [15]: tapi.vm_host_list()
Out[15]:
[{'vdc_id': 1,
  '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': 2299,
  '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': '[redacted]',
  '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': 'vmnix-x86',
  'vh_os_vendor': 'VMware, Inc.',
  'vh_os_version': '6.0.0',
  'vh_outside_ipv4_addr': '[redacted]',
  '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': '[redacted]',
  'vh_sys_vendor': 'Dell Inc.',
  'vh_uuid': '[redacted]}]
```

6. To list information pertaining to only a specified host, pass the *vh_id* to the method below.

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



```
In [16]: tapi.vm_host_get(vh_id=1)
Out[16]:
{'vdc_id': 1,
 'vdc_name': 'NETLAB',
 'vdt_name': 'VMware vSphere 5/6',
 'vdt_type': '[redacted]',
 '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': 2299,
 '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': '[redacted]',
 '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': 'vmnix-x86',
 'vh_os_vendor': 'VMware, Inc.',
 'vh_os_version': '6.0.0',
 'vh_outside_ipv4_addr': '[redacted]',
 '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': '[redacted]',
 'vh_sys_vendor': 'Dell Inc.',
 'vh_uuid': '[redacted]'}
```

The *vh_id* was identified in the previous method where a list was generated for all hosts.

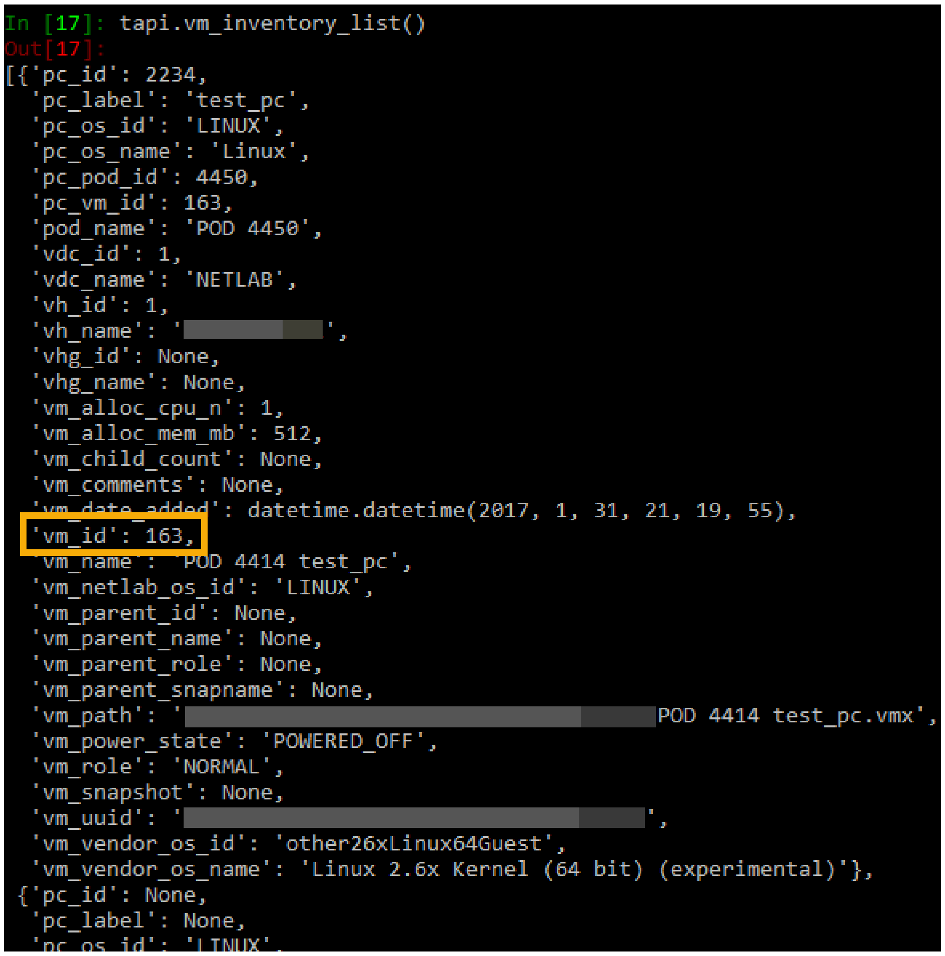
4.2.2 NETLAB+ API Methods

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.

```
tapi.vm_inventory_list()
```

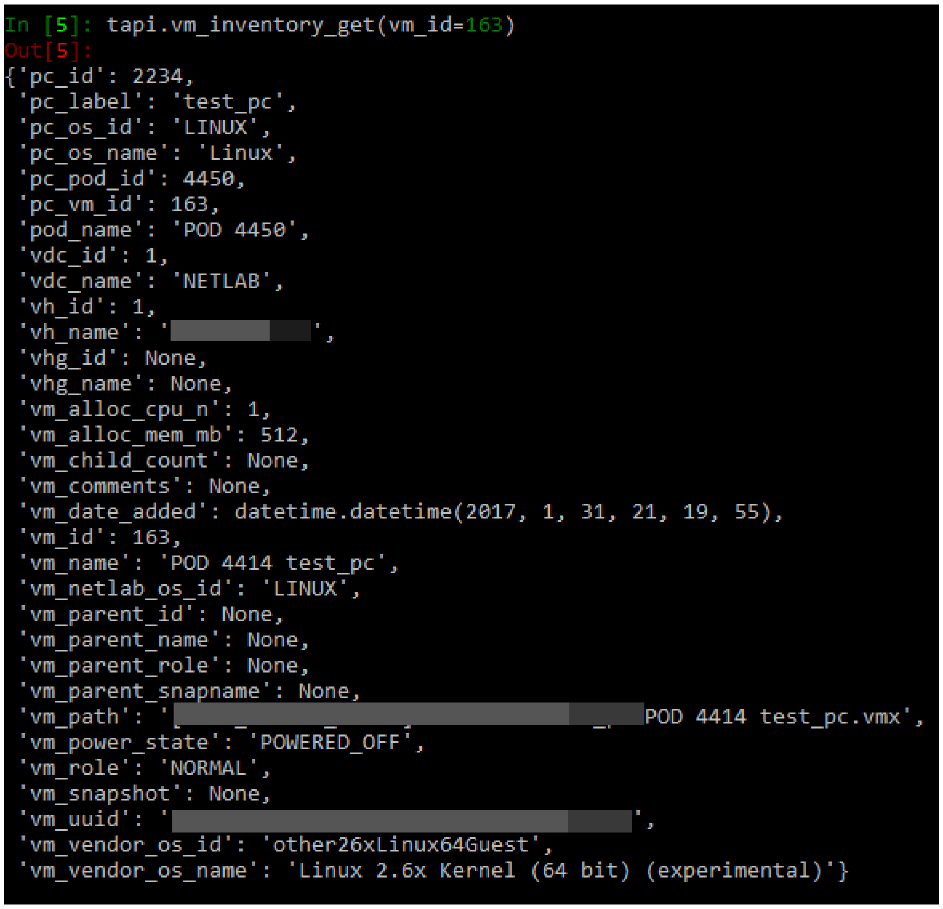


```
In [17]: tapi.vm_inventory_list()
Out[17]:
[{'pc_id': 2234,
  'pc_label': 'test_pc',
  'pc_os_id': 'LINUX',
  'pc_os_name': 'Linux',
  'pc_pod_id': 4450,
  'pc_vm_id': 163,
  'pod_name': 'POD 4450',
  'vdc_id': 1,
  'vdc_name': 'NETLAB',
  'vh_id': 1,
  'vh_name': '[redacted]',
  'vhg_id': None,
  'vhg_name': None,
  'vm_alloc_cpu_n': 1,
  'vm_alloc_mem_mb': 512,
  'vm_child_count': None,
  'vm_comments': None,
  'vm_date_added': datetime.datetime(2017, 1, 31, 21, 19, 55),
  'vm_id': 163,
  'vm_name': 'POD 4414 test_pc',
  'vm_netlab_os_id': 'LINUX',
  'vm_parent_id': None,
  'vm_parent_name': None,
  'vm_parent_role': None,
  'vm_parent_snapname': None,
  'vm_path': '[redacted]POD 4414 test_pc.vmx',
  'vm_power_state': 'POWERED_OFF',
  'vm_role': 'NORMAL',
  'vm_snapshot': None,
  'vm_uuid': '[redacted]',
  'vm_vendor_os_id': 'other26xLinux64Guest',
  'vm_vendor_os_name': 'Linux 2.6x Kernel (64 bit) (experimental)'},
 {'pc_id': None,
  'pc_label': None,
  'pc_os_id': 'LINUX',
```

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.

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



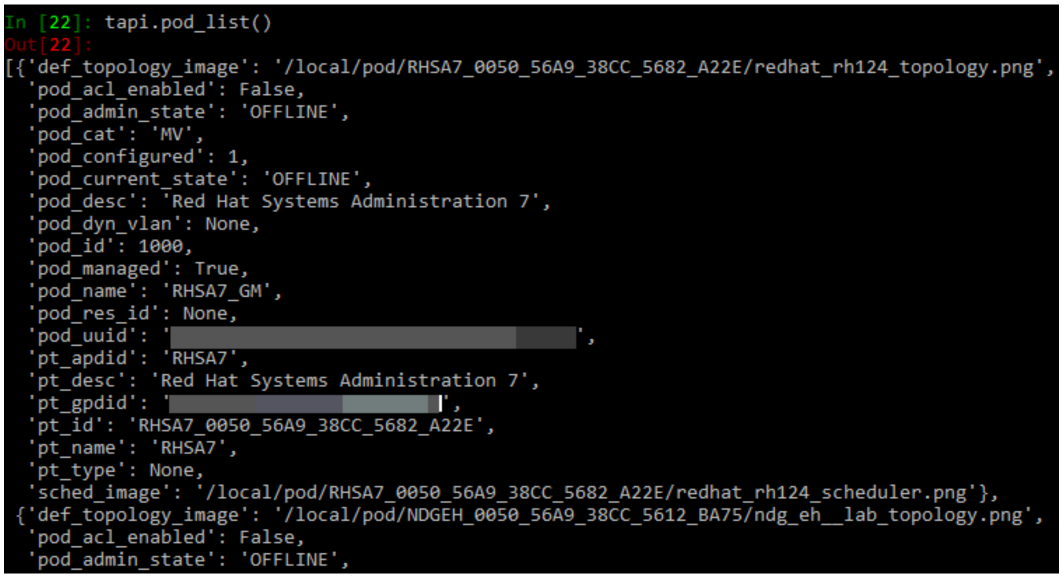
```
In [5]: tapi.vm_inventory_get(vm_id=163)
Out[5]:
{'pc_id': 2234,
 'pc_label': 'test_pc',
 'pc_os_id': 'LINUX',
 'pc_os_name': 'Linux',
 'pc_pod_id': 4450,
 'pc_vm_id': 163,
 'pod_name': 'POD 4450',
 'vdc_id': 1,
 'vdc_name': 'NETLAB',
 'vh_id': 1,
 'vh_name': '[redacted]',
 'vhg_id': None,
 'vhg_name': None,
 'vm_alloc_cpu_n': 1,
 'vm_alloc_mem_mb': 512,
 'vm_child_count': None,
 'vm_comments': None,
 'vm_date_added': datetime.datetime(2017, 1, 31, 21, 19, 55),
 'vm_id': 163,
 'vm_name': 'POD 4414 test_pc',
 'vm_netlab_os_id': 'LINUX',
 'vm_parent_id': None,
 'vm_parent_name': None,
 'vm_parent_role': None,
 'vm_parent_snapname': None,
 'vm_path': '[redacted]POD 4414 test_pc.vmx',
 'vm_power_state': 'POWERED_OFF',
 'vm_role': 'NORMAL',
 'vm_snapshot': None,
 'vm_uuid': '[redacted]',
 'vm_vendor_os_id': 'other26xLinux64Guest',
 'vm_vendor_os_name': 'Linux 2.6x Kernel (64 bit) (experimental)'}
```

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.

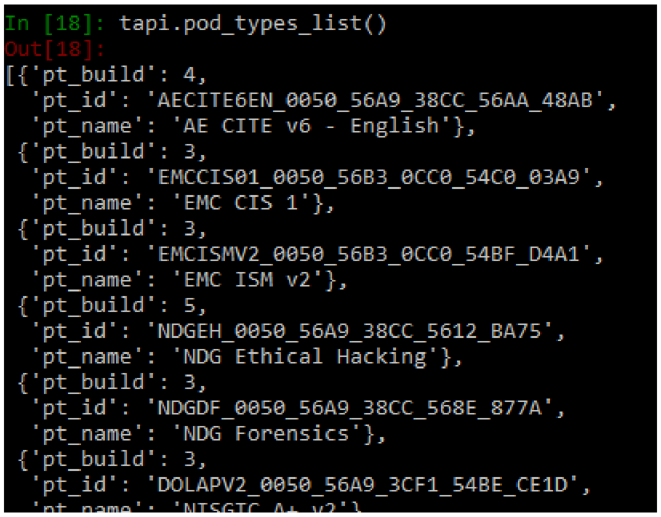
```
tapi.pod_list()
```



