Huawei Certification Training

HCIP-Datacom-Network Automation Developer
Python Programming Basics Lab Guide

V1.0

Huawei Technologies Co., Ltd.

Trademarks and Permissions


and other Huawei trademarks are trademarks of Huawei Technologies Co., Ltd.

All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The purchased products, services and features are stipulated by the contract made between Huawei and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

Huawei Technologies Co., Ltd.

Address: Huawei Industrial Base
Bantian, Longgang
Shenzhen 518129
People's Republic of China

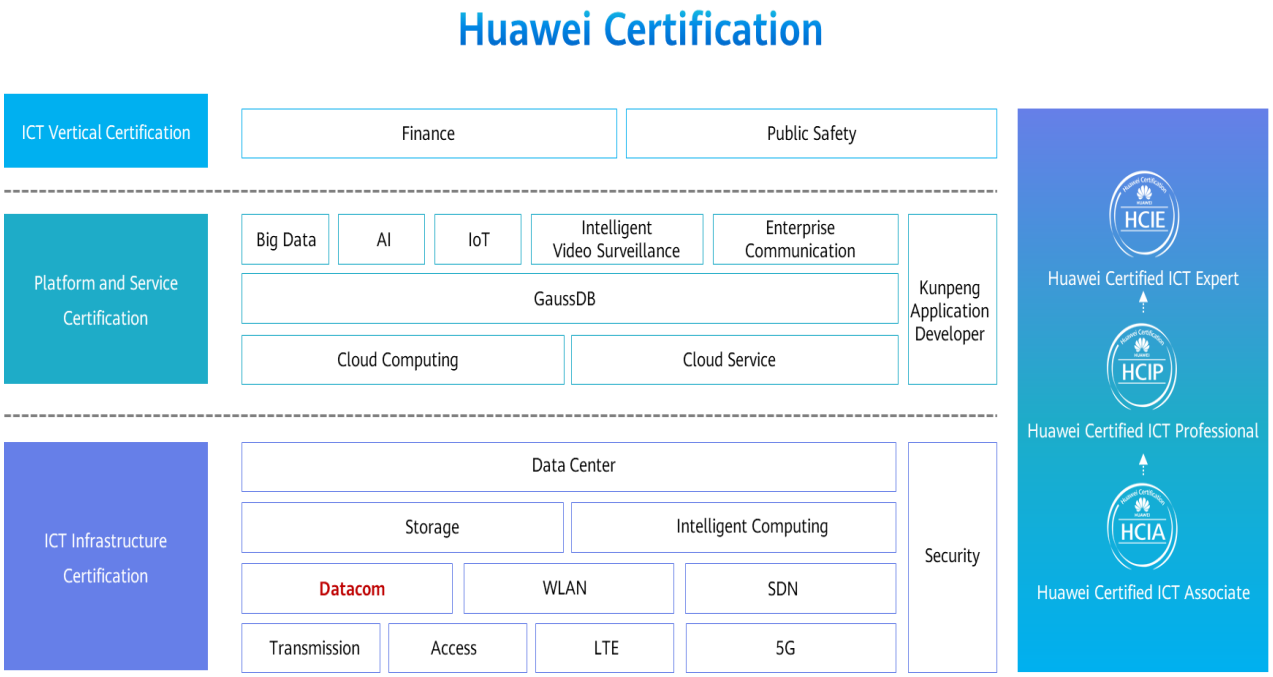
Website: https://e.huawei.com/

Huawei Certification System

Huawei Certification follows the "platform + ecosystem" development strategy, which is a new collaborative architecture of ICT infrastructure based on "Cloud-Pipe-Terminal". Huawei has set up a complete certification system consisting of three categories: ICT infrastructure certification, platform and service certification, and ICT vertical certification. It is the only certification system that covers all ICT technical fields in the industry. Huawei offers three levels of certification: Huawei Certified ICT Associate (HCIA), Huawei Certified ICT Professional (HCIP), and Huawei Certified ICT Expert (HCIE). Huawei Certification covers all ICT fields and adapts to the industry trend of ICT convergence. With its leading talent development system and certification standards, it is committed to fostering new ICT talent in the digital era, and building a sound ICT talent ecosystem.

HCIP-Datacom-Network Automation Developer is designed for professional engineers who are capable of network automation development. The HCIP-Datacom certification will qualify you as professionals in network automation development, who are capable of automatic deployment, development, and O&M of enterprise networks.

The Huawei certification system introduces the industry, fosters innovation, and imparts cutting-edge datacom knowledge.



About This Document

Introduction

This document is an HCIP-Datacom-Network Automation Developer certification training course and is intended for trainees who are going to take the HCIP-Datacom-Network Automation Developer exam or readers who want to understand Python programming basics and practices.

Background Knowledge Required

This course is intended for professional network automation engineers. The trainees are supposed to have the following background knowledge:

- HCIA-Datacom

Lab Environment

Description

This lab environment is intended for datacom engineers who are preparing for the HCIP-Datacom-Network Automation Developer exam. The experiments can be carried out on any Python compiler, and Juypter Notebook and Pycharm are recommended.

Compilation Environment

The compilation environment is based on the following compiler versions:

Compiler	Software Version
Anaconda3	2020.02
Pycharm Community Edition	2020.01

Contents

1 Setting up the Environment

1.1 Introduction to Anaconda

Anaconda is a distribution of Python for scientific computing, and it supports Linux, macOS, and Windows. It provides simplified package management and environment management, and easily deals with the installation issues when the system has multiple Python versions and third-party packages. Anaconda uses Conda or Pip to implement package and environment management. It also comes with Python and related tools. Anaconda is a Python tool for enterprise-level big data analytics. It contains more than 720 open-source data-science packages, including data visualization, machine learning, and deep learning. It can be used for data analysis, big data and AI fields.

Anaconda provides the following features for developers:

- Notebook is supported, which is friendly to beginners.
- Python has been integrated into Anaconda, and there is an option to select the Python version during download.
- You only need to add a virtual environment to Anaconda when different frameworks are required to support development. You can conduct development in different environments without worrying about compatibility issues. You can also configure environments for special projects to facilitate management.

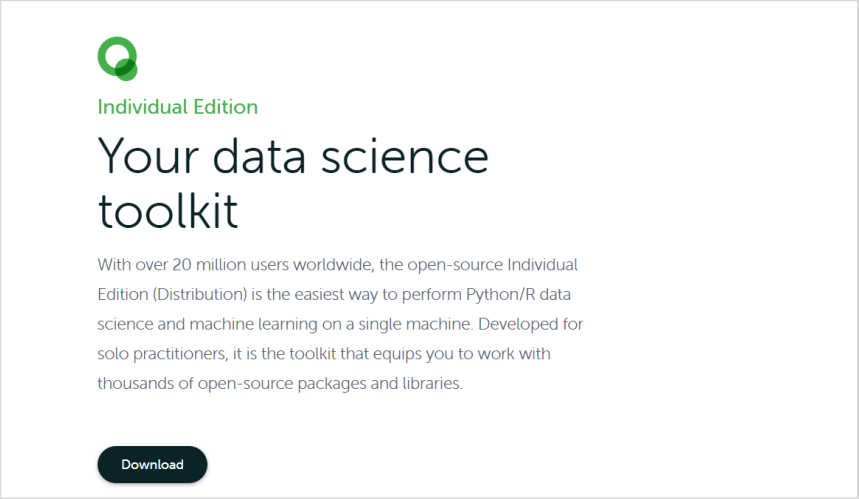
NOTE

You can also use other development environments (such as PyCharm) to run Python scripts. PyCharm is a Python IDE that provides a set of tools to help users improve efficiency in development using Python, such as debugging, syntax highlighting, project management, code jumping, intelligent tips, automatic completion, unit test, and version control. In addition, the IDE provides some advanced functions to support professional Web development under the Django framework.

1.2 Installation Procedure

Step 1 Download Anaconda.

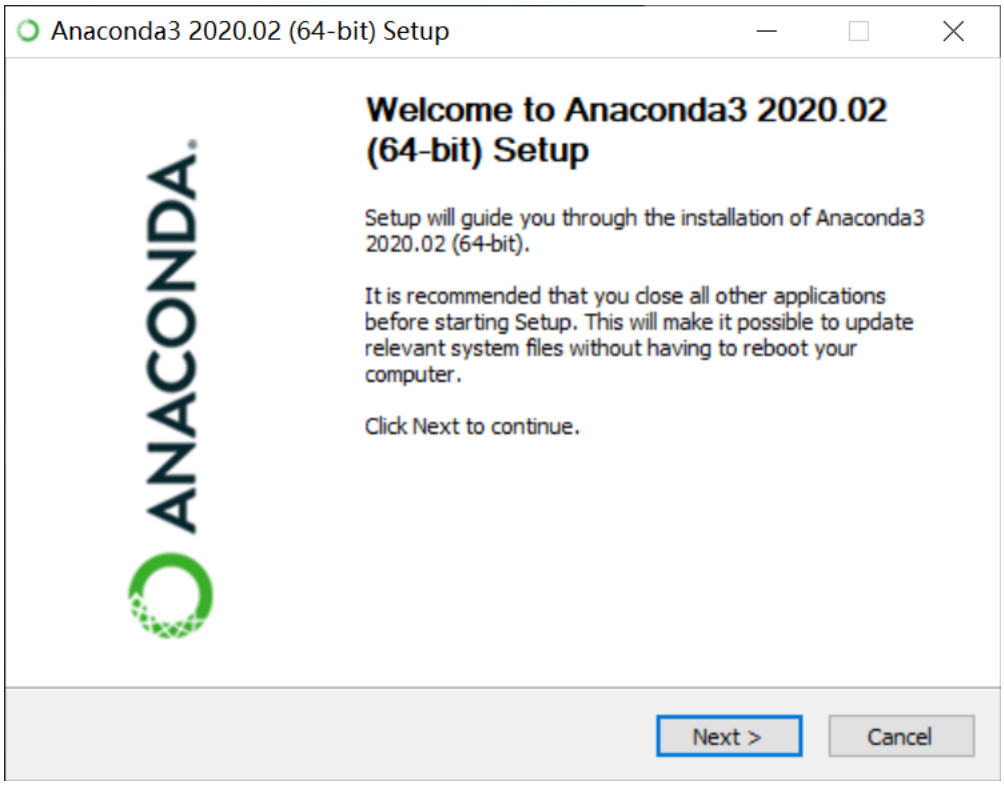
Log in to the official website of Anaconda (https://www.anaconda.com/products/individual), and click **Download** to download the installation package.



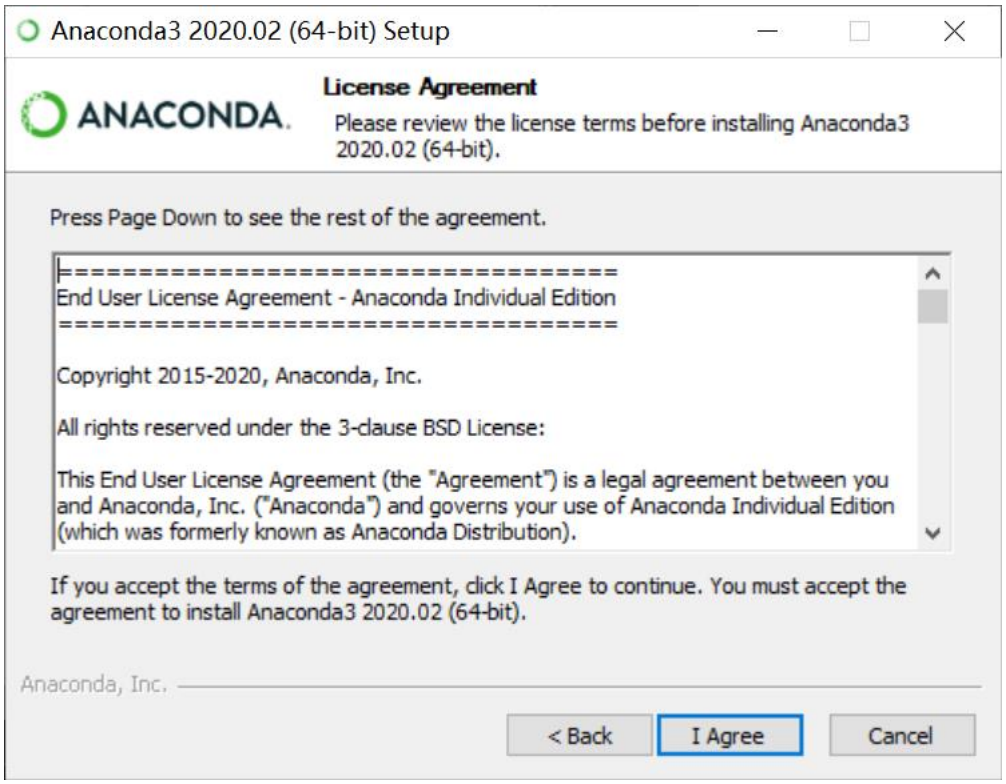
Select the version for Windows, macOS, or Linux. In this example, select **Windows**, select **Python 3.7** (recommended) or **Python 2.7**, and click **64-Bit Graphical Installer** (not supported by a 32-bit computer).

Step 2 Install Anaconda.

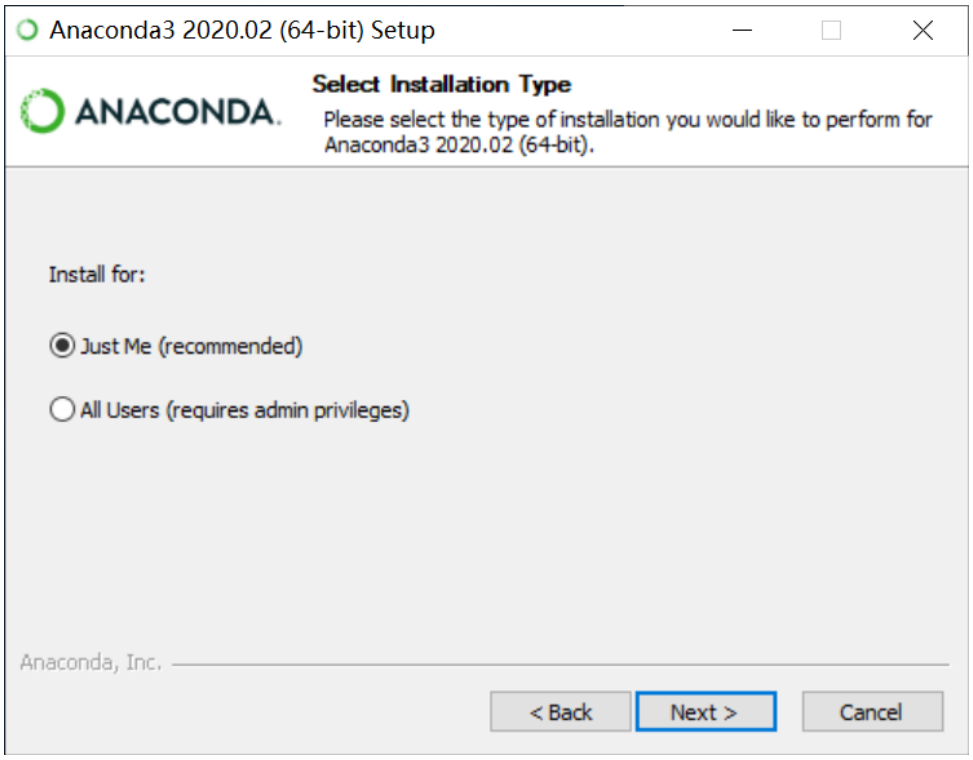
Double-click the **Anaconda3-x.x.x-Windows-x86_64.exe** file. In the dialog box that is displayed, click **Next**.



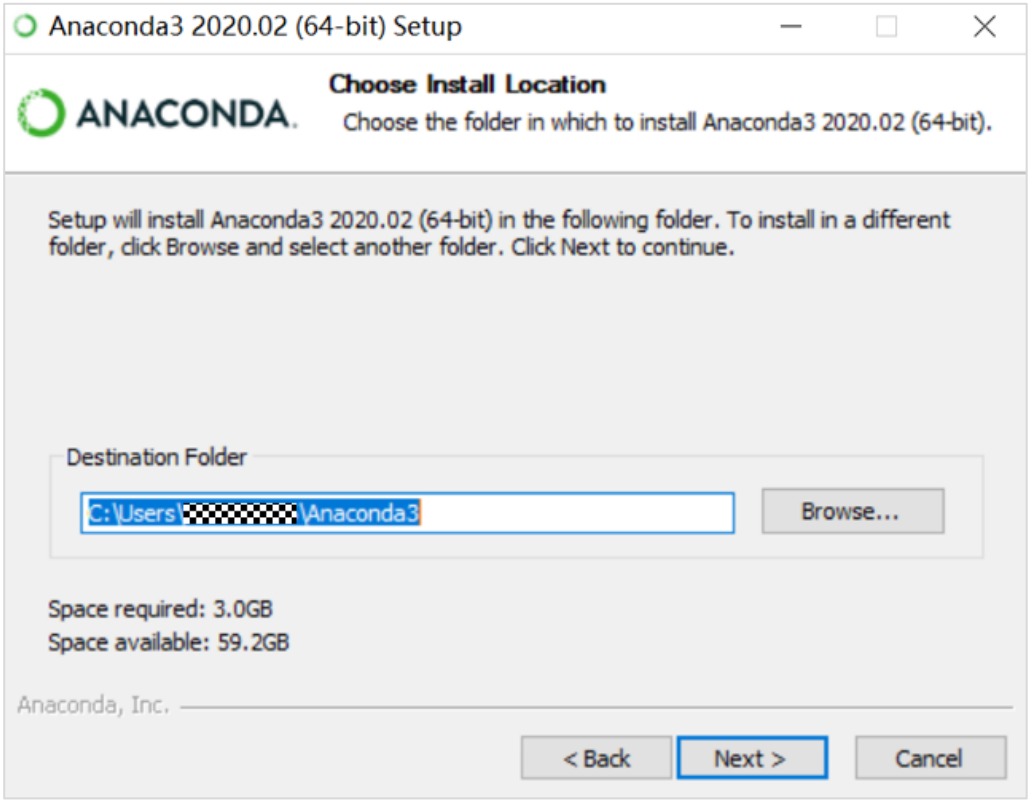
Click **I Agree**.



Install for: Select **Just Me** and click **Next**.

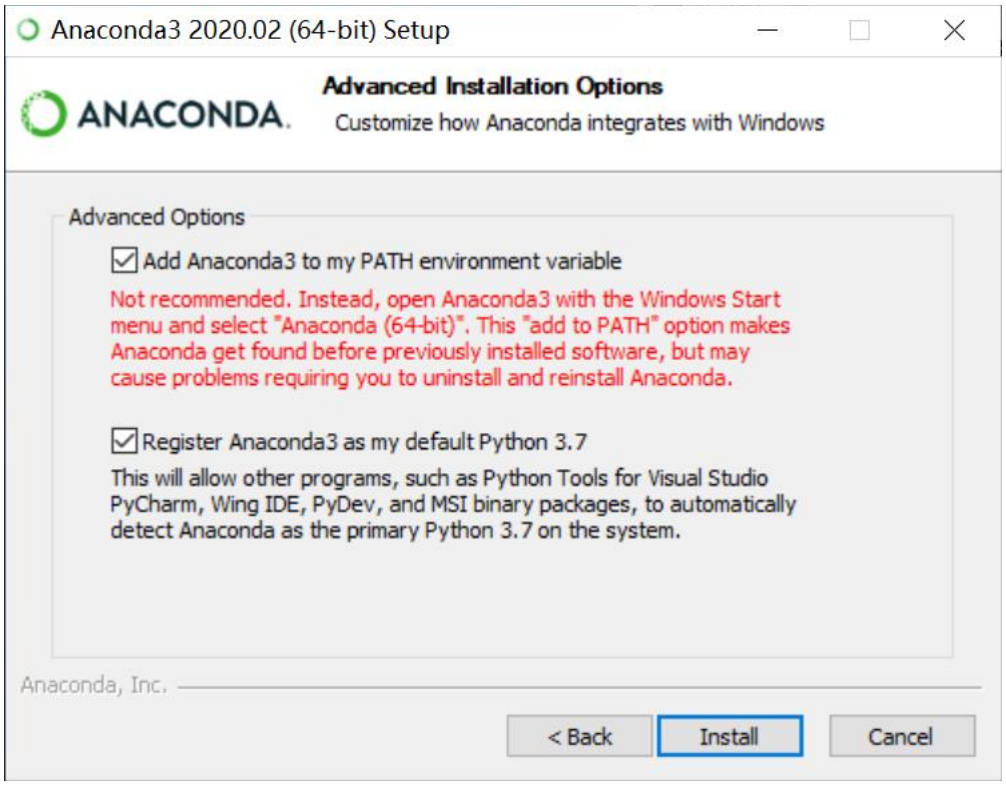


Step 3 Specify the software installation path and click **Next**.

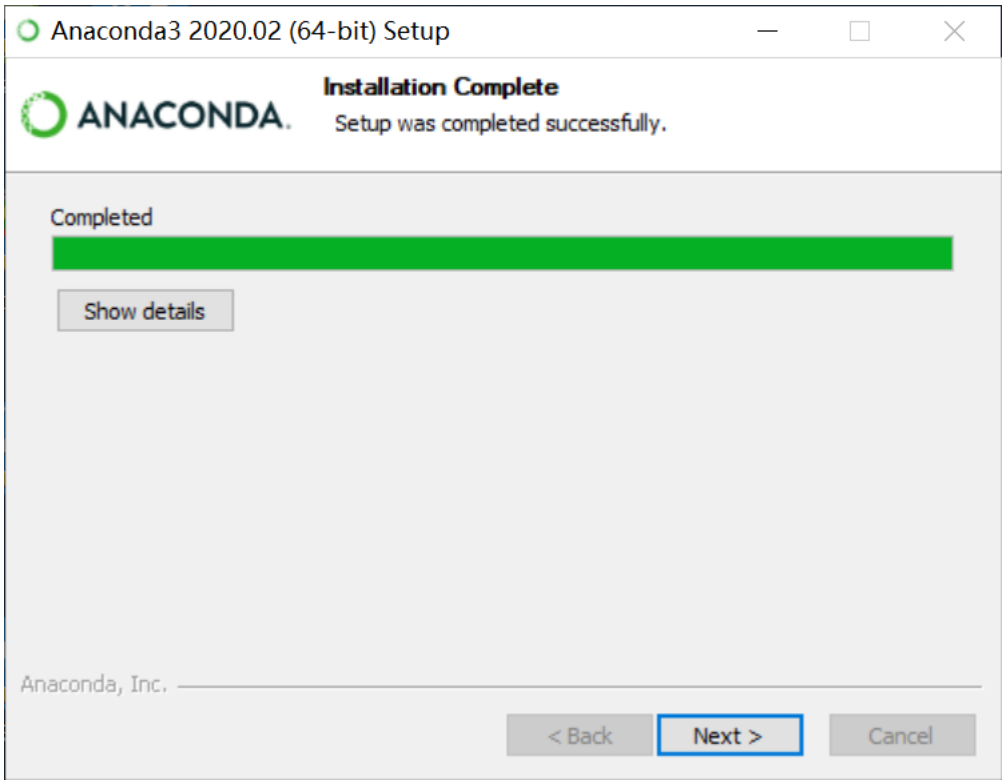


Step 4 Configure environment variables.

Select **Add Anaconda to my PATH environment variable** and **Register Anaconda as my default Python 3.7** to reduce subsequent configurations, and click **Install**. (Note: If Python of another version has been installed on the computer, you are advised to remove it and install Anaconda. If it is not deleted, you are advised to de-select the two options; otherwise, a path error may occur.)

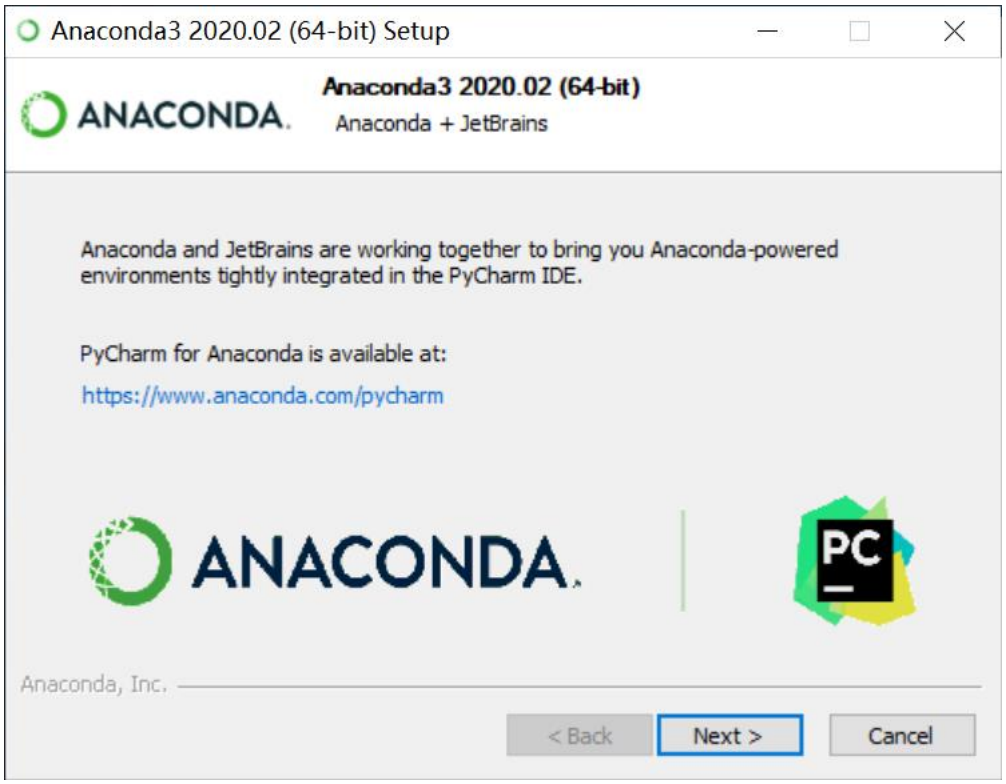


When the installation is complete, click **Next**.

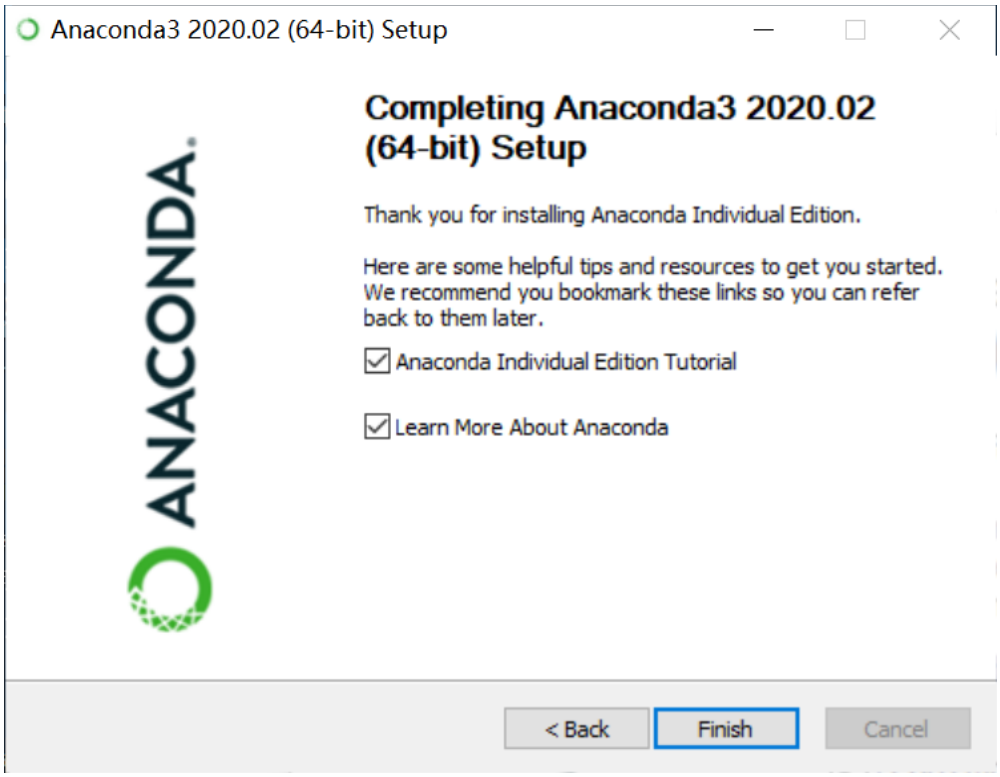


Step 5 View installation information.

Click **Next** to view related information.



Click **Finish** to view the Anaconda tutorial page.

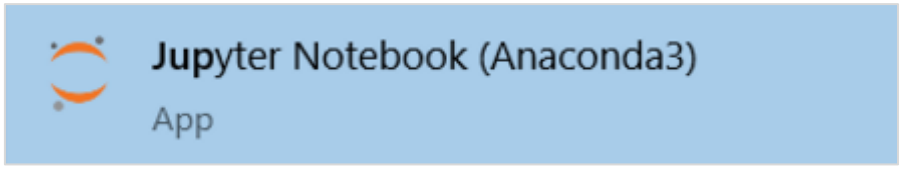


----End

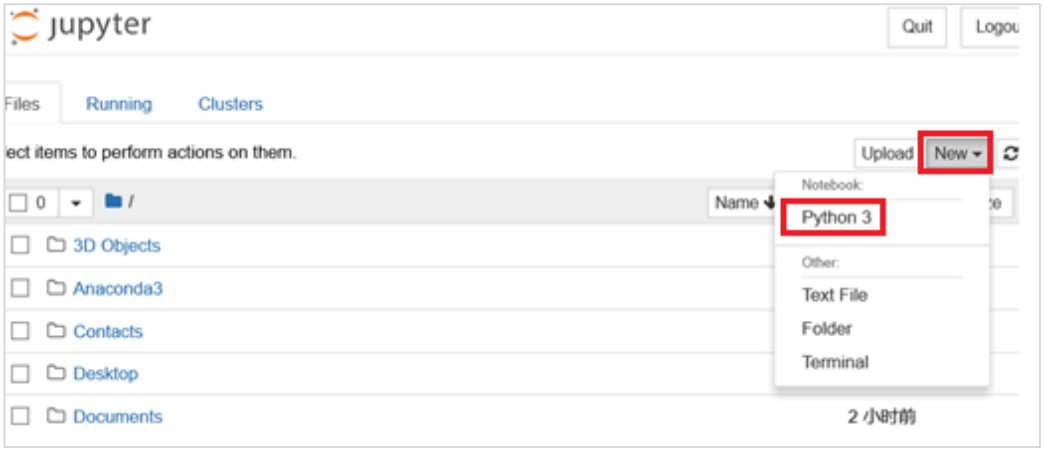
1.3 Verifying the Installation

Run the Python script.

1. Choose **Start** > **Jupyter Notebook** to display the Jupyter homepage.

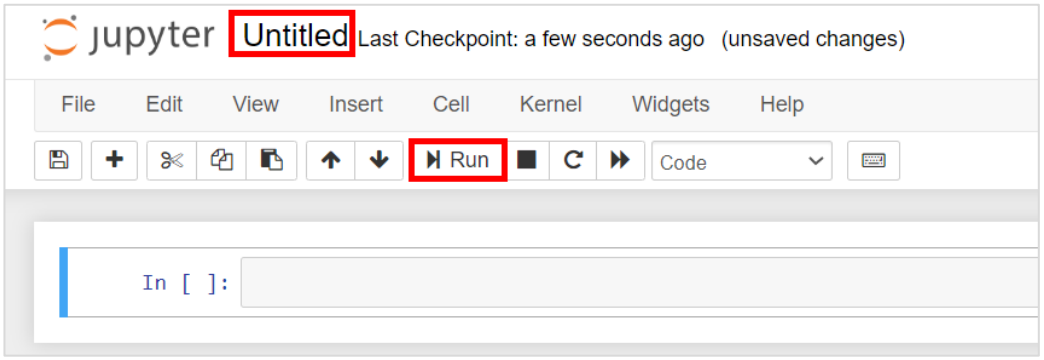


2. Click **New** in the upper right corner of the page and select **Python 3** to create a Jupyter file.



The title in the red box is the default title, which can be changed if you double-click the title.

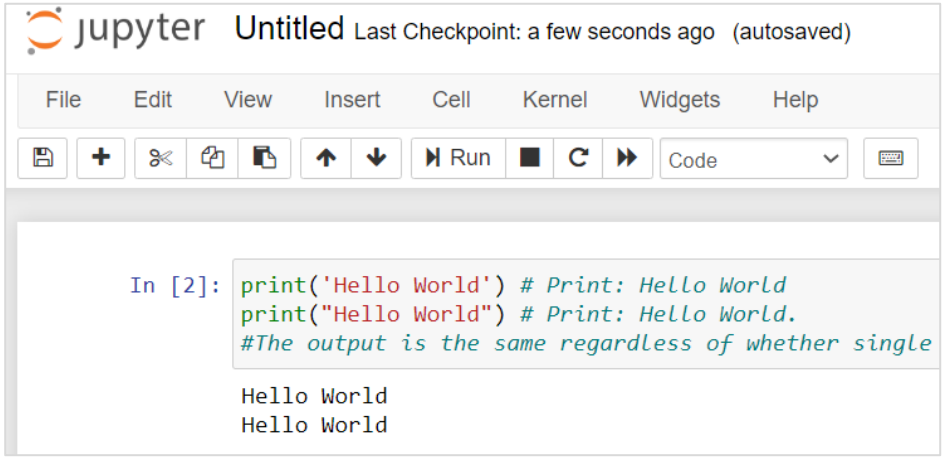
Enter the code in the green rectangle, and click **Run** to run the code.



3. Enter the following code in the text box:

```
print('Hello World') # Print: Hello World
print("Hello World") # Print: Hello World.
#The output is the same regardless of whether single or double quotation marks are carried in input.
```

4. Click **Run**, as shown in the following figure.



2 Python Programming Basics

2.1 Introduction to Experiment

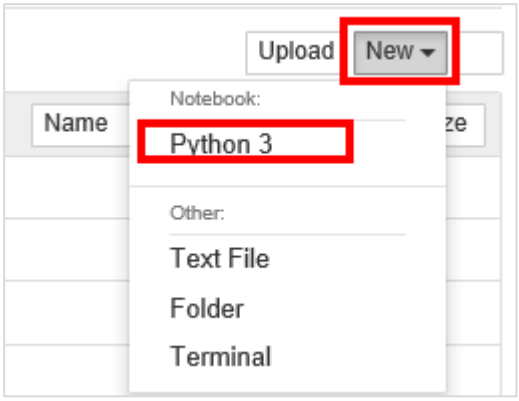
This experiment uses independent code practices of each module to help readers master the Python3 syntax.

- Basic data types of Python
- Deep copy and shallow copy
- Flow control
- Functions
- Operations on files
- Troubleshooting
- Object-oriented programming

It is recommended to learn this guide with the official Python Tutorial, https://docs.python.org/3/tutorial/index.html.

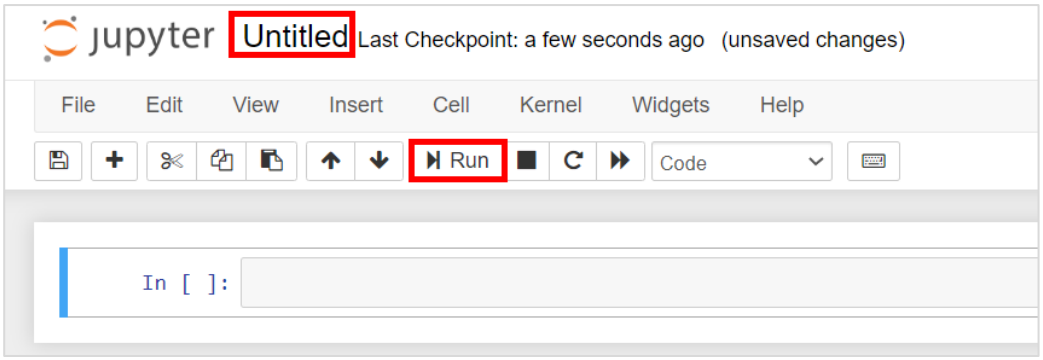
2.2 Code Practices

After Anaconda3 is installed on the local PC, open Jupyter Notebook and create a Python3 notebook file. (You can also use other compilers for the experiment, such as Pycharm.)



The title in the red box is the default title, and you can double-click the title to change the file name.

The blue box in the following figure is the code input box. After the code is complete, click **Run** to run the code.



2.2.1 Python Data Types

Python has rich data types, including numerics, strings, lists, tuples, dictionaries, and sets. This section describes the characteristics of different data types using code practices. For details, see "Built-in Types" in the Python standard library. https://docs.python.org/3.8/library/index.html.

2.2.1.1 Data Type: Numerics

This data type is used to store numeric values, and it is immutable, which means that, if it is changed, a new object will be assigned.

Note that Boolean operations in Python use keywords instead of operators to express “and/or/not”.

```
print(True+False) # The output is 1. By default, 1 indicates True, and 0 False.
print(True or False) # True is displayed, and the "or" operation is performed.
Print(5//2) # The output is 2, and “//” is the rounding operator.
Print(5%2) # The output is 1, and “%” is the modulo operator.
print(3**2) # The output is 9, and “**” indicates the power operation.
```

```
print(5+1.6) # The output is 6.6. By default, the sum of numbers in different precisions follow the highest precision of the numbers.
```

2.2.1.2 Data Type: Strings

A string consists of digits, letters, and underscores (_). Objects are created using single, double, or triple quotation marks.

Basic operations on strings:

```
S = 'python' # Assign ‘python’ to variable S.
# len(obj): Return the object length.
print(len(S)) # Output 6.

print(S[0],S[1],S[-1]) # Output pyn and obtain elements based on the index.
print(S+'1',S*2) # Output python1 pythonpython: merge and repeat.
```

Immutability of strings:

```
S = 'python'                  # Assign a value to the variable.
S[0] = 'Z'                    # Program exception, as the string cannot be changed.
S1 ='Z'+S[1:]                 # A new string, Zython, is generated and assigned to S1.
                              # S[1:] indicates the string following the first character, that is, ython.
print("S:%s,S1:%s"%(S,S1))    # Output S:python, S1:Zython. %s prints the character string.
```

Common operations on strings:

```
S = "python" # Assign a value to the variable.

#   str.split(str="", num=-1): The string is split by separator. If the “num”argument has a value, the string is split into num+1 substrings. The value “-1” indicates that all strings are split.
print(S.split('h')) # Output ['pyt','on'], and split the string based on “h”.

#   str.replace(old, new[, max]): A string generated after “old” in the string is replaced with “new”. If the third argument “max” is specified, this replacement can take place for not more than max times.
print(S.replace('py','PY')) # PYthon: Replace “py” in the string with “PY”.

#   str.upper(): Return the value after lowercase letters are converted to uppercase letters.
print(S.upper())                                    # PYTHON

#   str.lower(): Return the value after uppercase letters are converted to lowercase letters.
print('PYTHON'.lower()) # Output python. The string is converted into lowercase letters.

line='aa,bb,ccc,dd\n' #”\n” is a line feed character.

#   str.join(sequence): “sequence” indicates a sequence to be joined. In the output, the new string generated after the elements are specified in the sequence is returned.
print(''.join(['life', 'is' ,'short'])) # Output “life is short” and “join” joins the string.

hw12= '%s %s %d' % ('hello', 'world', 12) # Format the string.
print(hw12) # Output “hello world 12”.
```

2.2.1.3 Data Type: Lists

Lists can implement data structures of most set classes. It supports characters, numbers, strings, and even lists (that is, nesting). It is identified by [].

- Common operations on lists:

```
animals = ['cat', 'dog', 'monkey'] # Define the list animals.

#   list.append(obj): Add a new object to the end of the list.
animals.append('fish') # Append an element.
print(animals) # Output ['cat', 'dog', 'monkey', 'fish'].

#   list.remove(obj): Remove the first match for a value in the list.
animals.remove('fish') # Delete the element "fish".
print(animals) # Output ['cat', 'dog', 'monkey'].

#   list.insert(index, obj): Insert a specified object to a specified position in the list. The index indicates the position.
animals.insert(1,'fish') # Insert the element "fish" at the position of subscript 1.
print(animals) # Output ['cat', 'fish', 'dog', 'monkey'].

#   list.pop([index=-1]): Remove the element (the last element by default) corresponding to the subscript in the list. The index indicates the subscript.
animals.pop(1) # Delete the element whose subscript is 1.
print(animals) # Output ['cat', 'dog', 'monkey'].
```

- Traverse and obtain the elements and indexes.

```
#   enumerate(sequence): Return an index sequence consisting of a data object that can be traversed and list the data and subscripts. This function is usually used in the "for" loop.
for i in enumerate(animals):
        print(i) # Index formed by element subscripts and elements.
```

Output:

```
(0, cat)
(1, dog)
(2, monkey)
```

- List derivations

```
squares = [x*2 for x in animals] # Generate a list of elements that comply with rules in batches.
print(squares)                                          # ['catcat ', 'dogdog ', 'monkeymonkey ']
```

- Sorting

list.sort(cmp=None, key=None, reverse=False): The "cmp" parameter is optional. If it is specified, the method of this parameter is used for sorting. "key" is an element used for comparison. "reverse" indicates the sorting rule, and "False" indicates the ascending order.

```
list1 = [12,45,32,55] # Define a new list, list1.
list1.sort() # Sort the list.
print(list1) # Output [12,32,45,55].

#   list.reverse(): Element in the reverse list.
list1.reverse() # Reverse the list.
print(list1) # Output [55,45,32,12]
```

2.2.1.4 Data Type: Tuples

A tuple is identified by () and internal elements are separated by commas (,).

A tuple cannot be assigned a value for a second time, which is similar to a read-only list.

Common operations on tuples:

```
T=(1,2,3)                          # Create a tuple.
print(T+(4,5))                     # Combine tuples and output (1, 2, 3, 4, 5).
t=(42,)                            # A tuple with only one element, which is different from a number.
tuple1 = (12,45,32,55,[1,0,3])     # Create a tuple.
tuple1[0] = "good"                 # Program exception, as the tuple is immutable and a value cannot be
assigned to it.
tuple1[4][0] = 2                   # Values cannot be assigned to mutable elements. In the example, the
elements are in the [1,0,3] list.
print(tuple1)                      # (12,45,32,55,[2,0,3])
```

2.2.1.5 Data Type: Dictionaries

A dictionary is a flexible built-in data structure and is identified by {}.

A dictionary consists of indexes (keys) and their values. Compared with a list, elements in a dictionary are saved as keys instead of offsets.

Common operations on dictionaries:

```
# Three operations to assign values to dictionaries:
x = {'food':'Spam','quantity':4,'color':'pink'}
x = dict(food='Spam',quantity=4, color='pink')
x = dict([("food", "Spam"),("quantity", "4"),("color","pink")])

#   dict.copy(): Copy data.
d =x.copy()
d['color'] = 'red'
print(x)                           # {'food':'Spam','quantity':4,'color':'pink'}
print(d)                           # {'food':'Spam','quantity':4,'color':'red'}

# Element access.
print(d['name'])                   # Error information is obtained.
print(d.get('name'))               # Output none.
print(d.get('name','The key value does not exist.'))          # Output “The key value does not exist”.
print(d.keys())                    # Output “dict_keys(['food', 'quantity', 'color'])”.
print(d.values())                  # Output “dict_values(['Spam', 4, 'red'])”.
print(d.items())                   # Output “dict_items([('food', 'Spam'), ('quantity', 4), ('color', 'red')])”.
d.clear()                          # Clear all data in the dictionary.
print(d)                           # Output “{}”.
del(d)                             # Delete the dictionary.
```

2.2.1.6 Data Type: Sets

A set is an unordered sequence of unrepeatable elements. Sets can be created using braces {} or the set() function.

Common operations on sets:

```
sample_set = {'Prince', 'Techs'}
print('Data' in sample_set) # Output “False”. “in” is used to check whether an element exists in the set.

# set.add(obj): Add an element to a set. If the element to be added already exists in the set, no operation is performed.
sample_set.add('Data')              # Add element “Data” to the set.
print(sample_set)                   # Output “{'Prince', 'Techs', 'Data'}”.
print(len(sample_set))              # Output “3”.

# set.remove(obj): Remove a specified element from a set.
sample_set.remove('Data') # Delete element “Data”.
print(sample_set)                   # {'Prince', 'Techs'}

list2 = [1,3,1,5,3]
print(list(set(list2)))             # Output [1,3,5]. The unique set elements are used to de-duplicate the list.
print(list(set(list2)))             # Output the [1,3,5] list.
sample_set = frozenset(sample_set) # Immutable set.
```

2.2.2 Deep Copy and Shallow Copy

In Python, value assignment to an object is actually a reference to the object. When an object is created and assigned to another variable, Python does not copy the object but copies its reference.

In this experiment, the copy module in Python is used to differentiate deep copy from shallow copy.

```
import copy                                 # The current program calls the copy module.
Dict1 = {'name':'lee', 'age':89, 'num':[1,2,8]}  # Create a dictionary.
Dict_copy = Dict1.copy() # Shallow copy.
Dict_dcopy = copy.deepcopy(Dict1)           # Deep copy.
Dict2=Dict1                                 # Shallow copy. Directly assign a value to the object.
Dict1['num'][1] = 6                         # Change the value of the nested list in the original data.
print('Dict1:'+str(Dict1)+"\n",'Dict_copy:'+ str(Dict_copy)+"\n",'Dict_dcopy:'+ str(Dict_dcopy)+
"\n",'Dict2:'+str(Dict2))
```

Output:

```
Dict1:{‘name’:’lee’, ‘age’:89, ‘num’:[1,6,8]}
Dict_copy :{'name':'lee', 'age':89, 'num':[1,6,8]}      # Shallow copy modifies data..
Dict_dcopy :{'name':'lee', 'age':89, 'num':[1,2,8]}     # Deep copy does not modify data.
Dict2:{'name': 'lee', 'age': 89, 'num': [1, 6, 8]}      # Value assignment to an object is shallow copy, and data is modified.
```

2.2.3 if Statement

The programming language provides various control structures, which allow more complex execution paths. if statement is one of the control structures, and is used to execute subsequent instructions after a condition is matched.