```
In [22]: tapi.pod_list()
Out[22]:
[{'def_topology_image': '/local/pod/RHSA7_0050_56A9_38CC_5682_A22E/redhat_rh124_topology.png',
  'pod_acl_enabled': False,
  'pod_admin_state': 'OFFLINE',
  'pod_cat': 'MV',
  'pod_configured': 1,
  'pod_current_state': 'OFFLINE',
  'pod_desc': 'Red Hat Systems Administration 7',
  'pod_dyn_vlan': None,
  'pod_id': 1000,
  'pod_managed': True,
  'pod_name': 'RHSA7_GM',
  'pod_res_id': None,
  'pod_uuid': '[redacted]',
  'pt_apdid': 'RHSA7',
  'pt_desc': 'Red Hat Systems Administration 7',
  'pt_gpdid': '[redacted]',
  'pt_id': 'RHSA7_0050_56A9_38CC_5682_A22E',
  'pt_name': 'RHSA7',
  'pt_type': None,
  'sched_image': '/local/pod/RHSA7_0050_56A9_38CC_5682_A22E/redhat_rh124_scheduler.png'},
 {'def_topology_image': '/local/pod/NDGEH_0050_56A9_38CC_5612_BA75/ndg_eh__lab_topology.png',
  'pod_acl_enabled': False,
  'pod_admin_state': 'OFFLINE',
```

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

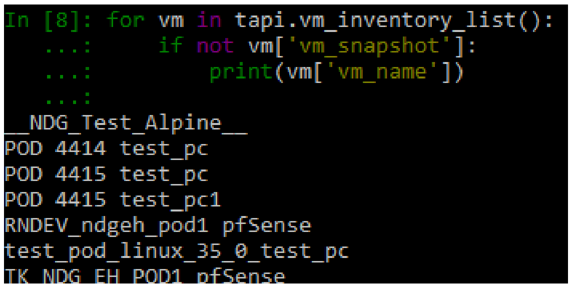
```
tapi.pod_types_list()
```



```
In [18]: tapi.pod_types_list()
Out[18]:
[{'pt_build': 4,
  'pt_id': 'AECITE6EN_0050_56A9_38CC_56AA_48AB',
  'pt_name': 'AE CITE v6 - English'},
 {'pt_build': 3,
  'pt_id': 'EMCCIS01_0050_56B3_0CC0_54C0_03A9',
  'pt_name': 'EMC CIS 1'},
 {'pt_build': 3,
  'pt_id': 'EMCISMV2_0050_56B3_0CC0_54BF_D4A1',
  'pt_name': 'EMC ISM v2'},
 {'pt_build': 5,
  'pt_id': 'NDGEH_0050_56A9_38CC_5612_BA75',
  'pt_name': 'NDG Ethical Hacking'},
 {'pt_build': 3,
  'pt_id': 'NDGDF_0050_56A9_38CC_568E_877A',
  'pt_name': 'NDG Forensics'},
 {'pt_build': 3,
  'pt_id': 'DOLAPV2_0050_56A9_3CF1_54BE_CE1D',
  'pt_name': 'NISGIC A+ v2'},
```

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.

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



```
In [8]: for vm in tapi.vm_inventory_list():
   ...:     if not vm['vm_snapshot']:
   ...:         print(vm['vm_name'])
   ...:
__NDG_Test_Alpine__
POD 4414 test_pc
POD 4415 test_pc
POD 4415 test_pc1
RNDEV_ndgeh_pod1 pfSense
test_pod_linux_35_0_test_pc
TK NDG EH POD1 pfSense
```

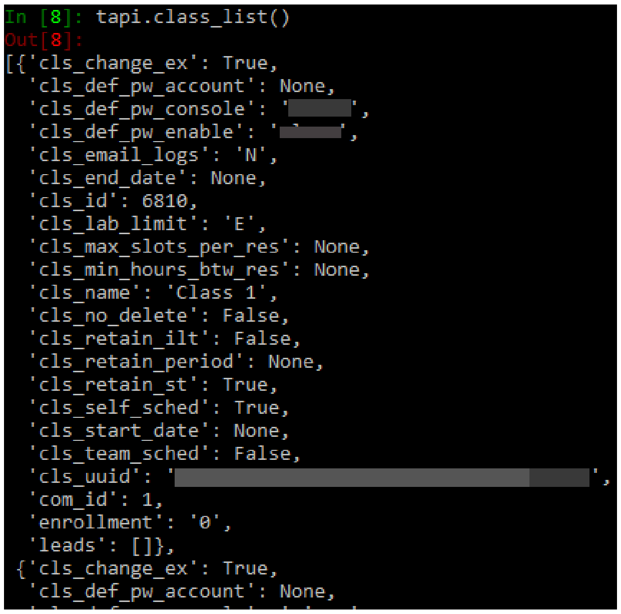
Notice a list of VMs that have a snapshot in place is outputted onto the screen.

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.

```
tapi.class_list()
```

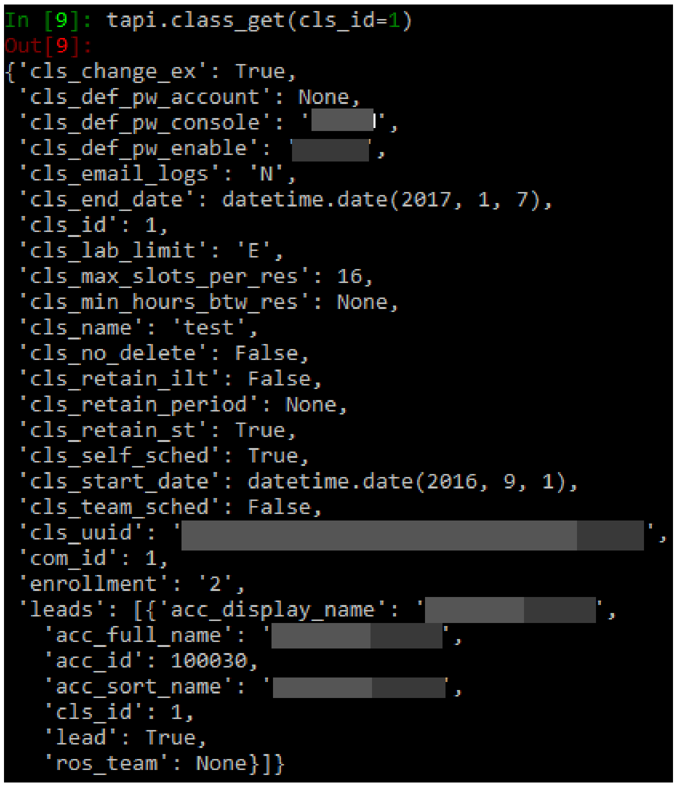


```
In [8]: tapi.class_list()
Out[8]:
[{'cls_change_ex': True,
  'cls_def_pw_account': None,
  'cls_def_pw_console': '[redacted]',
  'cls_def_pw_enable': '[redacted]',
  'cls_email_logs': 'N',
  'cls_end_date': None,
  'cls_id': 6810,
  'cls_lab_limit': 'E',
  'cls_max_slots_per_res': None,
  'cls_min_hours_btw_res': None,
  'cls_name': 'Class 1',
  'cls_no_delete': False,
  'cls_retain_ilt': False,
  'cls_retain_period': None,
  'cls_retain_st': True,
  'cls_self_sched': True,
  'cls_start_date': None,
  'cls_team_sched': False,
  'cls_uuid': '[redacted]',
  'com_id': 1,
  'enrollment': '0',
  'leads': []},
 {'cls_change_ex': True,
  'cls_def_pw_account': None,
```

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.

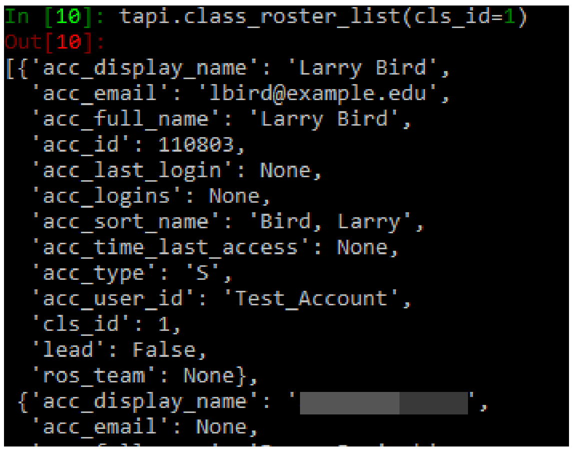
```
tapi.class_get(cls_id=1)
```



```
In [9]: tapi.class_get(cls_id=1)
Out[9]:
{'cls_change_ex': True,
 'cls_def_pw_account': None,
 'cls_def_pw_console': '[redacted]',
 'cls_def_pw_enable': [redacted],
 'cls_email_logs': 'N',
 'cls_end_date': datetime.date(2017, 1, 7),
 'cls_id': 1,
 'cls_lab_limit': 'E',
 'cls_max_slots_per_res': 16,
 'cls_min_hours_btw_res': None,
 'cls_name': 'test',
 'cls_no_delete': False,
 'cls_retain_ilt': False,
 'cls_retain_period': None,
 'cls_retain_st': True,
 'cls_self_sched': True,
 'cls_start_date': datetime.date(2016, 9, 1),
 'cls_team_sched': False,
 'cls_uuid': '[redacted]',
 'com_id': 1,
 'enrollment': '2',
 'leads': [{'acc_display_name': '[redacted]',
   'acc_full_name': '[redacted]',
   'acc_id': 100030,
   'acc_sort_name': '[redacted]',
   'cls_id': 1,
   'lead': True,
   'ros_team': None}]}
```

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

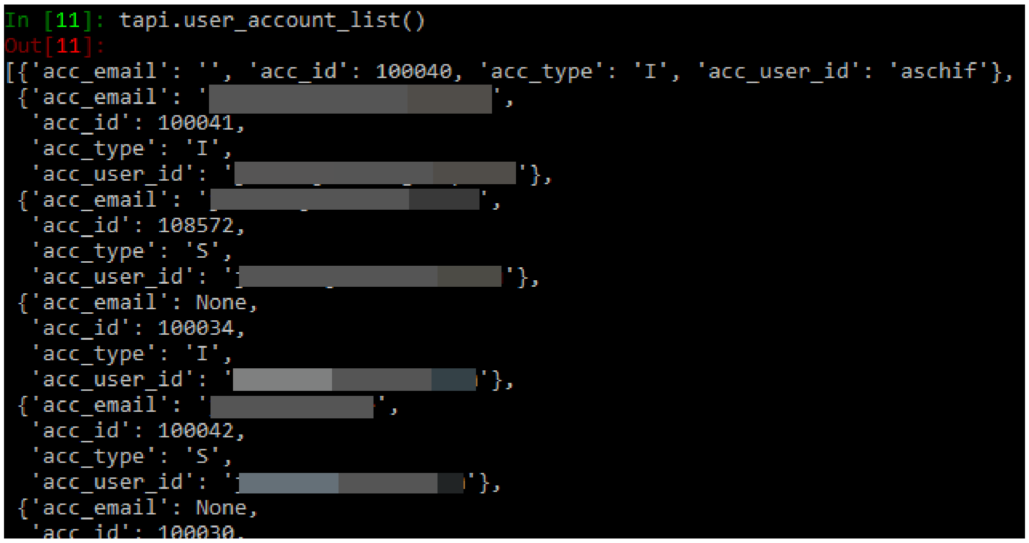
```
tapi.class_roster_list(cls_id=1)
```



```
In [10]: tapi.class_roster_list(cls_id=1)
Out[10]:
[{'acc_display_name': 'Larry Bird',
  'acc_email': 'lbird@example.edu',
  'acc_full_name': 'Larry Bird',
  'acc_id': 110803,
  'acc_last_login': None,
  'acc_logins': None,
  'acc_sort_name': 'Bird, Larry',
  'acc_time_last_access': None,
  'acc_type': 'S',
  'acc_user_id': 'Test_Account',
  'cls_id': 1,
  'lead': False,
  'ros_team': None},
 {'acc_display_name': '[redacted]',
  'acc_email': None,
```

4. To retrieve a list of all accounts on a *NETLAB+* system, call the method below.

```
tapi.user_account_list()
```

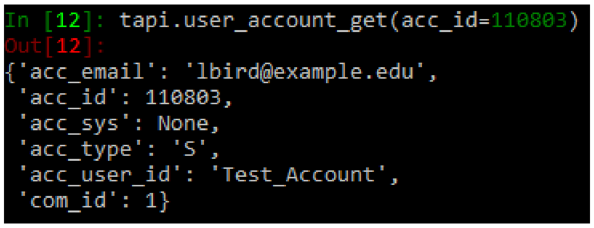


```
In [11]: tapi.user_account_list()
Out[11]:
[{'acc_email': '', 'acc_id': 100040, 'acc_type': 'I', 'acc_user_id': 'aschif'},
 {'acc_email': '[redacted]',
  'acc_id': 100041,
  'acc_type': 'I',
  'acc_user_id': '[redacted]'},
 {'acc_email': '[redacted]',
  'acc_id': 108572,
  'acc_type': 'S',
  'acc_user_id': '[redacted]'},
 {'acc_email': None,
  'acc_id': 100034,
  'acc_type': 'I',
  'acc_user_id': '[redacted]'},
 {'acc_email': '[redacted]',
  'acc_id': 100042,
  'acc_type': 'S',
  'acc_user_id': '[redacted]'},
 {'acc_email': None,
  'acc_id': 100030,
```

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.

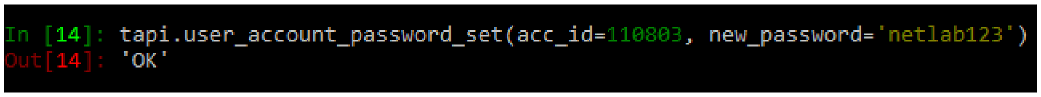
```
tapi.user_account_get(acc_id=110803)
```



```
In [12]: tapi.user_account_get(acc_id=110803)
Out[12]:
{'acc_email': 'lbird@example.edu',
 'acc_id': 110803,
 'acc_sys': None,
 'acc_type': 'S',
 'acc_user_id': 'Test_Account',
 'com_id': 1}
```

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

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

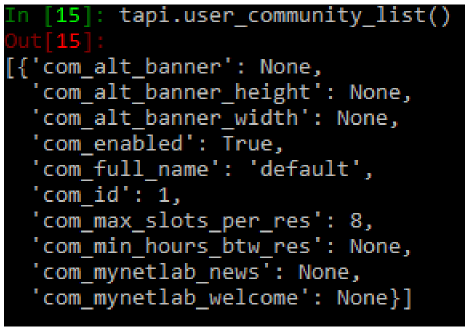


```
In [14]: tapi.user_account_password_set(acc_id=110803, new_password='netlab123')
Out[14]: 'OK'
```

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.

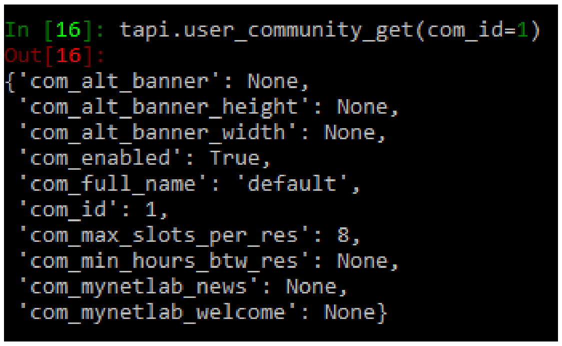
```
tapi.user_community_list()
```



```
In [15]: tapi.user_community_list()
Out[15]:
[{'com_alt_banner': None,
  'com_alt_banner_height': None,
  'com_alt_banner_width': None,
  'com_enabled': True,
  'com_full_name': 'default',
  'com_id': 1,
  'com_max_slots_per_res': 8,
  'com_min_hours_btw_res': None,
  'com_mynetlab_news': None,
  'com_mynetlab_welcome': None}]
```

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

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



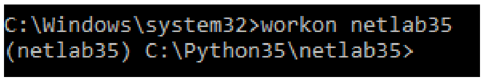
```
In [16]: tapi.user_community_get(com_id=1)
Out[16]:
{'com_alt_banner': None,
 'com_alt_banner_height': None,
 'com_alt_banner_width': None,
 'com_enabled': True,
 'com_full_name': 'default',
 'com_id': 1,
 'com_max_slots_per_res': 8,
 'com_min_hours_btw_res': None,
 'com_mynetlab_news': None,
 'com_mynetlab_welcome': None}
```

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.

```
workon netlab35
```



```
C:\Windows\system32>workon netlab35
(netlab35) C:\Python35\netlab35>
```

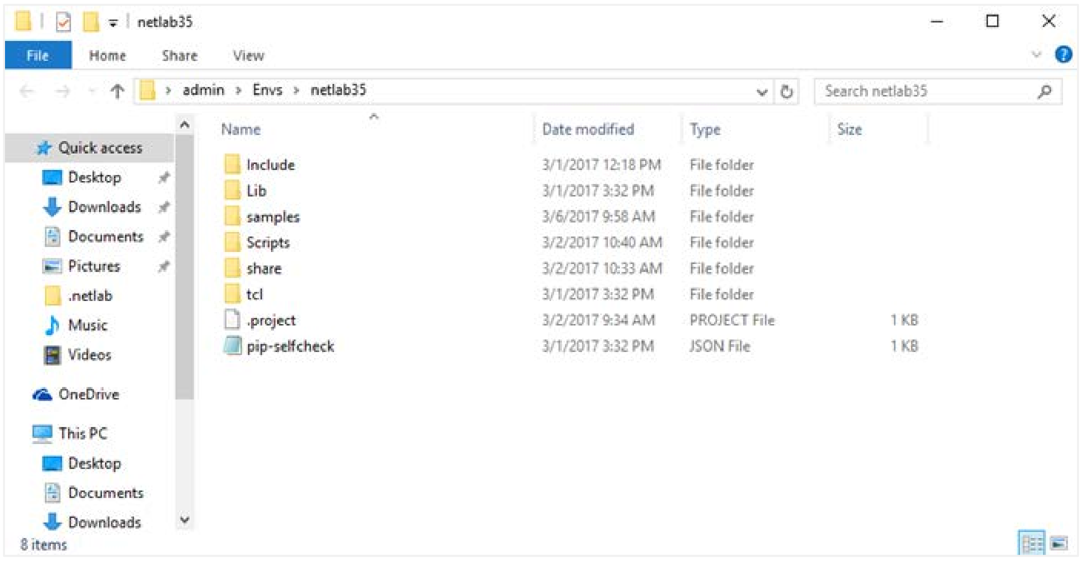
The *netlab35* can be replaced with a user specified project name dependent on the versioning of the project that is being worked on.

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
```

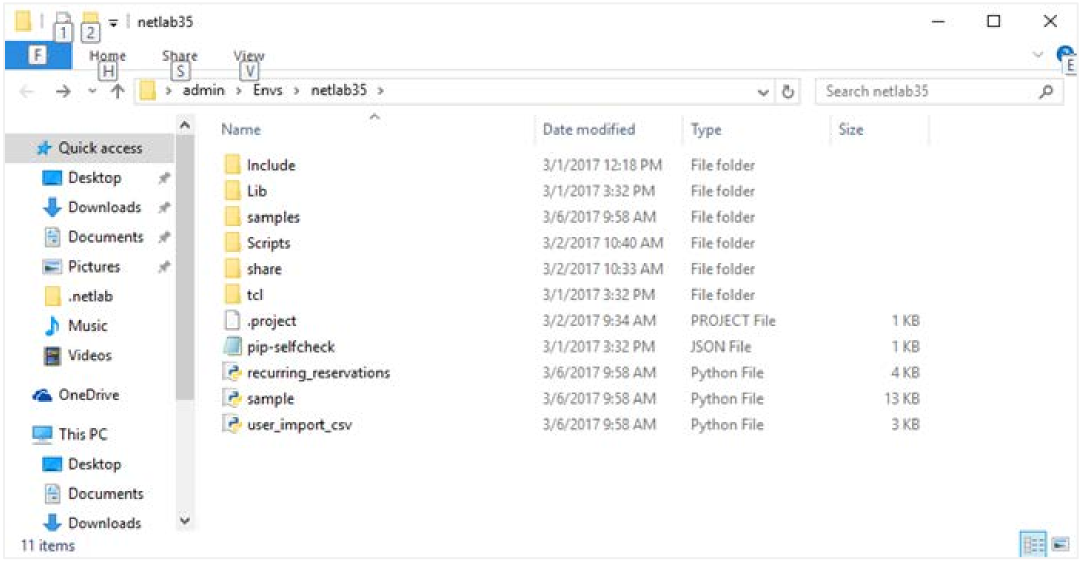
2. Once downloaded, extract the **netlab-samples-X.X.**X archive into the project directory that was configured in *Section 2.3* of this guide.



Name	Date modified	Type	Size
Include	3/1/2017 12:18 PM	File folder	
Lib	3/1/2017 3:32 PM	File folder	
samples	3/6/2017 9:58 AM	File folder	
Scripts	3/2/2017 10:40 AM	File folder	
share	3/2/2017 10:33 AM	File folder	
tcl	3/1/2017 3:32 PM	File folder	
.project	3/2/2017 9:34 AM	PROJECT File	1 KB
pip-selfcheck	3/1/2017 3:32 PM	JSON File	1 KB

For this example, the project directory was configured to *C:\Users\admin\Envs\netlab35*.

3. For easier accessibility, move all file contents from the *samples* subdirectory into the root directory of the specified project directory.