In this example, a Python script is compiled to receive a score entered by a user and determine the level of the score. You can use if statement in Python to implement this function.

```
# Determine the level of entered score.
# input(): Receive input data.
score = input("Please enter your score.") # The input function receives input, which is a string.

#   try:    except Exception: ... is a Python statement used to capture exceptions. If an exception occurs in the statement in “try”, the statement in “except “is executed.
try:
    score = float(score)                # Convert the score to a number.
    if 100>=score>=90:                  # Check whether the entered value is greater than the score of a level.
        print("Excellent")              # Generate the level when conditions are met.
    elif 90 > score >= 80:
        print("good")
    elif 80>score>=60:
        print("medium")
    elif 60>score>=0:
        print("bad")
    else:
        raise
except Exception:
    print("Enter a correct score.")
```

2.2.4 Loop Statement

A loop statement can execute a statement or statement group multiple times. This example describes two types of loops: for and while.

- for loop

  The for loop can traverse any sequence of items, such as a list or string.

  In this example, a Python script is compiled to print the multiplication table.

  ```
  for i in range(1,10):           # Define the outer loop.
      for j in range(1,i+1):      # Define the inner loop.
          print ("%d * %d = %2d" % (i, j, i*j), end=" \t ")      # Print a string in format to align the print.
      print()                 # The “end” attribute sets the end symbol of the print, which is /n by default.
  ```

  Output:

  ```
  1*1= 1
  2*1= 2 2*2= 4
  3*1= 3 3*2= 6 3*3= 9
  4*1= 4 4*2= 8 4*3=12 4*4=16
  5*1= 5 5*2=10 5*3=15 5*4=20 5*5=25
  6*1= 6 6*2=12 6*3=18 6*4=24 6*5=30 6*6=36
  7*1= 7 7*2=14 7*3=21 7*4=28 7*5=35 7*6=42 7*7=49
  8*1= 8 8*2=16 8*3=24 8*4=32 8*5=40 8*6=48 8*7=56 8*8=64
  9*1= 9 9*2=18 9*3=27 9*4=36 9*5=45 9*6=54 9*7=63 9*8=72 9*9=81
  ```

- while loop

In Python programming, the while statement is used to cyclically execute a program. That is, under a certain condition, a program is cyclically executed to process the same task that needs to be repeatedly processed.

When the condition is met, the statement is executed cyclically. To end the loop, use break or continue.

```
#   while loop
i = 0                                                   # Create the variable i.
while i<9:                                              # Set a condition for the loop.
        i+=1                                            # The value of i increases by 1 in each loop.
        if i == 3:                                      # Check whether the conditions are met.
                print("End the current loop.")
                continue                                # execute “continue” to end the current loop.
        if i == 5:
                print("End the current big loop.")
                break                                   # End the current big loop.
        print(i)
```

Output:

```
1
2
End the current loop.
4
End the current big loop.
```

2.2.5 Customizing Functions

A function is a piece of code, organized and reusable. It can improve code utilization and modularity of the program. Python provides many built-in functions, such as print(). You can also create functions, that is, user-defined functions.

This section describes how to customize functions.

- Define a function.

  Use the “def” keyword to define a function, which can call the function (print information).

```
def Print_Hello():                  # Function name, no input value.
    print("Hello, Python.")         # Output
Print_Hello()                       # Call function.
```

  Output:

```
Hello, Python.
```

- Parameter transfer for a function and default parameters

  A function can be customized to print parameters transferred in different ways.

```
def hello(greeting='hello',name='world'):      # Set default parameters.
    print('%s, %s!' % (greeting, name))        # Format the output.
hello()                         #hello, world.  If no parameter is specified, the default parameter is used.
Hello ('Greetings')                            # Greetings, world. This is a position parameter.
hello(name='Huawei')                           # hello, Huawei. This is a keyword parameter.
```

```
Hello ('Greetings', 'universe')                    # Greetings, universe. This is a position parameter.
```

Output:

```
hello, world!
Greetings, world!
hello, Huawei!
Greetings, universe!
```

- Length-variable parameters

  A function with length-variable parameters can process parameters more than defined. In this case, * indicates a tuple-type parameter and ** indicates a dictionary-type parameter.

```
def test (a, b, *c, **d):
    print (a)
    print (b)
    print (c)
    print (d)
test (1,2,3,4,5,6,x=10,y=11) # The transfered parameters are automatically distinguished as the tuple or dictionary type.
```

Output:

```
1
2
(3, 4, 5, 6)
{'x': 10, 'y': 11}
```

- Returned values

  The returned value of a function is the result that the function returns to the caller after completing the method. Use the return statement.

```
def plus_one (number):
    return int(number)+1
plus_one (10)
```

Output:

```
11
```

2.2.6 I/O Operations

The standard library of Python is large and provides a wide range of modules. File I/O is one of the built-in modules. For more operations on the standard library, see the official document at https://docs.python.org/3.8/library/index.html.

Python I/O operations are performed to read and write files. Generally, you can use the open function to perform the operations.

File Mode	Description
r	Read-only, default. The file pointer is placed in the file header.
w	Write-only. The file pointer is placed in the file header, and the file overwrites the original content.

File Mode	Description
a	Append. The pointer is placed at the end of the file. If no file exists, a file is created.
rb	Read-only files in binary format, such as audio or image files.
wb	Write-only files in binary format.
ab	Append files in binary format.
r+	Read and write. The file pointer is placed in the file header.
w+	Read and write. If a file exists, the file will be overwritten. If no file exists, a file is created.
a+	Append, read, and write. If no file exists, a file is created.

- Write a file

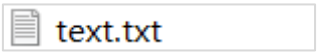
```
f = open("text.txt", 'w') # Open the text.txt file. If the file does not exist, a new file is created. w: write mode
Str = input("Please enter the content to be written.") # The built-in function “input” gets the input of the console.
f.write(Str)        # Write “Str” to file f.
f.close()           # Close the file.
```

Output:

```
Please enter the content to be written. python file operation
```

Enter the content to be written, “Python file operation”.

The **text.txt** file is generated in the same directory.



- Read a file

The read method is used to read files. If read() is not specified, all content is read.

```
f = open("text.txt", 'r')       # r indicates that the file is read.
print(f.read(6))                # Read six characters and move the cursor six characters backward.
print(f.read())                 # Read from the current position of the cursor to the end.
f.close()
```

Output:

```
Python
file operation
```

- Use the context manager to operate the file.

The context manager defines the scope of using the resources. Resources are allocated at the start of the code and released at the end of the code. This

simplifies code and improves its readability. The context manager works with the “with” statement. The basic syntax is as follows:

“with” context expression [“as” resource object]: # “as” indicates that the return value of the context is assigned to the resource object.

Object operation:

```
with open("text1.txt", 'w') as f:              # Use the “with” statement to write data to the file.
    f.write("python file operation")
with open("text1.txt", 'r') as f:              # Use the “with” statement to read the file content.
    print(f.read())
```

Output:

```
Python file operation
```

There are more methods for file operations, such as reading a line of readlines, positioning files with “tell”, and finding files with “seek”. For details, see Python Tutorial.

2.2.7 Exception Handling

Errors that occur during program execution are called exceptions. Common exceptions include syntax errors (SyntaxError), undeclared variables (NameError), and access to unknown attributes (AttributeError). Python has a powerful troubleshooting mechanism to accurately report error information, which helps developers locate faults.

Python uses the “try-except” statement to handle exceptions. The “try” statement is used to check exceptions, and the “except” statement to capture exceptions.

- Create exceptions.

  First, let's create an exception where the divisor in a division is 0.

```
num1 = input('Please enter the first digit:')
num2 = input('Please enter the second digit:')
print('The first number divided by the second number is %f'%(int(num1)/int(num2)) #%f    indicates a floating point number.
```

  Output:

```
Please enter the first digit: 10
Please enter the second digit: 0
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-10-5cc9c0d19753> in <module>
      1 num1 = input('Please enter the first digit:')
      2 num2 = input('Please enter the second digit:')
----> 3 print ('The first number divided by the second number is %f'%(int(num1)/int(num2)))
      4
ZeroDivisionError: division by zero
```

  The cause of exception is **ZeroDivisionError**. You can capture this exception to keep the program running properly.

- Use “try-except” to generate code.

The “try-except” statement is used to capture the ZeroDivisionError exception and display “The second digit cannot be 0”.

```
try:
    num1 = input('Please enter the first digit:’)
    num2 = input('Please enter the second digit:')
    print('The first number divided by the second number is %f'%(int(num1)/int(num2)))
except ZeroDivisionError:
    print('The second digit cannot be 0.')
```

Output:

```
Please enter the first digit: 10.
Please enter the second digit: 0.
The second digit cannot be 0.
```

The exception is successfully captured. If multiple exceptions need to be captured, you can use multiple “except” statements in parallel to capture different exceptions.

- Capture all exceptions.

  It is very difficult to define each exception. The exception class is the parent class of all exceptions and can represent all exceptions.

```
try:
    num1 = input('Please enter the first digit:')
    num2 = input('Please enter the second digit:')
    print('The first number divided by the second number is %f'%(int(num1)/int(num2)))
except Exception as result :
    print('Captured exception: %s' % result)
```

  Output:

```
Please enter the first digit: hello.
Please enter the second digit: 0.
Captured exception: invalid literal for int() with base 10: 'hello'
```

- No exception is captured (else and finally).

  If no exception is captured in the program, the program executes the “else” statement. Sometimes this capture operation has to be terminated regardless of whether an exception has been caught or not. In this case, the “finally” statement can be used.

  In Python, the complete exception handling format is as follows:

```
try:
    # Statement
except:
    # Code with exception
except:
    # Another code with exception
else:
    # Code without exception
finally:
    # Code that must be executed at last, regardless of exception
```

2.2.8 Object-Oriented Programming (Class)

Class and object are important concepts in object-oriented programming. An object is specific and a class is general to regard a group of objects with the same characteristics and behavior.

For example, in the first case, Dog is a class and Husky is an object of the class. Husky has all the features and behaviors of the Dog class.

2.2.8.1 Creating and Using Classes

Create the Dog class.

Each instance created based on the Dog class stores the name and age. We will give each dog the sit () and roll_over () abilities.

```
class Dog():                                    # Use the keyword class to declare a class.
    """Simulate dog behavior (method)"""
    def sit(self):                              # Use a function to define the method of a class. “self” indicates itself and is always the first parameter.
        """Simulate a dog sitting when ordered"""
        print("Dog is now sitting") # In the method, “self” is used to access the name attribute.
    def roll_over(self):
        """Simulate a dog rolling over when ordered."""
        print("Dog rolled over!")
dog = Dog()                         # Create an object and use Dog to save its reference.
dog.name = "Husky"                  # Add an attribute that indicates the name, which is Husky.
dog.sit()                           # Call the method.
dog.roll_over()
print (dog.name)
```

Output:

```
Dog is now sitting
Dog rolled over!
Husky
```

2.2.8.2 Attributes of Class

In the previous case, if you want to create an object of the Dog class, you need to use “object name.attribute name” again, which is troublesome. To solve this problem, Python provides a constructor method **__init__** (with two underscores before and after) to initialize the class.

In this example, the name and age attributes are initialized to the Dog class.

```
class Dog(): # Use the keyword class to declare a class.
        "Simulate the attributes and behaviors of dog (method)"
    def __init__ (self,name,age): # Initialize class attributes. The first parameter is always self, indicating the object itself.
        """Initialize the name and age attributes."""
        self.name = name
        self.age = age
    def sit(self): # Use the function to define a method for class to carry the self parameter.
        """Simulate a dog sitting when ordered."""
```

```
            self.name+" is now sitting") # The method uses “self” to access the name attribute.
    def roll_over(self):
            """Simulate a dog rolling over when ordered."""
            print(self.name+" rolled over!")
dog = Dog("Husky",2) # Create an object and use Dog to save its reference.
print (dog.age) # Access attributes.
dog.sit() # Call the method.
dog.roll_over()
```

Output:

```
2
Husky is now sitting
Husky rolled over!
```

2.2.8.3 Encapsulating Classes

The process of hiding the details of properties, methods, and method implementations is called encapsulation. Attributes are defined as private to prevent unauthorized value assignment.

Private attributes are implemented by __ (two underscores).

```
class Dog():
    def __init__ (self,name,age):
        self.name = name
        self.__age = age      # Set age to a private attribute __age.
    def get_age(self):
        return self.__age
dog = Dog ("Husky",2)
print (dog.name)
dog.get_age()                 # Call the get_age() method and return parameters.
print (dog.__age)             # The program reports an error, indicating that the __age attribute does not exist
and cannot be directly called by external systems.
```

Output:

```
Husky
2
AttributeError                            Traceback (most recent call last)
<ipython-input-23-374e764e1475> in <module>
      5 dog = Dog('Husky', '2')
      6 print (dog.name)
----> 7 print (dog.__age)
      8
AttributeError: 'Dog' object has no attribute '__age'
```

2.2.8.4 Inheriting Classes

One of the main benefits of object-oriented programming is code reuse, which is achieved through inheritance. Inheritance can be understood as the type and subtype relationships between classes.

In this example, the subclass Husky of class Dog is constructed to inherit the attributes and methods of the parent class. The method of subclass (parent class)

is used. If a subclass needs to inherit multiple classes, the method is subclass (parent class 1, parent class 2). In this example, a subclass inherits one class.

```
class Dog():
    def __init__ (self,name,age):
        self.name = name
        self.__age = age # Set age as a private attribute __age.
    def get_age(self):
        return self.__age
class Husky(Dog): # Declare a subclass Husky and a parent class Dog.
    pass
mydog = Husky('Larry','3') # The subclass inherits the construction method of the parent class.
print (mydog.name) # The subclass inherits the name attribute of the parent class, but cannot inherit private attributes.
mydog.get_age() # The subclass inherits the get_age() method of the parent class.
```

Output:

```
Larry
3
```

2.2.8.5 Class Polymorphism

In Python, polymorphism refers to the use of objects without considering object types. Python does not care about whether an object is a parent class or subclass, but only cares about the behavior (method) of the object.

In this example, the parent class is Dog and has the shout method. The subclasses of Dog, Husky and Tibetan Mastiff, rewrite the shout method of the parent class. A function sound can be defined. When calling the shout method, the program does not care whether the object is a parent class or subclass, but cares only whether they all have the shout method.

```
# The parent class is Dog and the sound is Dog shouts.
class Dog():
    def __init__ (self,name,age):
        self.name = name
        self.__age = age
    def get_age(self):
        return self.__age
    def shout(self):
        print ('Dog shouts')

# The subclass is Husky, shouting as woo.
class Husky(Dog):
    def shout(self):
        print('woo')

# The subclass is Tibetan Mastiff, shouting as barking.
class Tibetan_Mastiff(Dog):
    def shout(self):
        print('barking')

#   Define a function sound to call the shout method.
def sound(dog):
```

```
    dog.shout()

# dog1 is the object of Husky.
dog1 = Husky('Larry',3)
sound(dog1)

# dog2 is the object of Dog.
dog2=Dog('Richard',2)
sound(dog2)

# dog3 is the object of Tibetan Mastiff.
dog3= Tibetan_Mastiff('Sylvia',4)
sound(dog3)
```

Output:

```
Woo
Dog shouts
Barking
```

3 Python - Advanced

3.1 Introduction

This experiment uses independent code practices of each module to help readers master the advanced Python3 syntax.

- Regular expressions
- Decorators
- Iterators
- Generators
- Multiple tasks

3.2 Code Practices

3.2.1 Regular Expressions

A regular expression is a special character sequence that helps you easily check whether a string matches a pattern. In Python, the re module is used to implement the regular expression function.

Table 3-1 Basic regular expression syntax

Symbol	Implication	Example
^	Start position to match a string	
$	End position to match a string	
*	Zero to multiple matches	
+	One or more matches (at	

Symbol	Implication	Example
	least one match)	
?	Zero or one match	
.	Match with a single character	
|	Or	a|b: match with a or b
( )	All characters in the parentheses are matched.	(abc): match with abc
[ ]	One character in the brackets is matched.	[0-9 a-z A-Z]
{ }	Limit of matches	{n}: match with n characters {n,}: match with n or more characters {n,m}: match with n characters at least and m characters at most
\	Escape character	*: match with the asterisk (*)
\w	Match with letters and digits	
\W	Match with non-letters and digits	
\d	Match with digits	
\D	Match with non-digit characters	

Table 3-2 Flags of Python regular expressions

Flag	Description
re.I	Case-insensitive matching
re.M	Multi-line matching, which affects ^ and $.
re.S	Make . match with all characters, including newline characters.
re.U	Parses characters based on the Unicode character set. This flag affects \w, \W.

This guide briefly describes how to use regular expressions. For more information, see official documents.

3.2.1.1 re.match

The re.match function attempts to match a pattern from the start position of the string. If the match is not successful, match() returns none.

Function syntax:

re.match(pattern, string, flags=0)

pattern indicates an expression matched. string indicates the object to be matched. flags is a flag bit, for example, whether it is case sensitive. For details, see the tutorial.

Example:

```
import re
print(re.match('www', 'www.huawei.com'))                # Match www at the start position by default.
print(re.match('com', 'www.huawei.com'))                # No match
print(re.match('.*com$','www.huawei.com'))              # Match an object whose name ends with .com.
```

Output:

```
<re.Match object; span=(0, 3), match='www'>
None
<re.Match object; span=(0, 14), match='www.huawei.com'>
```

3.2.1.2 re.search

The re.search expression scans the entire string and returns the first successful match.

Function syntax:

re.search(pattern, string, flags=0)

Example:

```
import re
print(re.search('www', 'www.huawei.com'))      # Search for www from the start.
print(re.search('com', 'www.huawei.com'))      # Search for com to the end.
print(re.search('.*com$','www.huawei.com'))    # Match an object whose name ends with .com.
print(re.search('COM', 'www.huawei.com',re.I)) # Case insensitive.
```

Output:

```
<re.Match object; span=(0, 3), match='www'>
<re.Match object; span=(11, 14), match='com'>
<re.Match object; span=(0, 14), match='www.huawei.com'>
<re.Match object; span=(11, 14), match='com'>
```

3.2.1.3 Retrieval and Replacement

The re module of Python provides the re.sub expression to replace the matching items in the string.

Syntax:

re.sub(pattern, repl, string, count=0, flags=0)

pattern indicates regular expression, repl indicates the replacement string, which can also be a function, and string indicates the original object for search. Count refers to the maximum number of replacements (the value 0 indicates all).

Example:

```
import re
print(re.sub('^WWW','support', 'www.huawei.com',flags=re.I)) # Replace www with support.
```

Output:

```
support.huawei.com
```

3.2.1.4 re.compile

The compile function is used to compile regular expressions and generate a regular expression object (pattern), which can be called by the match() and search() functions.

The syntax format is as follows:

re.compile(pattern[, flags])

Example:

```
import re
pattern = re.compile('\d+') # Match with at least one digit.
print (pattern.match('www.huawei123.com')) # Check whether the header contains digits. No match.
The first digit is returned for print (pattern.search('www.huawei123.com')) # Search and return the first digit.
```

Output:

```
None
<re.Match object; span=(10, 13), match='123'>
```

3.2.1.5 findall

findall indicates that all substrings that match the regular expression are found in the string and a list is returned. If no substring is found, an empty list is returned.

The syntax format is as follows:

findall(string[, pos[, endpos]])

string: the string to be matched; Pos: start position of the string, 0 as default; Endpos: end position of the string, string length as default

Example:

```
import re
pattern = re.compile('\d+') # Match with at least one digit.
print(pattern.findall('www.huawei123.com \n 345')) # Search all strings.
```

Output:

```
['123', '345']
```

3.2.1.6 re.split()

The split expression splits a string based on the matched substring and returns a list. This expression is used as follows:

re.split(pattern, string[, maxsplit=0, flags=0])

pattern: regular expression; string: string to be matched; Maxsplit: maximum splitting times, 0 as default, indicating no limit on splitting times

Example:

```
import re
s = re.split('\W+', 'www.huawei.com') # \W+ indicates that non-English characters and digits are split for one or more times.
print(s)
```

Output:

```
['www', 'huawei', 'com']
```

3.2.2 Decorators

As essentially a Python function, a decorator is special as it returns a function. It can extend the functionality without changing the existing function. The syntax of a decorator starts with @. It is used as follows:

- Build an initial function.

  ```
  Here is a function to output “I am learning Decorators”.
  def test():
      print ("I am learning Decorators.")
  test()
  ```

  Output:

  ```
  I am learning Decorators.
  ```

- Extend functionality of the function.

  Before extending functionality of the function, “I learned Python Data Structures” has to be output.

  Generally, the solution is as follows:

  ```
  def test():
      print ("I learned Python Data Structures.")
      print ("I am learning Decorators.")
  ```

  This solves the problem, but changes the existing function. If you want to solve the problem without changing the original function, you can use a decorator.

- Decorate the function using a decorator.

  Define a new function decorator() whose parameter is a function and returned value is also a function. The func() function that is used as a parameter is executed inside the returned wrapper() function. Add @decoder to the start of the test() function to get a new function.

```
def decorator(func):
    def wrapper():
        print ("I learned Python Data Structures.")
        func()
    return wrapper
@decorator
def test():
    print ("I am learning Decorators.")
test()
```

  Output:

```
I learned Python Data Structures.
I am learning Decorators.
```

3.2.3 Generators

Sometimes, the number of elements in a set is large. If all the elements are read and stored in the memory, there will be excessive hardware requirements. In this case, you can use a generator to get the subsequent elements in a set through loop.

In Python, a function that uses the yield keyword is called generator. A generator can be used as follows:

- Create a generator using list derivation.

  Use parentheses to write a generator.

```
G = (x*2 for x in range(5)) #   Each x*2 in range(5)
print(type(G))
for i in G:
    print (i)
```

  Output:

```
<class 'generator'>
0
2
4
6
8
```

- Create a generator using yield.

  Use the yield keyword to create a generator to generate the Fibonacci sequence.

```
def fib(n):
    current, num1, num2= 0, 0, 1 # Initialize numbers to start from 0, 1. Current means a loop.
    while True:
```

```
        if current > n:                         #   The value is greater than the parameter value and
the loop ends.
            return
        yield num1 # yield turns the function into a generator, and the function is cyclically executed.
        num1, num2 = num2, num1+num2
        current += 1
fib(5)
for i in fib(5):
    print (i)
```

Output:

```
0
1
1
2
3
5
```

- Wake up the generator.

  The next and send methods can be used to wake up the generator. Different from the next method, the send method can also transfer a piece of data to the breakpoint when waking up the generator.

```
def gen():
    i = 0
    while i<5:
        temp = yield i
        print(temp)
        i+=1
f = gen()
next(f)
>>>0
f.send('haha')
>>>haha
>>>1
next(f)
>>>None
>>>2
```

3.2.4 Iterators

Iteration is a powerful function of Python and a way to access sets. An iterator is an object that remembers the traversed positions. An iterator starts from the first element for access until all elements are accessed.

An iterator has two basic methods: iter() and next(). The following is an example of iterator:

- Determine objects that can be iterated.

  An iterator supports only backward iterations. Strings, lists, and tuple objects can be used to create iterators. The value cannot be iterated.

```
# Use the isinstance() method to determine whether an object can be iterated.
from collections import Iterable # Iterable object
```

```
print(isinstance([], Iterable))
print(isinstance('abc', Iterable))
print(isinstance({'a':1,'b':2},Iterable))
print(isinstance(100, Iterable))
```

Output:

```
True
True
True
False
```

- Basic methods of iterator

  The iter() function is used to obtain the iterators of iterable objects. You can use the next() function repeatedly for the obtained iterator to obtain the next data record.

```
l = [1, 2, 3, 4, 5]
l_iter = iter(l)
next(l_iter)
>>>1
next(l_iter)
>>>2
```

- Class implementation iteration

  An iterator implements the __iter__() and __next__() methods in a class.

  The StopIteration exception identifies the completion of iteration and prevents infinite loops.

```
class Count:                          # Declare a class implementation iterator.
    def __iter__(self):               # Implement the __iter__ method.
        self.a = 1                    # The initial value is 1.
        return self
    The def __next__(self):           # Implement the __next__ method. The function can be +1.
        if self.a <= 5:
            x = self.a
            self.a += 1
            return x
        else:
            raise StopIteration       # Iteration is complete.
mycount = Count()
myiter = iter(mycount)
for i in myiter:                      # Use the for loop to traverse the iterator.
    print (i)
```

Output:

```
1
2
3
4
5
```

3.2.5 Multiple Tasks

Multi-task concurrent programming is a common function. Multi-task in Python refers to multi-thread or multi-process.

A thread is a flow of instructions to complete a particular task, and is the smallest executable unit that allocates time. A thread is usually located in a process and consists of a program counter, a stack, a set of registers, and an identifier. Common CPUs support multi-thread processing tasks.

A process contains a main thread and can generate multiple sub-threads. Each thread contains its own registers and stacks. Processes can be single-thread or multi-thread. Generally, an execution instance of a program is a process.

3.2.5.1 Multiple Threads

In Python, the threading module can be used to implement multiple threads.

- Use multiple threads to execute tasks.

```
import threading
from time import sleep,ctime # Use sleep to specify the program waiting time, and ctime to return the local time.
def work1(): # Define the work1 function and output the work1 string at an interval of 1s.
    for i in range(3):
        print("work1 is being executed...%d"%i)
        sleep(1)

def work2(): # Define the work2 function and output the work2 string at an interval of 1s.
    for i in range(3):
        print("work2 is being executed...%d"%i)
        sleep(1)

if __name__ == '__main__':
    print('---Start---:%s'%ctime())

    t1 = threading.Thread(target=work1)      # Thread 1 calls work1.
    t2 = threading.Thread(target=work2 )     # Thread 2 calss work2.
                                             #   Start the thread.
    t1.start()
    t2.start()

    sleep(5)
print('---End---:%s' %ctime())
```

Output:

```
---Start---:Mon Apr 15 10:55:16 2019
work1 is being executed...0
work2 is being executed...0
work1 is being executed...1
work2 is being executed...1
work1 is being executed...2
work2 is being executed...2
---End---:Mon Apr 15 10:55:21 2019
```

- Synchronize threads.

  If multiple threads modify a piece of data at the same time, unexpected results may occur. To ensure correct data, multiple threads need to be synchronized. Lock can be used to implement simple thread synchronization. This object has acquire and release methods. The concept of lock is introduced to implement synchronization. A lock has two states: locked and unlocked. Each time a thread, for example, wants to access shared data, it must get the locked state. If another thread has gotten the locked state, the thread is suspended. In this case, thread synchronization is blocked. Wait until other threads finish their accesses and release the lock, and then continue with the thread.

  Example:

```
import threading
import time
g_num = 0                # Define a global variable and set its value to 0.
def test1(num):          # Define the function test1 and perform the add operation to global variables.
    global g_num         # Use global variables.
    for i in range(num):
        mutex.acquire() # Lock
        g_num += 1
        mutex.release() # Unlock
    print("---test1--- g_num=%d" %g_num)

def test2(num):          # Define the function test2 and perform the add operation to global variables.
    global g_num
    for i in range(num):
        mutex.acquire() # Lock
        g_num += 1       # Perform the add operation.
        mutex.release() #   Unlock

    print("---test2--- g_num=%d" %g_num)
# Create a mutual exclusion lock.
# By default, the global variables are not locked. You can delete the lock and view resource contention.
mutex = threading.Lock()

# Create two threads and add 1,000,000 times for g_num.
p1 = threading.Thread(target=test1, args=(1000000,))
p1.start()

p2 = threading.Thread(target=test2, args=(1000000,))
p2.start()

# Wait for the calculation to complete.
time.sleep(5)

print("After two threads operate the same global variable, the final result is: %s" % g_num)
```

  Output:

```
---test1--- g_num=1868536
---test2--- g_num=2000000
After two threads operate the same global variable, the final result is: 2,000,000.
```

3.2.5.2 Multiple Processes

To perform multi-process operations in Python, the multiprocessing module is required. The process class is similar to the multi-thread class.

```
# coding=gbk                    # Coding declaration
from multiprocessing import Process
import os
import time

nums = [11, 22]                 # Set global variables.

def work1():
    "Code to be executed by subprocess 1. The nums list is extended."
    print("in process1 pid=%d ,nums=%s" % (os.getpid(), nums)) # Get the process ID.
    for i in range(3):
        nums.append(i)
        time.sleep(1)
        print("in process1 pid=%d ,nums=%s" % (os.getpid(), nums))

def work2():
    "Code to be executed by subprocess 2. Process ID is output and list of nums is called."
    print("in process2 pid=%d ,nums=%s" % (os.getpid(), nums))

if __name__ == '__main__':
    p1 = Process(target=work1)
    p1.start()
    p1.join()                   # After this thread is executed, execute other threads.

    p2 = Process(target=work2)
    p2.start()
```

Output:

```
D:\Python Learning>python test.py
in process1 pid=23384 ,nums=[11, 22]
in process1 pid=23384 ,nums=[11, 22, 0]
in process1 pid=23384 ,nums=[11, 22, 0, 1]
in process1 pid=23384 ,nums=[11, 22, 0, 1, 2]
in process2 pid=20588 ,nums=[11, 22]
```

Note: The Jupyter Notebook does not generate the result of this experiment because of the editor. You can use Pycharm to execute code or create a **.py** file to run it in CMD.

Huawei Certification Training

HCIP-Datacom-Network Automation Developer Git Operation Lab Guide

V1.0

Huawei Technologies Co., Ltd.

Trademarks and Permissions

 and other Huawei trademarks are trademarks of Huawei Technologies Co., Ltd.

All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The purchased products, services, and features are stipulated by the contract made between Huawei and the customer. All or part of the products, services, and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, expressed or implied.

Huawei Technologies Co., Ltd.

Address: Huawei Industrial Base
Bantian, Longgang,
Shenzhen 518129
People's Republic of China

Website: http://e.huawei.com

Huawei Certification System

Huawei Certification follows the "platform + ecosystem" development strategy, which is a new collaborative architecture of ICT infrastructure based on "Cloud-Pipe-Terminal". Huawei has set up a complete certification system consisting of three categories: ICT infrastructure certification, platform and service certification, and ICT vertical certification. It is the only certification system that covers all ICT technical fields in the industry.

Huawei offers three levels of certification: Huawei Certified ICT Associate (HCIA), Huawei Certified ICT Professional (HCIP), and Huawei Certified ICT Expert (HCIE). Huawei Certification covers all ICT fields and adapts to the industry trend of ICT convergence. With its leading talent development system and certification standards, it is committed to fostering new ICT talent in the digital era, and building a sound ICT talent ecosystem.

HCIP-Datacom-Network Automation Developer is designed for senior engineers with professional knowledge and skills in network automation development in the datacom field. Passing the HCIP-Datacom-Network Automation Developer certification proves that you are competent for the position of enterprise network automation development engineer and have the capability of using Huawei datacom devices for automatic deployment, development, and O&M of enterprise networks.

The Huawei certification system introduces the industry, fosters innovation, and imparts cutting-edge datacom knowledge.

Huawei Certification
ICT Vertical Certification
行业ICT认证
Finance
Public Safety
Platform and Service
Certification
平台与服务认证
Big Data
AI
IoT
Intelligent
Video Surveillance
Enterprise
Communication
GaussDB
Cloud Computing
Cloud Service
Kunpeng
Application
Developer
ICT Infrastructure
Certification
ICT技术架构认证
Data Center
Storage
Intelligent Computing
Datacom
WLAN
SDN
Transmission
Access
LTE
5G
Security
Huawei Certified ICT Expert
HCIE
Huawei Certified ICT Professional
HCIP
Huawei Certified ICT Associate
HCIA

About This Document

Introduction

This document is designed for the HCIP-Datacom-Network Automation Developer certification training course and is intended for trainees who are going to take the HCIP-Datacom-Network Automation Developer exam or readers who want to understand the basic knowledge and practices of the version control tool Git and Huawei code hosting service CodeHub.

Background Knowledge Required

This document is intended for senior network automation engineers. To better understand this course, familiarize yourself with the following requirements:

- HCIA-Datacom

Lab Environment Description

Environment Description

This lab environment is based on Git, graphical user interface (GUI)-based Git client TortoiseGit, and CodeHub provided on HUAWEI CLOUD.

This document describes how to implement version control using Git through the command-line interface (CLI) and TortoiseGit, how to use CodeHub, and how to perform a Git practice based on the Gitflow workflow.

Instructions

- Register a HUAWEI CLOUD account and complete real-name authentication at https://www.huaweicloud.com/.
- For CodeHub progressive knowledge, visit https://www.huaweicloud.com/en-us/product/codehub.html

Contents

1 Basic Git Operations

1.1 Introduction

In this experiment, you will perform basic Git operations using the CLI and GUI (TortoiseGit in this experiment).

On completion of this experiment, you will be able to:

- Create a local repository.
- Commit changes to the local repository.
- Create branches, merge branches, and resolve merge conflicts.
- Check version change records.

1.1.1 Procedure

The experiment procedure is as follows:

1. Prepare the environment: Install Git and TortoiseGit on your local PC.
2. Perform basic Git operations using the CLI.
3. Perform basic Git operations using TortoiseGit.

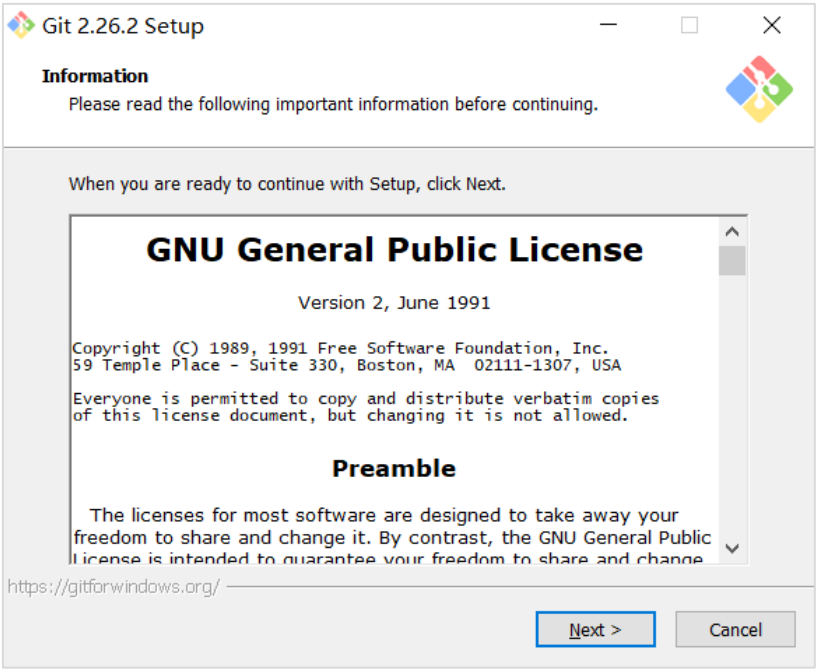
1.2 Environment Preparations

1.2.1 Installing and Configuring Git

1. Install Git.

   Visit https://git-scm.com/download/win to download the 32-bit or 64-bit Git installation package based on the operating system of your local PC.

   Double-click the downloaded installation package. In the displayed installation window, click **Next** and finally **Install**.



2. Configure Git.

   Click the start icon of Windows, enter **Git Bash** in the search box, and press **Enter** to open Git Bash. It is recommended that you fix it to the Windows taskbar.

   The first thing you need to do is to set your user name and email address. Enter the following commands on Git Bash:

```
git config --global user.name "<Your user name>"
git config --global user.email "<Your email address>"
```

NOTE

- Enter an email address in the standard format.
- If you specify **--global** in the preceding commands, Git will always use the user name for anything you do in repositories on the local PC. If you want to configure different names or email addresses for specific repositories, run the commands without specifying **--global**.

After the configuration is complete, you can run the following command to view the configuration:

```
git config -l
```

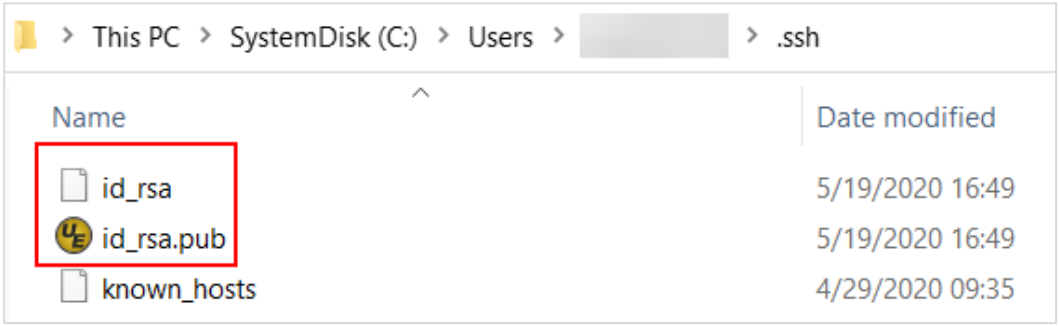
Generate an SSH key pair for authentication with the server where the remote repository resides. Run the following command on Git Bash:

```
ssh-keygen -t rsa -C "<Your email address>"
```

Press **Enter** for three times. By default, the generated private key and public key are saved in **~/.ssh/id_rsa** and **~/.ssh/id_rsa.pub**, respectively.

```
$ ssh-keygen -t rsa -C "wjj1@huawei.com"              # Generate an SSH key pair.
```

```
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/wjj1/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/wjj1/.ssh/id_rsa
Your public key has been saved in /c/Users/wjj1/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:XsRZZsryXcV+9ruJo8QUsYtexviMF9rPFjgVdpI6iSM wjj1@huawei.com
The key's randomart image is:
+---[RSA 3072]----+
|                .+ ...|
|              o *o= o.|
|             . Bo+ =. |
|           E *+=oo   +|
|            So+B=   .o|
|           ...@o..   .|
|            .+ B. . .|
|              o oo...|
|               .o+.o.|
+----[SHA256]-----+
```



So far, Git is installed and configured successfully.

1.2.2 Installing TortoiseGit

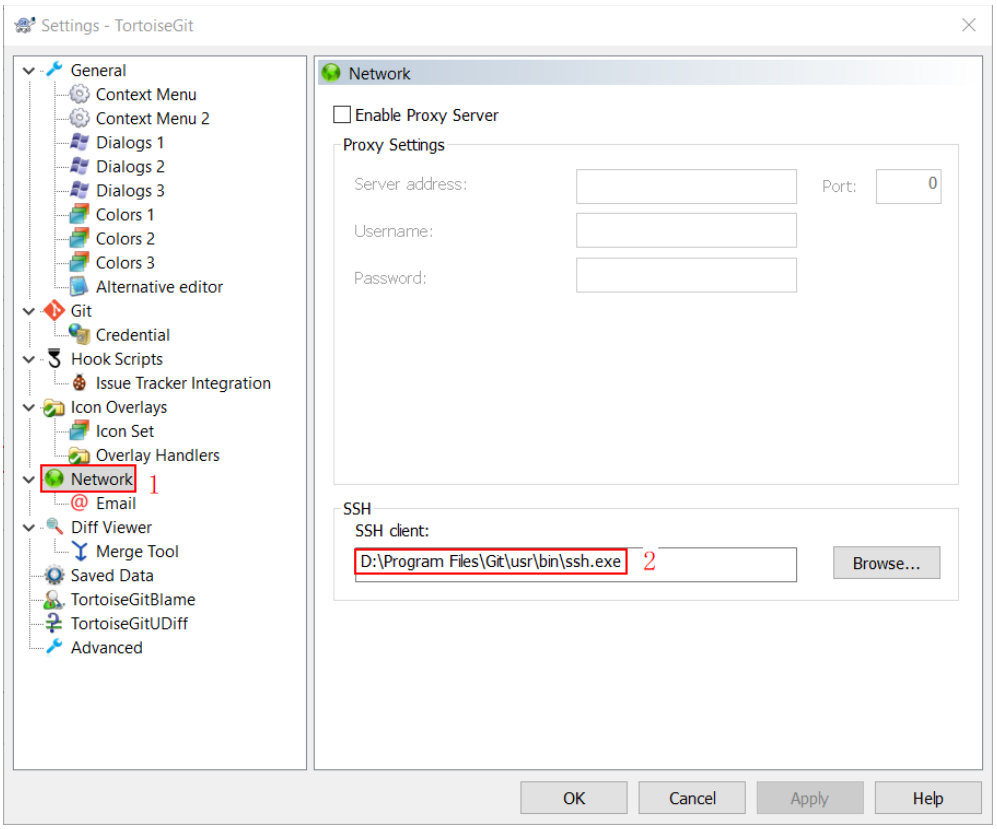
If you are not familiar with common Git commands or use a similar Subversion (SVN) client (such as TortoiseSVN), the TortoiseGit client, a GUI-based Git client, is your better choice. You need to install Git before running it. For details about how to install Git, see section 1.2.1.

1. Install TortoiseGit and start it for the first time.

a. Visit the TortoiseGit official website at https://tortoisegit.org/download to download a 32-bit or 64-bit TortoiseGit installation package based on the operating system of your local PC.
b. Double-click the installation package. In the window that is displayed, click **Next** and finally **Install**. After the installation is complete, click **Finish** to run TortoiseGit for the first time.
c. In the displayed first-startup wizard, select a language, configure the Git executable path (automatically filled with an available Git path), configure the user name and email address, and click **Next**.

2. Configure TortoiseGit.

   TortoiseGit also requires a key pair to authenticate with the code hosting server. The SSH key pair has been generated on Git Bash. Set **SSH client** on the **Network** page of TortoiseGit to **ssh.exe** in the Git installation directory. Then, the SSH key pair generated on Git Bash can be used.



1.3 Git Operations on the CLI

This section describes basic Git operations on the CLI, including creating a local Git repository, committing changes to the local Git repository, checking version change records, creating branches, merging branches, and resolving merge conflicts. Interactions between a local repository and a remote repository are explained in 2 Code Hosting Practice on HUAWEI CLOUD based on CodeHub.

1.3.1 Creating a Local Repository

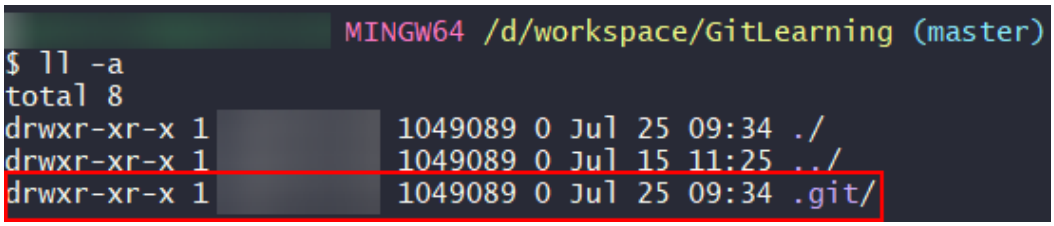
1. Create an empty folder on your local PC.

```
$mkdir GitLearning                    # Create a folder named GitLearning.
$cd GitLearning/                      # Enter the GitLearning folder.
```

2. Run the **git init** command to turn the folder into a local Git repository.

```
$git init                             # Create a local Git repository.
Initialized empty Git repository in D:/workspace/GitLearning/.git/
```

After this command is executed, an empty local Git repository is created. The **.git** folder is generated in the **GitLearning** folder, and is used by Git to track and manage the version library.



1.3.2 Committing Changes to the Local Git Repository

After the local Git repository is created, create a **readme.txt** file and add it to the repository. Run the following command to create the **readme.txt** file:

```
$ vim readme.txt                    # Create a readme.txt file.
```

Enter the following content in the **readme.txt** file.

```
This is a Git repository.
```

Run the **git status** command to view the status of each file in the local Git repository. The command output displays the files that are modified, deleted, and added.

```
$git status                         # Check the status of files in the local Git repository.
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        readme.txt

nothing added to commit but untracked files present (use "git add" to track)
```

Untracked files are the files that are not traced by Git. Run the **git add** command to add the **readme.txt** file to the staging area.

```
$ git add readme.txt                # Add the readme.txt file to the staging area.
warning: LF will be replaced by CRLF in readme.txt.
The file will have its original line endings in your working directory
```

Run the **git status** command again to check the status of files in the local Git repository.

```
$git status                         # Check the status of files in the local Git repository.
On branch master

No commits yet

Changes to be committed:
```

```
  (use "git rm --cached <file>..." to unstage)
        new file:   readme.txt
```

Changes to be committed indicates that changes are not committed to the local Git repository. The preceding command output shows that the **readme.txt** file is staged but not yet committed to the local Git repository. Run the **git commit** command to commit the **readme.txt** file to the local Git repository.

```
$ git commit -m "add readme.txt file"                    # Commit the readme.txt file to the local Git repository.
[master (root-commit) 28b687a] add readme.txt file
 1 file changed, 1 insertion(+)
 create mode 100644 readme.txt
```

The **-m** option indicates your commit message. Based on the commit message, you can easily find the desired commit record from history records.

1.3.3 Displaying Historical Commit Records

Now, you have created your first commit (the **readme.txt** file). It is hard to remember what was changed with each commit after you have created several commits. In this case, you can run the **git log** command to view historical commit records. To view the commit records of a specified file, specify the file name in the **git log** command.

```
$ git log                                                # Display historical commit records.
commit 28b687ac29cc4940710b3bb112b371c6ee46a260 (HEAD -> master)
Author: wjj1 <wjj1@huawei.com>
Date:   Mon May 18 16:35:43 2020 +0800

    add readme.txt file
```

A variety of options are available to the **git log** command. For example, the **--pretty==oneline** option displays each commit on a single line.

```
$ git log --pretty=oneline                               # Display each commit on a single line.
28b687ac29cc4940710b3bb112b371c6ee46a260 (HEAD -> master) add readme.txt file
```

For more information about the **git log** command, visit https://git-scm.com/book/en/v2/Git-Basics-Viewing-the-Commit-History.