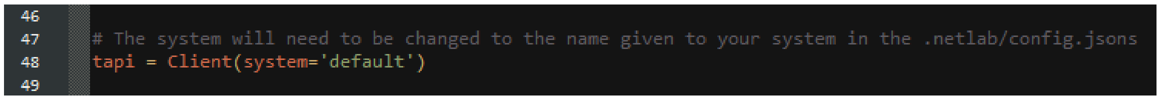
Name	Date modified	Type	Size
Include	3/1/2017 12:18 PM	File folder	
Lib	3/1/2017 3:32 PM	File folder	
samples	3/6/2017 9:58 AM	File folder	
Scripts	3/2/2017 10:40 AM	File folder	
share	3/2/2017 10:33 AM	File folder	
tcl	3/1/2017 3:32 PM	File folder	
.project	3/2/2017 9:34 AM	PROJECT File	1 KB
pip-selfcheck	3/1/2017 3:32 PM	JSON File	1 KB
recurring_reservations	3/6/2017 9:58 AM	Python File	4 KB
sample	3/6/2017 9:58 AM	Python File	13 KB
user_import_csv	3/6/2017 9:58 AM	Python File	3 KB

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

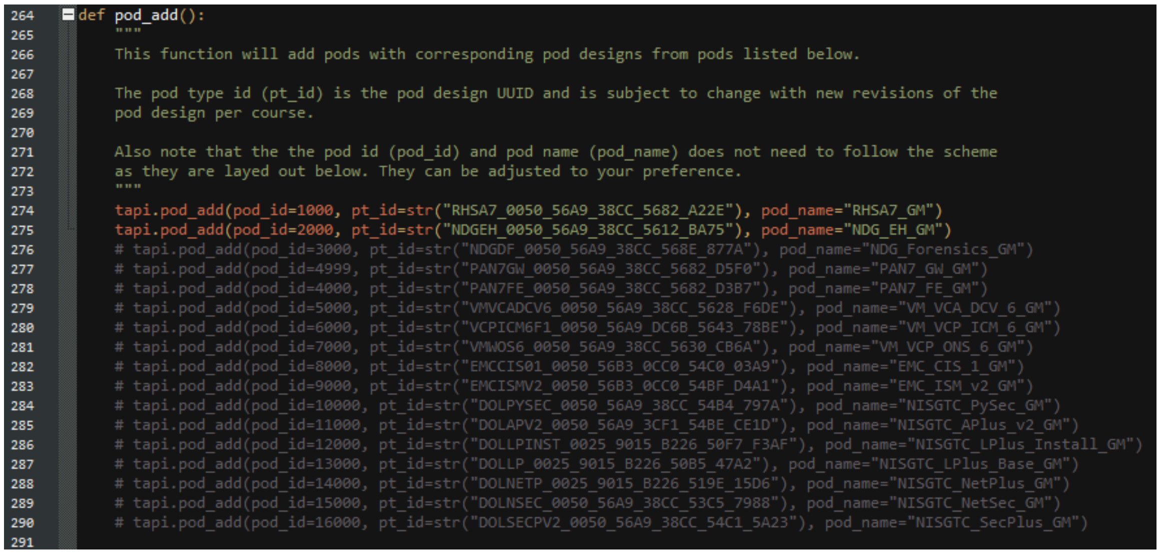
1. Navigate to where the **sample.py** file is saved and edit the file using a text editor.
2. Change the client system configuration to the appropriate value. In this example, the default config is used.



```

# The system will need to be changed to the name given to your system in the .netlab/config.jsons
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*.

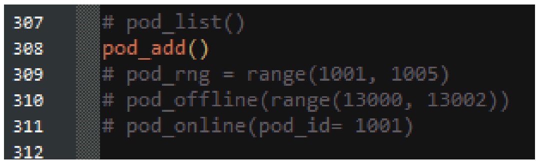
3. Scroll down to the **pod_add** method and uncomment any lines that you wish to use, depending on the type(s) of master pods.



```
def pod_add():
    """
    This function will add pods with corresponding pod designs from pods listed below.

    The pod type id (pt_id) is the pod design UUID and is subject to change with new revisions of the
    pod design per course.

    Also note that the the pod id (pod_id) and pod name (pod_name) does not need to follow the scheme
    as they are layed out below. They can be adjusted to your preference.
    """
    tapi.pod_add(pod_id=1000, pt_id=str("RHSA7_0050_56A9_38CC_5682_A22E"), pod_name="RHSA7_GM")
    tapi.pod_add(pod_id=2000, pt_id=str("NDGEH_0050_56A9_38CC_5612_BA75"), pod_name="NDG_EH_GM")
    # tapi.pod_add(pod_id=3000, pt_id=str("NDGDF_0050_56A9_38CC_568E_877A"), pod_name="NDG_Forensics_GM")
    # tapi.pod_add(pod_id=4999, pt_id=str("PAN7GW_0050_56A9_38CC_5682_D5F0"), pod_name="PAN7_GW_GM")
    # tapi.pod_add(pod_id=4000, pt_id=str("PAN7FE_0050_56A9_38CC_5682_D3B7"), pod_name="PAN7_FE_GM")
    # tapi.pod_add(pod_id=5000, pt_id=str("VMVCADCV6_0050_56A9_38CC_5628_F6DE"), pod_name="VM_VCA_DCV_6_GM")
    # tapi.pod_add(pod_id=6000, pt_id=str("VCPICM6F1_0050_56A9_DC6B_5643_78BE"), pod_name="VM_VCP_ICM_6_GM")
    # tapi.pod_add(pod_id=7000, pt_id=str("VMWOS6_0050_56A9_38CC_5630_CB6A"), pod_name="VM_VCP_ONS_6_GM")
    # tapi.pod_add(pod_id=8000, pt_id=str("EMCCIS01_0050_56B3_0CC0_54C0_03A9"), pod_name="EMC_CIS_1_GM")
    # tapi.pod_add(pod_id=9000, pt_id=str("EMCISMV2_0050_56B3_0CC0_54BF_D4A1"), pod_name="EMC_ISM_v2_GM")
    # tapi.pod_add(pod_id=10000, pt_id=str("DOLPYSEC_0050_56A9_38CC_54B4_797A"), pod_name="NISGTC_PySec_GM")
    # tapi.pod_add(pod_id=11000, pt_id=str("DOLAPV2_0050_56A9_3CF1_54BE_CE1D"), pod_name="NISGTC_APlus_v2_GM")
    # tapi.pod_add(pod_id=12000, pt_id=str("DOLLPINST_0025_9015_B226_50F7_F3AF"), pod_name="NISGTC_LPlus_Install_GM")
    # tapi.pod_add(pod_id=13000, pt_id=str("DOLLP_0025_9015_B226_50B5_47A2"), pod_name="NISGTC_LPlus_Base_GM")
    # tapi.pod_add(pod_id=14000, pt_id=str("DOLNETP_0025_9015_B226_519E_15D6"), pod_name="NISGTC_NetPlus_GM")
    # tapi.pod_add(pod_id=15000, pt_id=str("DOLNSEC_0050_56A9_38CC_53C5_7988"), pod_name="NISGTC_NetSec_GM")
    # tapi.pod_add(pod_id=16000, pt_id=str("DOLSECPV2_0050_56A9_38CC_54C1_5A23"), pod_name="NISGTC_SecPlus_GM")

```

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.



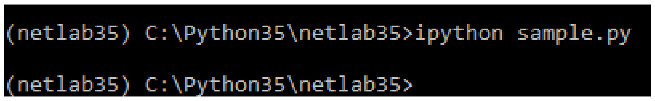
```
# pod_list()
pod_add()
# pod_rng = range(1001, 1005)
# pod_offline(range(13000, 13002))
# pod_online(pod_id= 1001)

```

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.

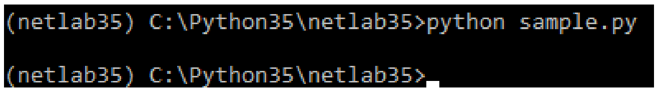
```
ipython sample.py
```



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

(netlab35) C:\Python35\netlab35>
```

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

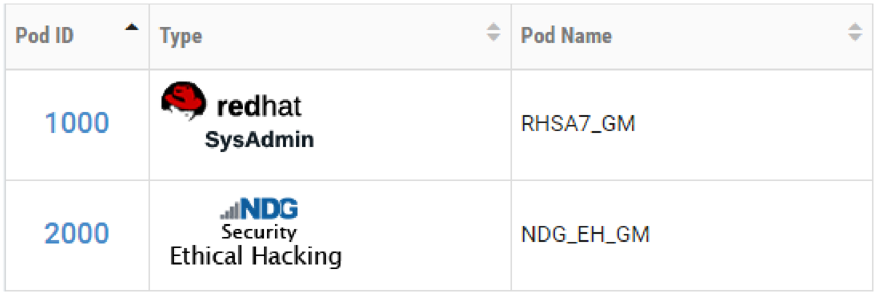
```
python sample.py
```



```
(netlab35) C:\Python35\netlab35>python sample.py

(netlab35) C:\Python35\netlab35>_
```

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.



Pod ID	Type	Pod Name
1000	redhat SysAdmin	RHSA7_GM
2000	NDG Security Ethical Hacking	NDG_EH_GM

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

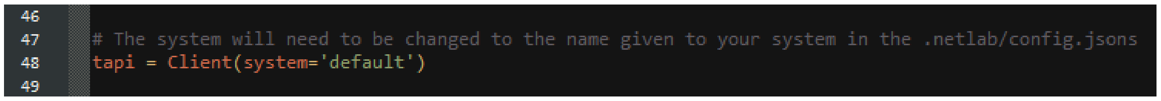
It is recommended to comment back the uncommented lines or any additional lines added to the sample.py file before reusing the script again for a different task. It may be better to save the script as a new file when making changes so that there is always an original available to work from.

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

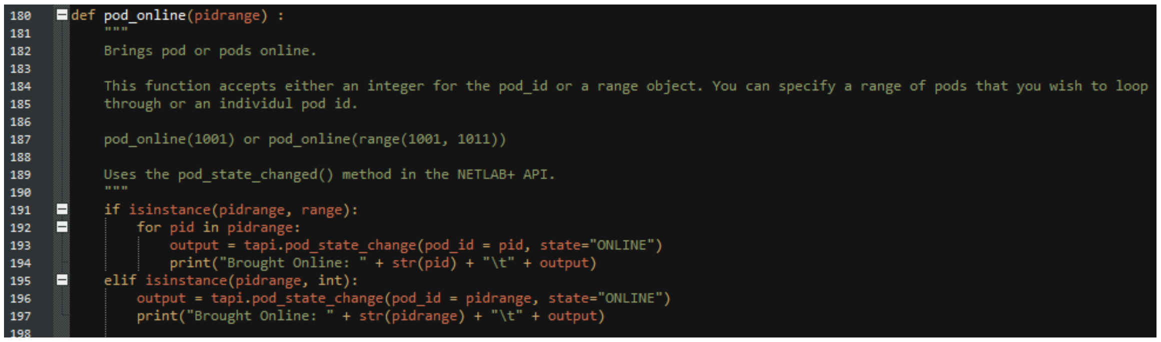
1. Navigate to where the **sample.py** file is saved and edit the file using a text editor.
2. Change the client system configuration to the appropriate value. In this example, the default config is used.



```
# The system will need to be changed to the name given to your system in the .netlab/config.jsons
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.

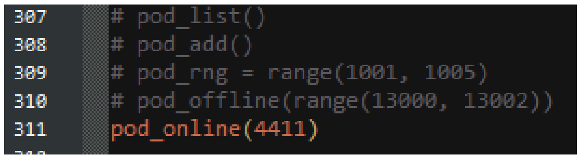
3. Scroll down to the **pod_online** method and take notice of the instructions.