1.3.4 Creating a Branch

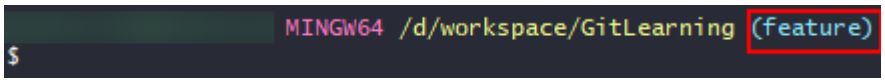
Branching is an important feature of Git. With this feature, you can diverge from the main line of development and continue to do work without messing with that main line. You can run the **git branch** command to create a branch. The following example creates a branch named **feature** and add functions in this branch.

```
$ git branch feature                          # Create a branch named feature.
```

Switch to the feature branch.

```
$ git switch feature                          # Switch to the feature branch.
Switched to branch 'feature'
```

The branch you are working on is displayed on the right of the CLI. The information in the red box in the following figure indicates that you are working on the feature branch.



Add the **helloworld.md** file to the feature branch. This file represents a new function.

```
$ echo "hello world !" > helloworld.md                    # Create the helloworld.md file.
$ git add helloworld.md                                   # Add the helloworld.md file to
the staging area.
warning: LF will be replaced by CRLF in helloworld.md.
The file will have its original line endings in your working directory
$ git commit -m "add helloworld.md file"                  # Commit the helloworld.md
file to the local Git repository.
[feature 9ce7bc3] add helloworld.md file
 1 file changed, 1 insertion(+)
 create mode 100644 helloworld.md
```

Check files in the **GitLearning** folder. You can see that the **helloworld.md** file is added to the feature branch.

```
$ ll                                      # Display files in the GitLearning folder.
total 2
-rw-r--r-- 1 wWX935519 1049089 14 May 18 17:06 helloworld.md
-rw-r--r-- 1 wWX935519 1049089 26 May 18 15:02 readme.txt
```

1.3.5 Merging Branches

After functions are developed in the feature branch, merge the feature branch to the master branch.

1. Switch to the master branch.

```
$ git switch master                       # Switch to the master branch.
Switched to branch 'master'
$ ll                                      # Check files in the master branch. Only the readme.txt
file exists.
total 1
-rw-r--r-- 1 wWX935519 1049089 26  May 18 15:02 readme.txt
```

2. Merge the feature branch to the master branch.

```
$ git merge feature
Updating 28b687a..9ce7bc3
Fast-forward
 helloworld.md | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 helloworld.md
```

The output of the **git merge** command indicates that the **helloworld.md** file is added to the master branch. This file is the one created in the feature branch.

Run the following command to check files in the master branch. The command output shows that the **helloworld.md** file is added to the master branch.

```
$ ll                                          # Check files in the master branch.
total 2
-rw-r--r-- 1 wWX935519 1049089 15   May 18 17:19 helloworld.md
-rw-r--r-- 1 wWX935519 1049089 26   May 18 15:02 readme.txt
```

1.3.6 Resolving Merge Conflicts

Conflicts may occur when branches are merged. If the same file is modified in two branches, Git may not let you merge the branches due to a conflict. In this case, you need to resolve the conflict. The following uses the feature branch and master branch as an example.

1. Switch to the feature branch, modify the **helloworld.md** file, and commit it to the local Git repository.

```
$ git switch feature
$ echo "This is feature branch." >> helloworld.md              # Write content in the
helloworld.md file.
$ git add helloworld.md
$ git commit -m "Modify helloworld.md in feature branch."
```

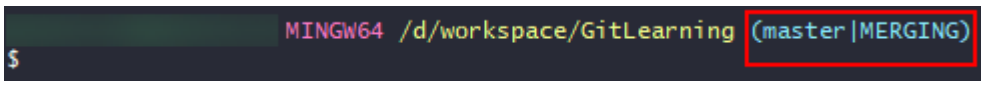
2. Switch to the master branch, modify the **helloworld.md** file, and commit it to the local Git repository.

```
$ git switch master
$ echo "This is master branch." >> helloworld.md
$ git add helloworld.md
$ git commit -m "Modify helloworld.md in master branch."
```

3. Merge the feature branch to the master branch.

```
$ git merge feature
Auto-merging helloworld.md
CONFLICT (content): Merge conflict in helloworld.md
Automatic merge failed; fix conflicts and then commit the result.
```

The **CONFLICT** field indicates that a conflict occurs during branch merging. In this case, the **MERGING** field is displayed at the end of the command.



4. Resolve the conflict.

Run the **git status** command to view the files that conflict with each other.

```
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)
```

```
Unmerged paths:
  (use "git add <file>..." to mark resolution)
        both modified:   helloworld.md

no changes added to commit (use "git add" and/or "git commit -a")
```

The preceding command output indicates that the **helloworld.md** file is modified in both branches. You need to manually modify the file to resolve the conflict. Run the following command to open the conflicting file.

```
$ vim helloworld.md                    # Open the helloworld.md file.
hello world !
<<<<<<< HEAD
This is master branch.
=======
This is feature branch.
>>>>>>> feature
```

The content between **<<<<<<<<HEAD** and **========** are the modification of the **helloworld.md** file in the master branch, and the content between **========** and **>>>>>>>Feature** are the modification of the **helloworld.md** file in the feature branch. In order to resolve the conflict, you must modify the content as expected. In this example, resolve this conflict by replacing the file content with the following:

```
hello world !
This is master branch.
```

Enter **:wq** and press **Enter** to save the configuration and exit.

After the file content is modified, run the **git add** command to add the modified **helloworld.md** file to the staging area. Then the file conflict is resolved.

```
$ git add helloworld.md
```

Run the **git commit** command. In the displayed window, enter **:q** to merge the feature and master branches.

```
$ git commit
[master e0cafd3] Merge branch 'feature'
```

You can run the **git log** command with the **-graph** option to display a graph that clearly shows the branch merging process.

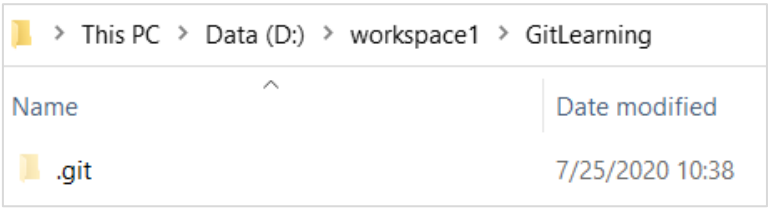
```
$ git log --pretty=oneline --graph
*   5719cfe6dcc9ea7588a980ab2bcbb95af45ef7e5 (HEAD -> master) Merge branch 'feature'
|\
| * c7687eea3e5730e47062e32abe17e3c84e928acc (feature) Modify helloworld.md in feature
branch.
* | 85d92eb800f2d0f8ada4f279142ef0e5120f2148 Modify helloworld.md in master branch.
```

1.4 Git Operations on the GUI

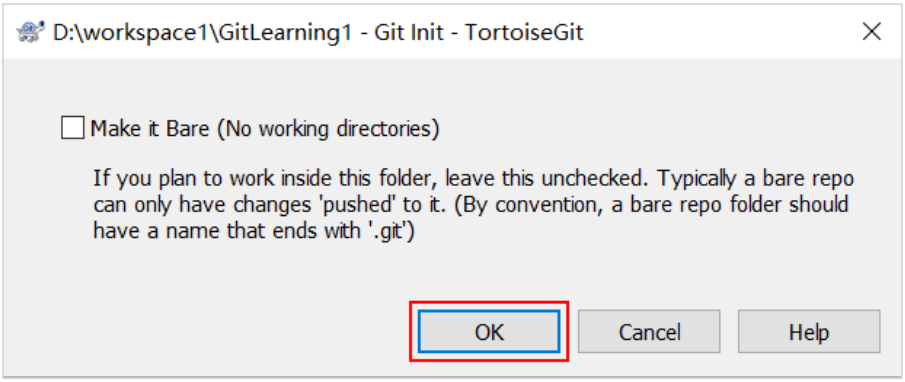
This section describes basic Git operations on TortoiseGit.

1.4.1 Creating a Local Git Repository

1. Create a folder named **GitLearning** and create a **.git** folder in the **GitLearning** folder. The **.git** folder is the folder where the local Git repository resides, and the **GitLearning** folder is the local workspace. Git will check out files to the **GitLearning** folder for developers to work on them.



2. Right-click in the blank area of the **GitLearning** folder, choose **Git Create repository here...** from the shortcut menu, and click **OK**.



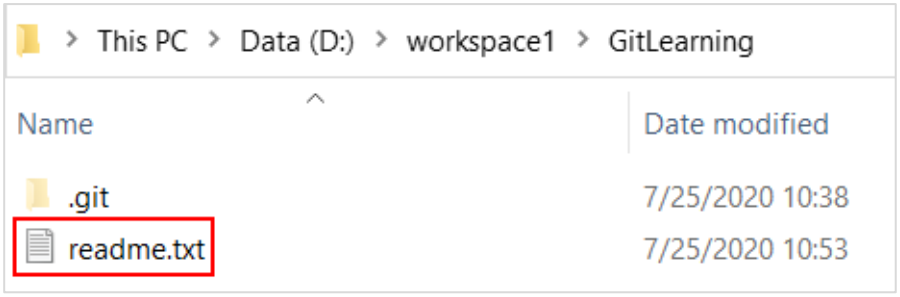
After the local Git repository is created, a green round icon is displayed in the lower left corner of the **GitLearning** folder. TortoiseGit uses this icon to identify the file status, for example, whether the file is modified or conflicted. Compared with the Git CLI, you can directly view the file status on the TortoiseGit GUI, without running the **git status** command.

The created **.git** folder is hidden and is displayed in semi-transparent mode. In addition, some files are generated in the **.git** folder. In this case, the **.git** folder is initialized as a Git repository.

1.4.2 Committing Changes to the Local Git Repository

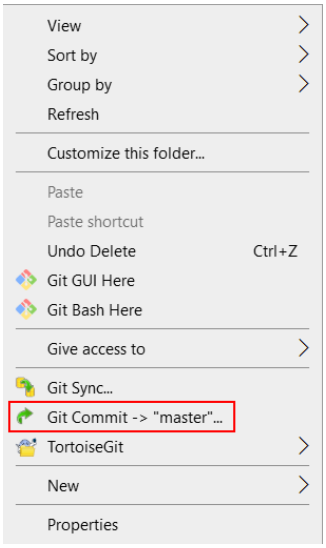
In the **GitLearning** folder, use Notepad to create a **readme.txt** file and enter the following content:

```
This is a Git repository.
```

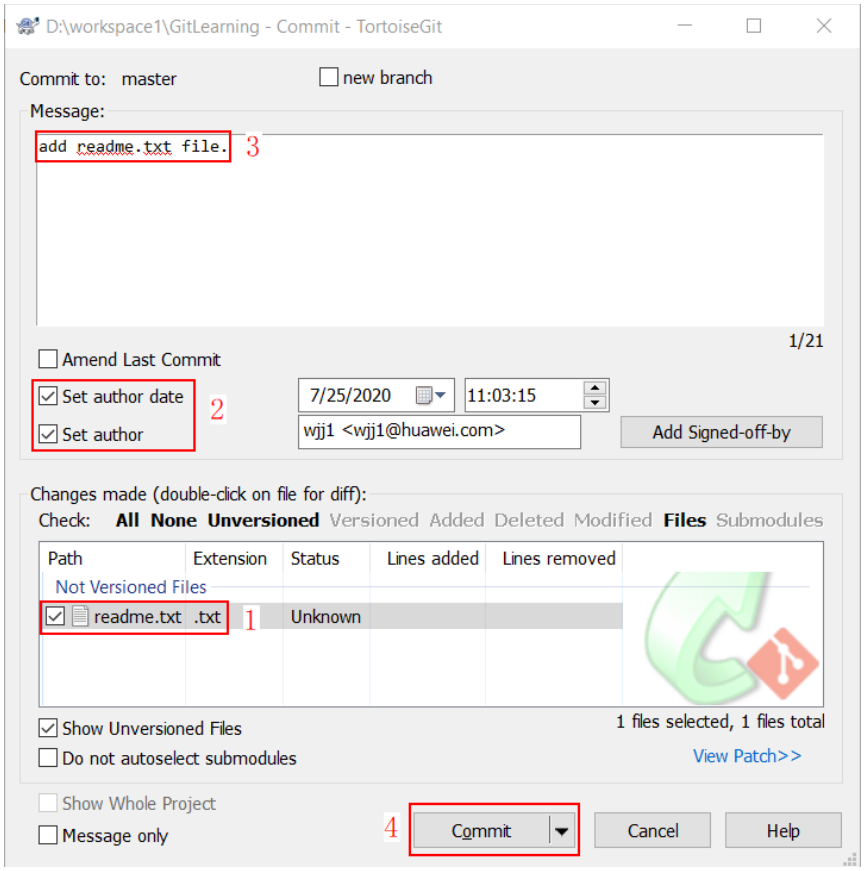


You can easily figure out that the **readme.txt** file is not managed by Git because the file icon does not have any status identifier, as shown in the preceding figure. However, on the Git CLI, you need to run the **git status** to check the file status to check whether a file has been managed by Git.

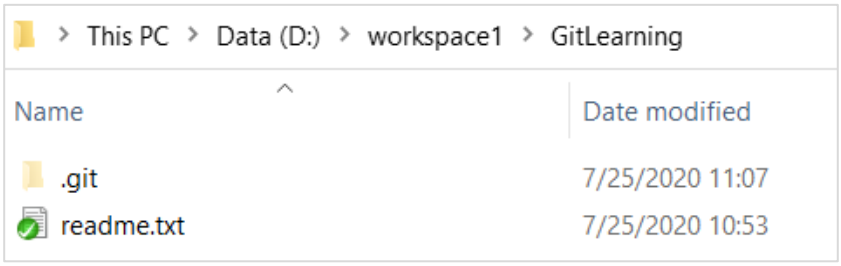
Right-click in the blank area of the **GitLearning** folder and choose **Git Commit** from the shortcut menu.



In the dialog box that is displayed, select the check boxes as shown in the following figure, enter a commit message in the **Message** text box, and click **Commit**.

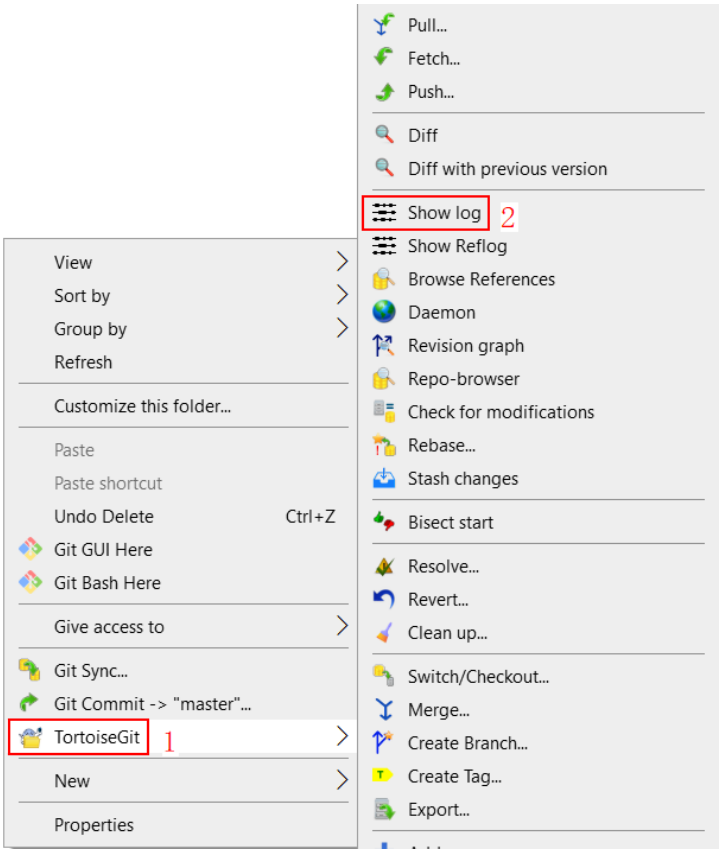


After the commit operation, the **readme.txt** file is committed to the local Git repository and a green icon is displayed on the folder icon, indicating that the file is the latest version in the Git repository. The commit function of TortoiseGit is equivalent to running the **git add** and **git commit** commands in the Git CLI.



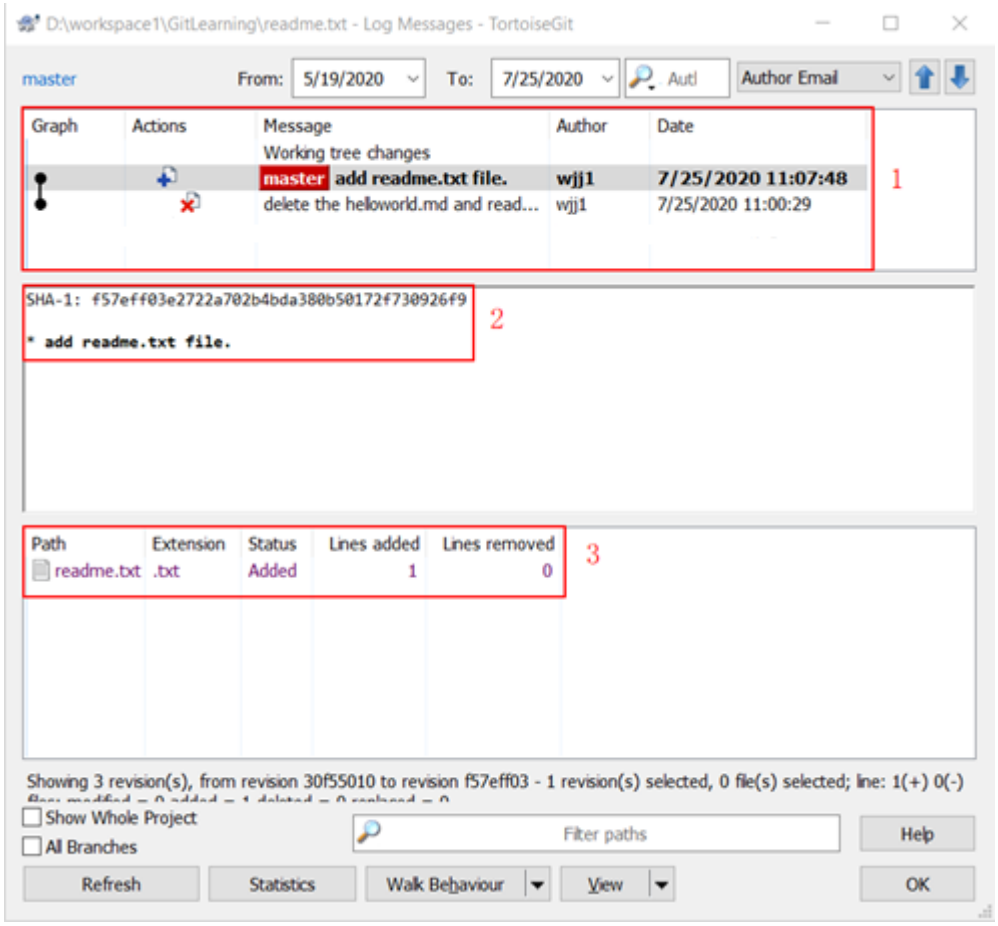
1.4.3 Displaying Historical Commit Records

Now, you have created your first commit (the **readme.txt** file). It is hard to remember what was changed with each commit after you have created several commits. You can use **TortoiseGit** to view historical commit records.



Right-click in the blank area of the folder where the working directory resides and choose **TortoiseGit** > **Show log** from the shortcut menu. Information about historical commit records is available in the displayed window, including:

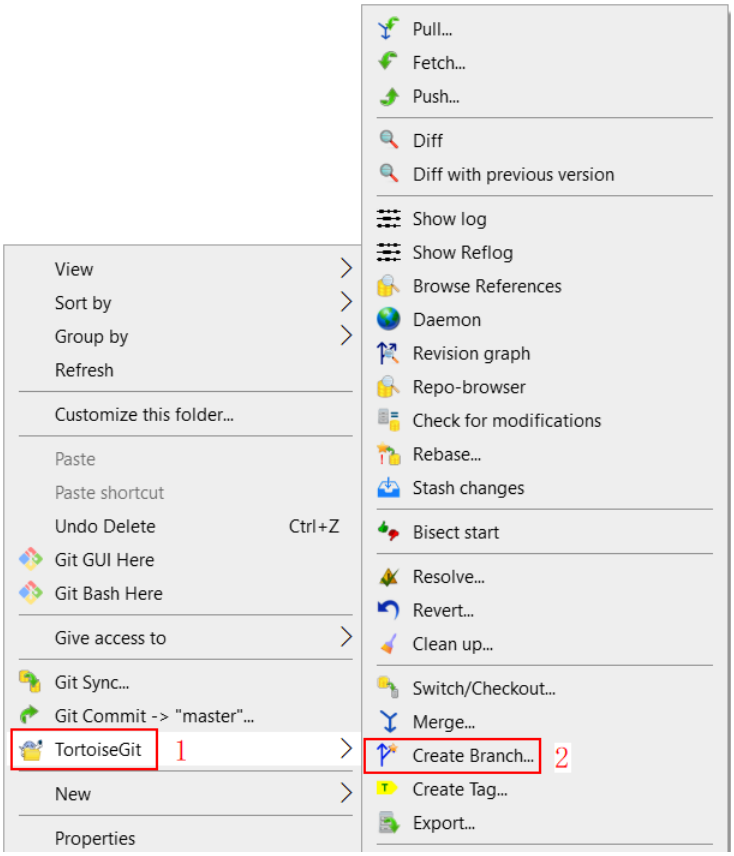
1. A summary of historical commit records, including the commit time, author, and commit message
2. Detailed information about a commit record
3. Files modified in a commit



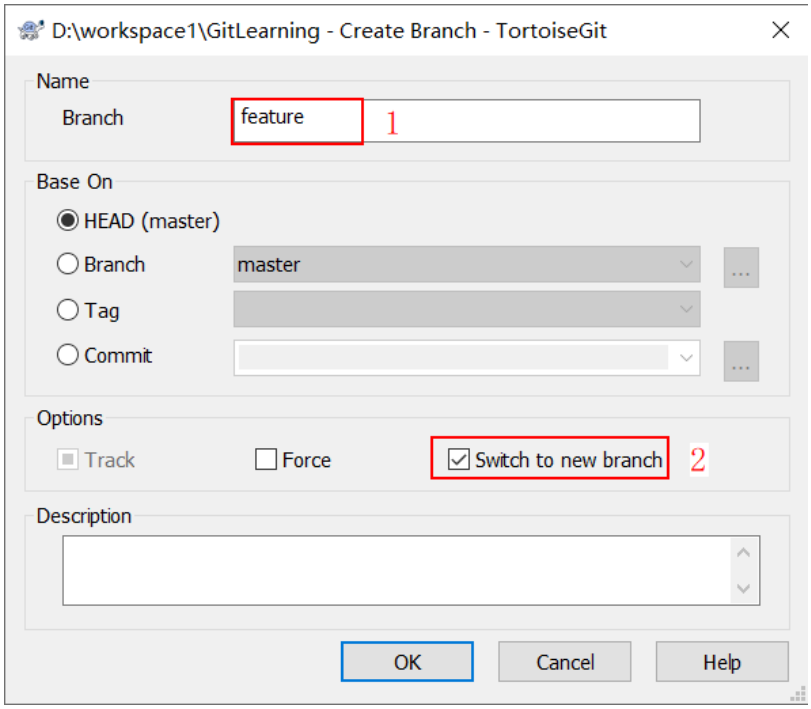
You can also filter historical commit records by the author, commit message, SHA1, and other conditions.

1.4.4 Creating a Branch

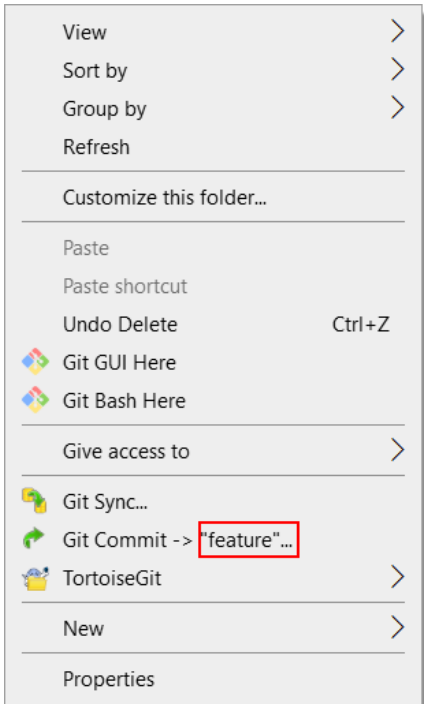
Create a branch named **feature** using TortoiseGit and develop functions in this branch.



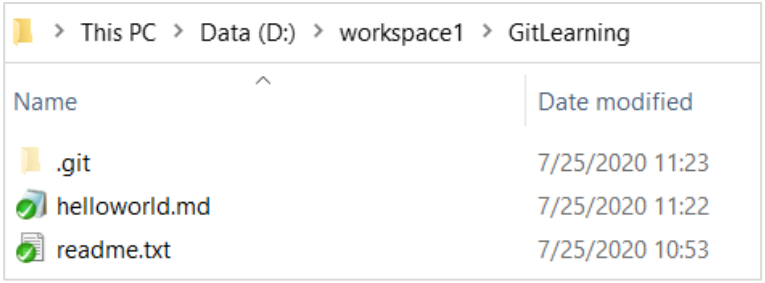
In the dialog box that is displayed, set the branch name to **feature** and select **Switch to new branch**. After the branch is created, the working directory is directly switched to the feature branch.



In the shortcut menu, **Git Commit** is followed by **feature**, indicating that you are working on the feature branch.



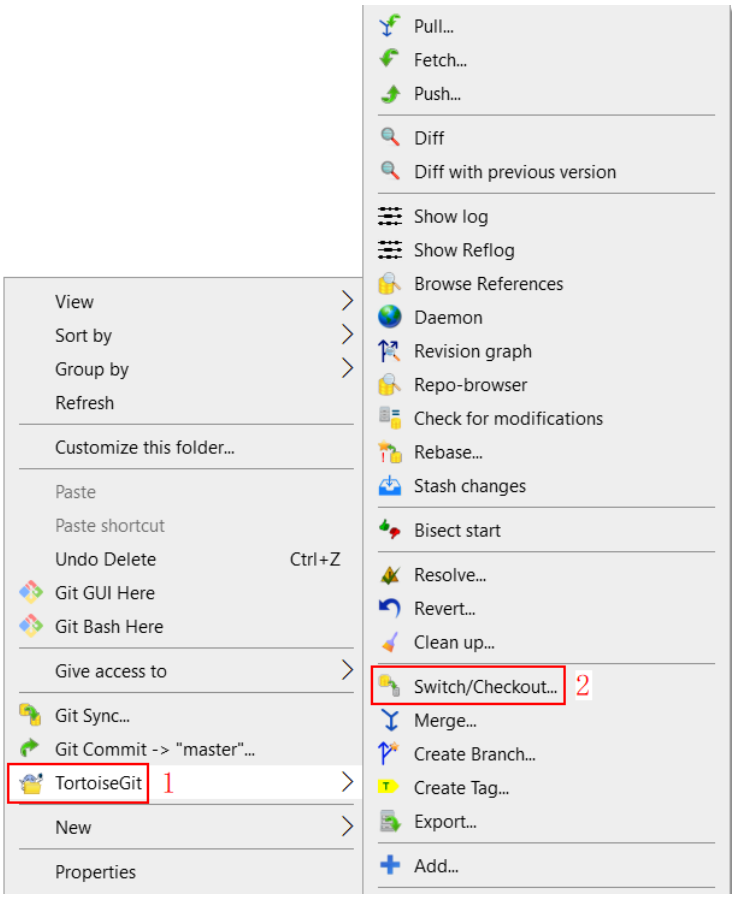
Create the **helloworld.md** file in the feature branch and commit the change to the local Git repository. This file represents a new function. All the involved operations are the same as those described in the previous section. After operations are complete, the structure of the **GitLearning** folder is as follows.

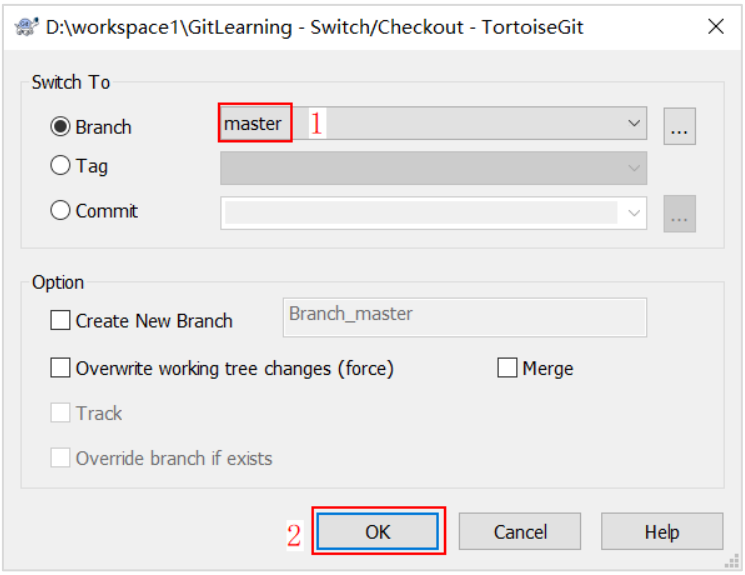


1.4.5 Merging Branches

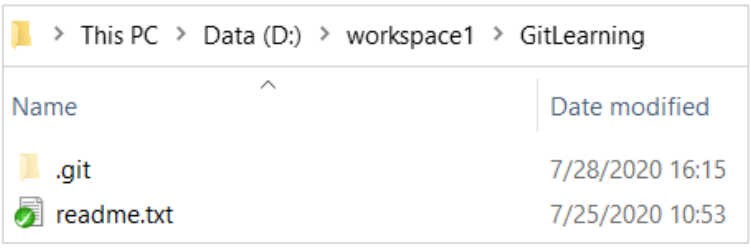
After functions are developed in the feature branch, merge the feature branch to the master branch.

1. Switch to the master branch on TortoiseGit.

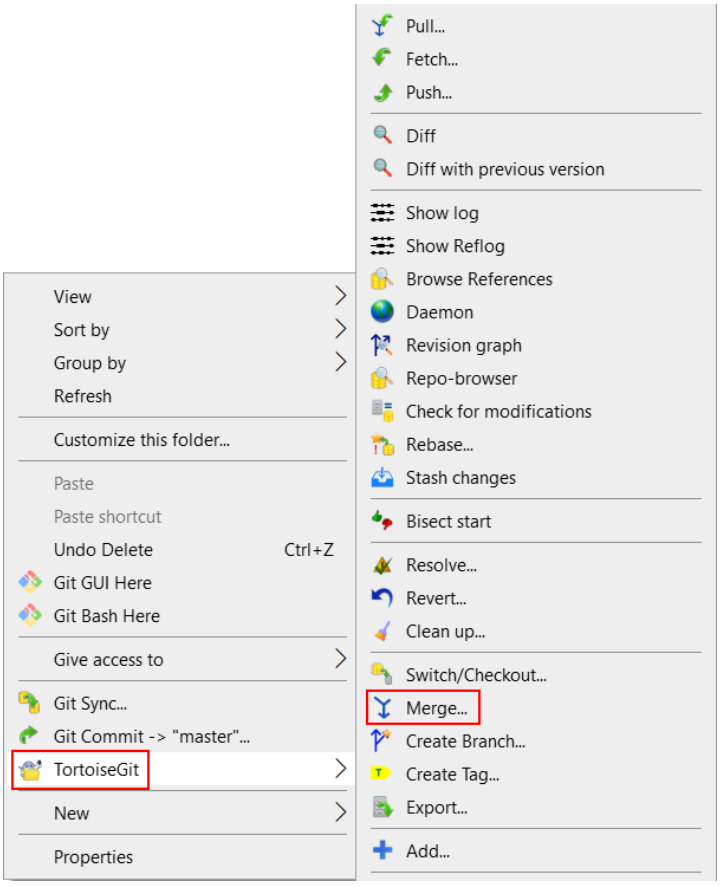




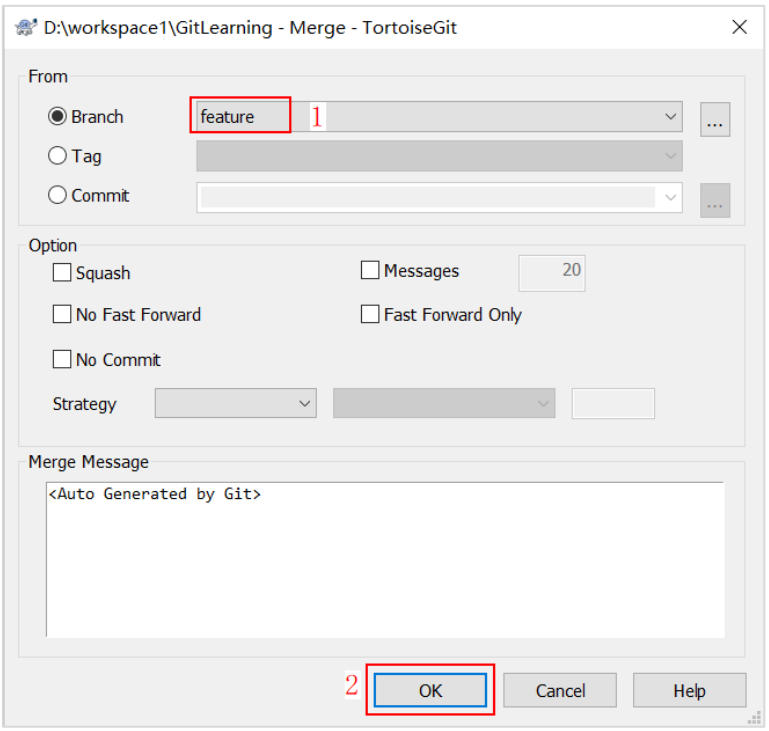
The workspace is switched to the master branch. Only the **readme.txt** file exists.



2. Merge the feature branch to the master branch.



In the dialog box that is displayed, select the feature branch to be merged to the master branch and click **OK**.

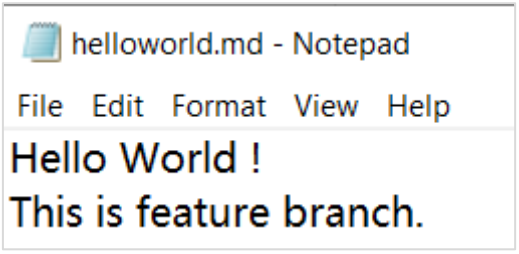


The **helloworld.md** file is added to the workspace of the master branch, which indicates that the feature branch is merged to the master branch successfully.

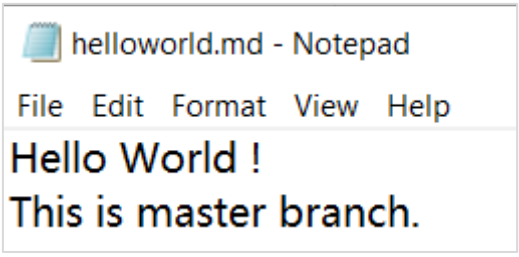
1.4.6 Resolving Merge Conflicts

The following uses the feature branch and master branch as an example to describe how to use TortoiseGit to resolve a merge conflict.

1. Switch to the feature branch, modify the **helloworld.md** file, and commit it to the local Git repository. The file is modified as follows.

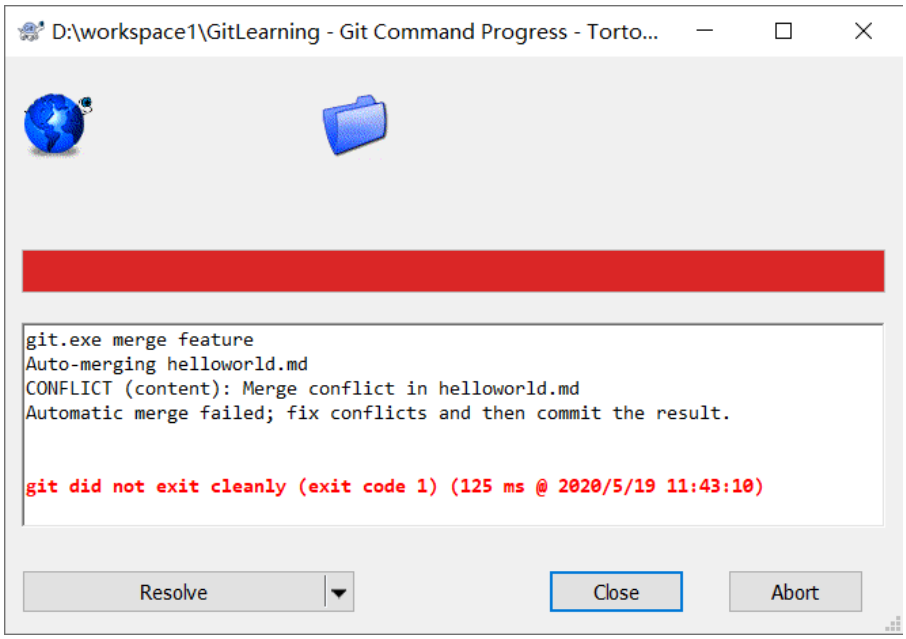


2. Switch to the master branch, modify the **helloworld.md** file, and commit it to the local Git repository. The file is modified as follows.



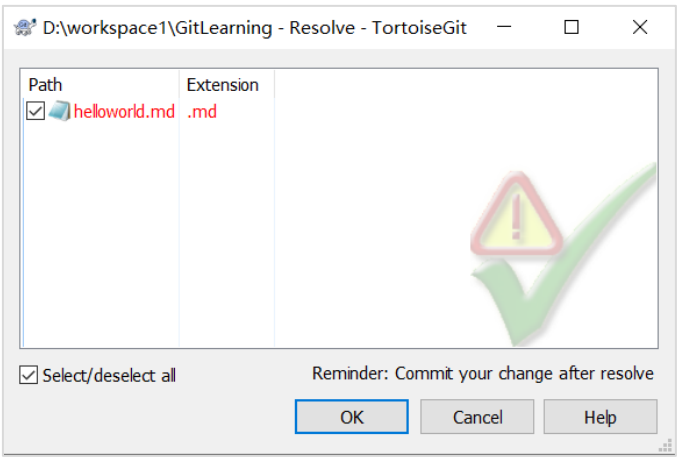
3. Merge the feature branch to the master branch.

   An error message indicating that a merge conflict occurs is displayed.

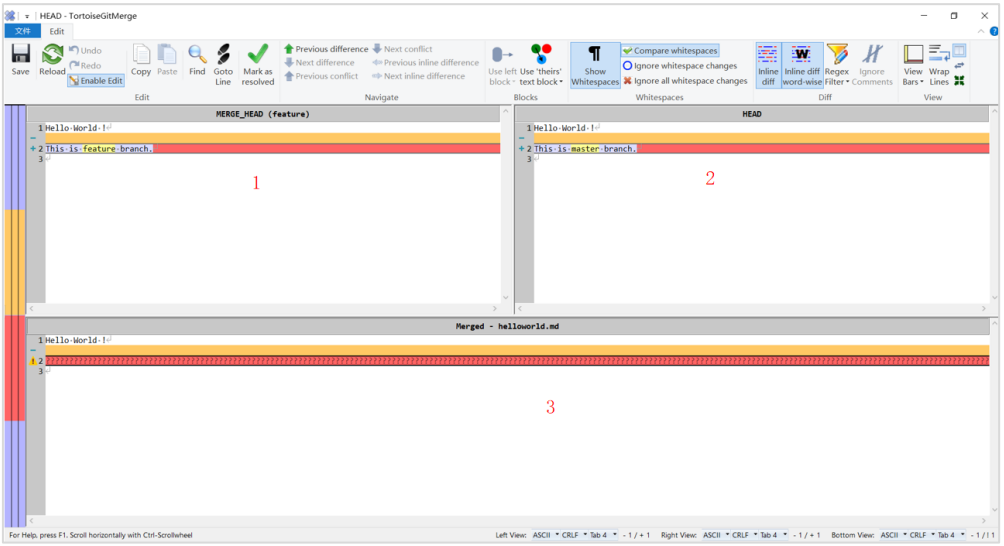


4. Resolve the merge conflict.

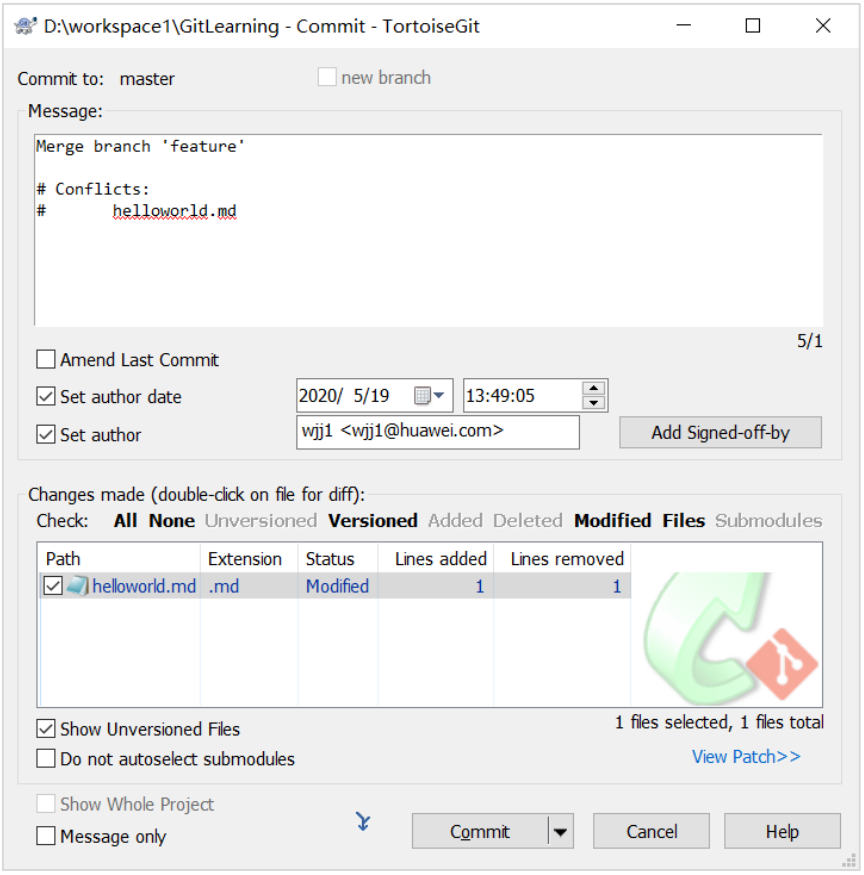
   Click **Resolve**. The following dialog box is displayed, with the conflicting file listed.



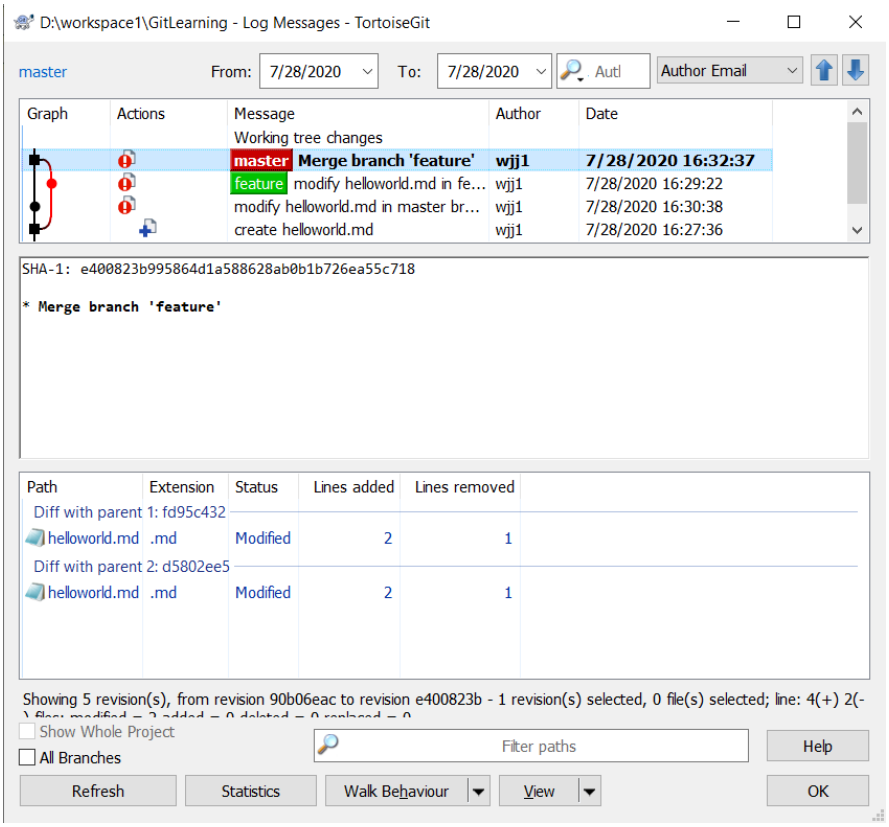
Double-click the file. The following window is displayed. The content in line 1 is the file content in the feature branch, and that in line 2 is the file content in the current branch, that is, the master branch. The content in line 3 is the merged content. Since a merge conflict occurs, you need to manually modify the merged file content. After modifying the merged content in line 3, click **Mark as resolved** on the toolbar and close the window.



Right-click in the blank area of the folder and choose **Git Commit** from the shortcut menu. In the displayed window, click **Commit**. The conflict is resolved and the feature branch is successfully merged to the master branch.



According to the commit record, the feature branch is merged to the master branch successfully.


D:\workspace1\GitLearning - Log Messages - TortoiseGit
master
From: 7/28/2020
To: 7/28/2020
Author Email
Graph
Actions
Message
Author
Date
Working tree changes
master Merge branch 'feature'
wjj1
7/28/2020 16:32:37
feature modify helloworld.md in fe...
wjj1
7/28/2020 16:29:22
modify helloworld.md in master br...
wjj1
7/28/2020 16:30:38
create helloworld.md
wjj1
7/28/2020 16:27:36
SHA-1: e400823b995864d1a588628ab0b1b726ea55c718
* Merge branch 'feature'
Path
Extension
Status
Lines added
Lines removed
Diff with parent 1: fd95c432
helloworld.md
.md
Modified
2
1
Diff with parent 2: d5802ee5
helloworld.md
.md
Modified
2
1
Showing 5 revision(s), from revision 90b06eac to revision e400823b - 1 revision(s) selected, 0 file(s) selected; line: 4(+) 2(-
Show Whole Project
All Branches
Filter paths
Help
Refresh
Statistics
Walk Behaviour
View
OK

2 Code Hosting Practice on HUAWEI CLOUD

2.1 Introduction

When a developer team uses Git for collaborative development, a remote Git repository is required for code commit and update of all developers. CodeHub is a Git-based online code hosting service for software developers. It provides a cloud code repository with functions such as security management, member and permission management, branch protection and merging, online editing, and statistics collection. The service aims to address issues such as cross-distance collaboration, multi-branch concurrent development, code version management, and security.