```
def pod_online(pidrange) :
    """
    Brings pod or pods online.

    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 individul pod id.

    pod_online(1001) or pod_online(range(1001, 1011))

    Uses the pod_state_changed() method in the NETLAB+ API.
    """
    if isinstance(pidrange, range):
        for pid in pidrange:
            output = tapi.pod_state_change(pod_id = pid, state="ONLINE")
            print("Brought Online: " + str(pid) + "\t" + output)
    elif isinstance(pidrange, int):
        output = tapi.pod_state_change(pod_id = pidrange, state="ONLINE")
        print("Brought Online: " + str(pidrange) + "\t" + output)

```

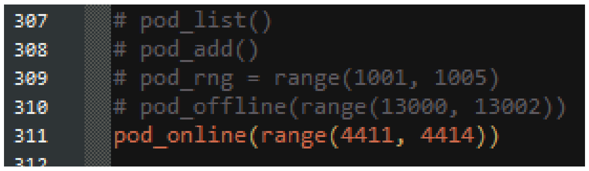
The *pod_online* method can be called upon using two different functions, either by a single pod ID or a range of pod IDs.

4. Scroll all the way to the bottom and add a line that includes **`pod_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.



```
# 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.



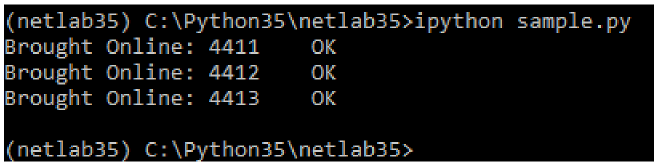
```
# pod_list()
# pod_add()
# pod_rng = range(1001, 1005)
# pod_offline(range(13000, 13002))
pod_online(range(4411, 4414))
```

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.

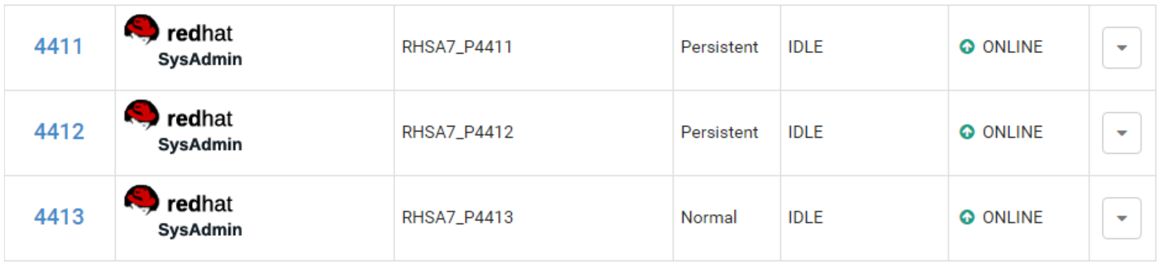
```
ipython sample.py
```



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

(netlab35) C:\Python35\netlab35>
```

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.



4411	redhat SysAdmin	RHSA7_P4411	Persistent	IDLE	ONLINE	▾
4412	redhat SysAdmin	RHSA7_P4412	Persistent	IDLE	ONLINE	▾
4413	redhat SysAdmin	RHSA7_P4413	Normal	IDLE	ONLINE	▾

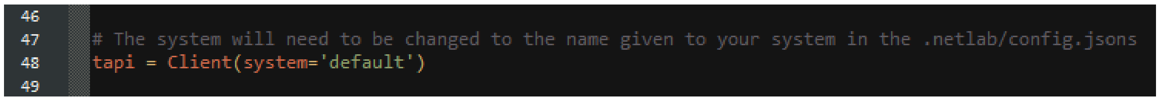
It is recommended to comment back the uncommented lines or any additional lines added to the sample.py file before reusing the script again for a different task. It may be better to save the script as a new file when making changes so that there is always an original available to work from.

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

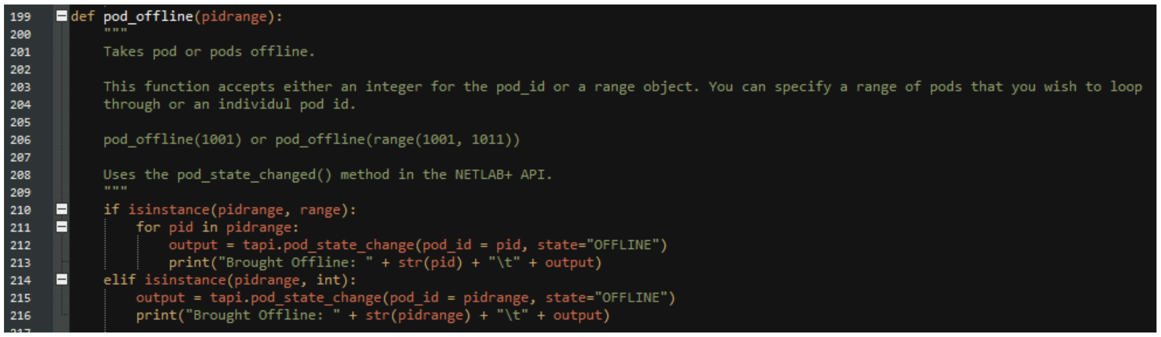
1. Navigate to where the **sample.py** file is saved and edit the file using a text editor.
2. Change the client system configuration to the appropriate value. In this example, the default config is used.



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

This value would reflect the specified --*system* value that was used when configuring the *config.json* file from *Section 4.3*.

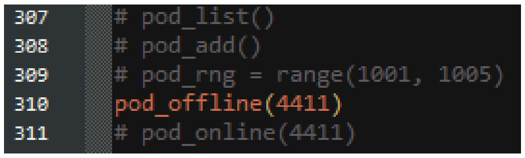
3. Scroll down to the **pod_offline** method and take notice of the instructions.



```
def pod_offline(pidrange):
    """
    Takes pod or pods offline.

    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 individul pod id.

    pod_offline(1001) or pod_offline(range(1001, 1011))

    Uses the pod_state_changed() method in the NETLAB+ API.
    """
    if isinstance(pidrange, range):
        for pid in pidrange:
            output = tapi.pod_state_change(pod_id = pid, state="OFFLINE")
            print("Brought Offline: " + str(pid) + "\t" + output)
    elif isinstance(pidrange, int):
        output = tapi.pod_state_change(pod_id = pidrange, state="OFFLINE")
        print("Brought Offline: " + str(pidrange) + "\t" + output)
```

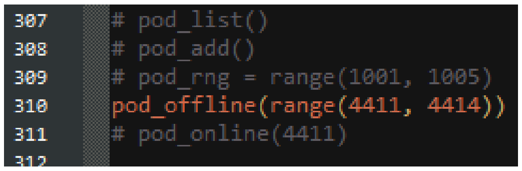
The *pod_offline* method can be called upon using two different functions, either by a single pod ID or a range of pod IDs.

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.



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

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



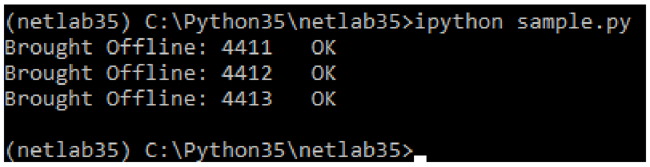
```
# pod_list()
# pod_add()
# pod_rng = range(1001, 1005)
pod_offline(range(4411, 4414))
# pod_online(4411)
```

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.

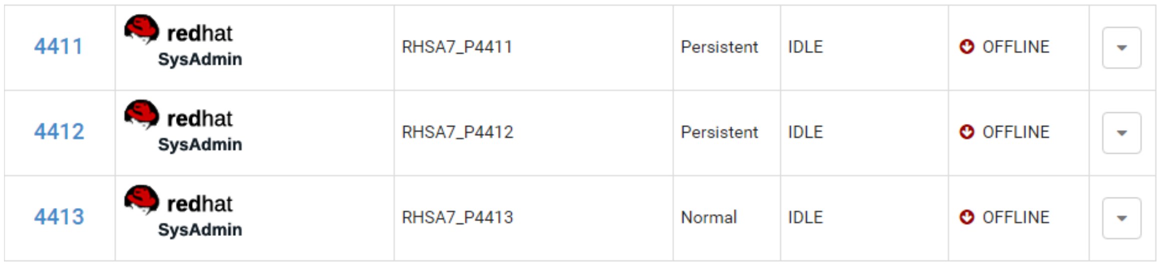
```
ipython sample.py
```



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

(netlab35) C:\Python35\netlab35>
```

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.



4411	redhat SysAdmin	RHSA7_P4411	Persistent	IDLE	OFFLINE	▾
4412	redhat SysAdmin	RHSA7_P4412	Persistent	IDLE	OFFLINE	▾
4413	redhat SysAdmin	RHSA7_P4413	Normal	IDLE	OFFLINE	▾

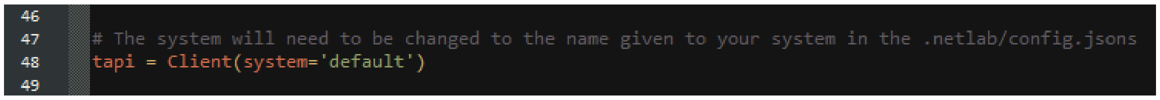
It is recommended to comment back the uncommented lines or any additional lines added to the sample.py file before reusing the script again for a different task. It may be better to save the script as a new file when making changes so that there is always an original available to work from.

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

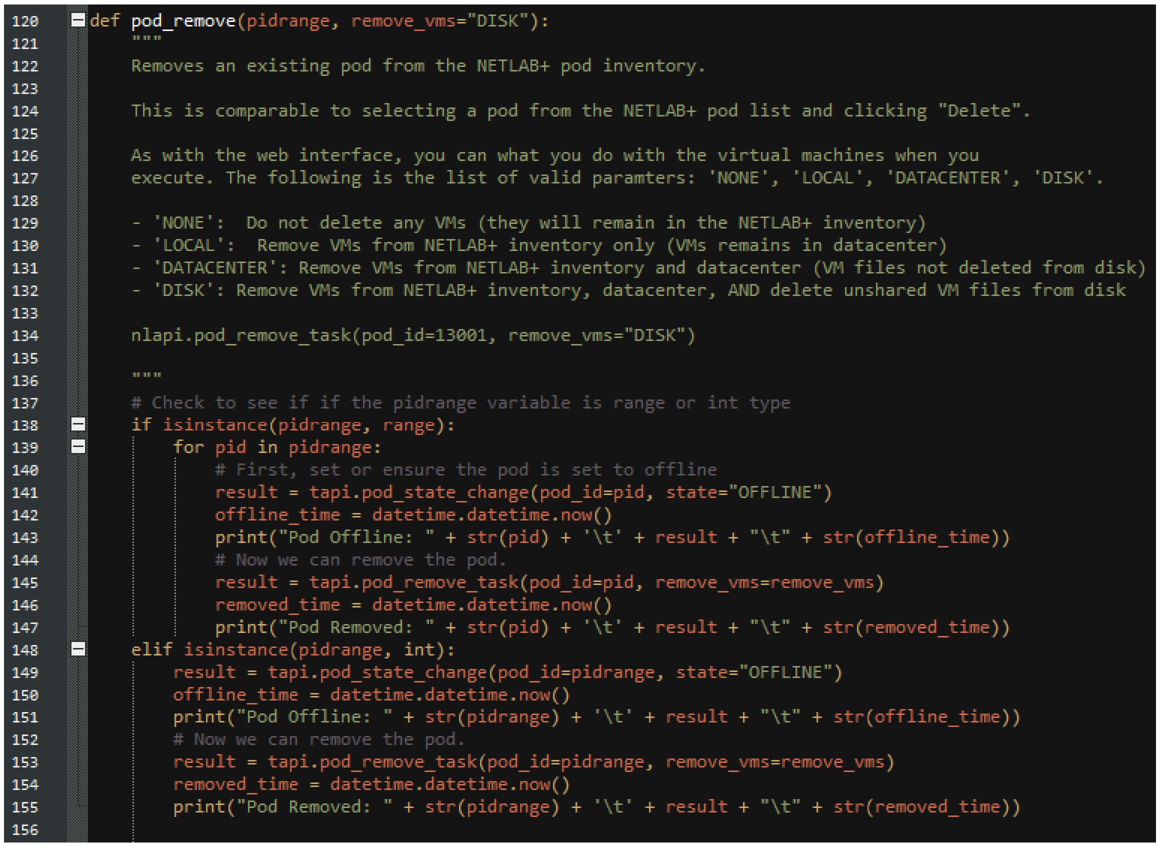
1. Navigate to where the **sample.py** file is saved and edit the file using a text editor.
2. Change the client system configuration to the appropriate value. In this example, the default config is used.



```
# The system will need to be changed to the name given to your system in the .netlab/config.jsons
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.

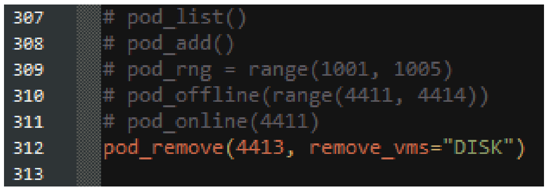
3. Scroll down to the **pod_remove** method and take notice of the instructions.



```
def pod_remove(pidrange, remove_vms="DISK"):
    """
    Removes an existing pod from the NETLAB+ pod inventory.

    This is comparable to selecting a pod from the NETLAB+ pod list and clicking "Delete".

    As with the web interface, you can what you do with the virtual machines when you
    execute. The following is the list of valid paramters: 'NONE', 'LOCAL', 'DATACENTER', 'DISK'.

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

    nlapi.pod_remove_task(pod_id=13001, remove_vms="DISK")

    """
    # Check to see if if the pidrange variable is range or int type
    if isinstance(pidrange, range):
        for pid in pidrange:
            # First, set or ensure the pod is set to offline
            result = tapi.pod_state_change(pod_id=pid, state="OFFLINE")
            offline_time = datetime.datetime.now()
            print("Pod Offline: " + str(pid) + '\t' + result + "\t" + str(offline_time))
            # Now we can remove the pod.
            result = tapi.pod_remove_task(pod_id=pid, remove_vms=remove_vms)
            removed_time = datetime.datetime.now()
            print("Pod Removed: " + str(pid) + '\t' + result + "\t" + str(removed_time))
    elif isinstance(pidrange, int):
        result = tapi.pod_state_change(pod_id=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(pod_id=pidrange, remove_vms=remove_vms)
        removed_time = datetime.datetime.now()
        print("Pod Removed: " + str(pidrange) + '\t' + result + "\t" + str(removed_time))
```

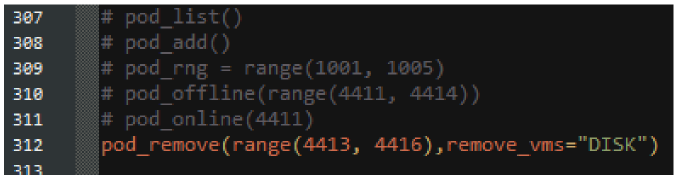
The *pod_remove* method can be called upon using three different functions, either by a single pod ID or a range of pod IDs as well as the type of deletion.

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.



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

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



```
# 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")
```

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

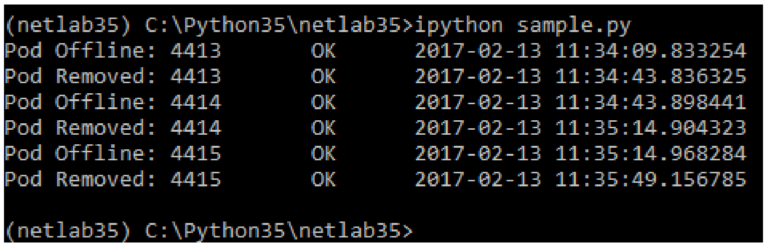
remove_vms="NONE": Do not delete any VMs (they will remain in the NETLAB+ inventory)
remove_vms="LOCAL": Remove VMs from NETLAB+ inventory only (VMs remains in datacenter)
remove_vms="DATACENTER": Remove VMs from NETLAB+ inventory and datacenter (VM files not deleted from disk)
remove_vms="DISK": Remove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



```
(netlab35) C:\Python35\netlab35>ipython sample.py
Pod Offline: 4413       OK      2017-02-13 11:34:09.833254
Pod Removed: 4413       OK      2017-02-13 11:34:43.836325
Pod Offline: 4414       OK      2017-02-13 11:34:43.898441
Pod Removed: 4414       OK      2017-02-13 11:35:14.904323
Pod Offline: 4415       OK      2017-02-13 11:35:14.968284
Pod Removed: 4415       OK      2017-02-13 11:35:49.156785

(netlab35) C:\Python35\netlab35>
```

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.

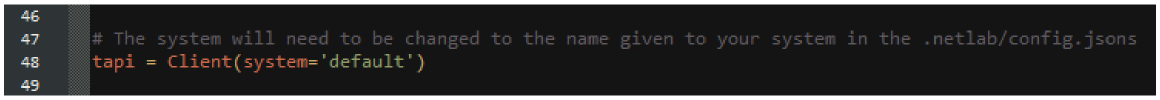
It is recommended to comment back the uncommented lines or any additional lines added to the sample.py file before reusing the script again for a different task. It may be better to save the script as a new file when making changes so that there is always an original available to work from.

4.3.6 Cloning Pods

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

4.3.6.1 Script Prep

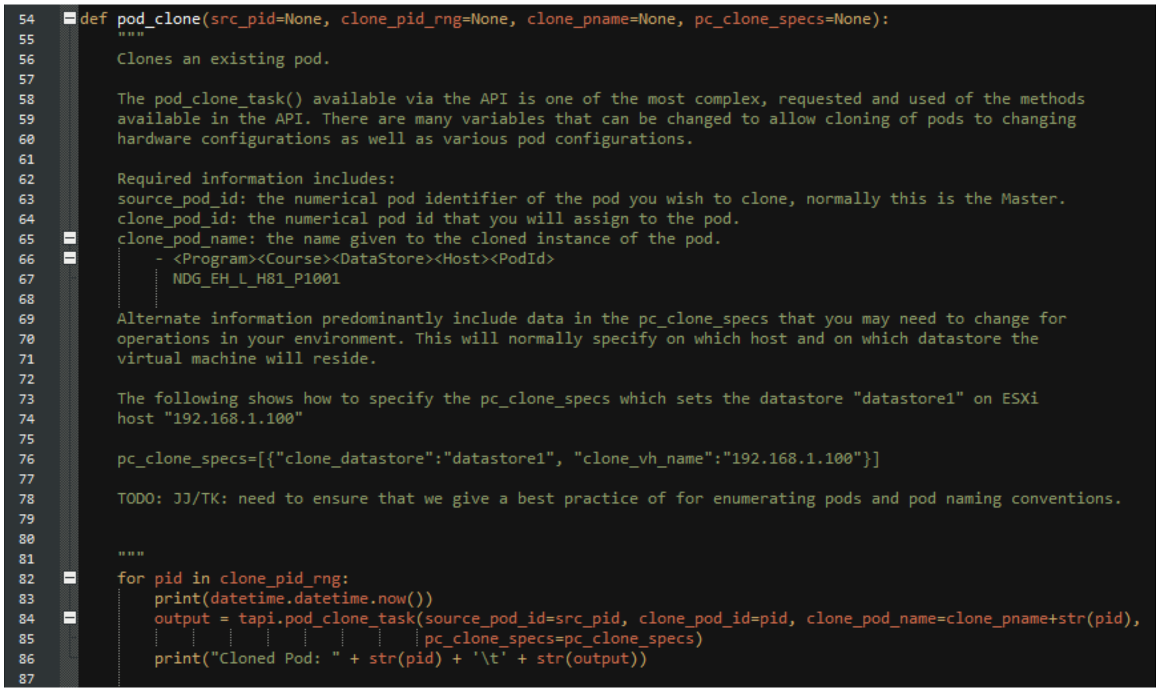
1. Navigate to where the **sample.py** file is saved and edit the file using a text editor.
2. Change the client system configuration to the appropriate value. In this example, the default config is used.



```
# The system will need to be changed to the name given to your system in the .netlab/config.jsons
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*.

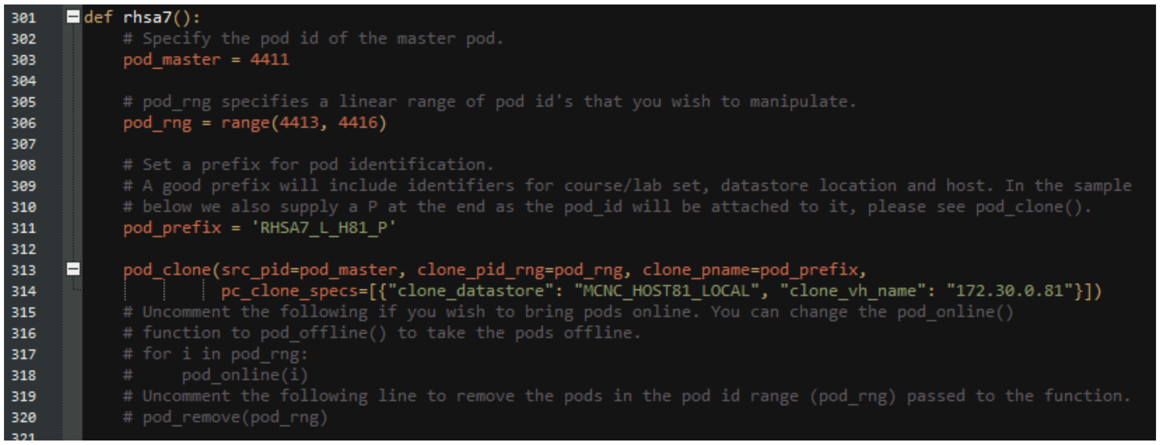
3. Scroll down to the **pod_clone** method and take notice of the instructions.



```
def pod_clone(src_pid=None, clone_pid_rng=None, clone_pname=None, pc_clone_specs=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.
        - <Program><Course><DataStore><Host><PodId>
          NDG_EH_L_H81_P1001

    Alternate information predominantly include data in the pc_clone_specs that you may need to change for
    operations in your environment. This will normally specify on which host and on which datastore the
    virtual machine will reside.

    The following shows how to specify the pc_clone_specs which sets the datastore "datastore1" on ESXi
    host "192.168.1.100"

    pc_clone_specs=[{"clone_datastore":"datastore1", "clone_vh_name":"192.168.1.100"}]

    TODO: JJ/TK: need to ensure that we give a best practice of for enumerating pods and pod naming conventions.


    """
    for pid in clone_pid_rng:
        print(datetime.datetime.now())
        output = tapi.pod_clone_task(source_pod_id=src_pid, clone_pod_id=pid, clone_pod_name=clone_pname+str(pid),
                                     pc_clone_specs=pc_clone_specs)
        print("Cloned Pod: " + str(pid) + '\t' + 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.

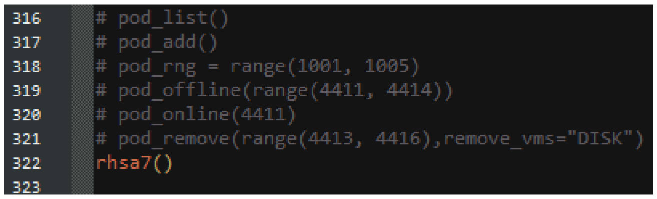


```
def rhsa7():
    # Specify the pod id of the master pod.
    pod_master = 4411

    # 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 P at the end as the pod_id will be attached to it, please see pod_clone().
    pod_prefix = 'RHSA7_L_H81_P'

    pod_clone(src_pid=pod_master, clone_pid_rng=pod_rng, clone_pname=pod_prefix,
              pc_clone_specs=[{"clone_datastore": "MCNC_HOST81_LOCAL", "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 i in pod_rng:
    #     pod_online(i)
    # 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.