This experiment guides you through common CodeHub operations. On completion of this experiment, you will be able to:

- Create a remote Git repository on CodeHub.
- Push code from the local Git repository to the remote Git repository on CodeHub.
- Pull code from the remote Git repository on CodeHub to the local Git repository.

2.1.1 Procedure

The experiment procedure is as follows:

1. Prepare the environment: Create a HUAWEI CLOUD account and complete real-name authentication.
2. Create a remote repository on CodeHub.
3. Run Git commands to implement the interaction between the local and remote Git repositories: clone, push, and pull.
4. Use TortoiseGit to implement the interaction between the local and remote Git repositories: clone, push, and pull.

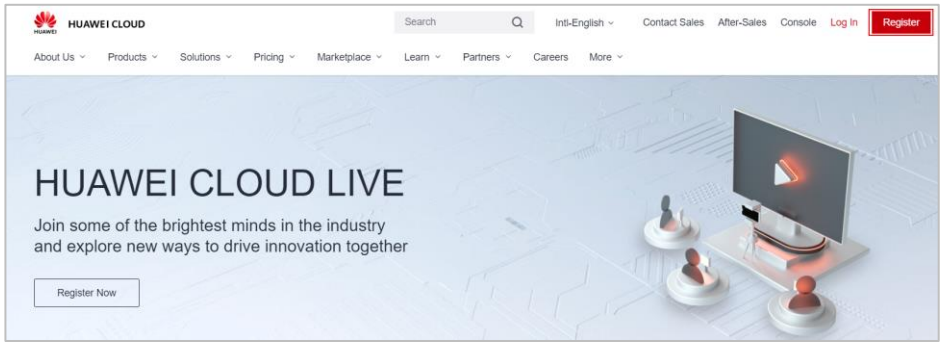
2.2 Environment Preparations

1. Register a HUAWEI CLOUD account and complete real-name authentication.

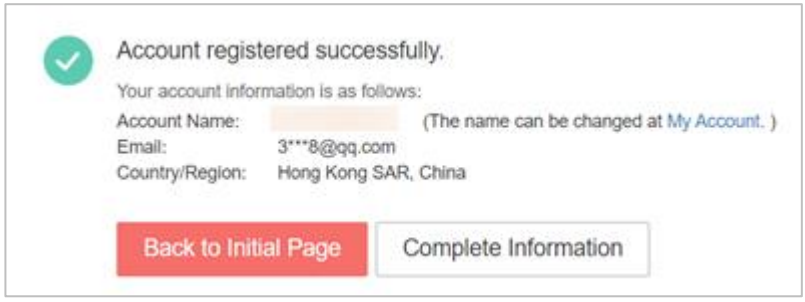
   To ensure account and resource security, you must register an account and pass real-name authentication before using Huawei DevCloud (DevCloud for short) services. DevCloud is a cloud-based R&D platform that integrates Huawei's R&D practices, cutting-edge R&D concept, and advanced R&D tools. CodeHub provided on DevCloud is a Git-based online code hosting service for software developers. It supports code cloning, downloading, committing, pushing, comparing, merging, branching, and reviewing. Currently, DevCloud is available only at the China site. You can select the China site and switch the language to English or download the code to your local host and run it on the local IDE. The following uses DevCloud as an example.

   If you have not registered a HUAWEI CLOUD account, perform the following operations.

- Visit https://www.huaweicloud.com/intl/en-us/.



- Enter your personal information and click **Complete Information**.



- Complete all the information as required.

2. Add an SSH key pair to the CodeHub server.

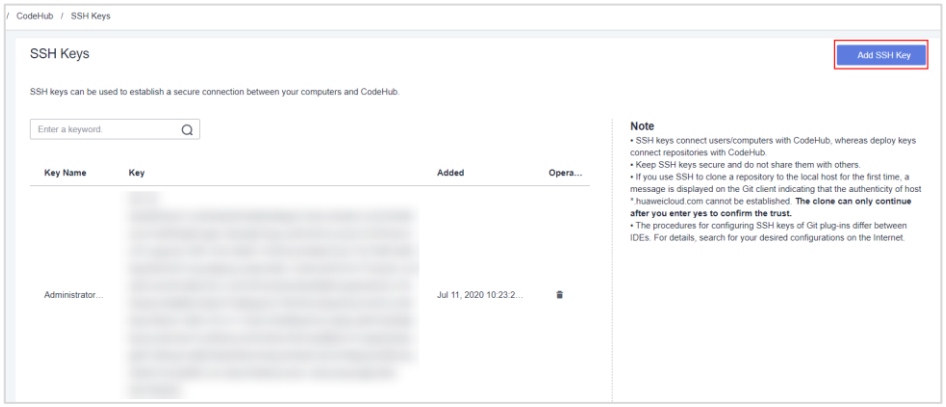
   a. Start Git Bash and run the following command to check the public SSH key in the **~/.ssh/id_rsa.pub** file.

```
$ cat ~/.ssh/id_rsa.pub                    # Display content in the id_rsa.pub file.
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABgQDKvEG8tcNEYW8hdTQrtw51ZF6Hpynj2qxUAGEBovdhH
AoRsxfDpumtKe88pRTqWk08gtoaVK54niNOBEE4jSP73nKdsajCoXlrnaVC/TBTxW+plRFoAk+DSYJE
kWLK/aBk7sjDWsLiZmU2z0M0ifx/Vn5tyyc5JC+IijZfuHX4r11P03DfrXXn21lfYfcXFXyvDrA9ojFlxgX0
Xuc53vo1nUcsng+hqQuXfRLe/mT4nQesS81BD3TAYOv5UUDVj60CdMW2ZqULCl6Yez/OvLWT2h9
```

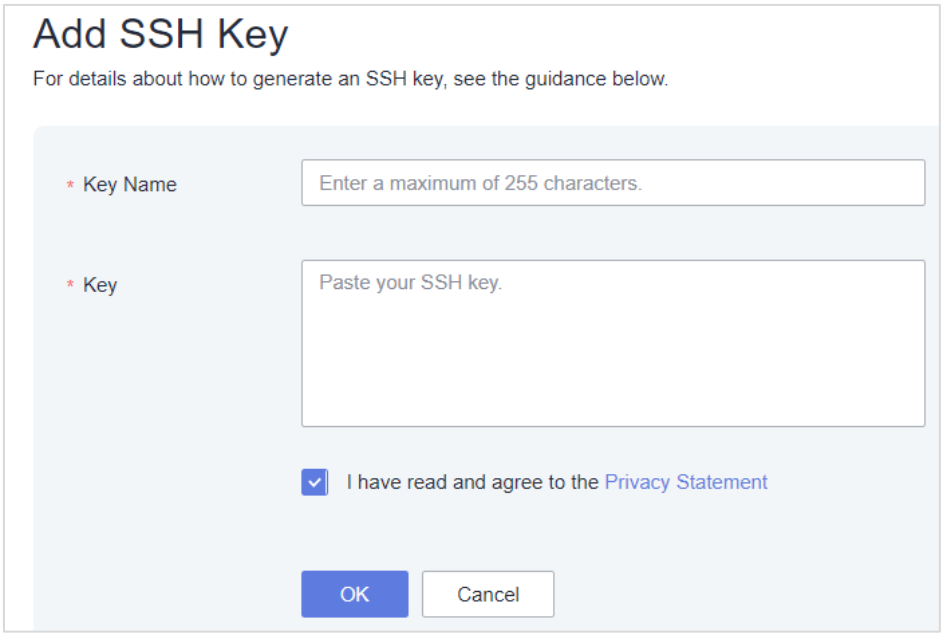
```
r4y2zG1DEOhhyDxXM++Z5HIE+qA11BKUFGPFQClL2BiNFRuTPag9dBAfeMhAjOVHW30pPEWK+B
CZgVyH/SQ4tu9xr3WGxJMmjaoQ+VOOl/u8B7oKI4iw4rOAYekv0HvFdXlfl3a9qJjdx+v1vkL9lss7ws
qVpf6tX5IAzglfZzsNWMOKx1KksbVGAXAcKONsHXDxyogXxvQ1h/Vmh4pZoTAwfLhpBNcvejLvblic
= wjj1@huawei.com
```

Copy the SSH public key in the command output and go to the CodeHub homepage (https://devcloud.huaweicloud.com/codehub/home). Click **Set SSH Key** to go to the SSH key management page and configure the SSH key.

The SSH key management page is shown in the following figure.



b. Click **Add SSH Key** paste the copied SSH public key, enter the title, and click **OK.**

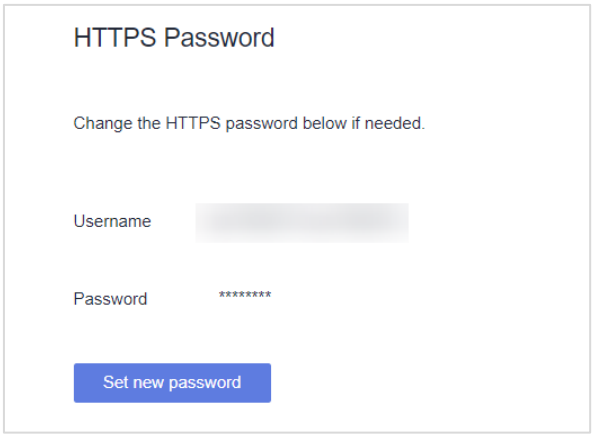


c. The SSH public key has been set. You can continue to set the HTTPS password.

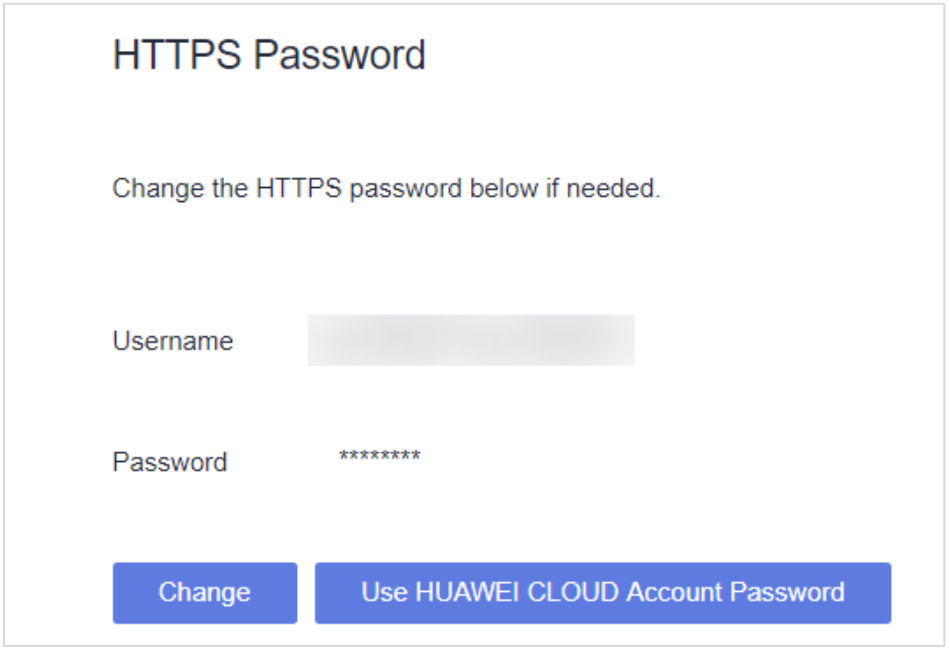
3. Set the HTTPS password.

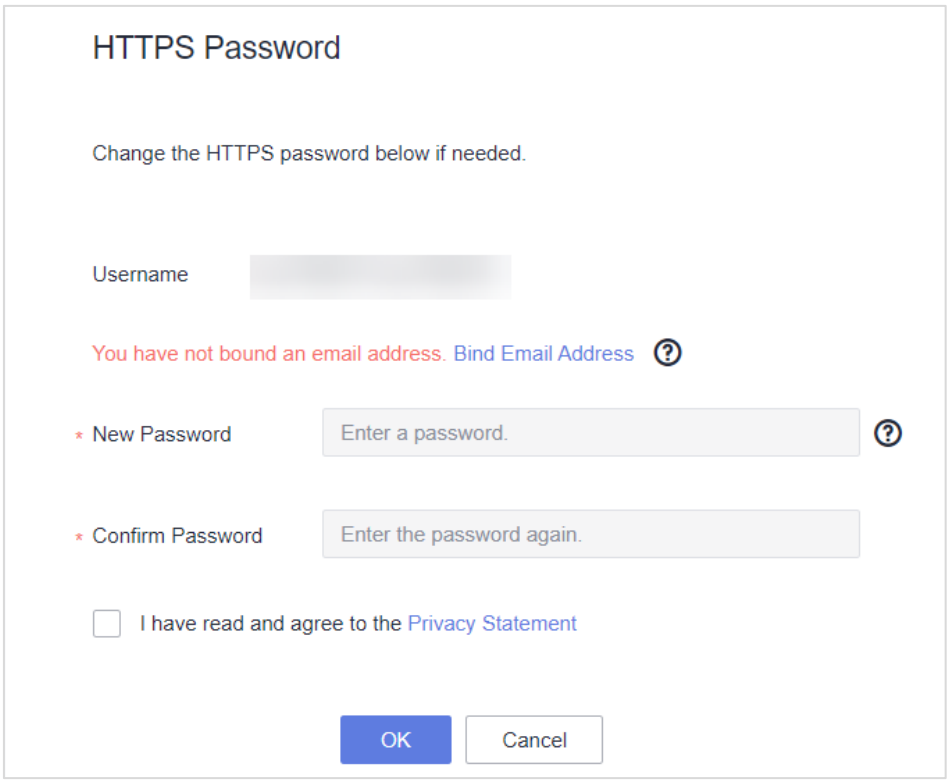
An HTTPS password is used when a client communicates with CodeHub over the HTTPS protocol. By default, it is same as the HUAWEI CLOUD login password. To set an HTTPS password, perform the following operations.

a. On the CodeHub homepage, click **Set HTTPS Password**. The HTTPS key management page is displayed. Click **Set new password.**



b. On the displayed page, click **Change.** To change the HTTPS password, you need to bind an email address to your account. After the email address is bound, reset the password and click **OK**.





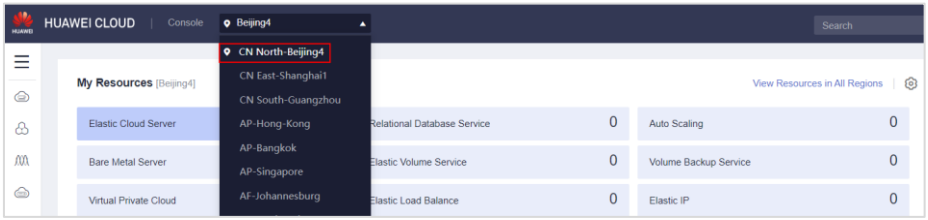
The HTTPS password is set successfully. Now, you can create a repository on CodeHub.

2.3 Creating a Code Repository on CodeHub

2.3.1 Creating a Project

Before creating a repository, you need to create a project.

1. Select a resource storage region.
   a. Visit https://devcloud.huaweicloud.com/home and go to the DevCloud homepage. Select **CN North-Beijing4** from the region drop-down list in the upper right corner.

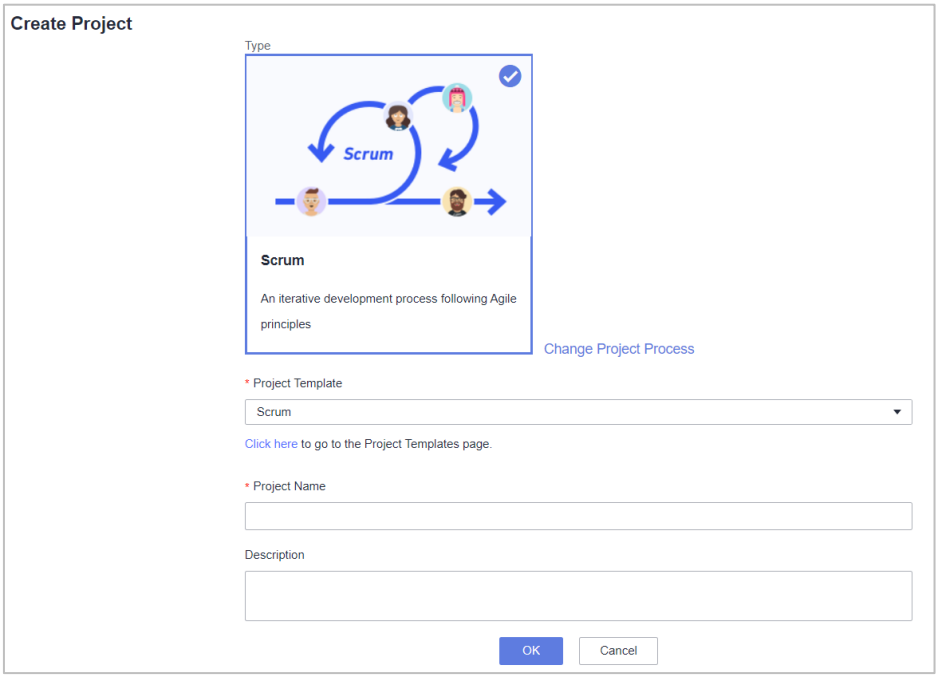


2. Create a Scrum project.

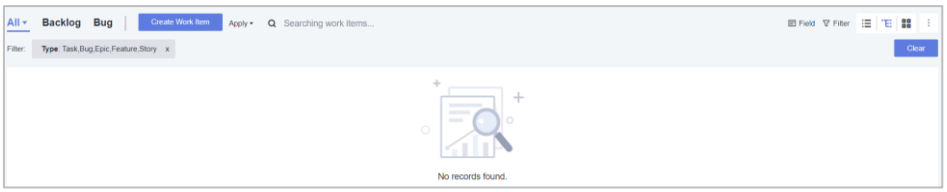
a. Click **Create Project** in the upper right corner of the homepage.

b. Click **Scrum**.

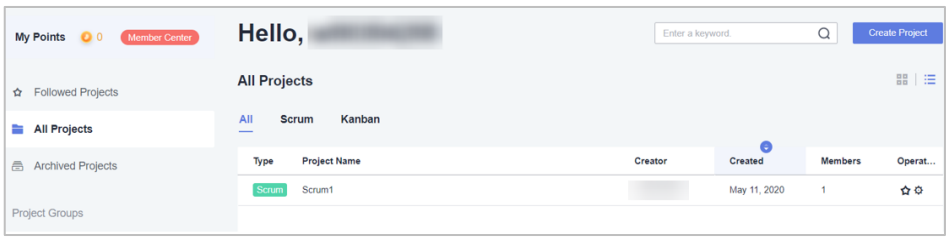
c. Enter the project name and description.



d. Click **OK.** The current workspace is displayed.



So far, you have a cloud-based development environment. After a project is created, you can view the new project in the project list on the homepage and access the project workspace.



2.3.2 Creating a Code Repository

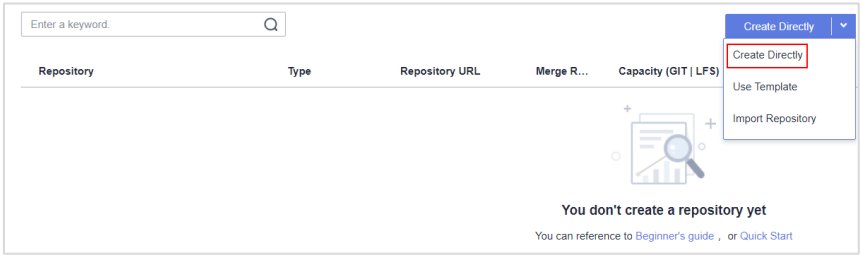
After a project is created, you can create a code repository in the project.

1. Go to the code hosting page.

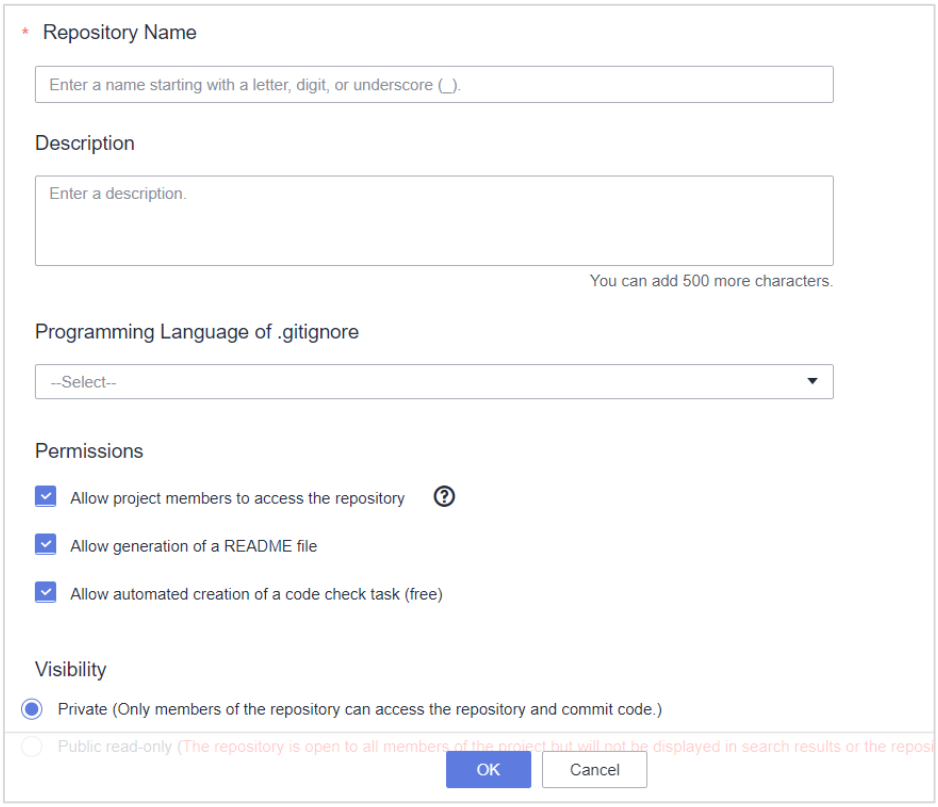
   Select a project from the project list on the homepage, move the pointer to the project, and click **Code.**

2. Create a code repository in common creation mode.

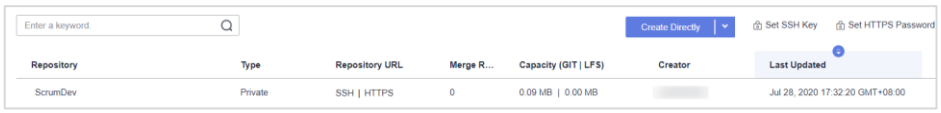
   a. Click **Create Directly** in the upper right corner and choose **Create Directly.**



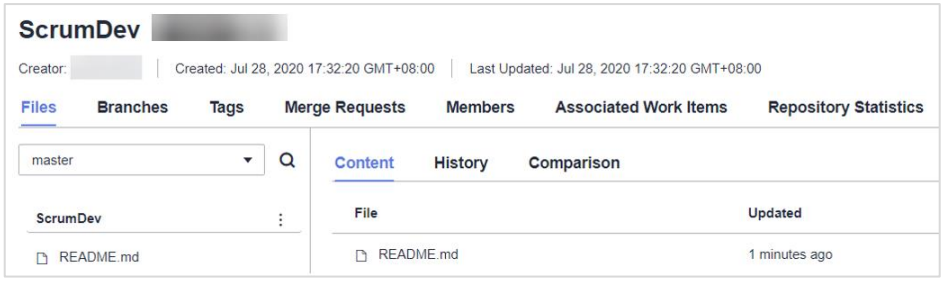
   b. Enter the code repository name and description, and click OK.



Now, you have created a code repository.



The new code repository has only one built-in **README.md** file.



Now, you can clone this remote repository to your local PC for collaborative development.

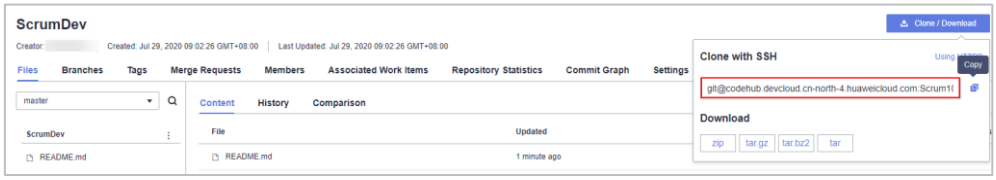
2.4 Interactions Between the Local and Remote Git Repositories

This section describes how to perform basic operations for interactions between the local and remote repositories on the Git CLI and TortoiseGit, including clone, push, and pull.

2.4.1 Operations on the Git CLI

2.4.1.1 Clone

1. Go to the remote repository, click **Clone/Download**, and copy the SSH clone address. Alternatively, you can use the HTTPS clone address. The SSH clone address is used as an example here.



2. Open Git Bash in the selected folder and run the **git clone** command to clone the remote repository to your local PC.

```
$ git clone git@codehub.devcloud.cn-north-4.huaweicloud.com:Scrum100005/ScrumDev.git
Cloning into 'ScrumDev'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
```

```
remote: Total 3 (delta 0), reused 0 (delta 0)
Receiving objects: 100% (3/3), done.
```

In the **git clone** command, you can add the local folder name behind the repository address. The following command clones the remote repository to the **MyDir** folder on the local PC: If no parameter is specified, the local folder name is the same as that of the remote repository by default.

```
$ git clone git@codehub.devcloud.cn-north-4.huaweicloud.com:Scrum100005/ScrumDev.git
MyDir
```

Check files in the **MyDir** folder. The command output shows that the remote repository is cloned to the local PC successfully.

```
$ ll                                  # Check files in the MyDir folder.
total 1
drwxr-xr-x 1 wWX111111 1049089 0    May 19 16:51 ScrumDev/
```

2.4.1.2 Push

In this section, you need to develop some functions in your local repository and push them to the remote repository. Create the **helloworld.py** file.

```
$ vim helloworld.py
```

Enter the following content (you can press **i** to start entering the file content). After the content is entered, enter **:wq** to save it and exit.

```
print("Hello World !")
```

Commit the **helloworld.py** to the local repository.

```
$ git add helloworld.py
$ git commit -m "add helloworld.py"
```

Run the **git push** command to push the file to the remote repository.

```
$ git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 301 bytes | 301.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To codehub.devcloud.cn-north-4.huaweicloud.com:Scrum100005/ScrumDev.git
    e8ca831..44391be    master -> master
```

Open the remote repository. The **helloworld.py** file has been pushed to the remote repository.

2.4.1.3 Pull

Assume that developer B modifies the **helloworld.py** file and pushes it to the remote repository. Now, you need to pull the modifications made by developer B to your local repository. You can create another repository on your local PC to simulate operations made by developer B.

```
$ git clone git@codehub.devcloud.cn-north-4.huaweicloud.com:Scrum100005/ScrumDev.git
ScrumDev_B
$ cd ScrumDev_B/
$ vim helloworld.py
```

Modify the **helloworld.py** file as follows, save the file, and exit.

```
print ("Hello World !\nI'm xxx")
```

Commit the modified **helloworld.py** file to the local repository and push it to the remote repository.

```
$ git add helloworld.py
$ git commit -m "Modify helloworld.py by developer B"
$ git push
```

Now, you need to run the **git pull** command to pull the modifications made by developer B to your local repository.

```
$ cd ../ScrumDev                                    # Go to the folder where the local
repository ScrumDev is located.
$ git pull                                          # Pull the modifications made by
developer B to the master branch in your local repository.
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), 306 bytes | 12.00 KiB/s, done.
From codehub.devcloud.cn-north-4.huaweicloud.com:Scrum100005/ScrumDev
   44391be..c618432  master     -> origin/master
Updating 44391be..c618432
Fast-forward
 helloworld.py | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
```

Open the **helloworld.py** file. You can see that the modifications made by developer B have been updated in this file.

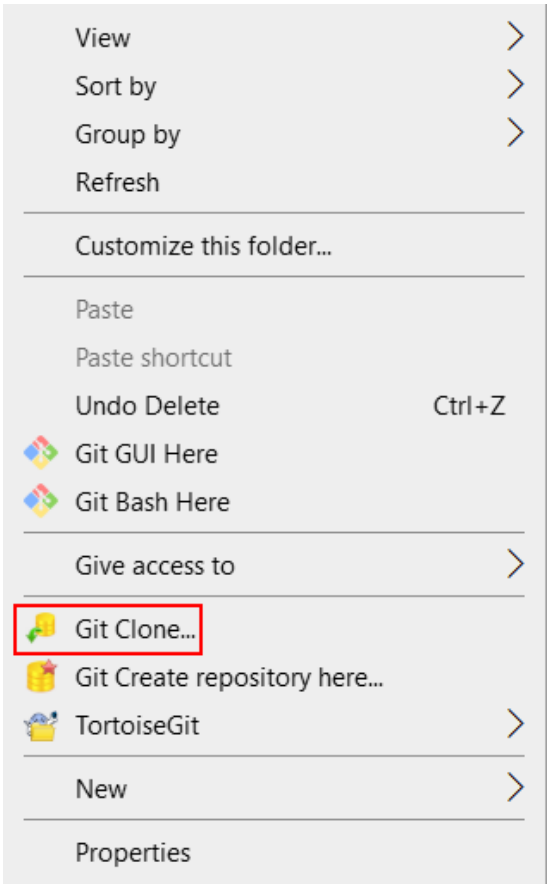
```
$ vim helloworld.py
print("Hello World !\nI'm xxx")                    # This is the content of the modified
```
helloworld.py file.

2.4.2 Operations on TortoiseGit

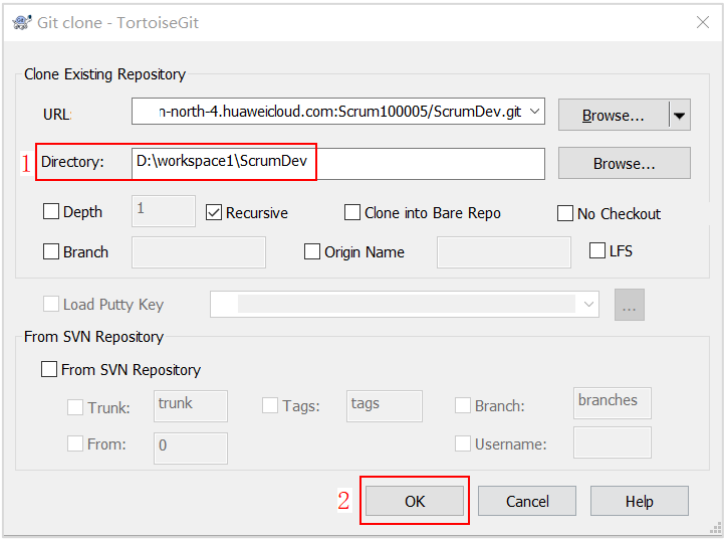
This section describes how to perform clone, push, and pull operations on TortoiseGit.

2.4.2.1 Clone

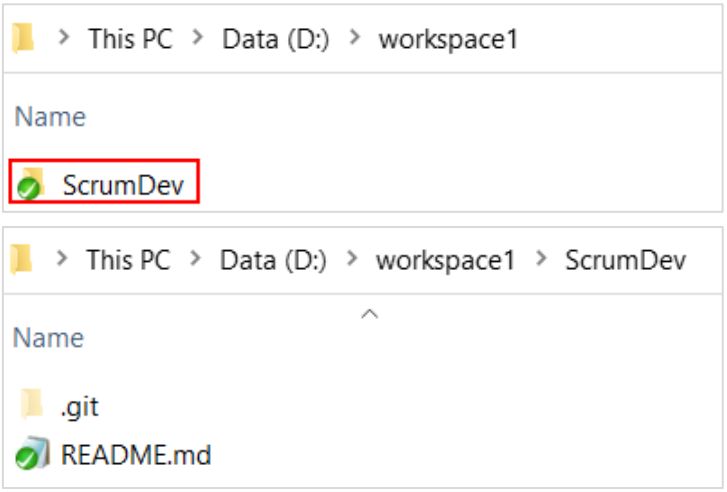
1. Go to the remote repository, click **Clone/Download,** and copy the SSH clone address.
2. Right-click the blank area of the desired local folder and choose **Git Clone** from the shortcut menu.



In the dialog box that is displayed, enter the address of the folder where the local repository is located. By default, the name of the folder where the local repository is located is the same as that of the remote repository. Then, click **OK**.

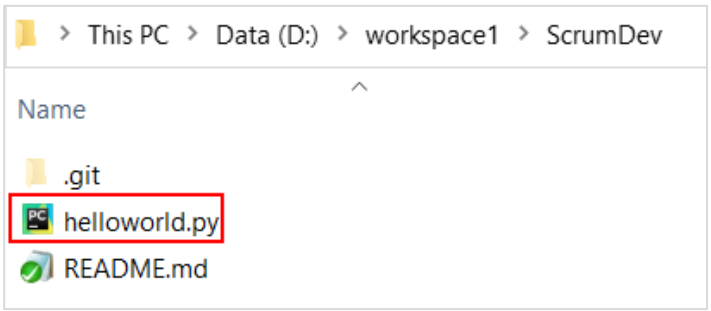


The remote repository has been cloned to your local PC.



2.4.2.2 Push

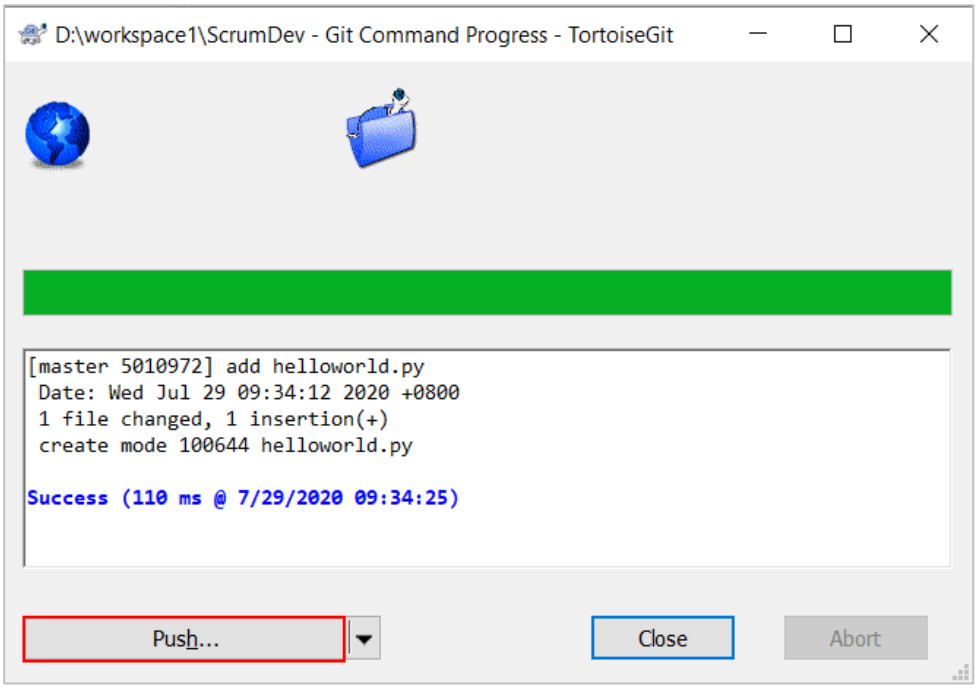
In this section, you need to develop some functions in your local repository and push them to the remote repository. Create the **helloworld.py** file to represent a new function.



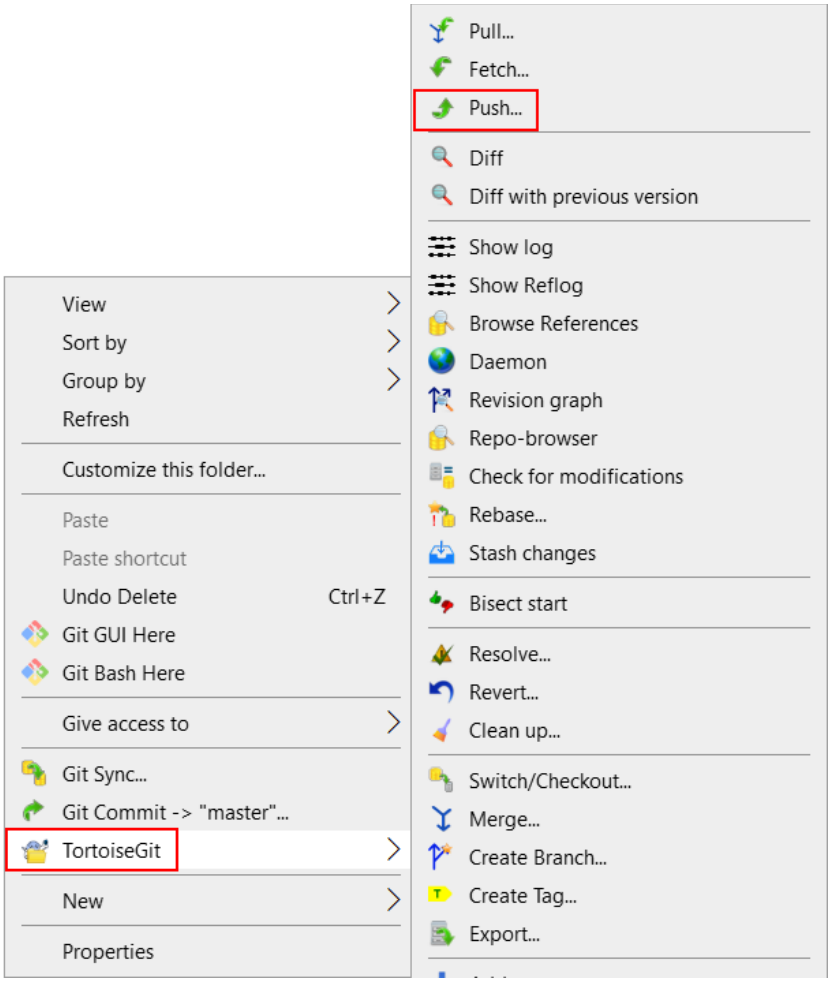
The file content is as follows:

```
print("Hello World !")
```

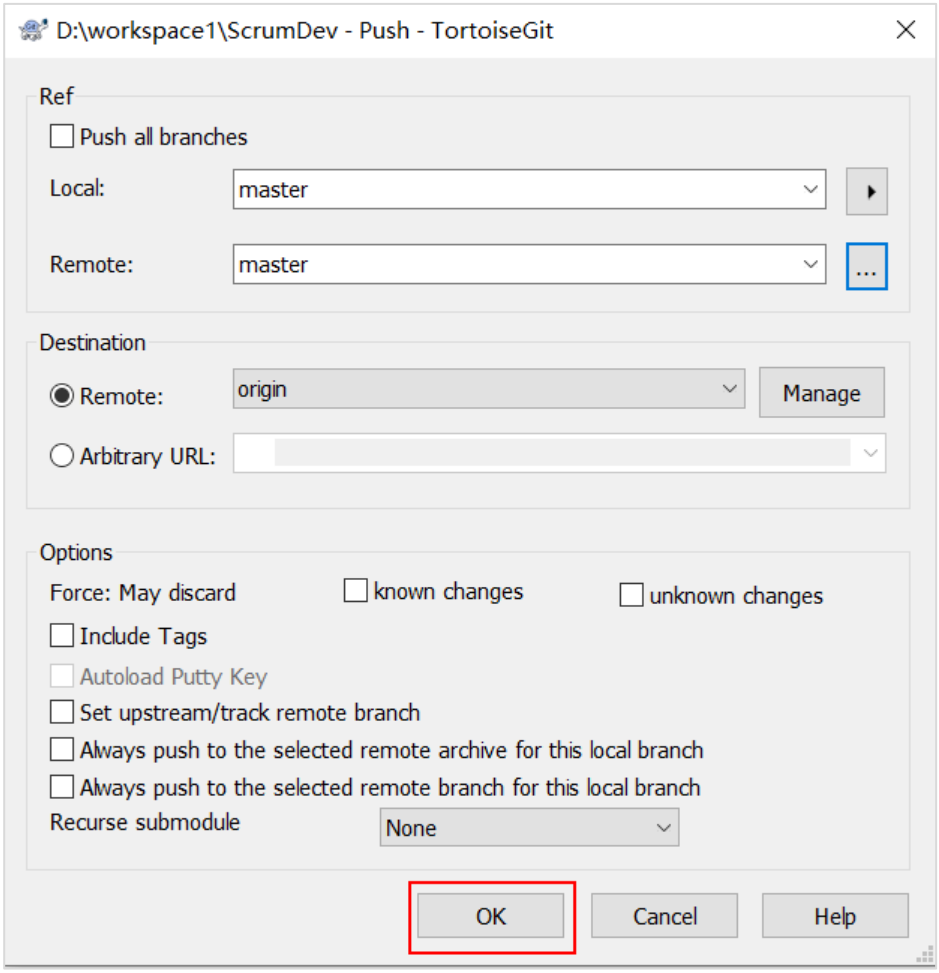
Commit the **helloworld.py** file to the local repository. For details, see 1.3.2.2. After you click **Commit**, the following dialog box is displayed. Click **Push** to push the **helloworld.py** file to the remote repository.



If you do not want to push the file to the remote repository immediately, click **Close**. You can push the file later by right-clicking the blank area in the desired folder and choosing **TortoiseGit** > **Push** from the shortcut menu.



In the displayed window, click **OK**.

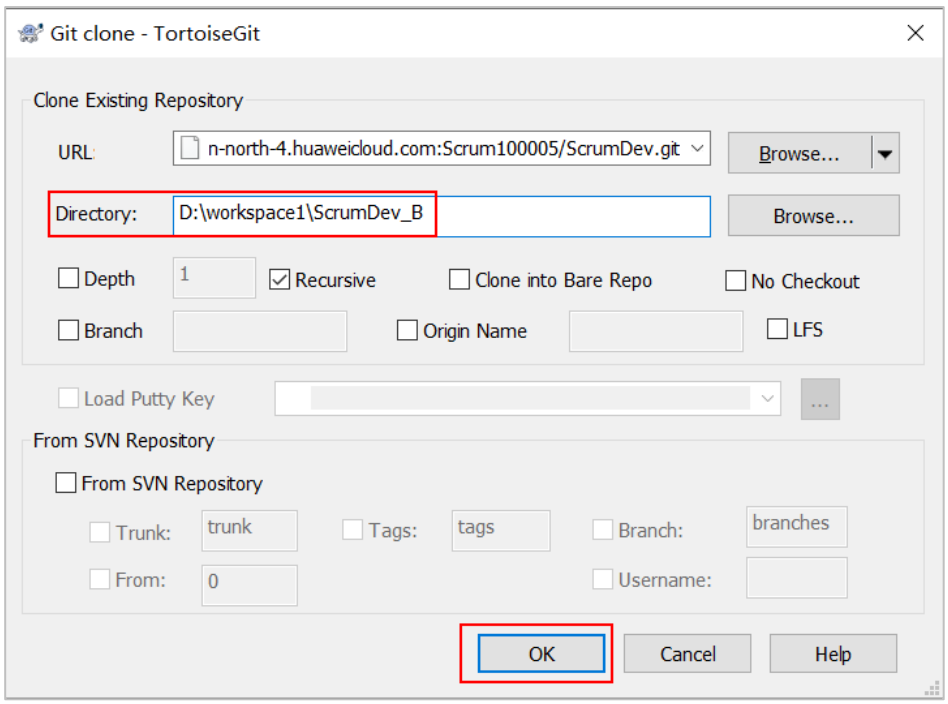


Open the remote repository. The **helloworld.py** file has been pushed.

2.4.2.3 Pull

Assume that developer B modifies the **helloworld.py** file and pushes it to the remote repository. Now, you need to pull the modifications made by developer B to your local repository. You can create another repository on your local PC to

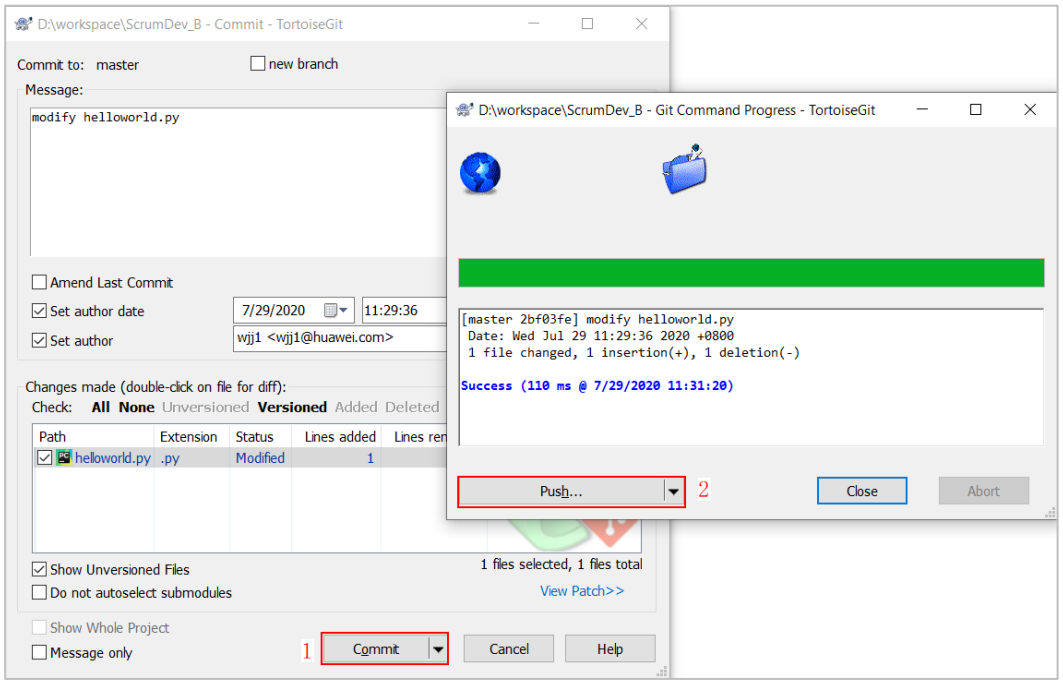
simulate operations made by developer B. On the local PC, use TortoiseGit to clone the remote repository to the **ScrumDev_B** folder.



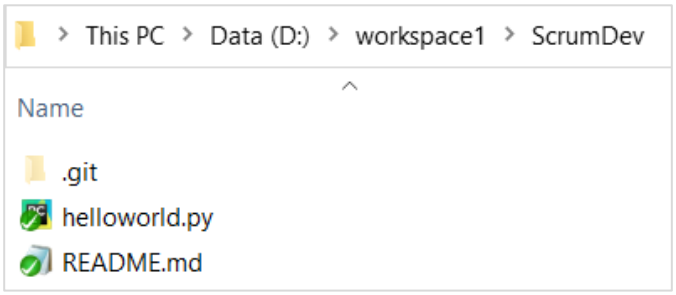
Modify the **helloworld.py** file as follows:

```
print("Hello World !\nI'm xxx")
```

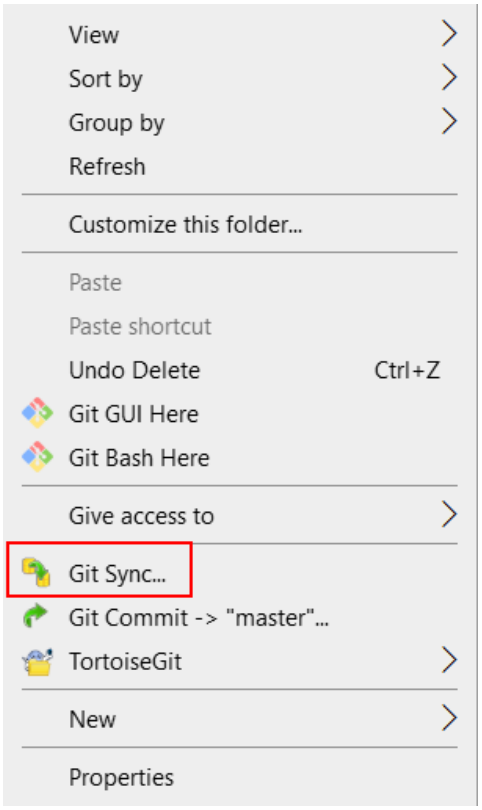
Commit the modified **helloworld.py** file to the local repository and then push it to the remote repository.



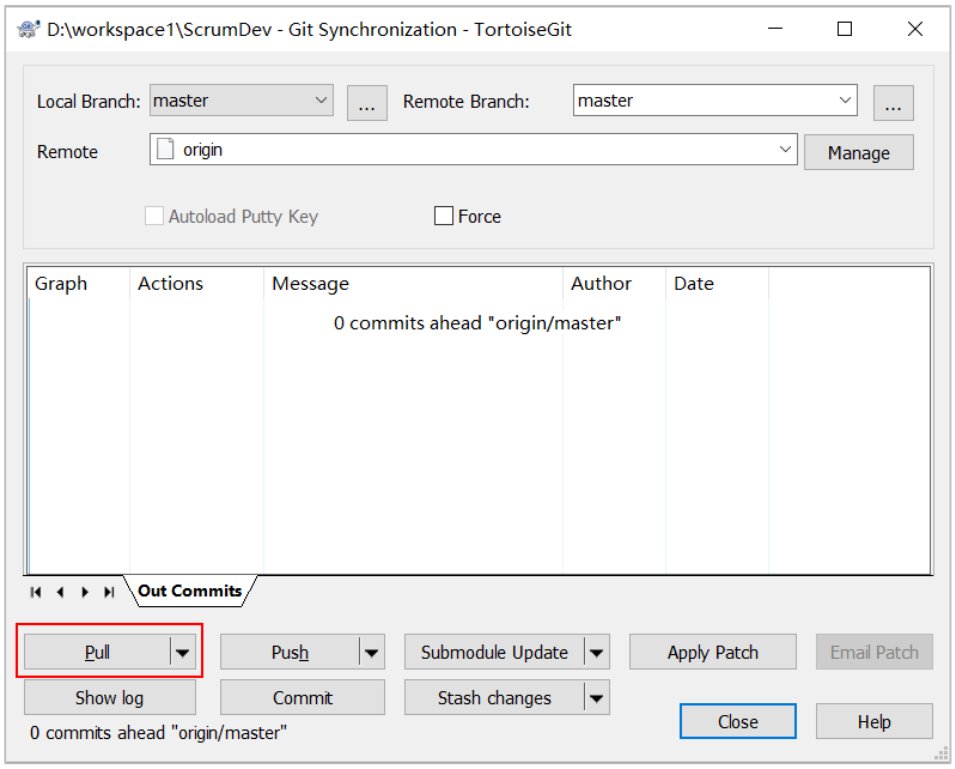
Now, you need to pull the modifications made by developer B to your local repository using TortoiseGit. Go to the folder where the local Git repository is located.



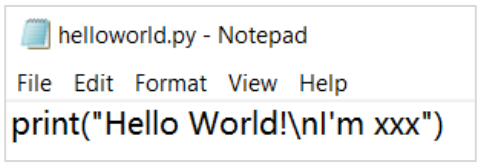
Right-click the blank area of the folder and choose **Git Sync** from the shortcut menu.



In the displayed window, click **Pull** to pull updates from the remote repository.



Open the **helloworld.py** file. You can see that the modifications made by developer B have been updated in this file.



3 Gitflow Workflow Practice

3.1 Background

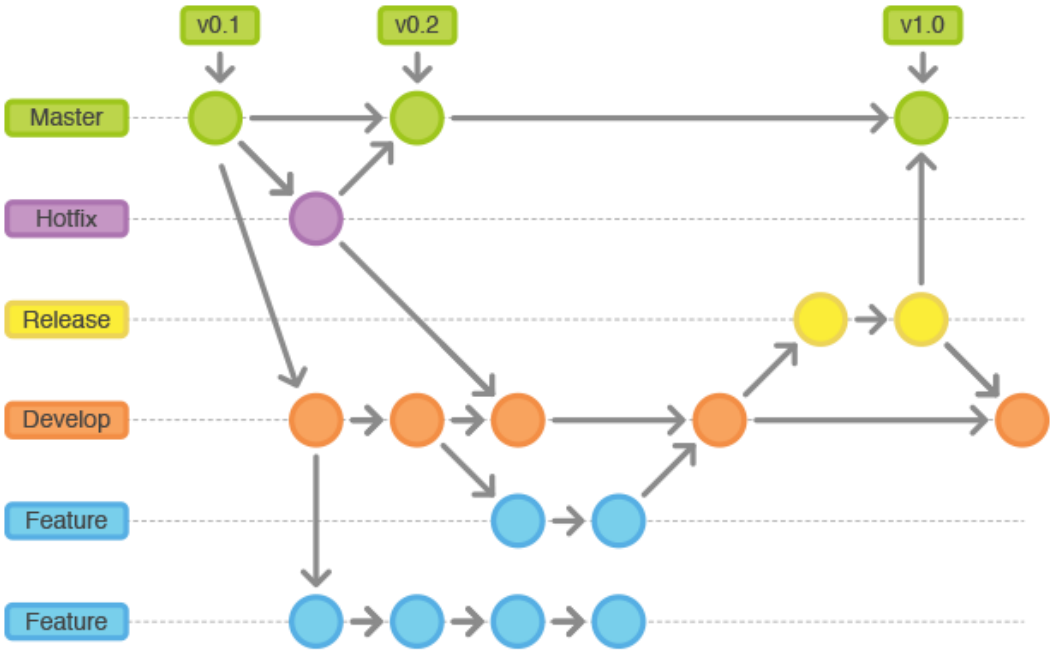
Now, you have learned the basic operations in the local Git repository and operations for interacting with the remote repository on CodeHub. In real-world software development, the use of Git is often more complex. A proper Git workflow can effectively improve project management and team collaborative development capabilities. A Git workflow is a branching strategy for code management and serves project process management and team collaborative development.

Gitflow is a Git workflow used to manage large projects. This experiment guides you through the Gitflow workflow process. On completion of this experiment, you will be able to understand:

- What is Gitflow.
- How Gitflow works.

3.1.1 Gitflow Overview

Gitflow is a branching strategy for code management and is ideal for large project management. It assigns specific roles to different branches and defines how and when they should interact. The following figure shows the Gitflow workflow.



How It Works

- **Master branch**

  It is the production-ready branch and the most stable. Developers cannot commit anything directly to the master branch. Instead, they should merge commits from other branches to the master branch. Many enterprises have the master branch protected and only the maintenance personnel can perform operations in this branch.

- **Hotfix branch**

  It is a temporary fix branch created from the master branch to fix frontline emergency bugs. As soon as the fix is complete, the hotfix branch should be merged to both master and develop branches, and the master branch needs to be tagged with a new version number.

- **Develop branch**

  It is created from the master branch and used for function integration. It contains all the code to be released in the next version and is used for development integration and system testing.

- **Release branch**

  Once it is time for a release, a release branch is created from the develop branch. Functions that are not in the release branch will be pushed to the next release. The release branch is used for release. No new features can be added to the release branch. Only bug fixes, documentation generation, and other release-oriented tasks should go in this branch. Once it is ready for release, the release branch gets merged to the master branch and the master branch is tagged with a version number. In addition, the release branch should be merged back to the develop branch.

- **Feature branch**

  It is used for function development. A feature branch uses the develop branch as its parent branch. After function development is complete, feature branches get merged back to the develop branch. Feature branches never interact directly with the master branch.

3.2 Experiment Introduction

3.2.1 Experiment Background

Assume that you are working in a Fortune 500 ICT company and you are responsible for developing a flow control function by team. To implement this function, you create a feature branch and push it to the remote repository so that team members can develop this function together. After the function is developed, you can merge the function to the develop branch and create a release branch for release. After the release is complete, you merge the release branch to the master and develop branches. However, you suddenly receive a call saying that there is a bug in the current version. To solve this bug, you create a hotfix branch based on the master branch, fix the bug in the hotfix branch, and merge your modifications to the master and develop branches.

3.2.2 Procedure

The experiment procedure is as follows:

1. Initialize a remote repository as your team's repository and create the develop branch based on the master branch.
2. Clone the remote repository to your local PC, create the featureA branch based on the develop branch, and push the featureA branch to the remote repository.
3. After the flow control function is developed in the featureA branch, push the changes to the remote repository.
4. Merge the featureA branch to the develop branch.
5. Pull the develop branch to your local PC, create the release-v1 branch based on the develop branch, and push the release-v1 branch to the remote repository.
6. Perform some release-oriented work in the release-v1 branch and push the changes to the remote repository.
7. After the release-v1 branch is ready, merge it to the master branch and tag the master branch with a version number. Then, merge the release-v1 branch to the develop branch.
8. Pull the master and develop branches to your local PC, and create a hotfix branch based on the master branch.

9. Fix the bug in the hotfix branch and push the hotfix branch to the remote repository.
10. Merge the hotfix branch to the master branch and tag the master branch with a new version number. Then, merge the hotfix branch to the develop branch.

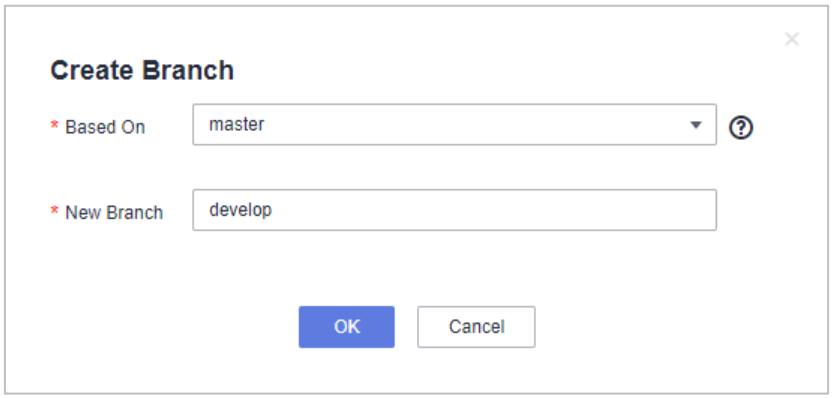
3.3 Experiment Operations

In this experiment, operations are performed in the Git CLI.

1. Create a remote repository in CodeHub and create the develop branch based on the master branch. This repository is the remote repository dedicated to your team. The flow control function that you need to develop is also developed in this repository.

   For details about how to create a code repository in CodeHub, see 2.3. As shown in the following figure, a remote repository named **GitflowPractice** has been created in CodeHub.

   Create the develop branch based on the master branch.



   Now, the remote repository has the master and develop branches.

The following figure describes this operation in the Git workflow.

2. Clone the remote repository to your local PC and create the featureA branch based on the develop branch to develop the flow control function. To ensure that team members can cooperate, push the featureA branch to the remote repository.

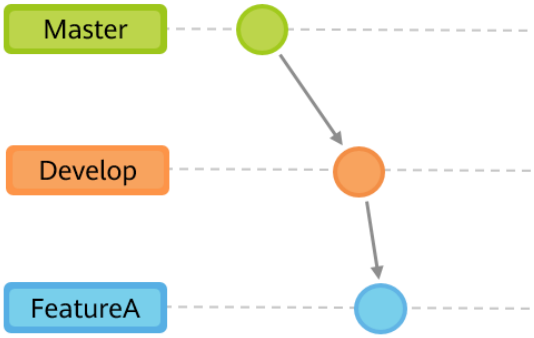
```
$ git clone git@codehub.devcloud.cn-north-4.huaweicloud.com:Scrum100005/GitflowPractice.git
                                        # Clone the remote repository to your local PC.
$ git checkout -b develop origin/develop        # Create the develop branch on your local PC
and track the remote branch origin/develop.
$ git checkout -b featureA                      # Create the featureA branch based on the
develop branch.
$ git push -u origin featureA                   # Push the featureA branch to the remote
repository.
```

Now, your local Git repository has three branches: develop, featureA, and master.

```
$git branch #View the branch.
  develop
* featureA
  master
```

There are also the three branches in the remote repository.

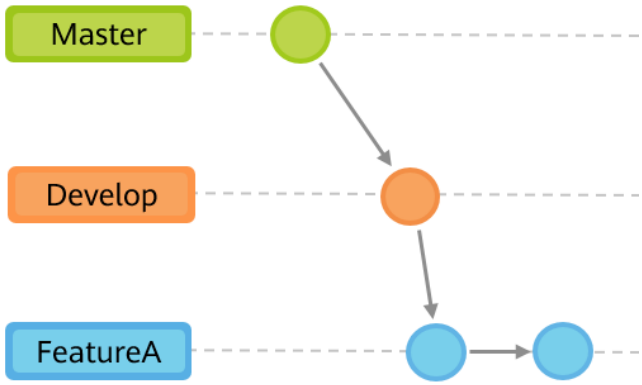
The following figure describes this operation in the Git workflow.



3. Develop the flow control function in the featureA branch. The **trafficControl.py** file represents the new function. After the function is developed, push the changes to the remote repository.

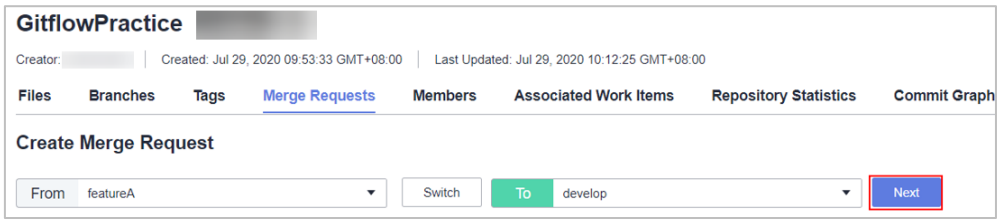
```
$ touch trafficControl.py                                  # Create the trafficControl.py file.
$ git add trafficControl.py                                # Add the trafficControl.py file to the
staging area.
$ git commit -m "add traffic control function"             # Commit the trafficControl.py file to the
local repository. -m indicates the commit message.
$ git push                                                 # Push the trafficControl.py file to the
remote repository.
```

The following figure describes this operation in the Git workflow.

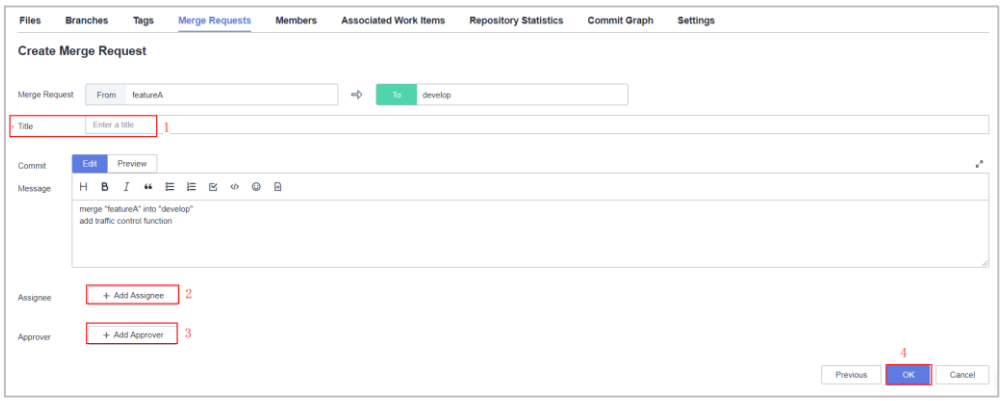


4. Now, you need to merge the code to the develop branch. To ensure the code quality, you need to ask other developers to review the code. In this case, you need to create a merge request on CodeHub and ask other developers to review the code.

   Click the **Merge Requests** tab and click **Create Merge Request**. Then, select the source and destination branches for the merge request and click **Next.**

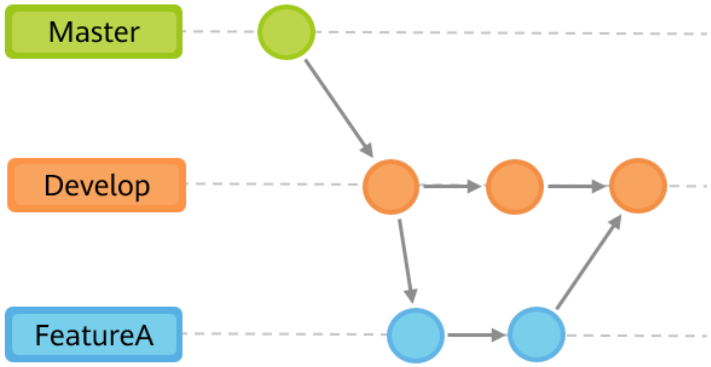


On the page that is displayed, set **Title**, **Assignee**, and **Approver**, and click **OK**. The assignee is the person who performs the merge operation, and the approver is the person who reviews the code.



The reviewer and the merger will receive a notification. In this experiment, you specify yourself as both reviewer and merger. The developer can modify the code based on the review comments made by the reviewer. Assume that the flow control function is reviewed. The merger needs to click **Merge** to merge the featureA branch to the develop branch.

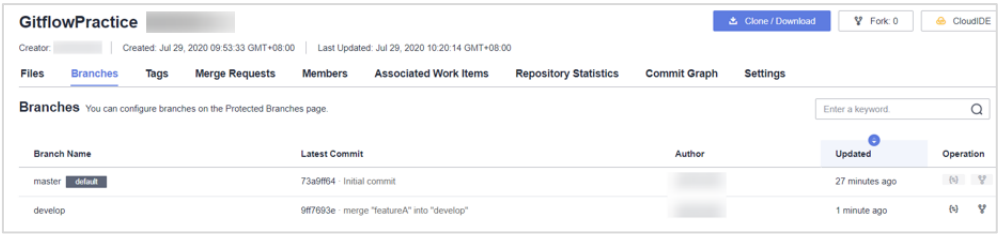
The following figure describes this operation in the Git workflow. Assume that another team has merged some functions to the develop branch before you merge the featureA branch. There will be an additional circle in the workflow of the develop branch.



After the flow control function is developed, you need to delete the featureA branch to ensure that the Git repository branch structure is clear.

```
$ git switch develop                # Switch to another branch before deleting the featureA branch.
$ git branch -d featureA            # Delete the featureA branch from your local repository.
$ git push origin -d featureA       # Delete the featureA branch from the remote repository.
```

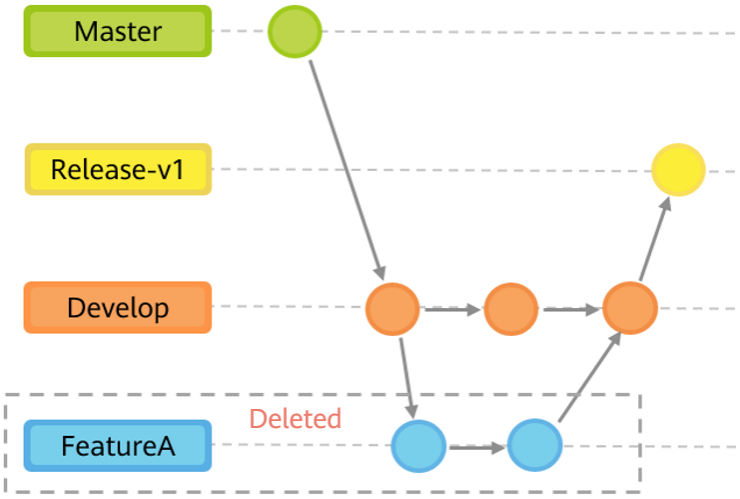
Now, the remote repository has only the master and develop branches.



5. Now, the develop branch has acquired enough features for a release. To initiate a release, pull the develop branch to your local repository, create the release-v1 branch based on the local develop branch, and push the release-v1 branch to the remote repository so that team members can work on the release together.

```
$ git switch develop                # Switch to the develop branch.
$ git pull                          # Pull the develop branch to your local repository.
$ git branch release-v1             # Create the release-v1 branch based on the
develop branch.
$ git switch release-v1             # Switch to the release-v1 branch.
$ git push -u origin release-v1     # Push the release-v1 branch to the remote
repository.
```

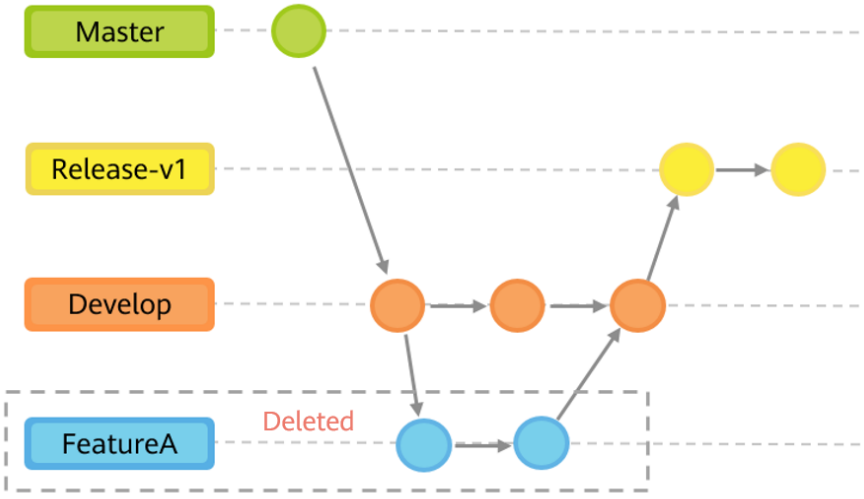
The following figure describes this operation in the Git workflow.



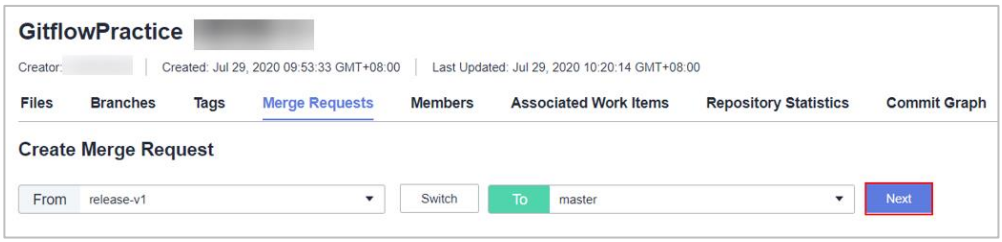
6. Now, you can perform release-oriented tasks in the release-v1 branch. Create the **release-v1.md** file and push it to the remote repository.

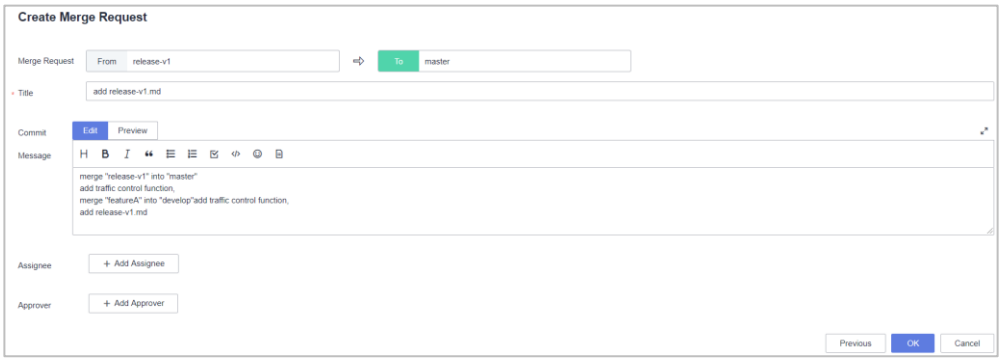
```
$ touch release-v1.md                          # Create the release-v1.md file.
$ git add release-v1.md                        # Add the file to the staging area.
$ git commit -m "add release-v1.md"            # Commit the file to the local repository.
$ git push                                     # Push the file to the remote repository.
```

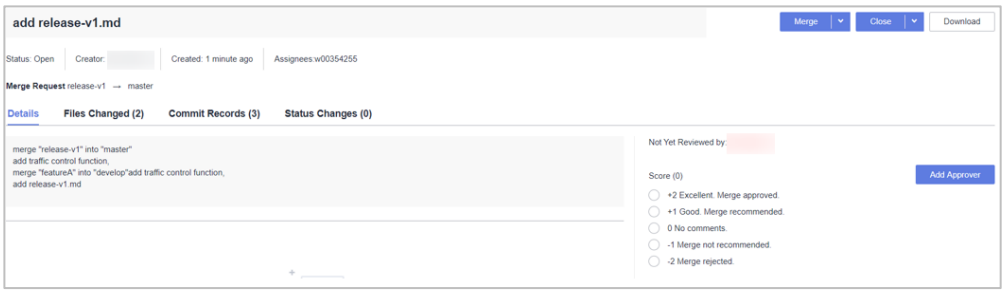
The following figure describes this operation in the Git workflow.



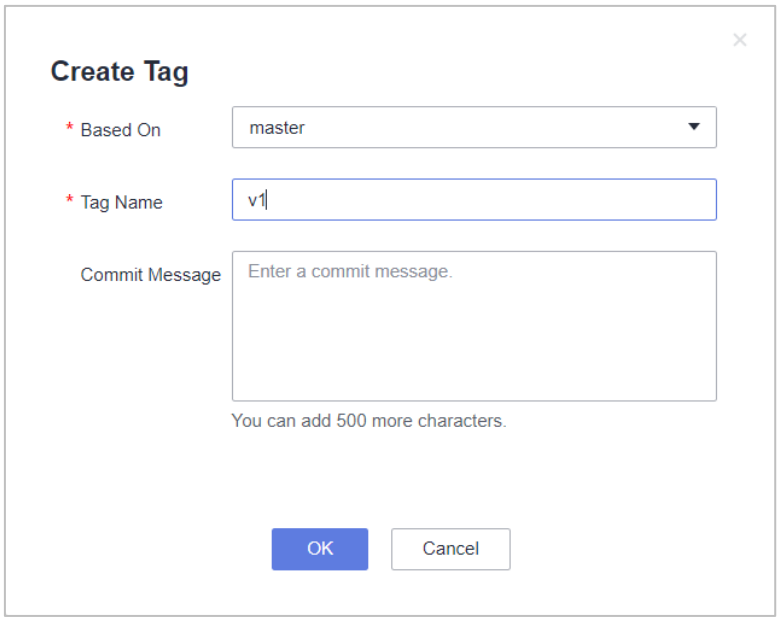
7. Now you have completed the release-oriented work related to the release-v1 branch. You need to merge the release-v1 branch to the master branch and tag the master branch. You also need to merge the release-v1 branch to the develop branch. These operations are performed in the remote repository.



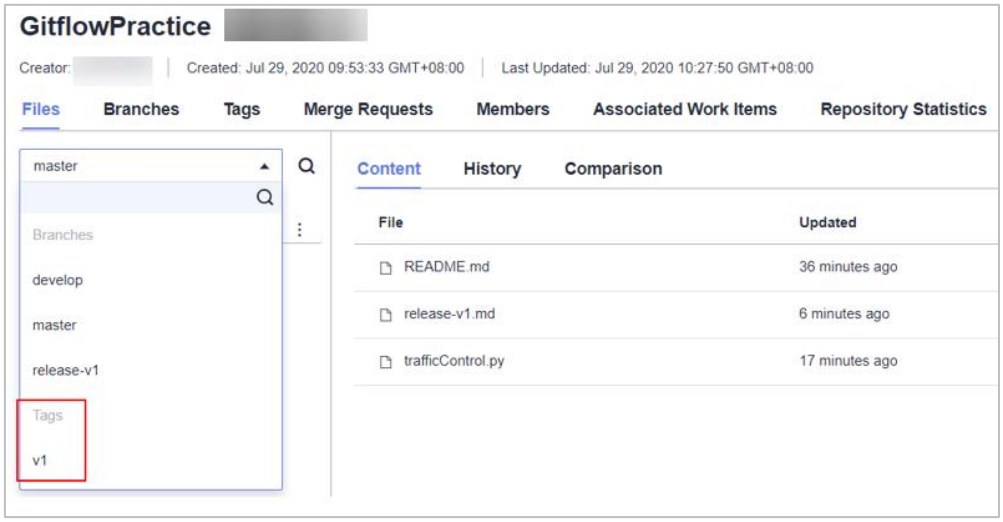




After the release-v1 branch is merged to the master branch, tag the master branch with a version number.



After the master branch is tagged, the v1 tag is displayed in the remote repository. The tag is a snapshot of the master branch and records the status of the master branch at this point.



Merge the release-v1 branch to the develop branch. The procedure is the same as that for merging the release-v1 branch to the master branch.

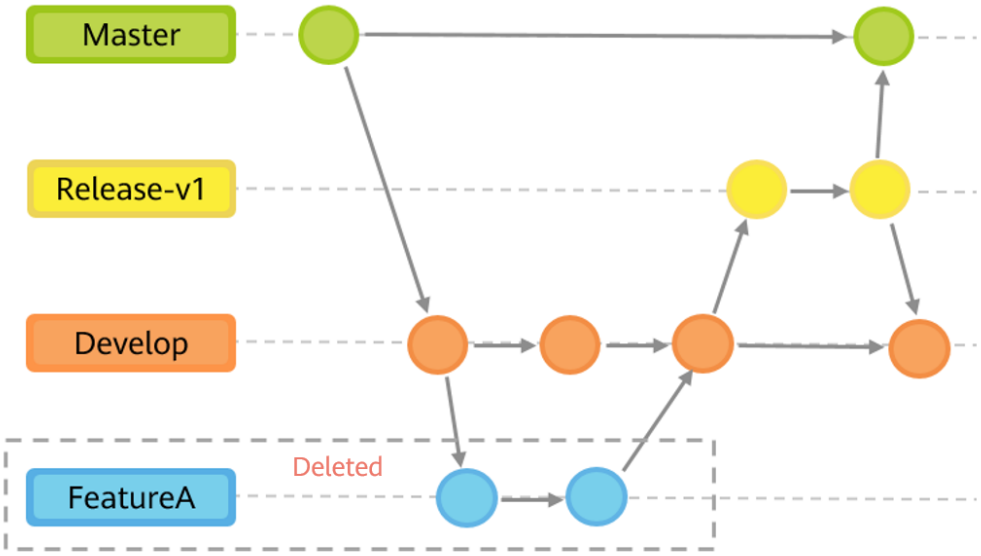
After the operation is complete, delete the release-v1 branch to ensure that the branch structure in Git repositories is clear.

```
$ git switch master                  # Switch to another branch before deleting the release-v1
branch.
$ git branch -d release-v1           # Delete the release-v1 branch from the local repository.
$ git push origin -d release-v1      # Delete the release-v1 branch from the remote repository.
```

After the release-v1 branch is deleted, only the master and develop branches exist in the local repository and remote repository.

```
$ git branch                    # Check branches.
  develop
* master
```

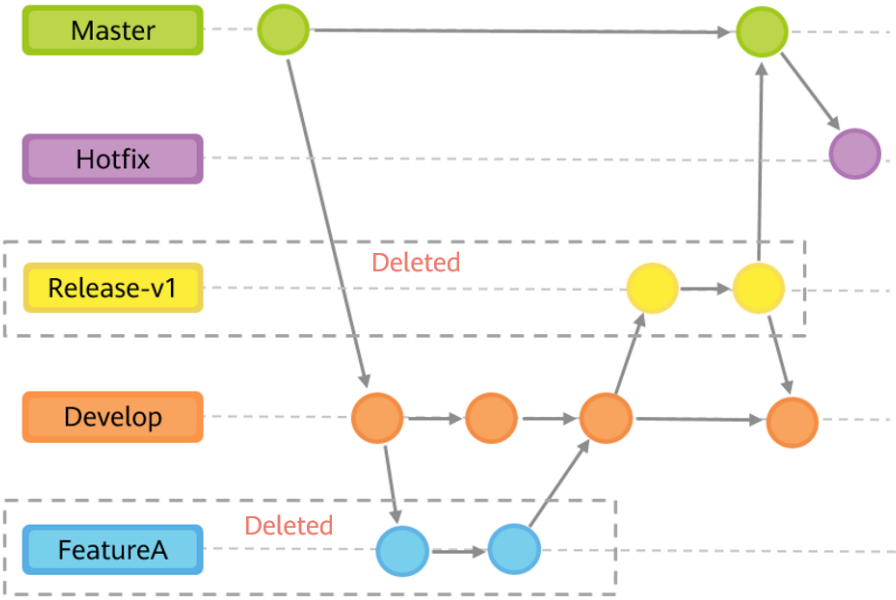
The following figure describes this operation in the Git workflow.



8. After the version is released, you suddenly receive a call saying that there is a bug in the released version. To resolve the problem, pull the master and develop branches to your local repository first, since there are updates in the two branches in the remote repository. Then, create the hotfix branch based on the master branch to fix the bug.

```
$ git switch develop            # Switch to the develop branch.
$ git pull                      # Pull the develop branch to the local repository.
$ git switch master             # Switch to the master branch.
$ git pull                      # Pull the master branch to the local repository.
$ git branch hotfix             # Create the hotfix branch based on the master branch.
$ git switch hotfix             # Switch to the hotfix branch.
```

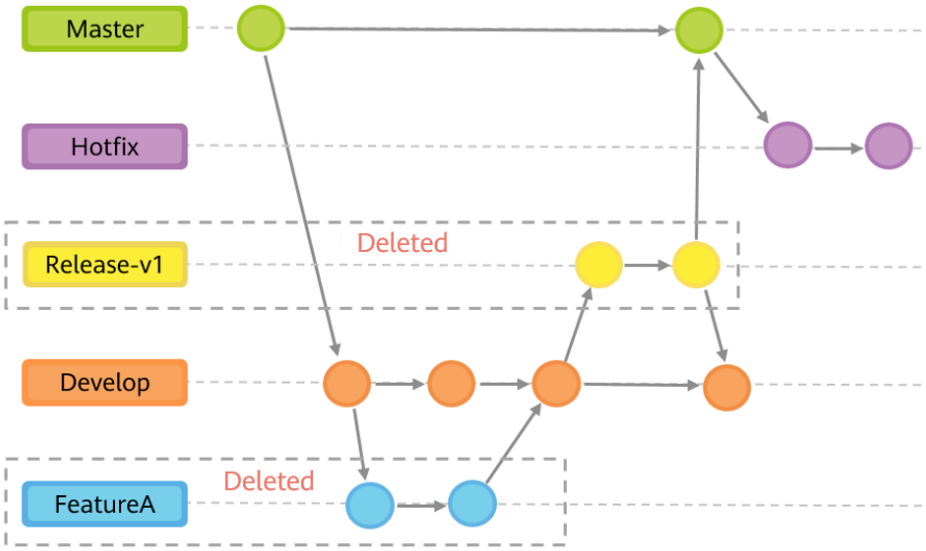
The following figure describes this operation in the Git workflow.



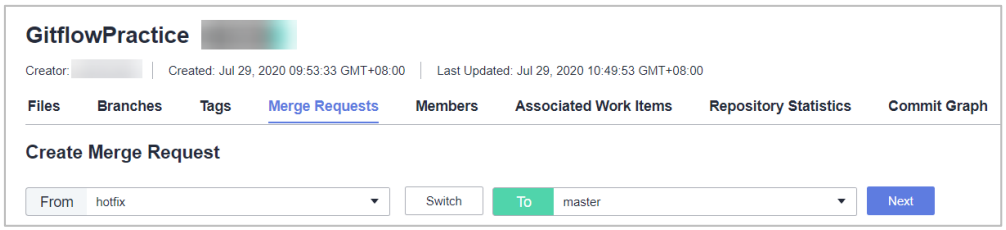
9. Fix the bug in the hotfix branch. In this example, the **bugfixed.pat** file is created to fix the bug. You need to push the hotfix branch to the remote repository.

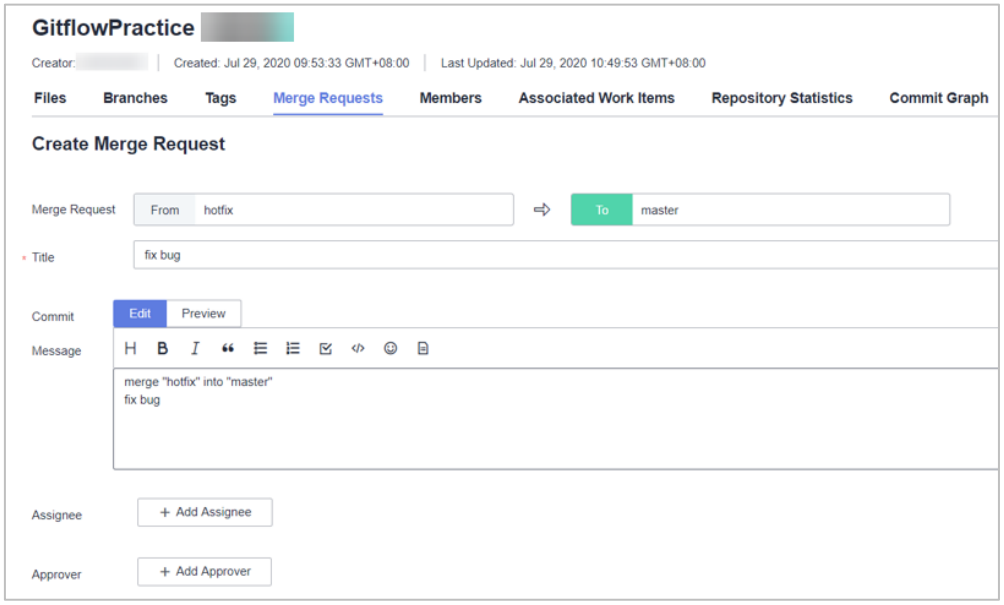
```
$ git switch hotfix
$ touch bugfixed.pat                    # Create the bugfixed.pat file.
$ git add bugfixed.pat
$ git commit -m "fix bug"
$ git push -u origin hotfix             # Push the hotfix branch to the remote
repository.
```

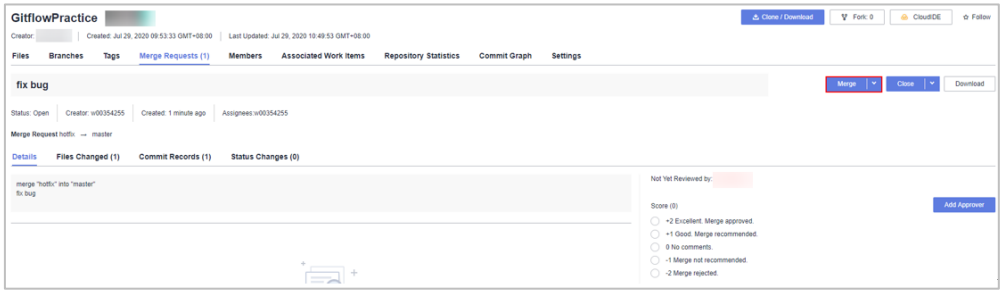
The following figure describes this operation in the Git workflow.



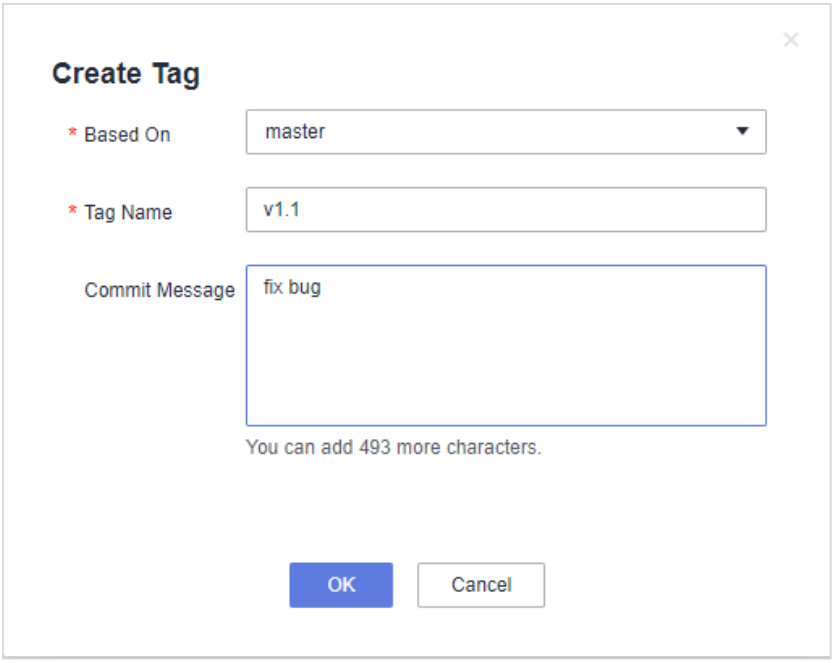
10. After the bug is fixed, merge the hotfix branch to the master branch and tag the master branch. Then, merge the hotfix branch to the develop branch.



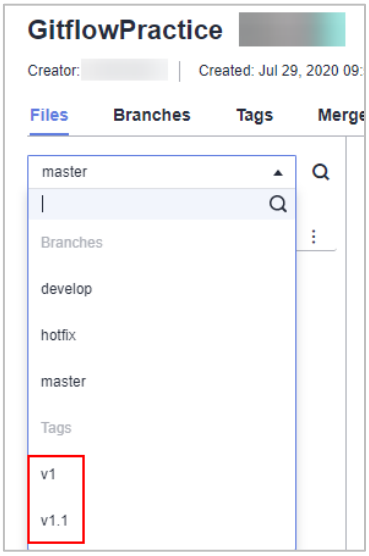




Tag the master branch with **v1.1**.

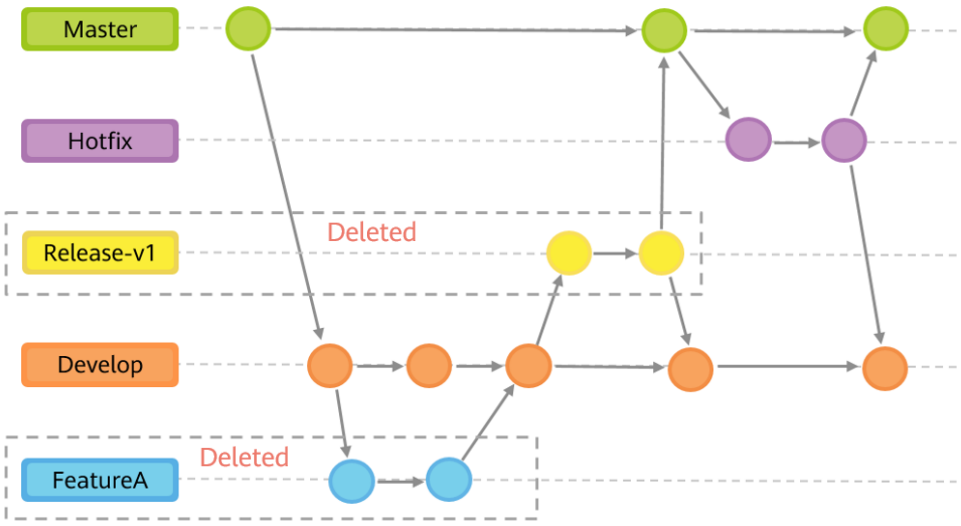


The master branch has two tags **v1** and **v1.1**.



Merge the hotfix branch to the develop branch. The procedure is the same as that for merging the hotfix branch to the master branch.