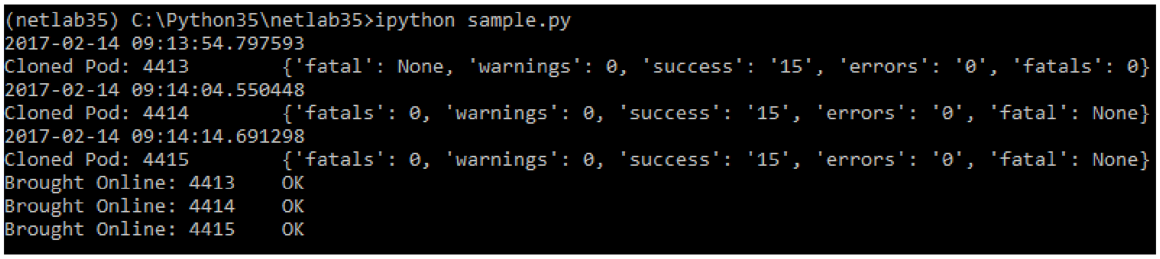
```
# pod_list()
# pod_add()
# pod_rng = range(1001, 1005)
# pod_offline(range(4411, 4414))
# pod_online(4411)
# pod_remove(range(4413, 4416),remove_vms="DISK")
rhsa7()
```

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.

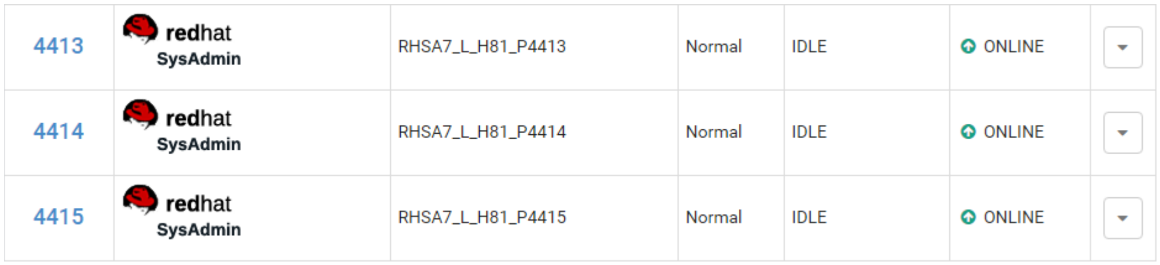
```
ipython sample.py
```



```
(netlab35) C:\Python35\netlab35>ipython sample.py
2017-02-14 09:13:54.797593
Cloned Pod: 4413        {'fatal': None, 'warnings': 0, 'success': '15', 'errors': '0', 'fatals': 0}
2017-02-14 09:14:04.550448
Cloned Pod: 4414        {'fatals': 0, 'warnings': 0, 'success': '15', 'errors': '0', 'fatal': None}
2017-02-14 09:14:14.691298
Cloned Pod: 4415        {'fatals': 0, 'warnings': 0, 'success': '15', 'errors': '0', 'fatal': None}
Brought Online: 4413    OK
Brought Online: 4414    OK
Brought Online: 4415    OK
```

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.



4413	redhat SysAdmin	RHSA7_L_H81_P4413	Normal	IDLE	ONLINE	
4414	redhat SysAdmin	RHSA7_L_H81_P4414	Normal	IDLE	ONLINE	
4415	redhat SysAdmin	RHSA7_L_H81_P4415	Normal	IDLE	ONLINE	

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

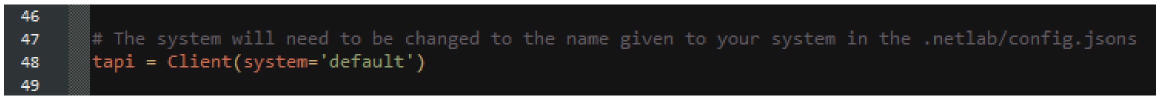
It is recommended to comment back the uncommented lines or any additional lines added to the sample.py file before reusing the script again for a different task. It may be better to save the script as a new file when making changes so that there is always an original available to work from.

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

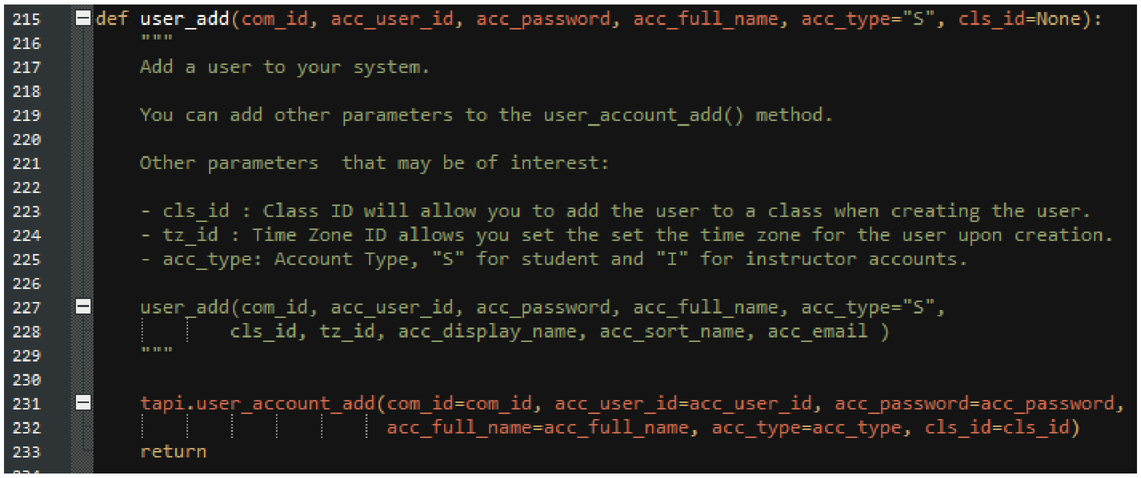
1. Navigate to where the **sample.py** file is saved and edit the file using a text editor.
2. Change the client system configuration to the appropriate value. In this example, the default config is used.



```

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

```

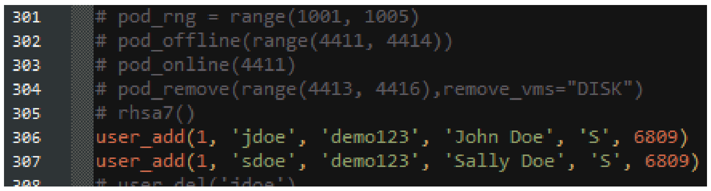
This value would reflect the specified --*system* value that was used when configuring the *config.json* file from *Section 3.4*.

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



```
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 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 )
    """

    tapi.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**.



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

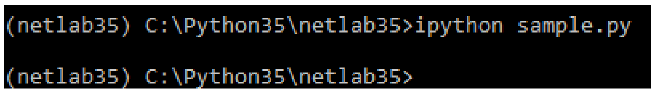
com_id: The community ID in which to add the user to, typically 1 is the *default* community.
acc_user_id: Assign a username to an account.
acc_password: Assign a password for the account.
acc_full_name: Assign a full name for the account.
acc_type: Assign (S) for student or (I) for instructor account 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.

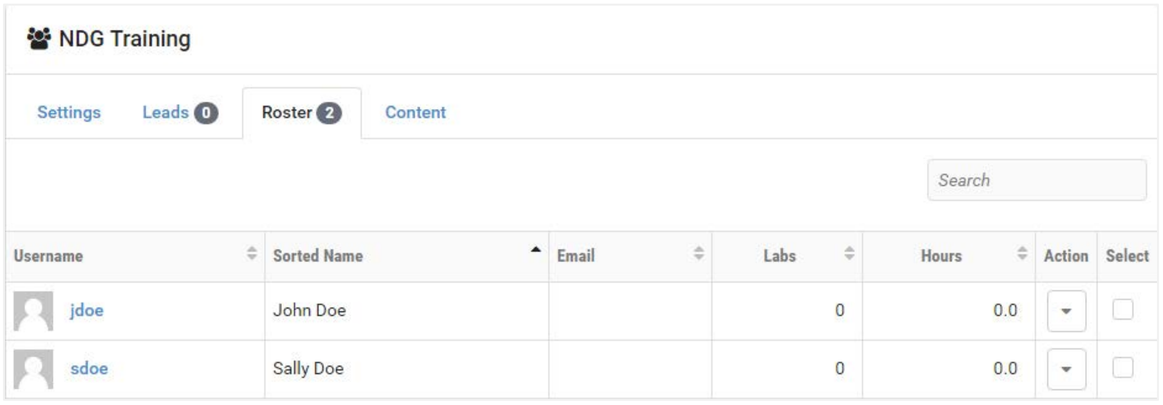
```
ipython sample.py
```



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

(netlab35) C:\Python35\netlab35>
```

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.



Username	Sorted Name	Email	Labs	Hours	Action	Select
jdoe	John Doe		0	0.0		
sdoe	Sally Doe		0	0.0		

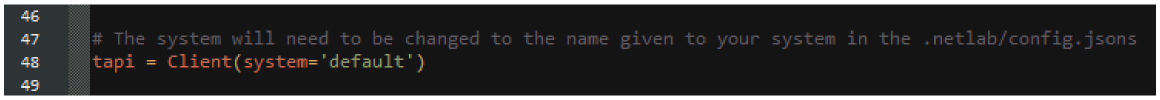
It is recommended to comment back the uncommented lines or any additional lines added to the sample.py file before reusing the script again for a different task. It may be better to save the script as a new file when making changes so that there is always an original available to work from.

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

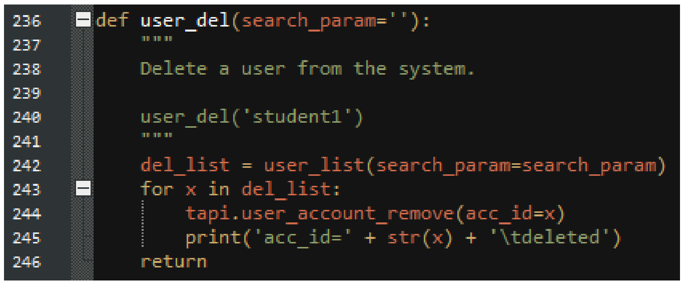
1. Navigate to where the **sample.py** file is saved and edit the file using a text editor.
2. Change the client system configuration to the appropriate value. In this example, the default config is used.



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

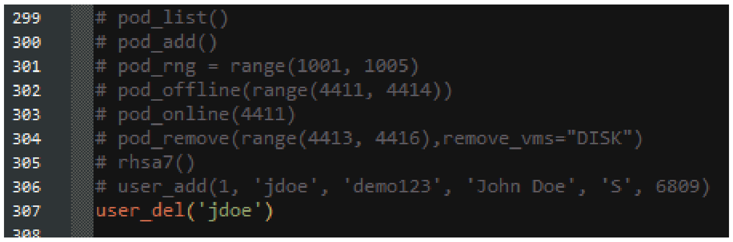
This value would reflect the specified --*system* value that was used when configuring the *config.json* file from *Section 3.4*.