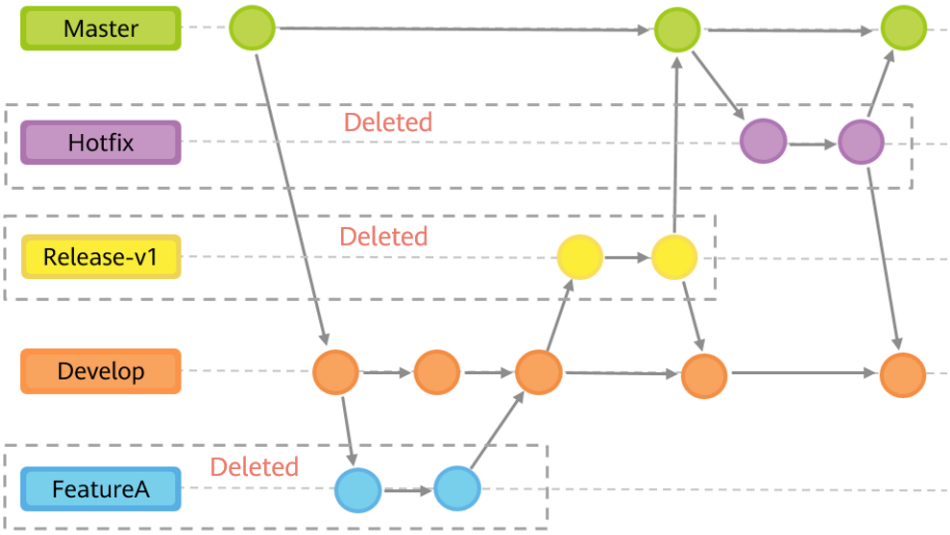
The following figure describes this operation in the Git workflow.



To ensure the clear branch structure of the Git repository, delete the hotfix branch after the bug is fixed.

```
$ git switch master                    # Switch to another branch.
$ git branch -d hotfix                 # Delete the hotfix branch from the local repository.
$ git push origin -d hotfix            # Delete the hotfix branch from the remote repository.
```

Now, the local and remote repositories have only the master and develop branches. The following figure shows the final branch status.



So far, the Gitflow workflow practice ends.

Huawei Certification Training

HCIP-Datacom-Network Automation Developer
Network Devices' Open Programmability Lab Guide

V1.0

Huawei Technologies Co., Ltd.

Trademarks and Permissions

 and other Huawei trademarks are trademarks of Huawei Technologies Co., Ltd.

All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The purchased products, services and features are stipulated by the contract made between Huawei and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

Huawei Technologies Co., Ltd.

Address: Huawei Industrial Base
Bantian, Longgang
Shenzhen 518129
People's Republic of China

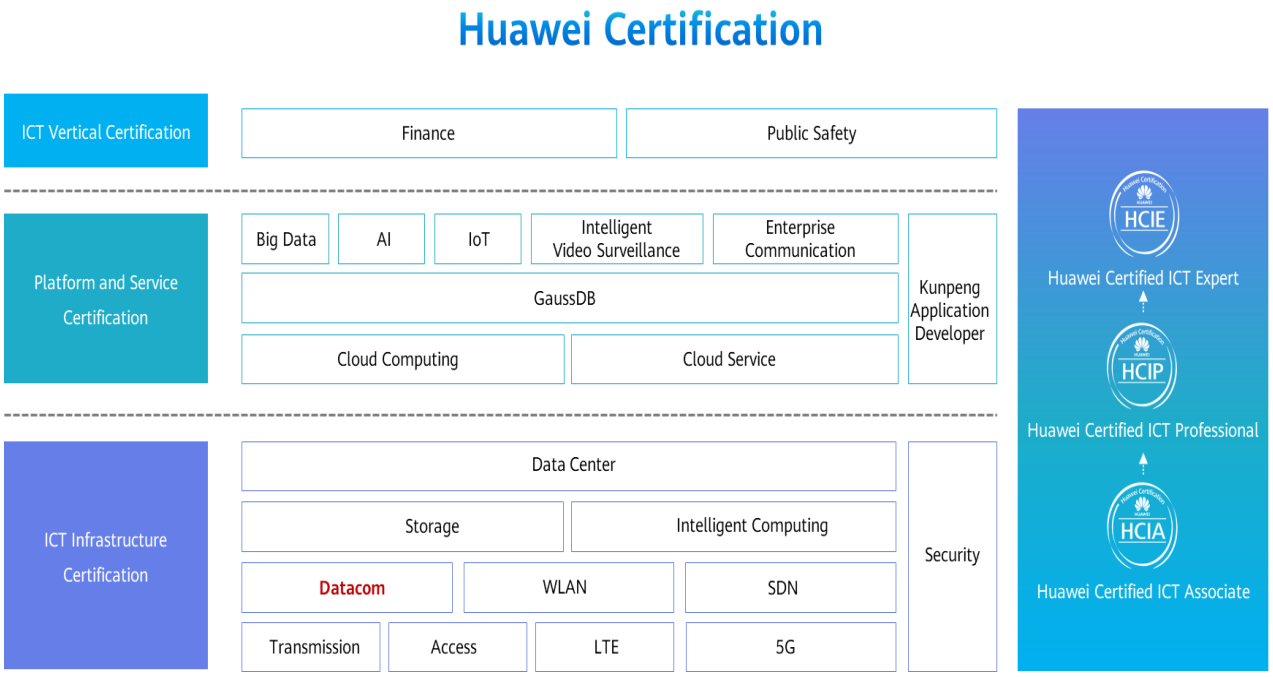
Website: https://e.huawei.com/

Huawei Certification System

Huawei Certification follows the "platform + ecosystem" development strategy, which is a new collaborative architecture of ICT infrastructure based on "Cloud-Pipe-Terminal". Huawei has set up a complete certification system consisting of three categories: ICT infrastructure certification, platform and service certification, and ICT vertical certification. It is the only certification system that covers all ICT technical fields in the industry. Huawei offers three levels of certification: Huawei Certified ICT Associate (HCIA), Huawei Certified ICT Professional (HCIP), and Huawei Certified ICT Expert (HCIE). Huawei Certification covers all ICT fields and adapts to the industry trend of ICT convergence. With its leading talent development system and certification standards, it is committed to fostering new ICT talent in the digital era, and building a sound ICT talent ecosystem.

Huawei Certified ICT Professional-Datacom (HCIP-Datacom) is designed for professional engineers who are capable of network automation development. The HCIP-Datacom certification will qualify you as professionals in network automation development, who are capable of automatic deployment, development, and O&M of enterprise networks.

The Huawei certification system introduces the industry, fosters innovation, and imparts cutting-edge datacom knowledge.



About This Document

Introduction

This document is an HCIP-Datacom-Network Automation Developer certification training course and is intended for trainees who are going to take the HCIP-Datacom- Network Automation Developer exam or readers who want to understand network automation knowledge and practices.

Experiments

This lab guide contains the following experiments:

- SSH experiment
- Automatic SNMP configuration experiment
- NETCONF configuration experiment
- Configuration file comparison experiment
- gRPC remote configuration query experiment
- Telemetry configuration experiment
- OPS experiment
- Network flow analysis experiment

Required Background Knowledge

This lab guide is intended for network automation engineers, who are supposed to have the following knowledge and skills:

- Python programming basics
- Basic network knowledge and Huawei switch configurations

Lab Environment

Description

This lab environment is based on Huawei switch devices of VRP8, development environment of Python 3.8, and compiler of Jupyter Notebook or Pycharm.

For the experiments, either physical devices or devices simulated on a network simulation platform can be used. This lab guide uses CE12800 as an example to how network automation based on Python.

Contents

1 SSH Experiment

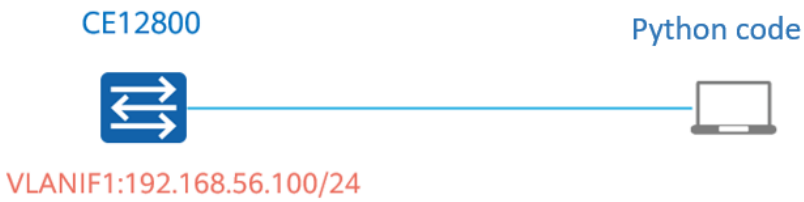
1.1 Background

A company deploys a CE12800 switch whose management IP address is 192.168.56.100/24. You will compile an automation script to capture the current configuration file of the device and upload and download the configuration file.

1.1.1 Objectives

In this experiment, you will learn the common methods of the **Paramiko** module, and be able to use SSH to log in to a specified device to implement automatic configuration and secure file transfer using SFTP.

1.1.2 Environment Preparations



In this experiment, you need to prepare a network device that communicates with the Python compilation environment at Layer 3. In this example, a local PC will set up the Python compilation environment.

1. Configure the CE12800 switch.

```
<HUAWEI>system-view immediately
Enter system view, return user view with return command.
[HUAWEI]sysname CE1
[CE1]interface Vlanif 1
[CE1-Vlanif1]ip add 192.168.56.100 24
[CE1-Vlanif1]quit
```

2. Verify connectivity between the switch and local PC.

   Open the **CMD** window on the local PC and test connectivity between the local PC and CE1.

```
C:\Users\XXX>ping 192.168.56.100
Pinging 192.168.56.100 with 32 bytes of data:
Reply from 192.168.56.100: Bytes = 32, time = 3 ms, TTL = 255
Reply from 192.168.56.100: Bytes = 32, time = 3 ms, TTL = 255
Reply from 192.168.56.100: Bytes = 32, time = 2 ms, TTL= 255
Reply from 192.168.56.100: Bytes = 32, time = 4 ms, TTL = 255
Statistics on ping to 192.168.56.100:
    Packets: 4 packets transmitted, 4 packets received, 0 packet loss (0.0%)
Estimated RTT (ms):
    Minimum = 2 ms, maximum = 4 ms, average = 3 ms
```

   The connection is successful.

1.2 Paramiko SSH Login

Paramiko is the most commonly used SSH module in Python. It implements secure remote command execution and file transfer in either password or public key mode.

This section uses RSA user authentication as an example to describe the configuration process that a client logs in to a server using Python Paramiko SSH

1.2.1 Configuration Roadmap

Configure an SSH server.

1. Configure STelnet. Specially, configure a management IP address, enable the STelnet function, and configure the user interface.
2. Configure users. Specially, create a local user and an SSH user, and configure service type and authentication mode for the users.
3. Configure a public key. Specially, add the public key generated by the client and allocate it to the user.

Configure an SSH client.

1. Create a key pair. Specifically, generate a public key and a private key locally.
2. Compile Python code.
3. Verify client login to the server.

1.2.2 Configurations

Before using a Python script to log in to a device through SSH, you need to create an SSH account and enable the STelnet function on the device.

Step 1 Enable STelnet on the SSH server and configure the VTY user interface.

```
[SSH Server] stelnet server enable
```

```
[SSH Server] user-interface vty 0 4
[SSH Server-ui-vty0-4] authentication-mode aaa
[SSH Server-ui-vty0-4] protocol inbound ssh
[SSH Server-ui-vty0-4] user privilege level 3
[SSH Server-ui-vty0-4] quit
```

Step 2 Create a local user, **python**, on the server, add the user to the administrator group, and configure service type for the user.

```
[SSH Server] aaa
[SSH Server-aaa] local-user python password irreversible-cipher Huawei12#$
[SSH Server-aaa] local-user python user-group manage-ug
[SSH Server-aaa] local-user python service-type ssh
[SSH Server-aaa] quit
```

Step 3 Create an SSH user on the SSH server and configure authentication type and service type for the user.

```
[SSH Server] ssh user python
[SSH Server] ssh user python authentication-type rsa
[SSH Server] ssh user python service-type stelnet
```

Step 4 On the client, create an RSA key pair using **Git Bash** and copy the public key.

In this step, press **Enter** to retain the default value.

```
exampleuser@exampleuser MINGW64 ~
$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/exampleuser/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/exampleuser/.ssh/id_rsa
Your public key has been saved in /c/Users/exampleuser/.ssh/id_rsa.pub
```

Display the public key.

```
$ cat /c/Users/exampleuser/.ssh/id_rsa.pub
```

Step 5 Add the public key to the SSH server and assign it to the user.

```
[SSH Server] rsa peer-public-key rsa01 encoding-type openssh
[SSH Server-rsa-public-key] public-key-code begin
[SSH Server-rsa-public-key-rsa-key-code] ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABgQDwLRx8MmuNs500dRemhFHdDbBmxco8Bp+wyqwaGuHJZBCjyFQV6
AB+ezu5t0eWE3mw57IZfgmvR+MjBcliZv/x3l8oUMLcQKlKslYQDtvfUCZd+za1suXAPB/dyPKMhYPAzSDA7K+xqC
WlmU3q06vxHEPLMv4A5IX54rKtBnK92fWjl9ACU+ak0ZlHxbKwOFn1tr0GJBazcInEs9DKGwkTTqJdu9+5hI5NxXT
SbM3an53805ZbCU18xPy57g7MZC89vbdsag/uvQmFkLJ3arts/Om2R7fhR92EU/SNPmVy+qDEdwZEVdubdqJIn
W+8zzVkPGlnb2oH5hwH78Ksklbxb0fEfmGR0mS1ZAi3ZHUGcEEjuFZona3+5Z0Un2OPxfXwvoljVDusbYcugJHo9
Ssurz05GzVuamQZlcO2JYY6FhtLUAImtXGQ80MpTjB0lcprkAZCib8agYOtVQNTZ7iB0g2EcBN9UTyMz7sh8RtrBD
j445r+XPaDE8LmpDRKHMk=
[SSH Server-rsa-public-key-rsa-key-code] public-key-code end
[SSH Server-key-code] peer-public-key end
[SSH Server] ssh user python assign rsa-key rsa01
```

Step 6 Compile and run the Python code to log in to the server in SSH mode.

----End

1.2.3 Complete Code and Code Execution Result

The Python script calls the **Paramiko** module to log in to CE1, runs the **display current-configuration** command, and displays the command output.

Step 1 Complete code:

```
import paramiko
import time

ssh = paramiko.SSHClient()
ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())

ssh.connect(hostname='192.168.56.100',port=22,username='python',key_filename=r'C:\Users\exampleuser\.ssh
\id_rsa')

cli = ssh.invoke_shell()
cli.send('screen-length 0 temporary\n')
cli.send('display cu\n')
time.sleep(3)

dis_cu = cli.recv(999999).decode()
print(dis_cu)

ssh.close()
```

Step 2 Code execution on the compiler:

Step 3 Output:

```
Info: The max number of VTY users is 5, the number of current VTY users online is 1, and total number of
terminal users online is 2.
        The current login time is 2020-05-12 11:04:56.
<SSH Server>screen-length 0 temporary
Info: The configuration takes effect on the current user terminal interface only.
<SSH Server>display cu
!Software Version V200R005C10SPC607B607
!Last configuration was updated at 2020-05-12 10:46:40+00:00
!Last configuration was saved at 2020-05-12 10:46:42+00:00
#
sysname SSH Server
```

```
#
device board 17 board-type CE-MPUB
device board 1 board-type CE-LPUE
#
rsa peer-public-key rsa01 encoding-type openssh
 public-key-code begin
   ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQDwLRx8MmuNs500dRemhFHdDbBmxco8Bp+wyqwaGuHJ
   ZBCjyFQV6AB+ezu5t0eWE3mw57IZfgmvR+MjBcliZv/x3l8oUMLcQKlKslYQDtvfUCZd+za1suXAPB/d
   yPKMhYPAzSDA7K+xqCWlmU3q06vxHEPLMv4A5IX54rKtBnK92fWjl9ACU+ak0ZlHxbKwOFn1tr0GJBaz
   cInEs9DKGwkTTqJdu9+5hI5NxXTSbM3an53805ZbCU18xPy57g7MZC89vbdsag/uvQmFkLJ3arts/Om2
   R7fhR92EU/SNPmVy+qDEdwZEVdubdqJInW+8zzVkPGlnb2oH5hwH78Ksklbxb0fEfmGR0mS1ZAi3ZHUG
   cEEjuFZona3+5Z0Un2OPxfXwvoljVDusbYcugJHo9Ssurz05GzVuamQZlcO2JYY6FhtLUAImtXGQ80Mp
   TjB0lcprkAZCib8agYOtVQNTZ7iB0g2EcBN9UTyMz7sh8RtrBDj445r+XPaDE8LmpDRKHMk= rsa-key

 public-key-code end
 peer-public-key end
#
aaa
 local-user python password irreversible-cipher
$1c$'&*wWaA2L3$iU"Y!^^Y}VJFkK=_E%)H({;pWU!Zr6]NO<LyLS,0$
 local-user python service-type ssh
 local-user python user-group manage-ug
 #
 authentication-scheme default
 #
 authorization-scheme default
 #
 accounting-scheme default
 #
 domain default
 #
 domain default_admin
#
interface Vlanif1
 ip address 192.168.56.100 255.255.255.0
#
interface MEth0/0/0
 undo shutdown
#
interface GE1/0/0
 undo shutdown
#
interface GE1/0/1
 undo shutdown
#
interface GE1/0/2
 shutdown
#
interface GE1/0/3
 shutdown
#
interface GE1/0/4
 shutdown
#
interface GE1/0/5
```

```
 shutdown
#
interface GE1/0/6
 shutdown
#
interface GE1/0/7
 shutdown
#
interface GE1/0/8
 shutdown
#
interface GE1/0/9
 shutdown
#
interface NULL0
#
stelnet server enable
ssh user python
ssh user python authentication-type rsa
ssh user python assign rsa-key rsa01
ssh user python service-type stelnet
ssh authorization-type default root
#
ssh server cipher aes256_gcm aes128_gcm aes256_ctr aes192_ctr aes128_ctr aes256_cbc aes128_cbc 3des_cbc
#
ssh server dh-exchange min-len 1024
#
ssh client cipher aes256_gcm aes128_gcm aes256_ctr aes192_ctr aes128_ctr aes256_cbc aes128_cbc 3des_cbc
#
user-interface con 0
#
user-interface vty 0 4
 authentication-mode aaa
 user privilege level 3
 protocol inbound ssh
#
vm-manager
#
return
<SSH Server>
```

----End

1.2.4 Code Explaination

Step 1 Import modules.

```
import paramiko
import time
```

Import the **Paramiko** and **time** modules required in the code segment. If the modules have not been installed, run the **pip install paramiko** command to install them.

This section describes the common classes and methods used by **Paramiko** as a client, for example, **SSHClient** as a common class and **AutoAddPolicy**, **connect**, **invoke_shell**, and **close** as methods related to the class. For more information about **Paramiko**, see http://docs.paramiko.org/.

By default, Python executes all code in sequence without any interval. When **Paramiko** sends configuration commands to the switch, the SSH response may be not as timely as usual or incomplete command output may be displayed. In this case, you can use the **sleep** method in the **time** module to manually suspend the program.

Step 2 Instantiate an SSH object.

Use **Paramiko SSHClient()** to instantiate an SSH object. In this example, the value **ssh** is assigned.

```
ssh = paramiko.SSHClient()
```

Step 3 Allow connections to unknown hosts. That is, you do not need to enter **yes** or **no** for confirmation when setting up a new SSH connection.

```
ssh.set_missing_host_key_policy(paramiko.client.AutoAddPolicy())
```

Step 4 Establish an SSH session connection. Specifically, set the IP address of the destination SSH server to 192.168.56.100, user name as **python**, and **key_filename** as the local private key file so that the user will be authenticated in key mode.

```
ssh.connect(hostname='192.168.56.100',username='python',key_filename=r'C:\Users\exampleuser\.ssh\id_rsa')
```

Step 5 Open an interactive session.

After logging in to the device through SSH, use the Python script to run commands on the SSH server.

Invoke **invoke_shell()** to assign a value to **cli**. **invoke_shell()** opens an interactive shell session, which is a logical channel established on an SSH session connection.

```
cli = ssh.invoke_shell()
```

Run the **send()** command to continue to enter commands.

Screen-length 0 temporary displays all the command output in one screen instead of split screens. **display cu** is short for **display current-configuration**.

time.sleep() sets the wait time (in seconds).

```
cli.send('screen-length 0 temporary\n')
cli.send('display cu\n')
time.sleep(3)
```

Step 6 Capture the channel output information.

A logical channel has been created using **invoke_shell()**. All input and output information will be saved in this channel. All information in this channel can be captured and displayed on the Python compiler.

```
dis_cu = cli.recv(999999).decode()
print (dis_cu)
```

Invoke **cli.recv()**, use **decode()** to decode it, and assign a value to **dis_cu**.

recv(999999) receives data over the channel, 999,999 bytes to the maximum.

decode() decodes the bytes object in the specified encoding format, which is UTF-8 by default.

The decoding function is to facilitate reading, as the result is displayed in a new line on the GUI.

```
Info: The max number of VTY users is 5, the number of current VTY users online is 2, and total number of terminal users online is 3.
      The current login time is 2019-11-05 16:43:26.
      The last login time is 2019-11-05 16:43:09 from 192.168.56.1 through SSH.
<CE1>screen-length 0 temporary
Info: The configuration takes effect on the current user terminal interface only.
<CE1>display cu
!Software Version V200R005C10SPC607B607
!Last configuration was updated at 2019-11-05 14:33:28+00:00
#
```

Otherwise, the following information is displayed:

```
nInfo: The max number of VTY users is 5, the number of current VTY users online is 2, and total number of terminal users online is 3.\r\n      The current login time is 2019-11-05 16:45:34.\r\n      The last login time is 2019-11-05 16:43:26 from 192.168.56.1 through SSH.\r\n<CE1>screen-length 0 temporary\r\nInfo: The configuration takes effect on the current user terminal interface only.\r\n<CE1>display cu\r\n!Software Version V200R005C10SPC607B607\r\n!Last configuration was updated at 2019-11-05 14:33:28+00:00\r\n#
```

Step 7 Close a session connection.

```
ssh.close()
```

Invoke **close()** to close the current session connection. As the number of VTY connections on the device is limited, close the SSH session after the script is executed.

----End

1.3 Paramiko SFTP File Transfer

SSH File Transfer Protocol (SFTP) is a secure file transfer protocol based on SSH.

This section uses RSA user authentication as an example to describe how to use **Python Paramiko SFTP** to upload and download files

1.3.1 Configuration Roadmap

Configure the SSH server.

1. Configure SFTP for the device. Specifically, configure the management IP address and enable SFTP on the device.
2. Configure users. Specifically, create an SSH user and configure service type, authentication mode, and SFTP path for the user.
3. Configure a public key. Specifically, add the public key generated by the client and allocate it to the user.
4. Verify the result by checking the uploaded files.

Configure the SSH client.

1. Create a key pair. Specifically, generate a public key and a private key locally.
2. Compile Python code.
3. Verify the result by checking the downloaded files.

1.3.2 Configurations

Before starting SFTP file transfer by running a Python script, you need to create an SFTP account and enable SFTP on the device.

Step 1 Enable the SFTP server function.

```
[SFTP Server] sftp server enable
```

Step 2 Create user **python** and configure authentication type and service type for the user.

```
[SFTP Server] ssh user python
[SFTP Server] ssh user python authentication-type rsa
[SFTP Server] ssh user python service-type sftp
[SFTP Server] ssh user python sftp-directory cfcard:
[SFTP Server] ssh authorization-type default root
```

Step 3 On the client, create an RSA key pair using **Git Bash** and copy the public key.

```
exampleuser@exampleuser MINGW64 ~
$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/exampleuser/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/exampleuser/.ssh/id_rsa
Your public key has been saved in /c/Users/exampleuser/.ssh/id_rsa.pub
```

Copy the displayed public key.

```
$ cat /c/Users/exampleuser/.ssh/id_rsa.pub
```

Step 4 Add the copied public key to the server and assign the public key to the user.

```
[SFTP Server] rsa peer-public-key rsa01 encoding-type openssh
```

```
[SFTP Server-rsa-public-key] public-key-code begin
[SFTP Server-rsa-public-key-rsa-key-code] ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABgQDwLRx8MmuNs500dRemhFHdDbBmxco8Bp+wyqwaGuHJZBCjyFQV6
AB+ezu5t0eWE3mw57IZfgmvR+MjBcliZv/x3l8oUMLcQKlKslYQDtvfUCZd+za1suXAPB/dyPKMhYPAzSDA7K+xqC
WlmU3q06vxHEPLMv4A5IX54rKtBnK92fWjl9ACU+ak0ZlHxbKwOFn1tr0GJBazcInEs9DKGwkTTqJdu9+5hI5NxXT
SbM3an53805ZbCU18xPy57g7MZC89vbdsag/uvQmFkLJ3arts/Om2R7fhR92EU/SNPmVy+qDEdwZEVdubdqJIn
W+8zzVkPGlnb2oH5hwH78Kskbxb0fEfmGR0mS1ZAi3ZHUGcEEjuFZona3+5Z0Un2OPxfXwvoljVDusbYcugJHo9
Ssurz05GzVuamQZlcO2JYY6FhtLUAImtXGQ80MpTjB0lcprkAZCib8agYOtVQNTZ7iB0g2EcBN9UTyMz7sh8RtrBD
j445r+XPaDE8LmpDRKHMk=
[SFTP Server-rsa-public-key-rsa-key-code] public-key-code end
[SFTP Server-key-code] peer-public-key end
[SFTP Server] ssh user python assign rsa-key rsa01
```

Step 5 The client compiles and runs Python code, and uses SFTP to upload files to and download files from the server.

----End

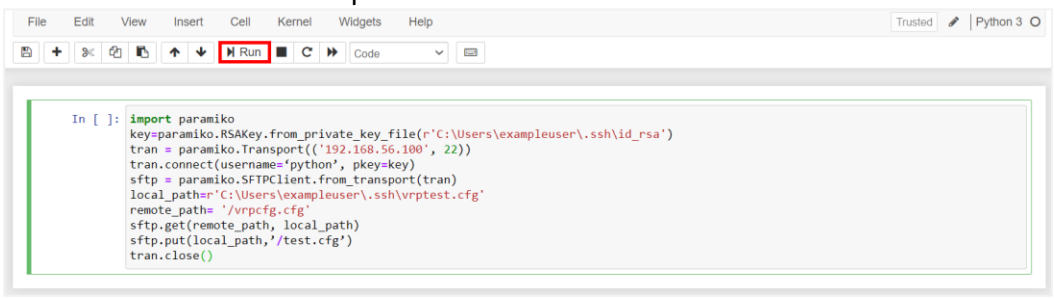
1.3.3 Complete Code and Code Execution Result

The Python script invokes the **Paramiko** module to log in to CE1, downloads the configuration file **vrpcfg.cfg**, and uploads the configuration file **test.cfg**.

Step 1 Complete code:

```
import paramiko
key=paramiko.RSAKey.from_private_key_file(r'C:\Users\exampleuser\.ssh\id_rsa')
tran = paramiko.Transport(('192.168.56.100', 22))
tran.connect(username='python', pkey=key)
sftp = paramiko.SFTPClient.from_transport(tran)
local_path=r'C:\Users\exampleuser\.ssh\vrptest.cfg'
remote_path= '/vrpcfg.cfg'
sftp.get(remote_path, local_path)
sftp.put(local_path,'/test.cfg')
tran.close()
```

Step 2 Code execution on the compiler:



Step 3 Result verification:

Get the downloaded file **vrptest.cfg** from the specified local path (**C:\Users\exampleuser\.ssh\vrptest.cfg** in this example) on the client.

Run the **dir** command on the server. The **test.cfg** file has been uploaded.

```
<SFTP Server>dir
Directory of cfcard:/

  Idx  Attr      Size(Byte)  Date        Time        FileName
    0  dr-x              -  May 13 2020 14:45:05   $_checkpoint
    1  dr-x              -  Apr 28 2020 20:20:09   $_install_mod
    2  dr-x              -  Apr 28 2020 20:20:37   $_license
    3  dr-x              -  May 13 2020 14:44:23   $_security_info
    4  dr-x              -  May 13 2020 14:44:22   $_system
    5  -rw-              0  May 13 2020 14:43:48   CE12800
    6  -rw-            104  Apr 28 2020 20:20:09   VRPV200R005C10SP
_ce12800.cc
    7  -rw-            251  May 13 2020 14:43:48   device.sys
    8  drwx              -  May 06 2020 17:22:02   test
    9  -rw-          2,172  May 13 2020 17:17:01   test.cfg
   10  -rw-          2,172  May 13 2020 14:43:48   vrpcfg.cfg
```

----End

1.3.4 Code Explaination

Step 1 Import the module.

```
import paramiko
```

Import the **paramiko** module to be used in the code segment. If the module has not been installed, run the **pip install paramiko** command to install it.

SFTP involves the **Transport**, **Key Handling**, and **SFTPClient** classes and their methods.

The **Transport** class instantiates session channels and establishes session connections.

The **Key handling** class instantiates key objects.

The **SFTPClient** class creates an SFTP session connection and performs remote file operations.

Step 2 Create an RSA key object.

Read the local RSA private key file on the client and assign a value to the key.

```
key=paramiko.RSAKey.from_private_key_file(r'C:\Users\exampleuser\.ssh\id_rsa')
```

Step 3 Instantiate a session channel. The destination SSH server IP address is 192.168.56.100 and the port number is 22.

```
tran = paramiko.Transport(('192.168.56.100', 22))
```

Step 4 Set up an SSH session connection. Set the user name to **python**, use **pkey** to specify a key object, and use the key for user authentication.

```
tran.connect(username='python', pkey=key)
```

Step 5 Create an SFTP channel based on the opened session connection and assign a value to **sftp**.

```
sftp = paramiko.SFTPClient.from_transport(tran)
```

Step 6 Set the local path and remote path

```
local_path=r'C:\Users\exampleuser\.ssh\vrptest.cfg'
remote_path='/vrpcfg.cfg'
```

Rename the **vrpcfg.cfg** file **vrptest.cfg**.

Step 7 Download the file.

```
sftp.get(remote_path, local_path)
```

Step 8 Upload the file.

```
sftp.put(local_path,'/test.cfg')
```

Step 9 Close the session connection.

```
tran.close()
```

----End

1.4 Quiz

How to use SSH to log in to multiple devices and view their configurations?

2 SNMP Experiment

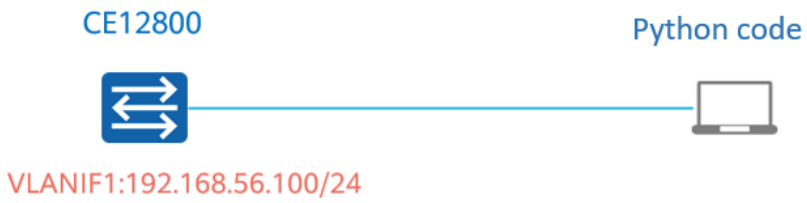
2.1 Background

A company deploys a CE12800 switch whose management IP address is 192.168.56.100/24. You can remotely log in to the switch using SSH. An SNMP configuration script, **snmp.txt**, is available. You will compile an automation script to configure SNMP and then collect SNMP information of the device.

2.1.1 Objectives

- Master the method of using **open** to read and write files.
- Master the common methods of using the **paramiko** module as the SSH client.
- Master the method of using **pysnmp** to collect SNMP information of devices.

2.1.2 Environment Preparations



In this experiment, you need to prepare a network device that communicates with the Python compilation environment at Layer 3.

1. Configure the CE12800 switch.

```
<HUAWEI>system-view immediately
Enter system view, return user view with return command.
[HUAWEI]sysname CE1
[CE1]interface Vlanif 1
[CE1-Vlanif1]ip add 192.168.56.100 24
[CE1-Vlanif1]quit
```

2. Verify connectivity between the switch and local PC.

   Open the **CMD** window on the local PC and test connectivity between the local PC and CE1.

```
C:\Users\XXX>ping 192.168.56.100
Pinging 192.168.56.100 with 32 bytes of data:
Reply from 192.168.56.100: Bytes = 32, time = 3 ms, TTL = 255
Reply from 192.168.56.100: Bytes = 32, time = 3 ms, TTL = 255
Reply from 192.168.56.100: Bytes = 32, time = 2 ms, TTL= 255
Reply from 192.168.56.100: Bytes = 32, time = 4 ms, TTL = 255
Ping statistics for 192.168.56.100:
    Packets: 4 packets transmitted, 4 packets received, 0 packet loss (0.0%)
Estimated RTT (ms):
    Minimum = 2 ms, maximum = 4 ms, average = 3 ms
```

   The connection is successful.

3. SSH has been configured on CE1. The user name is **python** and the password is **Huawei12#$**. (For details, see section 1.2.2 .)

4. Prepare the SNMP script, that is, **snmp.txt** file.

   In the SNMPv3 configuration script, the SNMP user name is **admin** and the password is **Huawei@123**. The complete script is as follows:

```
snmp-agent usm-user v3 admin group dc-admin
snmp-agen usm v3 admin au sha
Huawei@123
Huawei@123
snmp-a usm-user v3 admin pr aes128
Huawei@123
Huawei@123
snmp-agent trap source ME0/0/0
snmp-agent mib-view included nt iso
snmp-agent mib-view included rd iso
snmp-agent mib-view included wt iso
snmp-agent mib-view included iso-view iso
snmp-agent group v3 dc-admin privacy read-view rd write-view wt notify-view nt
```

2.2 Configuration Roadmap

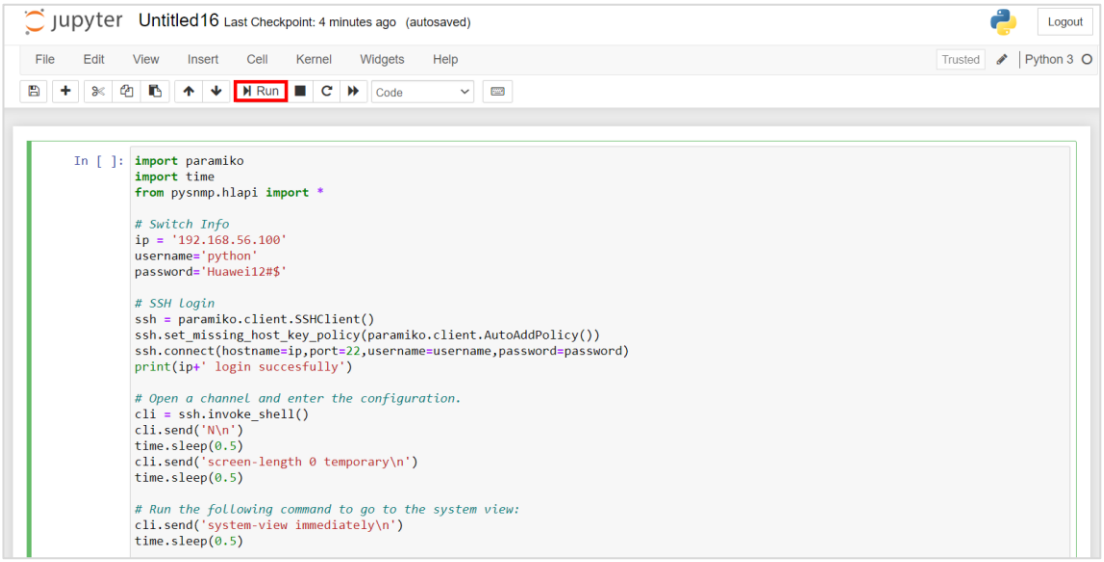
1. Prepare the lab environment.
2. Invoke **paramiko** to log in to the device.
3. Invoke **open** to open the local file **snmp.txt** and configure the device based on **paramiko**.
4. Invoke **pysnmp** to get the host name based on the device SNMP information.

2.3 Complete Code and Code Execution Result

Step 1 Complete code:

```
import paramiko
import time
from pysnmp.hlapi import *

# Switch Info
ip = '192.168.56.100'
username='python'
password='Huawei12#$'

# SSH login
ssh = paramiko.client.SSHClient()
ssh.set_missing_host_key_policy(paramiko.client.AutoAddPolicy())
ssh.connect(hostname=ip,port=22,username=username,password=password)
print(ip+' login succesfully')

# Open a channel and enter the configuration.
cli = ssh.invoke_shell()
cli.send('N\n')
time.sleep(0.5)
cli.send('screen-length 0 temporary\n')
time.sleep(0.5)

# Run the following command to go to the system view:
cli.send('system-view immediately\n')
time.sleep(0.5)

# Read the snmp.txt file in the same local folder line by line and write the file to the SSH channel.
f = open('snmp.txt','r')
snmp_config_list = f.readlines()
for i in snmp_config_list:
    cli.send(i)
    time.sleep(0.5)

# Set up an SNMP channel.
UdpTransportTarget((ip,161))
g = getCmd(SnmpEngine(),

# Obtain the host name of the device.
UsmUserData('admin','Huawei@123','Huawei@123',authProtocol=usmHMACSHAAuthProtocol,privProtocol=u
smAesCfb128Protocol),
                UdpTransportTarget((ip, 161)),
                ContextData(),
                ObjectType(ObjectIdentity('SNMPv2-MIB','sysName',0)))
errorIndication, errorStatus, errorIndex, varBinds =next(g)
for i in varBinds:
    print (i)
    print (str(i).split('=')[1].strip())

dis_this = cli.recv(999999).decode() # View the script interaction process.
print (dis_this)
# Close the session.
ssh.close()
```

Step 2 Code execution on the compiler:



Step 3 Output:

```
192.168.56.100 login succesfully
SNMPv2-MIB::sysName.0 = CE1
CE1

Warning: The initial password poses security risks.
The password needs to be changed. Change now? [Y/N]:N

Info: The max number of VTY users is 5, the number of current VTY users online is 1, and total number of terminal users online is 2.
        The current login time is 2019-11-05 19:43:51.
        The last login time is 2019-11-05 19:43:07 from 192.168.56.1 through SSH.
<CE1>screen-length 0 temporary
Info: The configuration takes effect on the current user terminal interface only.
<CE1>system-view immediately
Enter system view, return user view with return command.
[CE1]snmp-agent usm-user v3 admin group dc-admin
[CE1]snmp-agen usm v3 admin authentication-mode sha
Please configure the authentication password (8-255)
Enter Password:
Confirm Password:
Warning: The privacy and authentication passwords are the same, which is insecure. It is recommended that the privacy and authentication passwords be different.
[CE1]snmp-a usm-user v3 admin privacy-mode aes128
Please configure the privacy password (8-255)
Enter Password:
Confirm Password:
Warning: The privacy and authentication passwords are the same, which is insecure. It is recommended that the privacy and authentication passwords be different.
[CE1]snmp-agent trap source  Vlanif 1
[CE1]snmp-agent mib-view included nt iso
[CE1]snmp-agent mib-view included rd iso
[CE1]snmp-agent mib-view included wt iso
[CE1]snmp-agent mib-view included iso-view iso
```

```
[CE1]snmp-agent group v3 dc-admin privacy read-view rd write-view wt notify-view    nt
[CE1]
```

----End

2.4 Code Explaination

Step 1 Import modules.

```
import paramiko
import time
from pysnmp.hlapi import *
```

Import all modules used by this script. If the **pysnmp** module has not been installed, run the **pip install pysnmp** command to install it.

The **pysnmp** module collects SNMP information. This section describes the **Get** method for synchronizing SNMP to obtain device information. For more information, see http://snmplabs.com/pysnmp/.

Step 2 Log in to the device.

```
# Switch Info
ip = '192.168.56.100'
username='python'
password='Huawei12#$'
```

Create **variables ip**, **username**, and **password** to indicate the SSH host, user name, and password, respectively.

```
# SSH login
ssh = paramiko.client.SSHClient()
ssh.set_missing_host_key_policy(paramiko.client.AutoAddPolicy())
ssh.connect(hostname=ip,port=22,username=username,password=password)
print(ip+' login succesfully')
```

Invoke **paramiko** to log in to the device. If the login is successful, the host IP address login successfully is returned.

```
192.168.56.100 login succesfully
```

Step 3 Enable the SSH channel.

Enable the SSH channel and perform basic configurations.

```
# Open a channel and enter the configuration.
cli = ssh.invoke_shell()
cli.send('N\n')
time.sleep(0.5)
cli.send('screen-length 0 temporary\n')
time.sleep(0.5)

# Run the following command to go to the system view:
cli.send('system-view immediately\n')
```

```
time.sleep(0.5)
```

Invoke **invoke_shell()** to enable the SSH channel. Run the following commands to cancel the split-screen display of command output and go to the system view. The commands are executed at an interval of 0.5 seconds. The CLI interaction process is as follows:

```
<CE1>screen-length 0 temporary
Info: The configuration takes effect on the current user terminal interface only.
<CE1>system-view immediately
Enter system view, return user view with return command.
```

Step 4 Configure SNMP on the device.

Set SNMP parameters. Specifically, set the SNMP user name to **admin**, password to **Huawei@123**, authentication mode to **SHA**, and privacy mode to **AES128**.

```
# Read the snmp.txt file in the same local folder line by line and write the file to the SSH channel.
f = open('snmp.txt','r')
snmp_config_list = f.readlines()
for i in snmp_config_list:
    cli.send(i)
    time.sleep(0.5)
```

Invoke the built-in **open** function of Python to read **snmp.txt** to form a list and assign the value to **snmp_config_list**.

Use the “for” loop to read each line of command and write the command to the SSH channel.

The CLI interaction process is as follows:

```
[CE1]snmp-agent usm-user v3 admin group dc-admin
[CE1]snmp-agen usm v3 admin authentication-mode sha
Please configure the authentication password (8-255)
Enter Password:
Confirm Password:
Warning: The privacy and authentication passwords are the same, which is insecure. It is recommended that
the privacy and authentication passwords be different.
[CE1]snmp-a usm-user v3 admin privacy-mode aes128
Please configure the privacy password (8-255)
Enter Password:
Confirm Password:
Warning: The privacy and authentication passwords are the same, which is insecure. It is recommended that
the privacy and authentication passwords be different.
[CE1]snmp-agent trap source Vlanif 1
[CE1]snmp-agent mib-view included nt iso
[CE1]snmp-agent mib-view included rd iso
[CE1]snmp-agent mib-view included wt iso
[CE1]snmp-agent mib-view included iso-view iso
[CE1]snmp-agent group v3 dc-admin privacy read-view rd write-view wt notify-view    nt
[CE1]
```

Step 5 Set up an SNMP connection with the device.

```
# Set up an SNMP channel.
```

```
UdpTransportTarget((ip,161))
```

Invoke **UdpTransportTarget()** in **pysnmp**. The parameters include the destination IP address and port number.

For details about more parameters, see http://snmplabs.com/pysnmp/docs/api-reference.html#synchronous-snmp.

Step 6 Enable SNMP GET request and response.

The **pysnmp** module is used to implement the **GET** operation of SNMP to obtain the device information.

```
g = getCmd(SnmpEngine(),
UsmUserData('admin','Huawei@123','Huawei@123',authProtocol=usmHMACSHAAuthProtocol,privProtocol=usmAesCfb128Protocol),
            UdpTransportTarget((ip, 161)),
            ContextData(),
            ObjectType(ObjectIdentity('SNMPv2-MIB','sysName',0)))
```

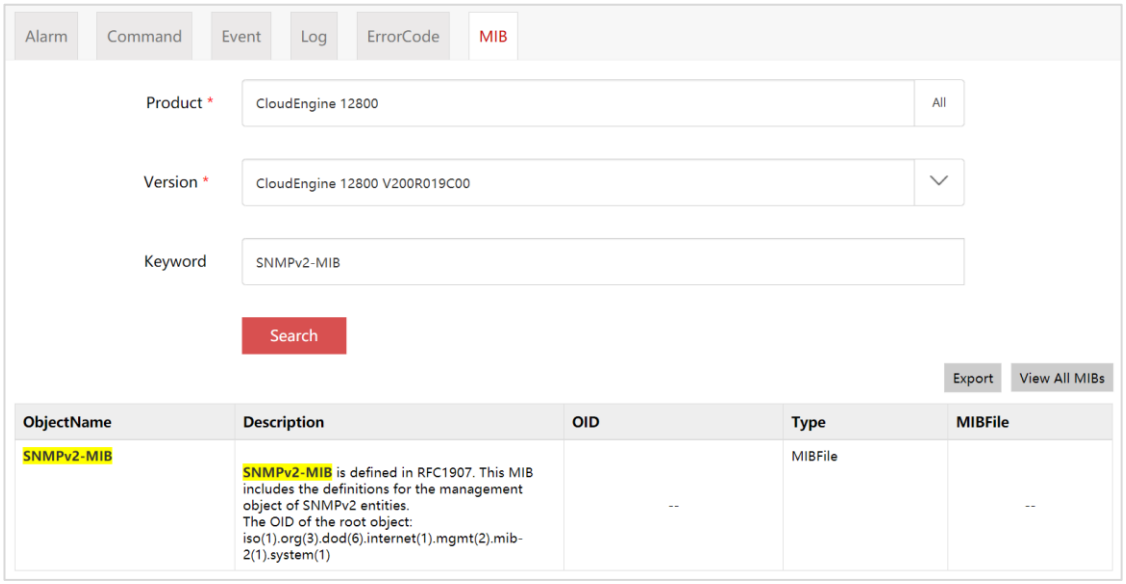
Invoke **getCmd()** to implement the **GET** operation of SNMP and assign the value to **g**.

- **UsmUserData** indicates SNMP user information, including SNMP user name, password, encryption mode, and authentication mode. The SNMP user information must be consistent with the SNMP configurations on the device.
- **UdpTransportTarget** is the transport layer information.
- **ContextData** is used in asynchronous mode. Leave this parameter empty.
- **ObjectType** indicates the device MIB object to be queried. You can use either the object name or OID. The object name is used here.

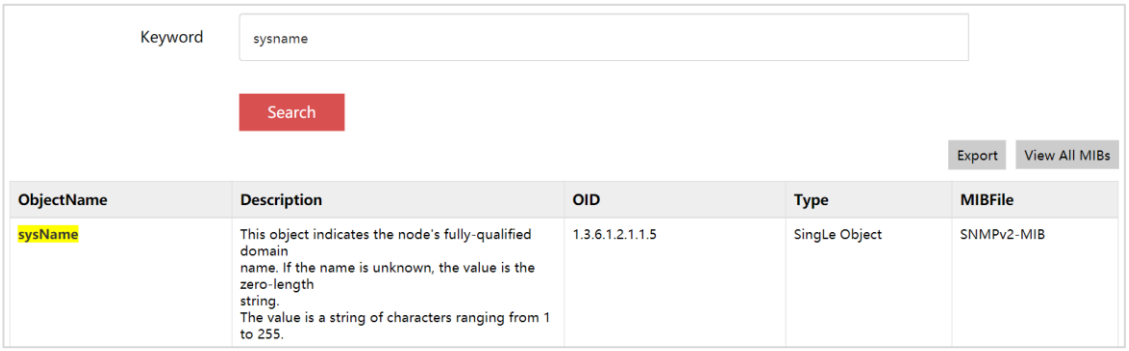
You can query the MIB information of Huawei devices at:

https://support.huawei.com/onlinetoolsweb/infoM/index.do?domain=1&lang=en&topicType=mib

The name of the MIB object file to be queried is **SNMPv2-MIB**.