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



```
def user_del(search_param=''):
    """
    Delete a user from the system.

    user_del('student1')
    """
    del_list = user_list(search_param=search_param)
    for x in del_list:
        tapi.user_account_remove(acc_id=x)
        print('acc_id=' + str(x) + '\tdeleted')
    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.



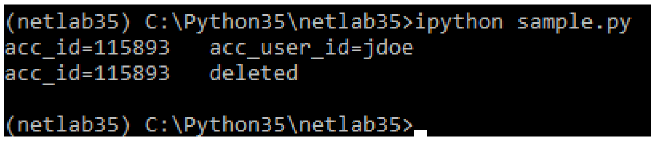
```
# pod_list()
# pod_add()
# pod_rng = range(1001, 1005)
# pod_offline(range(4411, 4414))
# pod_online(4411)
# pod_remove(range(4413, 4416),remove_vms="DISK")
# rhsa7()
# user_add(1, 'jdoe', 'demo123', 'John Doe', 'S', 6809)
user_del('jdoe')
```

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.

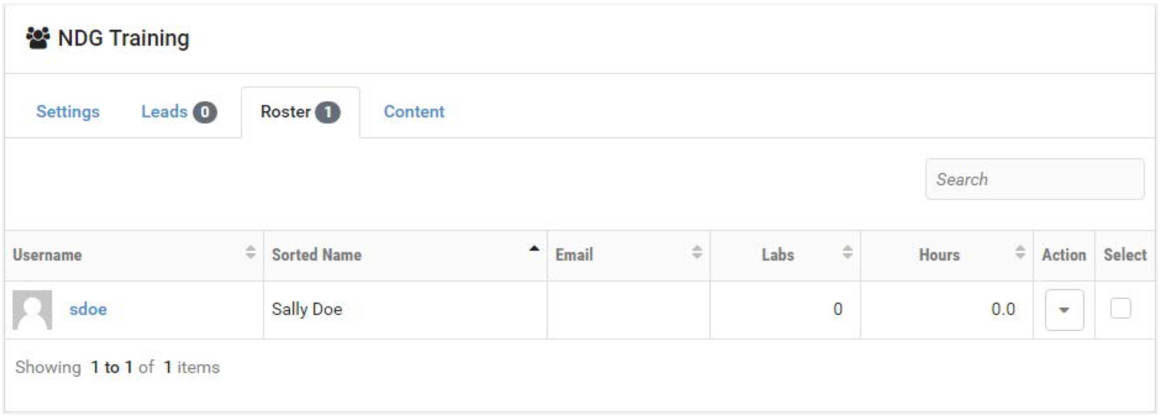
```
ipython sample.py
```



```
(netlab35) C:\Python35\netlab35>ipython sample.py
acc_id=115893   acc_user_id=jdoe
acc_id=115893   deleted

(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.



NDG Training

Username	Sorted Name	Email	Labs	Hours	Action	Select
sdoe	Sally Doe		0	0.0		

Showing 1 to 1 of 1 items

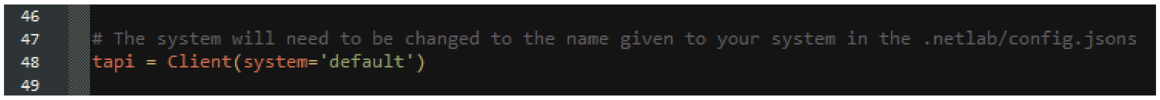
It is recommended to comment back the uncommented lines or any additional lines added to the sample.py file before reusing the script again for a different task. It may be better to save the script as a new file when making changes so that there is always an original available to work from.

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

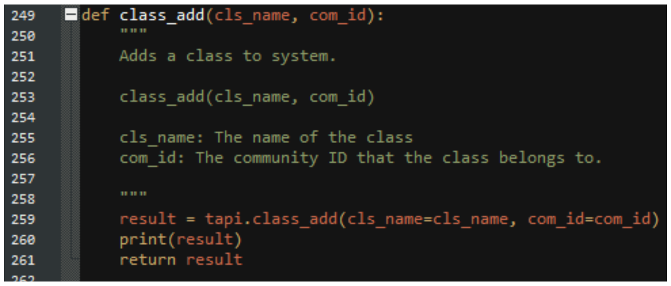
1. Navigate to where the **sample.py** file is saved and edit the file using a text editor.
2. Change the client system configuration to the appropriate value. In this example, the default config is used.



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

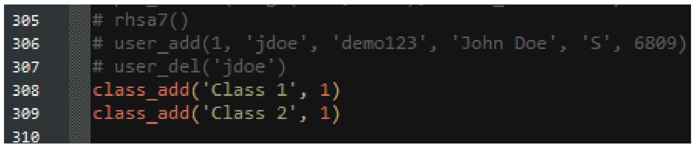
This value would reflect the specified --*system* value that was used when configuring the *config.json* file from *Section* 3.4.

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



```
def class_add(cls_name, com_id):
    """
    Adds a class to system.

    class_add(cls_name, com_id)

    cls_name: The name of the class
    com_id: The community ID that the class belongs to.

    """
    result = tapi.class_add(cls_name=cls_name, com_id=com_id)
    print(result)
    return result
```

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).



```
# rhsa7()
# user_add(1, 'jdoe', 'demo123', 'John Doe', 'S', 6809)
# user_del('jdoe')
class_add('Class 1', 1)
class_add('Class 2', 1)
```

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

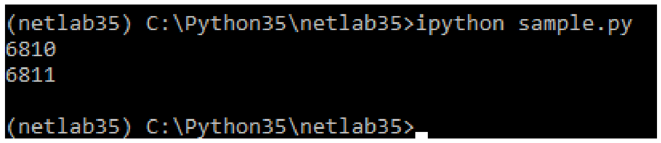
cls_name: The desired name for the new class.
com_id: The community ID in which to add the user to, typically 1 is the *default* community.

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
```

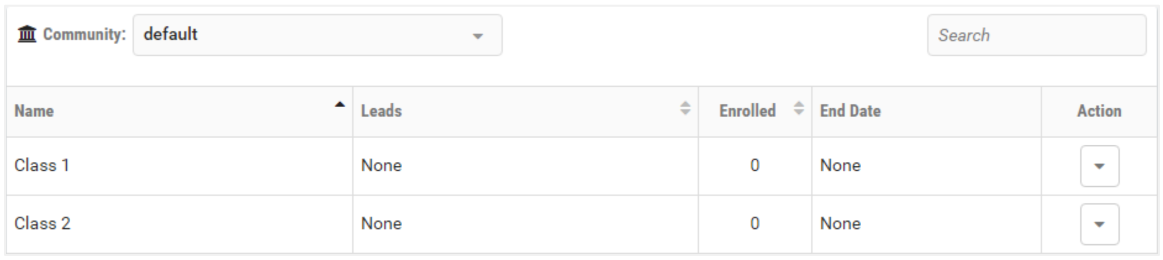


```
(netlab35) C:\Python35\netlab35>ipython sample.py
6810
6811

(netlab35) C:\Python35\netlab35>
```

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.



Community: default

Search

Name	Leads	Enrolled	End Date	Action
Class 1	None	0	None	
Class 2	None	0	None	

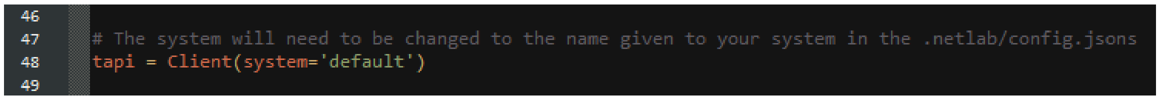
It is recommended to comment back the uncommented lines or any additional lines added to the sample.py file before reusing the script again for a different task. It may be better to save the script as a new file when making changes so that there is always an original available to work from.

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

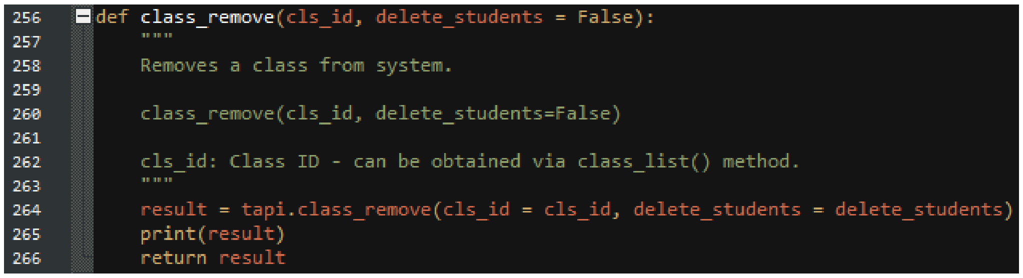
1. Navigate to where the **sample.py** file is saved and edit the file using a text editor.
2. Change the client system configuration to the appropriate value. In this example, the default config is used.



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

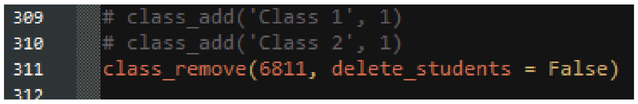
This value would reflect the specified --*system* value that was used when configuring the *config.json* file from *Section* 3.4.

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



```
def class_remove(cls_id, delete_students = False):
    """
    Removes a class from system.

    class_remove(cls_id, delete_students=False)

    cls_id: Class ID - can be obtained via class_list() method.
    """
    result = tapi.class_remove(cls_id = cls_id, delete_students = delete_students)
    print(result)
    return result
```

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.



```
# class_add('Class 1', 1)
# class_add('Class 2', 1)
class_remove(6811, delete_students = False)
```

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

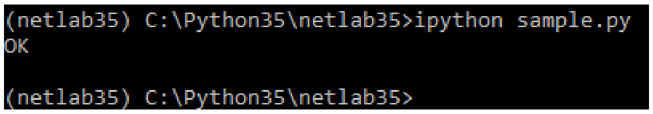
cls_id: Assign account to a specified class using an ID value. This value can be obtained from *Section* 4.2.2.3.
delete_students: Set either *True* (delete attached student accounts) or *False* (do no delete attached student accounts)

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.

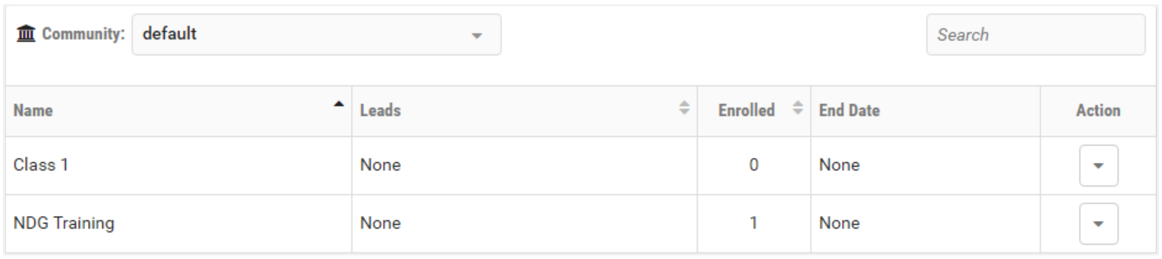
```
ipython sample.py
```



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

(netlab35) C:\Python35\netlab35>
```

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

Search

Name	Leads	Enrolled	End Date	Action
Class 1	None	0	None	
NDG Training	None	1	None	

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*.