The object is **sysName**.




```
errorIndication, errorStatus, errorIndex, varBinds = next(g)
```

Create variables **errorIndication**, **errorStatus**, **errorIndex**, and **varBinds** to obtain the returned information of **next(g)**.

For details about the returned information, see the official document at http://snmplabs.com/pysnmp/docs/hlapi/asyncore/sync/manager/cmdgen/getcmd.html. You can customize the variable names

Yields:

- **errorIndication** (*str*) – True value indicates SNMP engine error.
- **errorStatus** (*str*) – True value indicates SNMP PDU error.
- **errorIndex** (*int*) – Non-zero value refers to *varBinds [errorIndex-1]*
- **varBinds** (*tuple*) – A sequence of `ObjectType` class instances representing MIB variables returned in SNMP response.

The **varBinds** variable is a tuple, including the returned message in an SNMP query.

Step 7 Obtain the host name of device.

Process SNMP response data to separate the host name.

```
for i in varBinds:
    print (i)
    print (str(i).split('=')[1].strip())
```

The complete information in the **varBinds** tuple is as follows:

```
SNMPv2-MIB::sysName.0 = CE1
```

The returned message contains the MIB file, MIB object, ID, and host name.

Use the **split** and **strip** methods to obtain a single host name.

Step 8 Close the session.

```
dis_this = cli.recv(999999).decode() # View the script interaction process.
print (dis_this)
# Close the session.
ssh.close()
```

Check the SSH channel interaction process and close the session.

----End

2.5 Quiz

1. How to encapsulate a host name query into a function for other programs to invoke?
2. How to improve the SNMP query efficiency if there are a large number of devices on the live network?

3 NETCONF Experiment

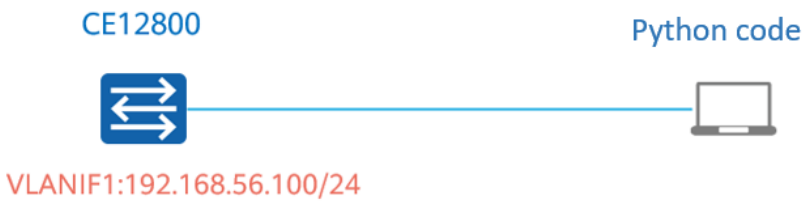
3.1 Background

A company deploys a CE12800 switch whose management IP address is 192.168.56.100/24. You can remotely log in to the switch using SSH. As delivering configurations through SSH is inefficient, you will deliver configurations through NETCONF to devices.

3.1.1 Objectives

- Master the basic usage of **paramiko**.
- Master the basic methods of **ncclient**.

3.1.2 Environment Preparations



In this experiment, you need to prepare a network device that communicates with the Python compilation environment at Layer 3.

1. Configure the CE12800 switch.

```
<HUAWEI>system-view immediately
Enter system view, return user view with return command.
[HUAWEI]sysname CE1
[CE1]interface Vlanif 1
[CE1-Vlanif1]ip add 192.168.56.100 24
[CE1-Vlanif1]quit
```

2. Verify connectivity between the switch and local PC.

   Open the **CMD** window on the local PC and test connectivity between the local PC and CE1.

```
C:\Users\XXX>ping 192.168.56.100
Pinging 192.168.56.100 with 32 bytes of data:
Reply from 192.168.56.100: Bytes = 32, time = 3 ms, TTL = 255
Reply from 192.168.56.100: Bytes = 32, time = 3 ms, TTL = 255
Reply from 192.168.56.100: Bytes = 32, time = 2 ms, TTL= 255
Reply from 192.168.56.100: Bytes = 32, time = 4 ms, TTL = 255
Ping statistics for 192.168.56.100:
    Packets: 4 packets transmitted, 4 packets received, 0 packet loss (0.0%)
Estimated RTT (ms):
    Minimum = 2 ms, maximum = 4 ms, average = 3 ms
```

   The connection is successful.

3. SSH has been configured on CE1. The user name is **python** and the password is **Huawei12#$**. (For details, see section 1.2.2)

4. Prepare the NETCONF pre-configuration script, **netconf.txt**.

   Create a NETCONF user with the user name as **netconf** and password as **Huawei12#$**. The complete script is as follows:

```
system-view immediately
aaa
local-user netconf password irreversible-cipher Huawei12#$
local-user netconf service-type ssh
local-user netconf level 3
quit
ssh user netconf authentication-type password
ssh user netconf service-type snetconf
snetconf server enable
netconf
protocol inbound ssh port 830
quit
```

3.2 Configuration Roadmap

1. Prepare the experiment environment.
2. Invoke **paramiko** to log in to the device.
3. Invoke **open** to open the local file **netconf.txt** and configure NETCONF for the device based on **paramiko**.
4. Invoke **ncclient** to configure the G1/0/2 interface through NETCONF. For the interface, the IP address is 192.168.2.1/24 and description is **Config by NETCONF**.

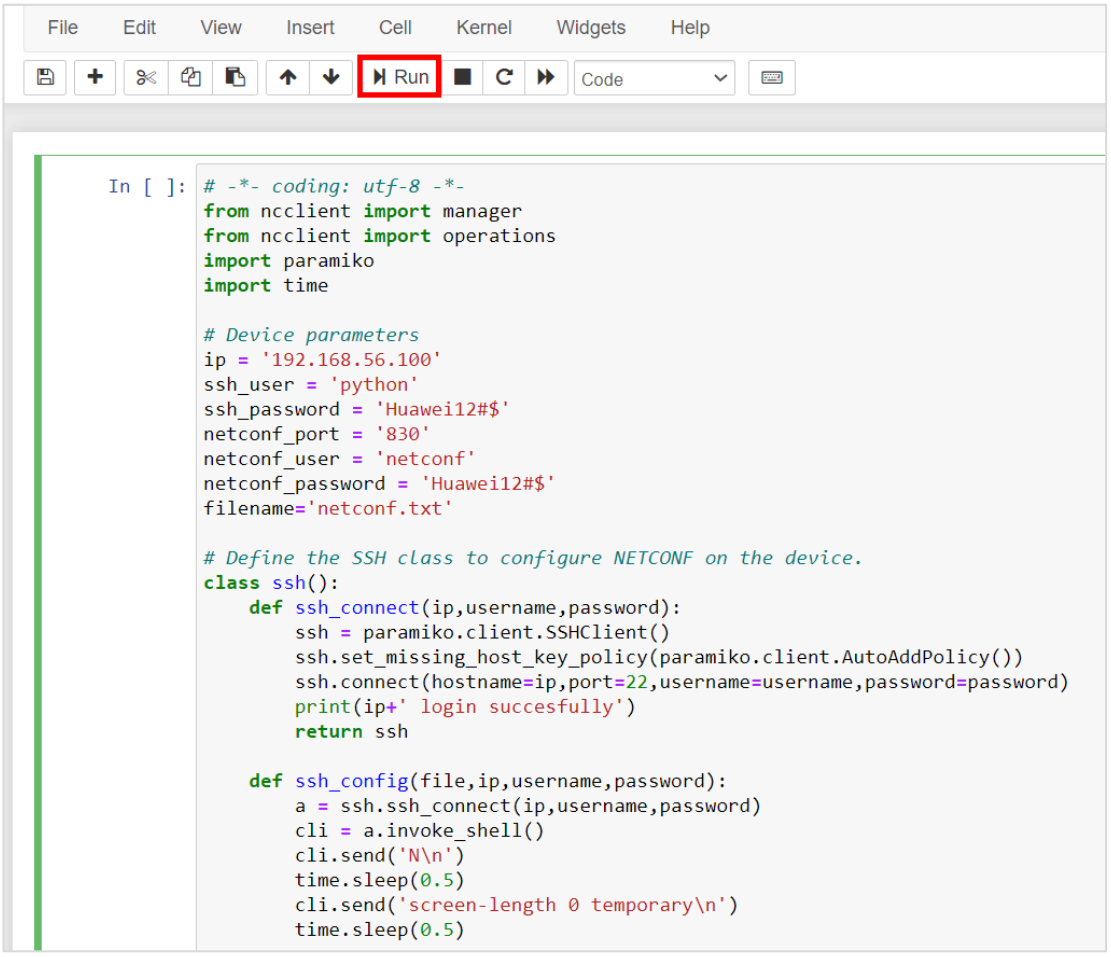
3.3 Complete Code and Code Execution Result

Step 1 Complete code:

```
# -*- coding: utf-8 -*-
from ncclient import manager
from ncclient import operations
import paramiko
import time

# Device parameters
ip = '192.168.56.100'
ssh_user = 'python'
ssh_password = 'Huawei12#$'
netconf_port = '830'
netconf_user = 'netconf'
netconf_password = 'Huawei12#$'
filename='netconf.txt'

# Define the SSH class to configure NETCONF on the device.
class ssh():
    def ssh_connect(ip,username,password):
        ssh = paramiko.client.SSHClient()
        ssh.set_missing_host_key_policy(paramiko.client.AutoAddPolicy())
        ssh.connect(hostname=ip,port=22,username=username,password=password)
        print(ip+' login succesfully')
        return ssh

    def ssh_config(file,ip,username,password):
        a = ssh.ssh_connect(ip,username,password)
        cli = a.invoke_shell()
        cli.send('N\n')
        time.sleep(0.5)
        cli.send('screen-length 0 temporary\n')
        time.sleep(0.5)

        f = open(file,'r')
        config_list = f.readlines()
        for i in config_list:
            cli.send(i)
            time.sleep(0.5)

        dis_this = cli.recv(999999).decode()
        print (dis_this)
        a.close()

# Define the huawei_connect function to establish a NETCONF connection.
def huawei_connect(host, port, user, password):
    return manager.connect(host=host,
                           port=port,
                           username=user,
                           password=password,
                           hostkey_verify = False,
                           device_params={'name': "huawei"},
                           allow_agent = False,
                           look_for_keys = False)

# Use NETCONF to send XML data and configure an IP address for an interface on the device.
CREATE_INTERFACE = '''<config>
```

```
        <ethernet xmlns="http://www.huawei.com/netconf/vrp" content-version="1.0" format-version="1.0">
          <ethernetIfs>
            <ethernetIf operation="merge">
              <ifName>GE1/0/2</ifName>
              <l2Enable>disable</l2Enable>
            </ethernetIf>
          </ethernetIfs>
        </ethernet>
        <ifm xmlns="http://www.huawei.com/netconf/vrp" content-version="1.0" format-version="1.0">
          <interfaces>
            <interface operation="merge">
              <ifName>GE1/0/2</ifName>
              <ifDescr>Config by NETCONF</ifDescr>
              <ifmAm4>
                <am4CfgAddrs>
                  <am4CfgAddr operation="create">
                    <subnetMask>255.255.255.0</subnetMask>
                    <addrType>main</addrType>
                    <ifIpAddr>192.168.2.1</ifIpAddr>
                  </am4CfgAddr>
                </am4CfgAddrs>
              </ifmAm4>
            </interface>
          </interfaces>
        </ifm>
      </config>'''

# Execute the main functions in sequence.
if __name__ == '__main__':
      ssh.ssh_config(filename,ip,ssh_user,ssh_password)
      m = huawei_connect(ip,netconf_port,netconf_user,netconf_password)
      m.edit_config(target='running',config=CREATE_INTERFACE)
```

Step 2 Code execution on the compiler:



Step 3 Output:

Compilation result:

```
192.168.56.100 login succesfully

Warning: The initial password poses security risks.
The password needs to be changed. Change now? [Y/N]:N

Info: The max number of VTY users is 5, the number of current VTY users online is 1, and total number of terminal users online is 2.
        The current login time is 2019-11-06 13:08:58.
        The last login time is 2019-11-06 11:48:16 from 192.168.56.1 through SSH.
<CE1>screen-length 0 temporary
Info: The configuration takes effect on the current user terminal interface only.
<CE1>system-view immediately
Enter system view, return user view with return command.
[CE1]aaa
[CE1-aaa]local-user netconf password irreversible-cipher Huawei12#$
Info: A new user is added.
[CE1-aaa]local-user netconf service-type ssh
[CE1-aaa]local-user netconf level 3
[CE1-aaa]quit
```

```
[CE1]ssh user netconf authentication-type password
[CE1]ssh user netconf service-type snetconf
[CE1]snetconf server enable
Info: The SNETCONF server is already started.
[CE1]netconf
[CE1-netconf]protocol inbound ssh port 830
Info: The ssh port 830 service is already started.
[CE1-netconf]quit
[CE1]
```

Log in to CE1 to check the configurations:

```
[CE1]interface GE 1/0/2
[CE1-GE1/0/2]display this
#
interface GE1/0/2
 undo portswitch
 description Config by NETCONF
 shutdown
 ip address 192.168.2.1 255.255.255.0
#
return
[CE1-GE1/0/2]
```

NETCONF configurations are delivered successfully.

----End

3.4 Code Explaination

Step 1 Import modules.

```
# -*- coding: utf-8 -*-
from ncclient import manager
from ncclient import operations
import paramiko
import time
```

Import all modules used by this script. If **ncclient** has not been installed, run the **pip install ncclient** command to install it.

ncclient is a NETCONF functional module of Python. This section briefly describes the **ncclient manager** and **operation** methods to establish and interact with NETCONF. For more information on **ncclient**, see https://ncclient.readthedocs.io/en/latest/.

Step 2 Define device parameter variables.

```
# Device parameters
ip = '192.168.56.100'
ssh_user = 'python'
ssh_password = 'Huawei12#$'
netconf_port = '830'
netconf_user = 'netconf'
```

```
netconf_password = 'Huawei12#$'
filename='netconf.txt'
```

Define parameter variables of the device, including the host IP address, SSH user, SSH password, NETCONF port, NETCONF user name, NETCONF password, and local file name.

Step 3 Perform NETCONF configurations on the device.

Use **paramiko** and **open** to send the local configuration script, **netconf.txt**, to CE1. In this example, the class encapsulation is used to facilitate invoking by the main function.

```
# Define the SSH class to configure NETCONF.
class ssh():
```

Declare the class name as **ssh**. The class contains two methods (functions), that is, **ssh_connect()** and **ssh_config()**, which are used to establish an SSH connection and send configurations, respectively.

```
    def ssh_connect(ip,username,password):
        ssh = paramiko.client.SSHClient()
        ssh.set_missing_host_key_policy(paramiko.client.AutoAddPolicy())
        ssh.connect(hostname=ip,port=22,username=username,password=password)
        print(ip+' login succesfully')
        return ssh
```

Define the first method **ssh_connect(ip,username,password)** in the **ssh** class and enter the IP address, user name, and password of SSH. This function encapsulates the **paramiko** method. For details, see the preceding sections.

```
    def ssh_config(file,ip,username,password):
        a = ssh.ssh_connect(ip,username,password) # invoke ssh_connect to assign a value to a.
        cli = a.invoke_shell()
        cli.send('N\n')
        time.sleep(0.5)
        cli.send('screen-length 0 temporary\n')
        time.sleep(0.5)

        f = open(file,'r')
        config_list = f.readlines()
        for i in config_list:
            cli.send(i)
            time.sleep(0.5)

        dis_this = cli.recv(999999).decode()
        print (dis_this)
        a.close()
```

The second method in the **ssh** class is **ssh_config(file,ip,username,password)**. Define four parameters, that is, IP address, user name, and password of SSH, and configuration file path.

ssh_config() invokes **ssh_connect()** to connect to the device and then sends configuration commands. Use the **open** function to open the local file,

netconf.txt, and write data to the SSH channel line by line. Finally, check the process of interaction with the device and close the session.

Step 4 Establish a NETCONF connection.

```
# Define the huawei_connect function to establish a NETCONF connection.
def huawei_connect(host, port, user, password):
    return manager.connect(host=host,
                           port=port,
                           username=user,
                           password=password,
                           hostkey_verify = False,
                           device_params={'name': "huawei"},
                           allow_agent = False,
                           look_for_keys = False)
```

Define the **huawei_connect(host, port, user, password)** function. The four input parameters of the function are the IP address, port number, user name, and password of the NETCONF host. The function returns the **manager.connect** method of **ncclient**.

manager.connect is used to establish a NETCONF connection. The parameters are defined in RFC4741. **device_params** has two options for Huawei devices. That is, the value can be **huawei** or **huaweiyang**, corresponding to the IETF YANG model or Huawei YANG model, respectively.

Step 5 Construct an XML configuration file.

Construct an XML configuration file by referring to the *NETCONF Schema API Reference* on the Huawei official website.
https://support.huawei.com/enterprise/en/switches/cloudengine-12800-pid-7542409

```
# Use NETCONF to send XML data and configure an IP address for an interface on the device.
CREATE_INTERFACE = '''<config>
      <ethernet xmlns="http://www.huawei.com/netconf/vrp" content-version="1.0" format-version="1.0">
        <ethernetIfs>
          <ethernetIf operation="merge">
            <ifName>GE1/0/2</ifName>
            <l2Enable>disable</l2Enable>
          </ethernetIf>
        </ethernetIfs>
      </ethernet>
      <ifm xmlns="http://www.huawei.com/netconf/vrp" content-version="1.0" format-version="1.0">
        <interfaces>
          <interface operation="merge">
            <ifName>GE1/0/2</ifName>
            <ifDescr>Config by NETCONF</ifDescr>
            <ifmAm4>
              <am4CfgAddrs>
                <am4CfgAddr operation="create">
                  <subnetMask>255.255.255.0</subnetMask>
                  <addrType>main</addrType>
                  <ifIpAddr>192.168.2.1</ifIpAddr>
                </am4CfgAddr>
```

```
                </am4CfgAddrs>
              </ifmAm4>
            </interface>
          </interfaces>
        </ifm>
      </config>'''
```

NETCONF uses XML files to send configurations. XML is a commonly used text format that allows you to nest and expand data. A complete NETCONF session consists of the transport layer, message layer, operation layer, and content layer. The current XML configuration file contains only the operation layer and content layer.

Basic NETCONF operations include **get-config**, **get**, **edit-config**, **copy-config**, **delete-config**, **lock**, **unlock**, **close-session**, and **kill session**. In this example, the operation layer information is **edit-config**, and the operation attribute is **merge**. This operation edits the target data, which may exist or not in the database. If the target data does not exist, target data is created; if the target data exists, the target data is edited.

The NETCONF content layer is used to edit specific parameters. In this example, disable the Layer 2 function on GE 1/0/2 by running the **undo portswitch** command, change the description to **Config by NETCONF**, and set the IP address to 192.168.2.1/24.

Step 6 Run the main functions.

```
# Execute the main functions in sequence.
if __name__ == '__main__':
    ssh.ssh_config(filename,ip,ssh_user,ssh_password)
    m = huawei_connect(ip,netconf_port,netconf_user,netconf_password)
    m.edit_config(target='running',config=CREATE_INTERFACE)
```

Run the main functions.

Run **ssh.ssh_config(filename,ip,ssh_user,ssh_password)** to invoke the **ssh_config** method in the **ssh** class. The input parameters are the variables defined in Step 2.

Then run **huawei_connect(ip,netconf_port,netconf_user,netconf_password)** to assign a value to **m**. Enter NETCONF parameters to establish a NETCONF connection.

Finally run **m.edit_config(target='running',config=CREATE_INTERFACE)**. Use the **edit_config** method to send the XML configuration file constructed in Step 5 to the device. Now you can log in to the device to verify the result. The configurations are successfully delivered.

----End

3.5 Quiz

1. Why if has to be used to build a main function?
2. What is the difference between the **get** operation by NETCONF and the **query** operation by SNMP to obtain device information?

4 Configuration File Comparison Experiment

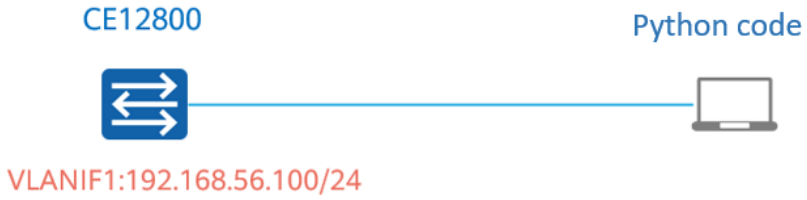
4.1 Background

A company has multiple network devices (one in this example). To better manage configurations, you need to use code to compare the configurations of each device and output the configuration comparison result.

4.1.1 Objectives

- Master the **difflib** method for text comparison.
- Master data processing by regular expressions.
- Master device configurations by **paramiko**.

4.1.2 Environment Preparations



In this experiment, you need to prepare a network device that communicates with the Python compilation environment at Layer 3.

1. Configure the CE12800 switch.

```
<HUAWEI>system-view immediately
Enter system view, return user view with return command.
[HUAWEI]sysname CE1
[CE1]interface Vlanif 1
```

```
[CE1-Vlanif1]ip add 192.168.56.100 24
[CE1-Vlanif1]quit
```

2. Verify connectivity between the switch and local PC.

   Open the **CMD** window on the local PC and test connectivity between the local PC and CE1.

```
C:\Users\XXX>ping 192.168.56.100
Pinging 192.168.56.100 with 32 bytes of data:
Reply from 192.168.56.100: Bytes = 32, time = 3 ms, TTL = 255
Reply from 192.168.56.100: Bytes = 32, time = 3 ms, TTL = 255
Reply from 192.168.56.100: Bytes = 32, time = 2 ms, TTL= 255
Reply from 192.168.56.100: Bytes = 32, time = 4 ms, TTL = 255
Ping statistics for 192.168.56.100:
    Packets: 4 packets transmitted, 4 packets received, 0 packet loss (0.0%)
Estimated RTT (ms):
    Minimum = 2 ms, maximum = 4 ms, average = 3 ms
```

   The connection is successful.

3. SSH has been configured on CE1. The user name is **python** and the password is **Huawei12#$**. (For details, see section 1.2.2 .)

4.2 Complete Code and Code Execution Result

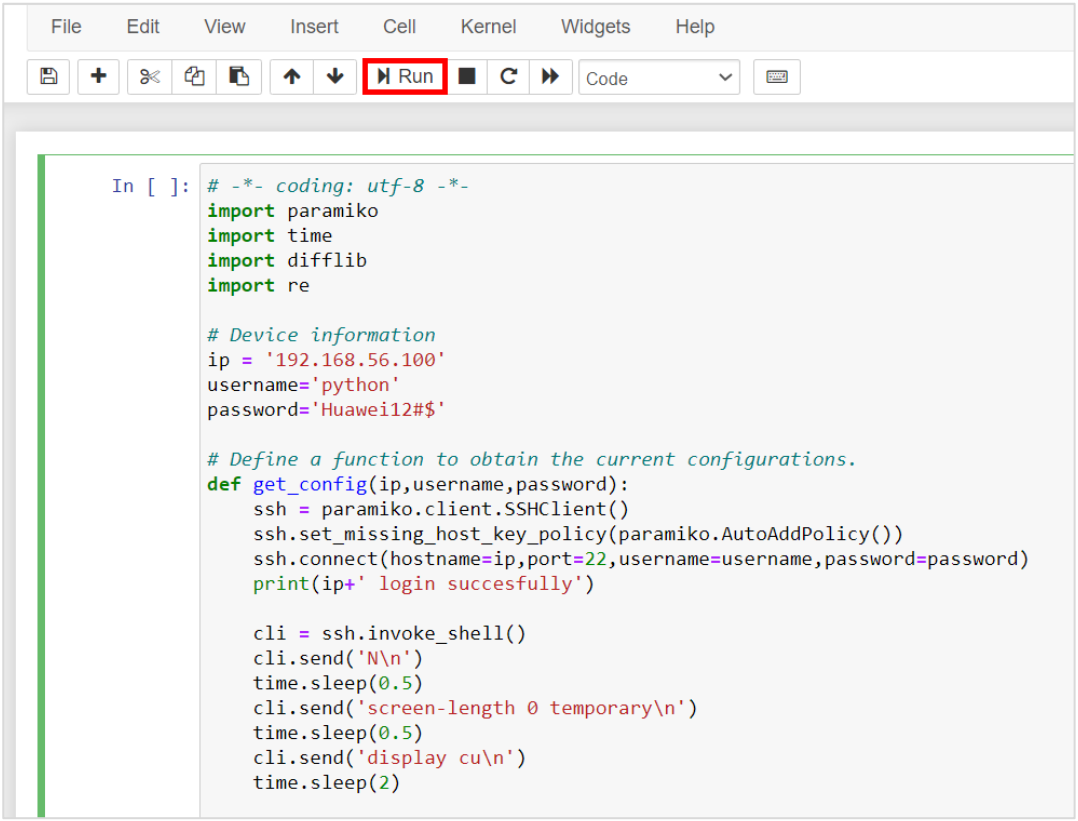
Step 1 Complete code:

```
# -*- coding: utf-8 -*-
import paramiko
import time
import difflib
import re

# Device information
ip = '192.168.56.100'
username='python'
password='Huawei12#$'

# Define a function to obtain the current configurations.
def get_config(ip,username,password):
    ssh = paramiko.client.SSHClient()
    ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh.connect(hostname=ip,port=22,username=username,password=password)
    print(ip+' login succesfully')

    cli = ssh.invoke_shell()
    cli.send('N\n')
    time.sleep(0.5)
    cli.send('screen-length 0 temporary\n')
    time.sleep(0.5)
    cli.send('display cu\n')
    time.sleep(2)

    dis_cu = cli.recv(999999).decode()
```

```
    return (dis_cu)
    ssh_client.close()

# Define the ssh_config function to write the script to the device.
def ssh_config(file,ip,username,password):
    ssh = paramiko.client.SSHClient()
    ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh.connect(hostname=ip,port=22,username=username,password=password)
    print(ip+' login succesfully')

    cli = ssh.invoke_shell()
    cli.send('N\n')
    time.sleep(0.5)
    cli.send('screen-length 0 temporary\n')
    time.sleep(0.5)

    f = open(file,'r')
    config_list = f.readlines()
    for i in config_list:
        cli.send(i)
        time.sleep(0.5)

    dis_this = cli.recv(999999).decode()
    #print (dis_this)
    ssh.close()

# Invoke get_config to assign a value to output.
output = get_config(ip,username,password)
#print (output)

# Process data and use regular expressions to obtain only configurations.
config = re.findall(r'(<CE1>display cu[\d\D]+<CE1>$)',output)
#print (config)

# Save the configurations to local file1.
with open(r'D:\Config\file1','w') as f:
    f.writelines(config[0])

# Invoke ssh_config to write the netconf.txt configurations to the device.
ssh_config('netconf.txt',ip,username,password)

# Read the configurations again and save them to local file2.
output = get_config(ip,username,password)
config = re.findall(r'(<CE1>display cu[\d\D]+<CE1>$)',output)

with open(r'D:\Config\file2','w') as f:
    f.writelines(config[0])

# Compare configurations.
d = difflib.HtmlDiff()

# Define a function to read files.
def read_file(filename):
    try:
        with open(filename,'r') as f:
```

```
                return f.readlines()
        except IOError:
            print('%s The file is not found.'% filename)
            sys.exit(1)

# Define the compare_files function to compare configurations and save the comparison as result.html.
def compare_files(file1,file2,out_file):
    file1_content = read_file(file1)
    file2_content = read_file(file2)
    d = difflib.HtmlDiff()
    result = d.make_file(file1_content,file2_content)
    with open(r'D:\Config\result.html','w') as f:
        f.writelines(result)
    print ()

# Invoke compare_files.
compare_files(r'D:\Config\file1',r'D:\Config\file2',r'D:\Config\result.html')
```

Step 2 Code execution on the compiler:



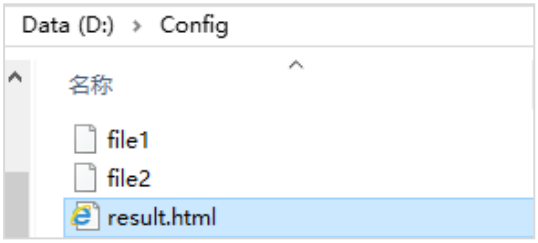
Step 3 Output:

Compilation result:

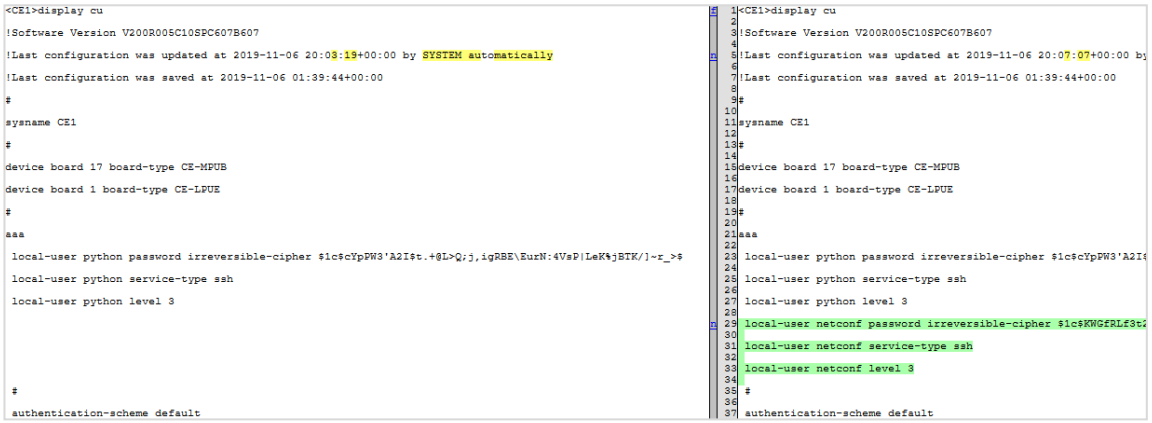
```
192.168.56.100 login successfully
192.168.56.100 login successfully
```

```
192.168.56.100 login successfully
```

You will find the configuration files and comparison file in the directory.



Open the **result.html** file.



----End

4.3 Code Explaination

Step 1 Import modules.

```
# -*- coding: utf-8 -*-
import paramiko
import time
import difflib
import re
```

Import all modules used by this script. If a module does not exist, run **pip install** [*module name*] to install the module.

In this example, the **difflib** module is used to compare texts. The **re** module is used to process data using regular expressions.

Step 2 Define device parameter variables.

```
# Device information
ip = '192.168.56.100'
```

```
username='python'
password='Huawei12#$'
```

Define the parameter variables of the device, that is, host IP address, SSH user, and SSH password.

Step 3 Define the **get_config()** function.

Define the **get_config()** function to obtain the current device configurations.

```
# Define a function to obtain the current configurations.
def get_config(ip,username,password):
    ssh = paramiko.client.SSHClient()
    ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh.connect(hostname=ip,port=22,username=username,password=password)
    print(ip+' login succesfully')

    cli = ssh.invoke_shell()
    cli.send('N\n')
    time.sleep(0.5)
    cli.send('screen-length 0 temporary\n')
    time.sleep(0.5)
    cli.send('display cu\n')
    time.sleep(2)

    dis_cu = cli.recv(999999).decode()
    return (dis_cu)
    ssh_client.close()
```

In the **get_config(ip,username,password)** function, enter the IP address, user name, and password. Invoke **paramiko** in the function to log in to the device and run **display current-configuration** to collect the command output. Return the command output. For details about the **paramiko** method, see the experiment of device login by **Paramiko**.

Step 4 Define the **ssh_config()** function.

Define the **ssh_config()** function to send the configuration script to the device.

```
# Define the ssh_config function to write the script to the device.
def ssh_config(file,ip,username,password):
    ssh = paramiko.client.SSHClient()
    ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh.connect(hostname=ip,port=22,username=username,password=password)
    print(ip+' login succesfully')

    cli = ssh.invoke_shell()
    cli.send('N\n')
    time.sleep(0.5)
    cli.send('screen-length 0 temporary\n')
    time.sleep(0.5)
# Read the file and write the content to the device line by line.
    f = open(file,'r')
    config_list = f.readlines()
    for i in config_list:
```

```
            cli.send(i)
            time.sleep(0.5)

        dis_this = cli.recv(999999).decode()
#       print (dis_this)
ssh.close()
```

----End

4.4 Quiz

1. Why is “192.168.56.100 login successfully” displayed for three times in the compilation result?
2. In this case, some scripts are redundant. Which part can be optimized?
3. Can the configuration files be saved and compared as scheduled? Is there any other way easier to get configuration files?

5 gRPC Remote Configuration Query Experiment

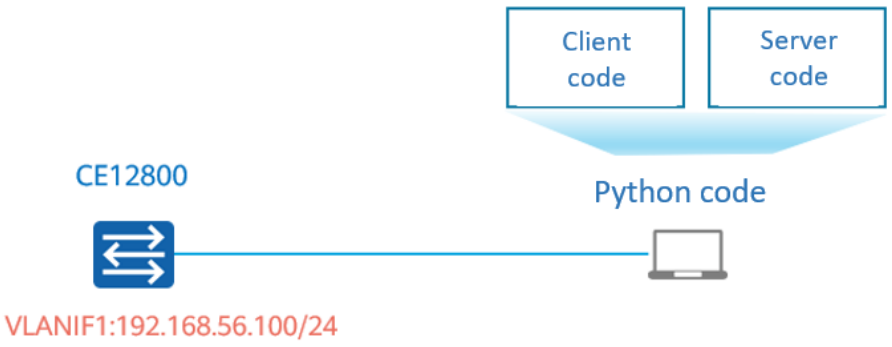
5.1 Background

A company deploys a CE12800 switch whose management IP address is 192.168.56.100. In a special requirement, the client needs to invoke the server through **gRPC** to query and return the current device configurations.

5.1.1 Objectives

- Master the method of user-defining a **.proto** file.
- Master gRPC server code compilation.
- Master gRPC client code compilation.

5.1.2 Environment Preparations



Prepare a simulated environment using a simulator or prepare a real environment. Code is compiled on a local computer for the client and server.

For details about how to set up an environment, see Chapter 1 SSH Experiment.

1. SSH has been configured on CE1. The user name is **python** and the password is **Huawei12#$**.
2. Set the device management address to 192.168.56.100/24.
3. Create two Python files, one as the client and the other as the server, on Juypter Notebook or any other compiler.

5.2 Configuration Roadmap

1. Compile a **.proto** file. Define the services and methods of gRPC request and response.
2. Generate Python code for the client and server based on the **.proto** file.
3. Compile the server code.
4. Compile the client code.

5.3 Configuration Procedure and Complete Code

5.3.1 Compiling the .proto File

```
syntax = "proto3";
package get_config;
// The get_config service definition.
service get_config {
   // RFC request and reply
   rpc Login_info (Request) returns (Reply) {}
}
// The request message containing the login information.
message Request {
   string host = 1;
   string username = 2;
   string password = 3;
}
// The response message containing the string reply.
message Reply {
   string message = 1;
}
```

Save the file in **get_config.proto** format.

The **get_config** service is defined in the **.proto** file. For the **Login_info()** method, the input parameter **Request** contains three string parameters: host, username, and password. For the returned parameter **Reply**, there is only one string parameter, that is, message.

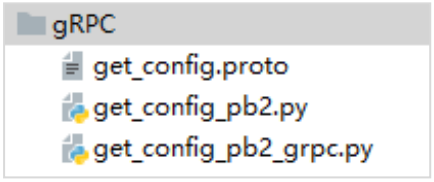
5.3.2 Generating Client and Server Codes

Install and run **grpcio-tools** to generate code. For details, see https://grpc.io/docs/tutorials/basic/python/.

Run the following command in the directory where the **.proto** file is saved:

```
C:\Users\Richard\HCIP\gRPC>python -m grpc_tools.protoc -I./ --python_out=. --
grpc_python_out=. ./get_config.proto
```

After the command is executed, the client and server code is automatically generated.



5.3.3 Complete Server Code

```
from concurrent import futures
import time
import grpc
import get_config_pb2
import get_config_pb2_grpc
import paramiko

class Display_Config(get_config_pb2_grpc.get_configServicer):
    # Invoke paramiko to log in to the device and obtain the current configurations.
    def Login_info(self, request, context):
        ssh = paramiko.client.SSHClient()
        ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
        ssh.connect(hostname=request.host, port=22, username=request.username,
password=request.password)
        cli = ssh.invoke_shell()
        cli.send('N\n')
        time.sleep(0.5)
        cli.send('screen-length 0 temporary\n')
        time.sleep(0.5)
        cli.send('display cu\n')
        time.sleep(3)
        data = cli.recv(999999).decode()
        ssh.close()
        # Return the configuration information in the command output.
        return get_config_pb2.Reply(message=data)

def serve():
    # Create the gRPC service.
    server = grpc.server(futures.ThreadPoolExecutor(max_workers=10))
    # Deploy the gRPC server in the defined service.
    get_config_pb2_grpc.add_get_configServicer_to_server(Display_Config(),server)
```

```
    # Start the server.
    server.add_insecure_port('localhost:8080')
    server.start()
    _ONE_DAY_IN_SECONDS = 60 * 60 * 24
    try:
        while True:
            time.sleep(_ONE_DAY_IN_SECONDS)
    except KeyboardInterrupt:
        server.stop()

if __name__ == "__main__":
    serve()
```

Run the server and keep listening.

5.3.4 Complete Client Code

```
import grpc
import get_config_pb2
import get_config_pb2_grpc

def run():
    # Instantiate stub on the client.
    connect = grpc.insecure_channel('localhost:8080')
    stub = get_config_pb2_grpc.get_configStub(channel=connect)
    # Invoke the Login_info method of the server through stub.
    response =
stub.Login_info(get_config_pb2.Request(host='192.168.56.100',username='python',password='Huawei12#$'))
    print (response.message)

if __name__ == "__main__":
    run()
```

Run the client and enter the IP address, user name, and port number of the device of login. The server returns the following information:

```
Warning: The initial password poses security risks.
The password needs to be changed. Change now? [Y/N]:N

Info: The max number of VTY users is 5, the number of current VTY users online is 1, and total number of
terminal users online is 2.
        The current login time is 2020-01-31 15:58:01.
        The last login time is 2020-01-31 15:57:20 from 192.168.56.1 through SSH.
<CE1>screen-length 0 temporary
Info: The configuration takes effect on the current user terminal interface only.
<CE1>display cu
!Software Version V200R005C10SPC607B607
!Last configuration was updated at 2020-01-31 15:31:54+00:00 by SYSTEM automatically
!Last configuration was saved at 2019-11-06 01:39:44+00:00
#
sysname CE1
#
device board 17 board-type CE-MPUB
device board 1 board-type CE-LPUE
#
```

```
aaa
 local-user python password irreversible-cipher
$1c$cYpPW3'A2I$t.+@L>Q;j,igRBE\EurN:4VsP|LeK%jBTK/]~r_>$
 local-user python service-type ssh
 local-user python level 3
 #
 authentication-scheme default
 #
 authorization-scheme default
 #
 accounting-scheme default
 #
 domain default
 #
 domain default_admin
#
interface Vlanif1
 ip address 192.168.56.100 255.255.255.0
#
interface MEth0/0/0
 undo shutdown
#
interface GE1/0/0
 undo shutdown
#
interface GE1/0/1
 shutdown
#
interface GE1/0/2
 shutdown
#
interface GE1/0/3
 shutdown
#
interface GE1/0/4
 shutdown
#
interface GE1/0/5
 shutdown
#
interface GE1/0/6
 shutdown
#
interface GE1/0/7
 shutdown
#
interface GE1/0/8
 shutdown
#
interface GE1/0/9
 shutdown
#
interface NULL0
#
snmp-agent
```

```
snmp-agent local-engineid 800007DB03707BE82F1330
#
snmp-agent sys-info version v3
snmp-agent group v3 dc-admin privacy read-view rd write-view wt notify-view nt
#
snmp-agent mib-view included nt iso
snmp-agent mib-view included rd iso
snmp-agent mib-view included wt iso
snmp-agent mib-view included iso-view iso
snmp-agent usm-user v3 admin
snmp-agent usm-user v3 admin group dc-admin
snmp-agent usm-user v3 admin authentication-mode sha
cipher %^%#FQ]J>Ba"e*F{#/+rx$UUZ2^:BaG>V/LoKc8No%DE%^%#
snmp-agent usm-user v3 admin privacy-mode aes128
cipher %^%#BEj8L``EE,88ziP^'jFW5A+EB"uqg'wdaS+Y9$F.%^%#
#
snmp-agent trap source Vlanif1
#
stelnet server enable
ssh authorization-type default aaa
#
ssh server cipher aes256_gcm aes128_gcm aes256_ctr aes192_ctr aes128_ctr aes256_cbc aes128_cbc 3des_cbc
#
ssh server dh-exchange min-len 1024
#
ssh client cipher aes256_gcm aes128_gcm aes256_ctr aes192_ctr aes128_ctr aes256_cbc aes128_cbc 3des_cbc
#
user-interface con 0
#
user-interface vty 0 4
 authentication-mode aaa
 user privilege level 3
#
vm-manager
#
return
<CE1>

Process finished with exit code 0
```

5.4 Code Explaination

5.4.1 Server Code

To compile the gRPC server code, perform the following steps:

1. Invoke the service defined by **.proto**. The generated **.py** file has implemented the service, and the server only needs to invoke the service.
2. Compile the functional code of the server.
3. Run the server.

Create the subclass **Display_Config()**, which inherits the attributes and methods of the **get_config_pb2_grpc** class.

```
class Display_Config(get_config_pb2_grpc.get_configServicer):
```

The **Login_info** function defines the RPC request and reply.

```
def Login_info(self, request, context):
```

The structure of RPC request has been defined in **protocol buffers**, and it has three attributes, that is, host, username, and password. In this case, the server receives parameters transferred from the client, invoke the **paramiko** module, and execute the commands for logging in to the device through SSH and querying the current configurations. Finally, data is returned.

```
    # Invoke paramiko to log in to the device and obtain the current configurations.
        ssh = paramiko.client.SSHClient()
        ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
        ssh.connect(hostname=request.host, port=22, username=request.username,
password=request.password)
        cli = ssh.invoke_shell()
        cli.send('N\n')
        time.sleep(0.5)
        cli.send('screen-length 0 temporary\n')
        time.sleep(0.5)
        cli.send('display cu\n')
        time.sleep(3)
        data = cli.recv(999999).decode()
        ssh.close()
        # Return the configuration information in the command output.
        return get_config_pb2.Reply(message=data)
```

Create the **serve()** function to instantiate the gRPC service, which is invoked and executed by the main function.

In this case, start port 8080 of the server and listen on this port for a whole day. Wait for the client to invoke the service.

```
def serve():
    # Create the gRPC service.
    server = grpc.server(futures.ThreadPoolExecutor(max_workers=10))
    # Deploy the gRPC server in the defined service.
    get_config_pb2_grpc.add_get_configServicer_to_server(Display_Config(),server)
    # Start the server.
    server.add_insecure_port('localhost:8080')
    server.start()
    _ONE_DAY_IN_SECONDS = 60 * 60 * 24
    try:
        while True:
            time.sleep(_ONE_DAY_IN_SECONDS)
    except KeyboardInterrupt:
        server.stop()

if __name__ == "__main__":
    serve()
```

5.4.2 Client Code

To compile the gRPC client code, perform the following steps:

1. Create **stub** to invoke services later.
2. Invoke service methods through **stub**.

Create **stub** to connect to the server.

```
def run():
    # Instantiate stub on the client.
    connect = grpc.insecure_channel('localhost:8080')
    stub = get_config_pb2_grpc.get_configStub(channel=connect)
```

Use **stub** to invoke the **Login_info** method of the server.

In this example, enter the IP address, user name, and password of the device for query. Then, the current device configurations are returned.

```
    response =
stub.Login_info(get_config_pb2.Request(host='192.168.56.100',username='python',password='Huawei12#$'))
    print (response.message)

if __name__ == "__main__":
    run()
```

In this experiment, the configuration query is executed on the server and the query result is returned to the client.

5.5 Quiz

What are the benefits of gRPC? Are there any other applications of gRPC in your mind?

6 Telemetry Experiment

6.1 Background

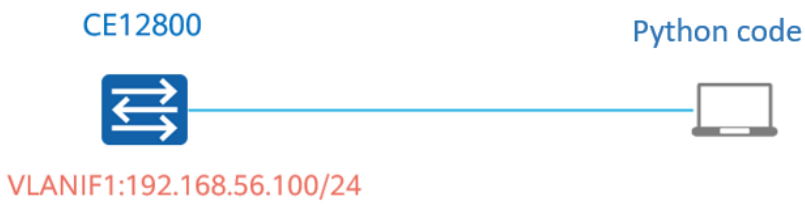
A company deploys a CE12800 switch whose management IP address is 192.168.56.100. To better collect device performance data, it is required that the device push CPU information to the server in Telemetry static subscription mode.

6.1.1 Objectives

- Master server code compilation for static collection by Huawei Telemetry.
- Master the common methods defined in a Huawei **.proto** file.

6.1.2 Environment Preparations

Prepare a simulated environment using a simulator or prepare a physical environment. Run the Python script on the local PC as the server (in this example, the IP address as 192.168.56.1). The switch pushes data to the server (in this example, the IP address of CE1 as 192.168.56.100).



1. Configure the CE12800 switch.

```
<HUAWEI>system-view immediately
Enter system view, return user view with return command.
[HUAWEI]sysname CE1
[CE1]interface Vlanif 1
[CE1-Vlanif1]ip add 192.168.56.100 24
[CE1-Vlanif1]quit
```

2. Verify connectivity between the switch and local PC.

   Open the **CMD** window on the local PC and test connectivity between the local PC and CE1.

```
C:\Users\XXX>ping 192.168.56.100
Pinging 192.168.56.100 with 32 bytes of data:
Reply from 192.168.56.100: Bytes = 32, time = 3 ms, TTL = 255
Reply from 192.168.56.100: Bytes = 32, time = 3 ms, TTL = 255
Reply from 192.168.56.100: Bytes = 32, time = 2 ms, TTL= 255
Reply from 192.168.56.100: Bytes = 32, time = 4 ms, TTL = 255
Ping statistics for 192.168.56.100:
    Packets: 4 packets transmitted, 4 packets received, 0 packet loss (0.0%)
Estimated RTT (ms):
    Minimum = 2 ms, maximum = 4 ms, average = 3 ms
```

   The connection is successful.

3. SSH has been configured on CE1. The user name is **python** and the password is **Huawei12#$**. (For details, see section 1.2.2 .)

6.2 Configuration Roadmap

1. Set up an environment.
2. Configure Telemetry static subscription on the device, including content to be collected, push object, and push interval.
3. Download the **.proto** file of Huawei device from Huawei official website. (Download the **.proto** file in the same way as you download the system software.) Visit Huawei enterprise technical support website (http://support.huawei.com/enterprise) or Huawei carrier technical support website (http://support.huawei.com/carrier) and search for the device model and version. Go to the software download page and obtain the **.proto** file of the required version. Compile the **.proto** file to obtain the method invoked by the server.
4. Compile the server code to listen on the specified port to obtain data.
5. Based on the data to be sent, select an appropriate method to decode the data.

6.3 Configuration Procedure and Complete Code

6.3.1 Configuring the Switch

Ensure that the switch and server are reachable at Layer 3, and configure Telemetry static subscription on the switch.

Step 1 Go to the Telemetry view.

```
<CE1> system-view immediately
Enter system view, return user view with return command.
```

```
[CE1] telemetry
[CE1-telemetry]
```

Step 2 Configure the push object on the device.

In this example, create a destination group, Dest1. The target IP address is 192.168.56.1 and the port number is 20000.

```
[CE1-telemetry] destination-group Dest1
[CE1-telemetry-destination-group-Dest1] ipv4-address 192.168.56.1 port 20000 protocol grpc no-tls
```

Step 3 Configure sampling data on the device.

When you configure static telemetry subscription to the data sampled, you need to create a sampling sensor group and specify a sampling path.

In this example, the sampling group Sensor1 is created. The sampling path is CPU information.

```
[CE1-telemetry] sensor-group Sensor1
[CE1-telemetry-sensor-group-Sensor1] sensor-path huawei-devm:devm/cpuInfos/cpuInfo
```

Step 4 Create a static subscription.

Create a subscription and associate the configured destination group with the sampling sensor group to send data. In this example, associate the destination group Dest1 with the sensor group Sensor1, and set the sampling interval to 1,000 ms.

```
[CE1-telemetry]subscription Sub1
[CE1-telemetry-subscription-Sub1]destination-group Dest1
[CE1-telemetry-subscription-Sub1]sensor-group Sensor1 sample-interval 1000
```

In this way, the device continuously pushes data to the target.

----End

6.3.2 Compiling the .proto File

You need to generate the gRPC client and server interfaces from the **.proto** service definition. To do this, you can use the protocol buffer compiler(protoc), and a special gRPC Python plug-in. Ensure that you have installed Protoc and the gRPC Python plug-in. For more information, see https://grpc.io/docs/tutorials/basic/python/.

In this example, compile the **.proto** file using the **run_codegen.py** script. Place all **.proto** files in the **/protos** directory. This script compiles **huawei-grpc-dialout.proto**, **huawei-telemetry.proto**, and **huawei-devm.proto** at a time. If you want to compile more files, you can add them in the same format.

```
"""Generates protocol messages and gRPC stubs."""

from grpc_tools import protoc

protoc.main(
```

```
    (
        '',
        '-I./protos',
        '--python_out=.',
        '--grpc_python_out=.',
        './protos/huawei-grpc-dialout.proto', # Specify the file path.
    )
)
protoc.main(
    (
        '',
        '-I./protos',
        '--python_out=.',
        '--grpc_python_out=.',
        './protos/huawei-telemetry.proto',
    )
)
protoc.main(
    (
        '',
        '-I./protos',
        '--python_out=.',
        '--grpc_python_out=.',
        './protos/huawei-devm.proto',
    )
)
```

After the compilation is complete, the following Python file is generated in the **run_codegen.py** directory:

6.3.3 Complete Code of Python Server

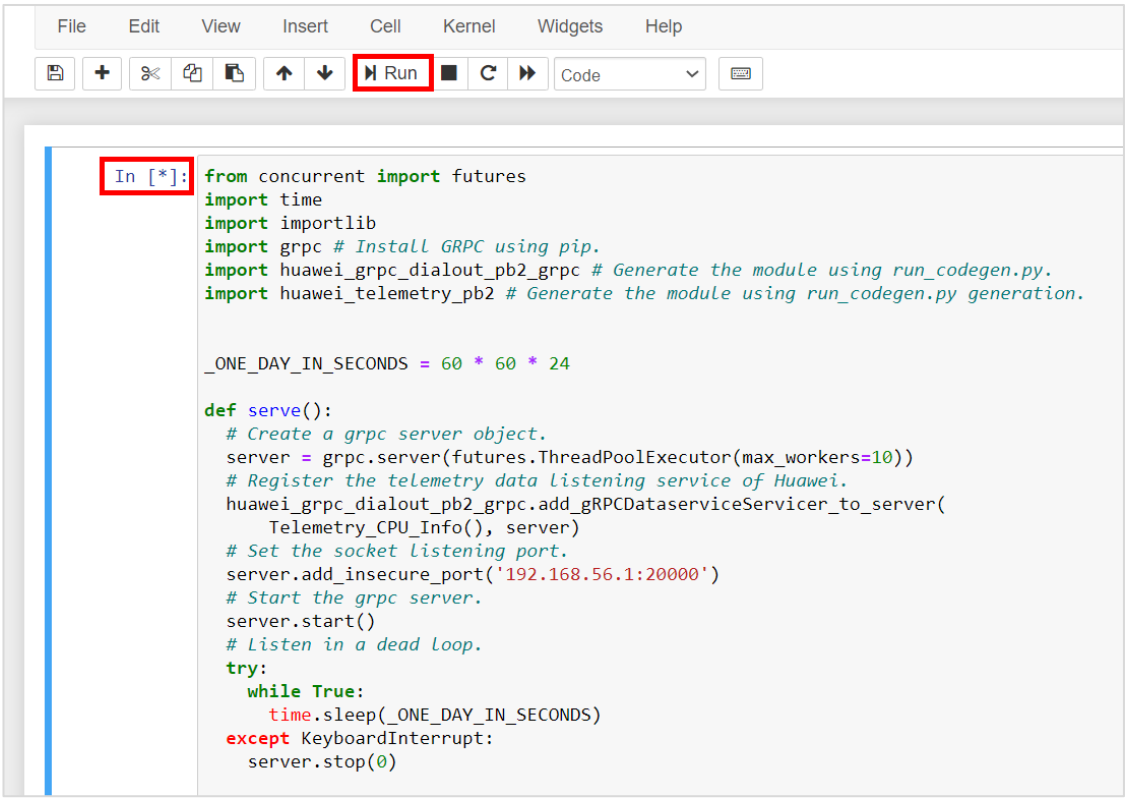
Step 1 Complete code:

```
from concurrent import futures
import time
import importlib
import grpc #installed by pip.
import huawei_grpc_dialout_pb2_grpc # Generated by run_codegen.py.
import huawei_telemetry_pb2 # Generated by run_codegen.py generation.
```

```
_ONE_DAY_IN_SECONDS = 60 * 60 * 24

def serve():
  # Create a grpc server object.
  server = grpc.server(futures.ThreadPoolExecutor(max_workers=10))
  # Register the telemetry data listening service of Huawei.
  huawei_grpc_dialout_pb2_grpc.add_gRPCDataserviceServicer_to_server(
      Telemetry_CPU_Info(), server)
  # Set the socket listening port.
  server.add_insecure_port('192.168.56.1:20000')
  # Start the grpc server.
  server.start()
  # Listen in a dead loop.
  try:
    while True:
      time.sleep(_ONE_DAY_IN_SECONDS)
  except KeyboardInterrupt:
    server.stop(0)

# Create a class that inherits the servicer method in huawei_grpc_dialout_pb2_grpc.
class Telemetry_CPU_Info(huawei_grpc_dialout_pb2_grpc.gRPCDataserviceServicer):
    def __init__(self):
        return

    def dataPublish(self, request_iterator, context):
        for i in request_iterator:
            print ('############ start ############\n')
            telemetry_data = huawei_telemetry_pb2.Telemetry.FromString(i.data)
            print (telemetry_data)

            for row_data in   telemetry_data.data_gpb.row:
                print ('------------------')
                print ('The proto path is :'+telemetry_data.proto_path)
                print ('------------------')
                module_name = telemetry_data.proto_path.split('.')[0]
                root_class   = telemetry_data.proto_path.split('.')[1]

# Dynamically load the module for obtaining telemetry data. In this example, the module is:
                decode_module = importlib.import_module( module_name+'_pb2')
                print (decode_module)
# Define the decoding method getattr to obtain the attribute values in the dynamically-loaded module and
invoke the decoding method FromString for this attribute.
                decode_func = getattr(decode_module,root_class).FromString

                print ('----------- content is -----------\n')
# Decode and output the content in row_data using this method.
                print (decode_func(row_data.content))
                print ('----------- done ------------------')


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

Step 2 Code execution on the compiler:

The server keeps listening unless interrupted.



Step 3 Output:

Data is pushed at the specified interval. This example uses one output.

```
############ start #############

node_id_str: "CE1"
subscription_id_str: "Sub1"
sensor_path: "huawei-devm:devm/cpuInfos/cpuInfo"
collection_id: 260
collection_start_time: 1580278055740
msg_timestamp: 1580278055824
data_gpb {
  row {
    timestamp: 1580278055740
    content: "*\022\n\020\"\0011\010\201\200\204\010(\006\030Z0K\020\010"
  }
  row {
    timestamp: 1580278055740
    content: "*\023\n\021\"\00217\010\201\200\304\010(\006\030Z0K\020\010"
  }
}
collection_end_time: 1580278055740
current_period: 60000
except_desc: "OK"
product_name: "CE12800"
proto_path: "huawei_devm.Devm"
```

```
-----------------
The proto path is :huawei_devm.Devm
-----------------
<module 'huawei_devm_pb2' from 'D:\\10 Python Learning\\Telemetry\\huawei_devm_pb2.py'>
---------- content is ----------

cpuInfos {
  cpuInfo {
    entIndex: 16842753
    interval: 8
    ovloadThreshold: 90
    position: "1"
    systemCpuUsage: 6
    unovloadThreshold: 75
  }
}

---------- done ----------------
-----------------
The proto path is :huawei_devm.Devm
-----------------
<module 'huawei_devm_pb2' from 'D:\\10 Python Learning\\Telemetry\\huawei_devm_pb2.py'>
---------- content is ----------

cpuInfos {
  cpuInfo {
    entIndex: 17891329
    interval: 8
    ovloadThreshold: 90
    position: "17"
    systemCpuUsage: 6
    unovloadThreshold: 75
  }
}

---------- done ----------------
```

----End

6.4 Code Explaination

The server code invokes the Python file generated by compiling the **.proto** file. The **run_codegen.py** script is simple and is not described here. An important part of writing server code is learning to invoke these generated classes and methods.

Step 1 Import the modules.

```
from concurrent import futures
import time
import importlib
import grpc # Install GRPC using pip.
import huawei_grpc_dialout_pb2_grpc # Generate code using run_codegen.py.
import huawei_telemetry_pb2 # Generate code using run_codegen.py generation
```

Import all modules used by this script. If gRPC has not been installed, make the preparations by referring to the following link:

https://grpc.io/docs/tutorials/basic/python/

concurrent.futures implements multiple processes or threads on the server.

importlib dynamically imports modules.

Step 2 Define the main function.

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

The program runs the main function first. The **serve()** function is invoked in the main function.

Step 3 Define the **serve()** function.

As the main part of the server, the **serve()** function sets threads and listening IP addresses, and start or stop services.

```
# Define variables for one day in seconds (60 x 60 x 24 seconds).
_ONE_DAY_IN_SECONDS = 60 * 60 * 24

def serve():
    # Create a gRPC server object with a maximum of 10 threads.
    server = grpc.server(futures.ThreadPoolExecutor(max_workers=10))
    # Register with the telemetry data listening service of Huawei.
    huawei_grpc_dialout_pb2_grpc.add_gRPCDataserviceServicer_to_server(
        Telemetry_CPU_Info(), server)
    # Set the socket listening port.
    server.add_insecure_port('192.168.56.1:20000')
    # Start the grpc server.
    server.start()
    # Perform listening in an infinite loop.
    try:
        while True:
            time.sleep(_ONE_DAY_IN_SECONDS)
    except KeyboardInterrupt:
        server.stop(0)
```

In the **serve()** function, futures is used to allow the server to receive a maximum of 10 threads and assign a value to the server.

```
server = grpc.server(futures.ThreadPoolExecutor(max_workers=10))
```

Then, invoke the **add_gRPCDataserviceServicer_to_server** method in **huawei_grpc_dialout_pb2_grpc**. The input **Telemetry_CPU_Info()** class will be defined later. (You can open the **huawei_grpc_dialout_pb2_grpc.py** file to view the specific function.)

```
huawei_grpc_dialout_pb2_grpc.add_gRPCDataserviceServicer_to_server(
        Telemetry_CPU_Info(), server)
```

server() is used to set the listening IP address and port. In this example, the IP address 192.168.56.1 is used for communication with the switch.

```
server.add_insecure_port('192.168.56.1:20000')
```

Start the gRPC service. Run the **while** function continuously within the time specified by **time.sleep()**.

```
server.start()
  # Perform listening in an infinite loop.
  try:
    while True:
      time.sleep(_ONE_DAY_IN_SECONDS)
  except KeyboardInterrupt:
    server.stop(0)
```

Step 4 Obtain Telemetry data.

The **serve()** function invokes the **Telemetry_CPU_Info()** class, which obtains and parses Telemetry data. First, let's see how to obtain Telemetry data.

```
# Create a class that inherits the servicer method in huawei_grpc_dialout_pb2_grpc.
class Telemetry_CPU_Info(huawei_grpc_dialout_pb2_grpc.gRPCDataserviceServicer):
    def __init__(self):
        return

    def dataPublish(self, request_iterator, context):
        for i in request_iterator:
            print ('############ start ############\n')
            telemetry_data = huawei_telemetry_pb2.Telemetry.FromString(i.data)
            print (telemetry_data)
```

Telemetry_CPU_Info () inherits the **gRPCDataserviceServicer** method in **huawei_grpc_dialout_pb2_grpc**. For details, see the **huawei_grpc_dialout_pb2_grpc.py** file.

```
class Telemetry_CPU_Info(huawei_grpc_dialout_pb2_grpc.gRPCDataserviceServicer):
```

The subclass inherits the **dataPublish(self, request_iterator, context)** function of the parent class.

Telemetry data of the device is continuously pushed. Each push is identified by the **#####start#####** field.

```
for i in request_iterator:
            print ('############ start ############\n')
```

The obtained telemetry data is expressed by the **telemetry_data** variable. Note that the **FromString** method in the **huawei_telemetry_pb2.Telemetry** class needs to be used, and only the data attribute in **request_iterator** needs to be entered.

```
telemetry_data = huawei_telemetry_pb2.Telemetry.FromString(i.data)
print (telemetry_data)
```

The output of Telemetry_data is as follows:

```
############ start #############

node_id_str: "CE1"
subscription_id_str: "Sub1"
sensor_path: "huawei-devm:devm/cpuInfos/cpuInfo"
collection_id: 260
collection_start_time: 1580278055740
msg_timestamp: 1580278055824
data_gpb {
  row {
    timestamp: 1580278055740
    content: "*\022\n\020\"\0011\010\201\200\204\010(\006\030Z0K\020\010"
  }
  row {
    timestamp: 1580278055740
    content: "*\023\n\021\"\00217\010\201\200\304\010(\006\030Z0K\020\010"
  }
}
collection_end_time: 1580278055740
current_period: 60000
except_desc: "OK"
product_name: "CE12800"
proto_path: "huawei_devm.Devm"
```

The complete Telemetry data contains parameters such as node_id, subscription ID, and time. The **data_gpb** contains multiple rows, and each row contains the **timestamp** and **content** fields. In this case, the data in the **content** field cannot be identified and needs to be further decoded.

Step 5 Decode Telemetry data.

In this example, the CPU information to be obtained is contained in the **content** field and cannot be identified. Therefore, the **content** field has to be decoded in the **.proto** file.

As each data push contains multiple rows, data is read and pushed in the **for** loop.

```
for row_data in telemetry_data.data_gpb.row:
    print ('------------------')
    print ('The proto path is :'+telemetry_data.proto_path)
    print ('------------------')
```

Check the value of **proto_path** for each data record. The command output is as follows:

```
------------------
The proto path is :huawei_devm.Devm
------------------
```

Different modules are dynamically loaded based on the value of **proto_path**. Use the **split()** method to split **huawei_devm.Devm** into **huawei_devm** and **Devm**.

```
module_name = telemetry_data.proto_path.split('.')[0]
root_class = telemetry_data.proto_path.split('.')[1]
```

```
# Dynamically load the module for obtaining telemetry data. In this example, the module is...
decode_module = importlib.import_ module( module_name+'_pb2')
print (decode_module)
```

Use **importlib.import_module** to dynamically load the module, and output the content. In this example, the module name is **huawei_devm_pb2**.

<module 'huawei_devm_pb2' from 'D:\\10 Python Learning\\Telemetry\\huawei_devm_pb2.py'>

```
# Define the decoding method: getattr to obtain the attribute value in the dynamically-loaded module and invoke the decoding method FromString of this attribute.
decode_func = getattr(decode_module,root_class).FromString
```

Define the **decode_func** method to decode **huawei_devm data**.

```
print ('----------- content is -----------\n')
# Use this method to decode the content in row_data and output the decoded content.
print (decode_func(row_data.content))
print ('----------- done ----------')
```

Finally, input the content to be decoded to this method.

The content that cannot be identified...

```
data_gpb {
  row {
    timestamp: 1580278055740
    content: "*\022\n\020\"\0011\010\201\200\204\010(\006\030Z0K\020\010"
  }
  row {
    timestamp: 1580278055740
    content: "*\023\n\021\"\00217\010\201\200\304\010(\006\030Z0K\020\010"
  }
}
```

Is decoded as follows...

```
cpuInfos {
  cpuInfo {
    entIndex: 16842753
    interval: 8
    ovloadThreshold: 90
    position: "1"
    systemCpuUsage: 6
    unovloadThreshold: 75
  }
}
and
cpuInfos {
  cpuInfo {
    entIndex: 17891329
    interval: 8
    ovloadThreshold: 90
    position: "17"
    systemCpuUsage: 6
    unovloadThreshold: 75
```

```
  }
}
```

The display corresponds to the content in two rows, respectively.

----End

6.5 Quiz

1. Why is decoding required instead of transferring complete data in **data_gpb**?
2. In this example, one function obtains and decodes data. Is there any way to optimize the function?

7 OPS Experiment

7.1 Background

A company deploys a CE12800 switch whose management IP address is 192.168.56.100. You will test the Open Programmability System (OPS) function of the device, and query and clear the startup configuration file of the device.

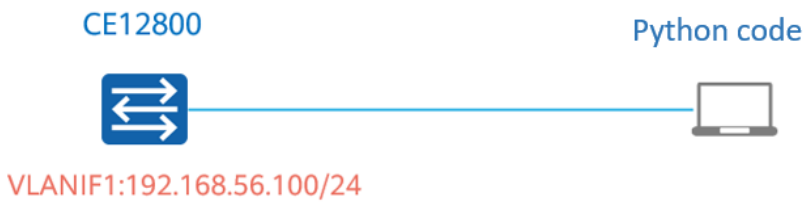
7.1.1 Objectives

- Understand the common HTTP operations.
- Master Huawei OPS development capabilities.

7.1.2 Environment Preparations

The FTP server and software has to be prepared additionally (no FTP server operation is involved in this guide.)

Simulate an environment using a simulator or prepare a physical environment. In this experiment, the local PC runs the FTP server (IP address: 192.168.56.1), and the switch functions as the FTP client (IP address of CE1: 192.168.56.100), which downloads the Python script from the FTP server.



1. Configure the CE12800 switch.

```
<HUAWEI>system-view immediately
Enter system view, return user view with return command.
[HUAWEI]sysname CE1
```

```
[CE1]interface Vlanif 1
[CE1-Vlanif1]ip add 192.168.56.100 24
[CE1-Vlanif1]quit
```

2. Verify connectivity between the switch and local PC.

   Open the **CMD** window on the local PC and test connectivity between the local PC and CE1.

```
C:\Users\XXX>ping 192.168.56.100
Pinging 192.168.56.100 with 32 bytes of data:
Reply from 192.168.56.100: Bytes = 32, time = 3 ms, TTL = 255
Reply from 192.168.56.100: Bytes = 32, time = 3 ms, TTL = 255
Reply from 192.168.56.100: Bytes = 32, time = 2 ms, TTL= 255
Reply from 192.168.56.100: Bytes = 32, time = 4 ms, TTL = 255
Ping statistics for 192.168.56.100:
    Packets: 4 packets transmitted, 4 packets received, 0 packet loss (0.0%)
Estimated RTT (ms):
    Minimum = 2 ms, maximum = 4 ms, average = 3 ms
```

3. Test FTP connectivity.

```
<CE1>ftp 192.168.56.1
Trying 192.168.56.1 ...
Press CTRL + K to abort
Connected to 192.168.56.1.
User(192.168.56.1:(none)):
```

   The FTP connectivity test is successful.

7.2 Configuration Roadmap

1. Compile the OPS script **demo.py** on the PC.
2. Upload the OPS script to the switch.
3. Run **demo.py** on the switch.

7.3 Configuration Procedure and Complete Code

7.3.1 Complete Code

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import traceback
import httplib
import string


# Define a class for invoking RESTful API. The class contains some methods to perform operations during HTTP connection establishment. This part can be used directly without any modification.
# Invoke this part without any modification.
class OPSConnection(object):
```

```
    """Make an OPS connection instance."""

    # Initialize a class and create an HTTP connection.
    def __init__(self, host, port=80):
        self.host = host
        self.port = port
        self.headers = {
            "Content-type": "text/xml",
            "Accept": "text/xml"
        }
        self.conn = None

    # Close the HTTP connection.
    def close(self):
        """Close the connection"""
        self.conn.close()

    # Create operations on device resources.
    def create(self, uri, req_data):
        """Create operation"""
        ret = self.rest_call("POST", uri, req_data)
        return ret

    # Delete operations on device resources.
    def delete(self, uri, req_data):
        """Delete operation"""
        ret = self.rest_call("DELETE", uri, req_data)
        return ret

    # Get operations on device resources.
    def get(self, uri, req_data=None):
        """Get operation"""
        ret = self.rest_call("GET", uri, req_data)
        return ret

    # Modify operations on device resources.
    def set(self, uri, req_data):
        """Set operation"""
        ret = self.rest_call("PUT", uri, req_data)
        return ret

    # Define methods invoked in a class.
    def rest_call(self, method, uri, req_data):
        """REST call"""
        print('|---------------------------------- request: ----------------------------------|')
        print('%s %s HTTP/1.1\n' % (method, uri))
        if req_data == None:
            body = ""
        else:
            body = req_data
            print(body)
        if self.conn:
            self.conn.close()
        self.conn = httplib.HTTPConnection(self.host, self.port)
```

```
            self.conn.request(method, uri, body, self.headers)
            response = self.conn.getresponse()
            response.status = httplib.OK    # stub code
            ret = (response.status, response.reason, response.read())
            print('|---------------------------------- response: ----------------------------------|')
            print('HTTP/1.1 %s %s\n\n%s' % ret)
            print('|-------------------------------------------------------------------------------|')
            return ret

def clear_startup_info(ops_conn):
    # Specify URI of the system startup information. URI is the managed object defined in RESTful API.
Different managed objects have different URIs.
    # Modify URI as required. For details about the URIs supported by the device, see RESTful API.
    uri = "/cfg/clearStartup"

    # Specify the request content to be sent. The content of this part corresponds to URIs and different URIs
identify different request content.
    # Modify the request content based on used URIs. For details on the format of the request content, see
RESTful API.
    req_data = \
        '''<?xml version="1.0" encoding="UTF-8"?>
        <clearStartup>
        </clearStartup>
        '''

    # Execute a POST operation request. uri and req_data indicate a URI request and a content request,
respectively. ret indicates whether the request is successful. rsp_data indicates the response data of the
system after the request is executed. For details about the response data format, see RESTful API.

    # Change request type get() as required, for example, to set() or create().
    ret, _, rsp_data = ops_conn.create(uri, req_data)
    if ret != httplib.OK:
        return None

# Define a function to obtain system startup information.
def get_startup_info(ops_conn):
    # Specify URI of the system startup information. The URI is the managed object defined in RESTful API,
and different managed objects have different URIs.
    # Modify URI as required. For details about the URIs supported by the device, see RESTful API.
    uri = "/cfg/startupInfos/startupInfo"

    # Specify the request content to be sent. The content of this part corresponds to URIs and different URIs
identify different request content.
    # Modify the request content based on used URIs. For details on the format of the request content, see
RESTful API.
    req_data = \
        '''<?xml version="1.0" encoding="UTF-8"?>
        <startupInfo>
        </startupInfo>
        '''

    # Execute a GET operation request. uri and req_data indicate a URI request and a content request,
respectively. ret indicates whether the request is successful. rsp_data indicates the response data of the
system after the request is executed. For details about the response data format, see RESTful API.
```

```
    # Change request type get() as required, for example, to set() or create().
    ret, _, rsp_data = ops_conn.get(uri, req_data)
    if ret != httplib.OK:
        return None

    return rsp_data

    # The main() function defines the operations to be performed by this script. You can modify this
function according to service requirements.


def main():
    """The main function."""

    # host indicates the loop address. Currently, RESTful API is invoked inside the device, whose value is
localhost.
    host = "localhost"
    try:
        # Establish an HTTP connection.
        ops_conn = OPSConnection(host)
        # Invoke the function to obtain system startup information.
        rsp_data = get_startup_info(ops_conn)
        rsp_data = clear_startup_info(ops_conn)
        rsp_data = get_startup_info(ops_conn)
        # Close the HTTP connection.
        ops_conn.close()
        return

    except:
        errinfo = traceback.format_exc()
        print(errinfo)
        return


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

7.3.2 Uploading Code

The preceding code is saved as **demo.py** on the FTP server.

The switch functions as an FTP client to download the file.

```
<CE1>ftp 192.168.56.1
Trying 192.168.56.1 ...
Press CTRL + K to abort
Connected to 192.168.56.1.
220
User(192.168.56.1:(none)):admin
Enter password:
230
[ftp]get demo.py
Warning: The file may not transfer correctly in ASCII mode.
213 5554
200 PORT                    .
```

```
150 starting
/       0% [                ]
\       100% [**********]
226 Transfer complete.
```

The file is downloaded successfully.

7.3.3 Execution Result

Multiple methods are available to configure OPS on a switch. In this example, a Python script is manually installed and run for this purpose.

<CE1>ops install file demo.py

<CE1>ops run python demo.py

```
|---------------------------------- request: ----------------------------------|
GET /cfg/startupInfos/startupInfo HTTP/1.1

<?xml version="1.0" encoding="UTF-8"?>
          <startupInfo>
          </startupInfo>

|---------------------------------- response: ----------------------------------|
HTTP/1.1 200 OK

<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply>
  <data>
    <cfg xmlns="http://www.huawei.com/netconf/vrp" format-version="1.0" content-
version="1.0">
      <startupInfos>
        <startupInfo>
          <position>17</position>
          <nextStartupFile>cfcard:/vrpcfg.cfg</nextStartupFile>
          <configedSysSoft>cfcard:/VRPV200R005C10SPC607B607D0306_ce12800.cc</con
figedSysSoft>
          <curSysSoft>cfcard:/VRPV200R005C10SPC607B607D0306_ce12800.cc</curSysSo
ft>
          <nextSysSoft>cfcard:/VRPV200R005C10SPC607B607D0306_ce12800.cc</nextSys
Soft>
          <curStartupFile>cfcard:/vrpcfg.cfg</curStartupFile>
          <curPatchFile>NULL</curPatchFile>
          <nextPatchFile>NULL</nextPatchFile>
          <boardInfo>101</boardInfo>
          <curPafFile>default</curPafFile>
          <nextPafFile>default</nextPafFile>
        </startupInfo>
      </startupInfos>
    </cfg>
  </data>
</rpc-reply>

|-----------------------------------------------------------------------------|
|---------------------------------- request: ----------------------------------|
```

```
POST /cfg/clearStartup HTTP/1.1

<?xml version="1.0" encoding="UTF-8"?>
          <clearStartup>
          </clearStartup>

|---------------------------------- response: ----------------------------------|
HTTP/1.1 200 OK

<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply>
  <ok/>
</rpc-reply>

|-------------------------------------------------------------------------------|
|---------------------------------- request: ----------------------------------|
GET /cfg/startupInfos/startupInfo HTTP/1.1

<?xml version="1.0" encoding="UTF-8"?>
          <startupInfo>
          </startupInfo>

|---------------------------------- response: ----------------------------------|
HTTP/1.1 200 OK

<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply>
  <data>
    <cfg xmlns="http://www.huawei.com/netconf/vrp" format-version="1.0" content-
version="1.0">
      <startupInfos>
        <startupInfo>
          <position>17</position>
          <nextStartupFile>NULL</nextStartupFile>
          <configedSysSoft>cfcard:/VRPV200R005C10SPC607B607D0306_ce12800.cc</con
figedSysSoft>
          <curSysSoft>cfcard:/VRPV200R005C10SPC607B607D0306_ce12800.cc</curSysSo
ft>
          <nextSysSoft>cfcard:/VRPV200R005C10SPC607B607D0306_ce12800.cc</nextSys
Soft>
          <curStartupFile>NULL</curStartupFile>
          <curPatchFile>NULL</curPatchFile>
          <nextPatchFile>NULL</nextPatchFile>
          <boardInfo>101</boardInfo>
          <curPafFile>default</curPafFile>
          <nextPafFile>default</nextPafFile>
        </startupInfo>
      </startupInfos>
    </cfg>
  </data>
</rpc-reply>

|-------------------------------------------------------------------------------|
```

7.4 Code Explaination

Step 1 Import the modules.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import traceback
import httplib
import string
```

Import the modules required by the OPS function. As the code runs on the switch, no module need to be installed on the local PC

Step 2 Parse the **main** function.

Parse the **main** function according to the code execution sequence.

```
def main():
    """The main function."""

    # host indicates the loop address. Currently, RESTful API is invoked inside the device, whose value is
localhost.
    host = "localhost"
    try:
        # Establish an HTTP connection.
        ops_conn = OPSConnection(host)
        # Invoke the function to obtain system startup information.
        rsp_data = get_startup_info(ops_conn)
        rsp_data = clear_startup_info(ops_conn)
        rsp_data = get_startup_info(ops_conn)
        # Close the HTTP connection.
        ops_conn.close()
        return

    except:
        errinfo = traceback.format_exc()
        print(errinfo)
        return

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

The **main** function defines the local host. Run this script on the switch to invoke the local RESTful interface.

Run the **try... except...** command to capture exceptions. First, set up an HTTP connection by invoking the **OPSConnection(host)** class. Then invoke the **get_startup_info(ops_conn)** function to obtain the configuration information of the current device, the **clear_startup_info(ops_conn)** function to clear the next startup configuration of the device, and the **get_startup_info(ops_conn)** function to obtain the configuration information of the current device again for verification.

Invoke **ops_conn.close()** to close the HTTP connection.

Step 3 Parse the OPSConnection() class

```
# Define a class to invoke RESTful API. The class contains some methods to perform operations during HTTP connection establishment. This part can be used directly without any modification.
# Invoke this part without any modification.
class OPSConnection(object):
    """Make an OPS connection instance."""

    # Initiate a class to create an HTTP connection.
    def __init__(self, host, port=80):
        self.host = host
        self.port = port
        self.headers = {
            "Content-type": "text/xml",
            "Accept": "text/xml"
        }
        self.conn = None

    # Close the HTTP connection.
    def close(self):
        """Close the connection"""
        self.conn.close()

    # Create operations on device resources.
    def create(self, uri, req_data):
        """Create operation"""
        ret = self.rest_call("POST", uri, req_data)
        return ret

    # Delete operations on device resources.
    def delete(self, uri, req_data):
        """Delete operation"""
        ret = self.rest_call("DELETE", uri, req_data)
        return ret

    # Get operations on device resources.
    def get(self, uri, req_data=None):
        """Get operation"""
        ret = self.rest_call("GET", uri, req_data)
        return ret

    # Modify operations on device resources.
    def set(self, uri, req_data):
        """Set operation"""
        ret = self.rest_call("PUT", uri, req_data)
        return ret

    # Define methods invoked in a class.
    def rest_call(self, method, uri, req_data):
        """REST call"""
        print('|---------------------------------- request: ----------------------------------|')
        print('%s %s HTTP/1.1\n' % (method, uri))
        if req_data == None:
```

```
            body = ""
        else:
            body = req_data
            print(body)
        if self.conn:
            self.conn.close()
        self.conn = httplib.HTTPConnection(self.host, self.port)

        self.conn.request(method, uri, body, self.headers)
        response = self.conn.getresponse()
        response.status = httplib.OK    # stub code
        ret = (response.status, response.reason, response.read())
        print('|---------------------------------- response: ----------------------------------|')
        print('HTTP/1.1 %s %s\n\n%s' % ret)
        print('|-------------------------------------------------------------------------------|')
        return ret
```

OPSConnection() is the class for the device to locally invoke RESTful APIs. This class defines some methods to perform operations (including GET, POST, DELETE, and PUT) when an HTTP connection is set up. This part can be used directly without any modification.

Step 4 Parse the **get_startup_info()** function

This function is used to obtain the current startup information of the device.

```
def get_startup_info(ops_conn):
    # Specify URI of the system startup information. The URI is the managed object defined in the RESTful API, and different managed objects have different URIs.
    # Modify URI as required. For details about the URIs supported by the device, see RESTful API.
    uri = "/cfg/startupInfos/startupInfo"
```

The URI varies depending on the operation. For details, see RESTful API. In this example, the content is as follows:

Configuration Management

System Startup Information

Operation	URI	Description
GET	/cfg/startupInfos/startupInfo	Obtain system startup information.

- Request example

```
<?xml version="1.0" encoding="UTF-8"?>
<startupInfo>
</startupInfo>
```

The GET operation is performed this time. Therefore, the URI is set to **/cfg/startupInfos/startupInfo**.

```
    # Specify the request content to be sent. The content of this part corresponds to URIs and different URIs identify different request content.
    # Modify the request content based on used URIs. For details on the format of the request content, see RESTful API.
```

```
    req_data = \
        '''<?xml version="1.0" encoding="UTF-8"?>
        <startupInfo>
        </startupInfo>
        '''
```

Define **req_data** as the request to be sent. Fill in the request based on the document requirements.

```
    # Execute a GET operation request. uri and req_data indicate a URI request and a content request, respectively. ret indicates whether the request is successful. rsp_data indicates the response data of the system after the request is executed. For details about the response data format, see RESTful API.

    # Change request type get() as required, for example, to set() or create().
    ret, _, rsp_data = ops_conn.get(uri, req_data)
    if ret != httplib.OK:
        return None

    return rsp_data
```

After defining the URI and request, invoke **ops_conn.get()** to perform the GET operation.

When the main function invokes the **get_startup_info()** function in step 4 for the first time, the following output information is displayed:

```
|---------------------------------- request: ----------------------------------|
GET /cfg/startupInfos/startupInfo HTTP/1.1

<?xml version="1.0" encoding="UTF-8"?>
        <startupInfo>
        </startupInfo>

|---------------------------------- response: ----------------------------------|
HTTP/1.1 200 OK

<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply>
  <data>
    <cfg xmlns="http://www.huawei.com/netconf/vrp" format-version="1.0" content-
version="1.0">
      <startupInfos>
        <startupInfo>
          <position>17</position>
          <nextStartupFile>cfcard:/vrpcfg.cfg</nextStartupFile>
          <configedSysSoft>cfcard:/VRPV200R005C10SPC607B607D0306_ce12800.cc</con
figedSysSoft>
          <curSysSoft>cfcard:/VRPV200R005C10SPC607B607D0306_ce12800.cc</curSysSo
ft>
          <nextSysSoft>cfcard:/VRPV200R005C10SPC607B607D0306_ce12800.cc</nextSys
Soft>
          <curStartupFile>cfcard:/vrpcfg.cfg</curStartupFile>
          <curPatchFile>NULL</curPatchFile>
          <nextPatchFile>NULL</nextPatchFile>
          <boardInfo>101</boardInfo>
```

```
            <curPafFile>default</curPafFile>
            <nextPafFile>default</nextPafFile>
          </startupInfo>
        </startupInfos>
      </cfg>
    </data>
</rpc-reply>
|-----------------------------------------------------------------------------|
```

The current configuration file and the configuration file for the next startup are **cfcard:/vrpcfg.cfg**.

Step 5 Parse the **clear_startup_info()** function.

This function is used to clear the current configuration file of the device.

```
def clear_startup_info(ops_conn):
    # Specify URI of the system startup information. The URI is the managed object defined in the RESTful API, and different managed objects have different URIs.
    # Modify the URI as required. For details about the URIs supported by the device, see RESTful API.
    uri = "/cfg/clearStartup"
```

The operation is to clear the configuration file for the next startup. Therefore, the corresponding URI is **/cfg/clearStartup**.

Configuration File for Next Startup

Operation	URI	Description
POST	/cfg/setStartup	Specify the configuration file for next startup.
POST	/cfg/deleteStartup	Delete the configuration file for next startup.
POST	/cfg/clearStartup	Cancels the configuration for the next file startup. After this parameter is specified, the system clears the current and next file startup configurations.

```
    # Specify the request content to be sent. The content of this part corresponds to URIs and different URIs identify different request content.
    # Modify the request content based on used URIs. For details on the format of the request content, see RESTful API.
    req_data = \
        '''<?xml version="1.0" encoding="UTF-8"?>
        <clearStartup>
        </clearStartup>
        '''
```

Continue to check the document to obtain the format of the request for clearing the configuration file.

4. Clears the next file startup configuration.

- Request example

```
<?xml version="1.0" encoding="UTF-8"?>
<clearStartup>
</clearStartup>
```

```
    # Execute a POST operation request. uri and req_data indicate a URI request and a content request, respectively. ret indicates whether the request is successful. rsp_data indicates the response data of the system after the request is executed. For details about the response data format, see RESTful API.

    # Change request type get() as required, for example, to set() or create().
    ret, _, rsp_data = ops_conn.create(uri, req_data)
    if ret != httplib.OK:
        return None
```

According to the document, the HTTP operation for clearing the configuration is POST. Check the method in **OPSConnection()**, which is **ops_conn.create**.

After the main function executes the **clear_startup_info()** function, the following information is displayed:

```
|---------------------------------- request: ----------------------------------|
POST /cfg/clearStartup HTTP/1.1

<?xml version="1.0" encoding="UTF-8"?>
          <clearStartup>
          </clearStartup>

|---------------------------------- response: ----------------------------------|
HTTP/1.1 200 OK

<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply>
  <ok/>
</rpc-reply>

|-----------------------------------------------------------------------------|
```

The HTTP request header is the POST operation, which clears the configuration. The HTTP response is successful.

Run the **get_startup_info()** function again to view the modifications.

```
|---------------------------------- request: ----------------------------------|
GET /cfg/startupInfos/startupInfo HTTP/1.1

<?xml version="1.0" encoding="UTF-8"?>
          <startupInfo>
          </startupInfo>

|---------------------------------- response: ----------------------------------|
HTTP/1.1 200 OK

<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply>
  <data>
    <cfg xmlns="http://www.huawei.com/netconf/vrp" format-version="1.0" content-version="1.0">
      <startupInfos>
        <startupInfo>
          <position>17</position>
          <nextStartupFile>NULL</nextStartupFile>
```

```
            <configedSysSoft>cfcard:/VRPV200R005C10SPC607B607D0306_ce12800.cc</con
figedSysSoft>
            <curSysSoft>cfcard:/VRPV200R005C10SPC607B607D0306_ce12800.cc</curSysSo
ft>
            <nextSysSoft>cfcard:/VRPV200R005C10SPC607B607D0306_ce12800.cc</nextSys
Soft>
            <curStartupFile>NULL</curStartupFile>
            <curPatchFile>NULL</curPatchFile>
            <nextPatchFile>NULL</nextPatchFile>
            <boardInfo>101</boardInfo>
            <curPafFile>default</curPafFile>
            <nextPafFile>default</nextPafFile>
          </startupInfo>
        </startupInfos>
      </cfg>
    </data>
</rpc-reply>

|-----------------------------------------------------------------------------|
```

The current configuration file and the configuration file for the next startup are displayed as NULL, indicating that the modification is successful.

----End

7.5 Quiz

Switches provide the Zero Touch Provisioning (ZTP) function. ZTP enables a newly delivered or un-configured device to automatically load version files (including the system software, configuration file, license file, patch file, and user-defined file) during startup after the device is powered on. How to use OPS to implement ZTP?

8 Network Flow Analysis Experiment

8.1 Background

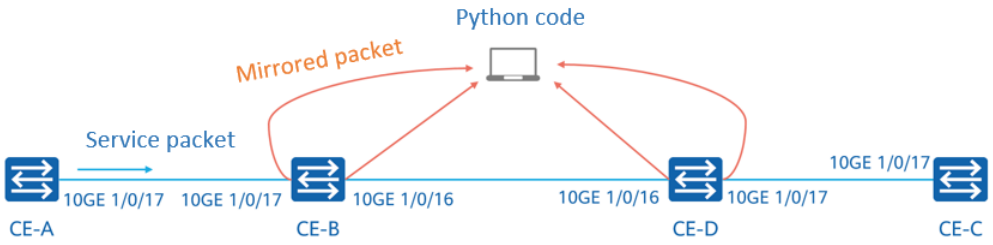
In network management requirements, the forwarding path of specific flow on the network needs to be identified. For this purpose, the network administrator will enable remote mirroring on the interfaces of network devices, compile a Python script, and use the **Scapy** module to capture and analyze mirrored packets, finally to identify the forwarding path.

8.1.1 Objectives

- Configure remote mirroring on network devices.
- Master the basic usage of the **Scapy** module.

8.1.2 Environment Preparations

Use the CE12800 to set up the experiment environment. As shown in the following figure, service packets are sent from CE-A to CE-C. Enable remote mirroring on 10GE 1/0/16 and 10GE 1/0/17 of CE-B and CE-D so that mirrored packets are sent to the flow collector (Python code).



1. Install the remote mirroring plug-in.

   The CE12800 of V200R001C00 and later versions provides remote mirroring as a plug-in, instead of a part of system software. For details on how to

install a plug-in, see the *CloudEngineXXX Plug-in Operation Guide - Remote Mirroring*. After the plug-in is installed, the remote mirroring function is available on the CE12800.

2. Configure remote mirroring on CE-B.

```
<CE-B>system-view immediately
Enter system view, return user view with return command.
[CE-B]observe-port 2 destination-ip 10.166.231.109 source-ip 10.154.186.102 erspan-id 2
[CE-B]interface 10GE 1/0/17
[CE-B-10GE1/0/17]port-mirroring observe-port 2 both
[CE-B-10GE1/0/17]interface 10GE 1/0/16
[CE-B-10GE1/0/16]port-mirroring observe-port 2 both
[CE-B-10GE1/0/16]quit
```

In this example, the IP address of CE-B is 10.154.186.102, and the IP address of the flow collector (Python code) is 10.166.231.109.

3. Configure remote mirroring on CE-D.

```
<CE-D>system-view immediately
Enter system view, return user view with return command.
[CE-D]observe-port 3 destination-ip 10.166.231.109 source-ip 10.154.186.104 erspan-id 3
[CE-D]interface 10GE 1/0/17
[CE-D-10GE1/0/17]port-mirror observe-port 3 both
[CE-D-10GE1/0/17]interface 10GE 1/0/16
[CE-D-10GE1/0/16]port-mirror observe-port 3 both
[CE-D-10GE1/0/16]quit
```

In this example, the IP address of CE-D is 10.154.186.104, and the IP address of the flow collector (Python code) is 10.166.231.109.

8.2 Configuration Roadmap

1. Compile a Python script on the PC, use **Scapy** to capture packets, and analyze the mirrored packets to identify the flow forwarding path.
2. On CE-A, test ping and telnet connections to CE-C to simulate service packets.
3. Run the script on the PC when CE-A is testing ping or telnet connections to CE-C.

8.3 Configuration Procedure and Complete Code

8.3.1 Complete Code

```
# -*- coding: utf-8 -*-

from scapy.all import *
from scapy.contrib.erspan import *
import re, paramiko

# Create a dictionary to save networking information about CE-B and CE-D.
```

```
interfaceConnect_CE_B = {
    "10GE1/0/17": {"CE-A": "10GE1/0/17"},
    "10GE1/0/14": {"CE-C": "10GE1/0/14"},
    "10GE1/0/16": {"CE-D": "10GE1/0/16"},
}

interfaceConnect_CE_D = {
    "10GE1/0/16": {"CE-B": "10GE1/0/16"},
    "10GE1/0/17": {"CE-C": "10GE1/0/17"},
}

# Create the NetDevice class to abstract a device and get its information.
class NetDevice:

    def __init__(self, deviceName, deviceIp, username, password, interfaceConnect):
        self.deviceName = deviceName
        self.deviceIp = deviceIp
        self.username = username
        self.password = password
        self.interfaces = []
        self.interfacesConnect = {}
        self.macUnderInterface = {}
        self.interfaceMac = {}
        try:
            ssh = paramiko.client.SSHClient()
            ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
            ssh.connect(hostname=deviceIp, username=username, password=password)
            cli = ssh.invoke_shell()
            cli.send('N\n')
            time.sleep(0.5)
            # Disable split-screen display temporarily so that the command output can be displayed on one
screen.
            cli.send('screen-length 0 temporary\n')
            time.sleep(1)
        except:
            print("An exception occurred during SSH login to device {}.".format(deviceIp))
            raise

        self.get_interfaces_info(cli)
        self.get_mac_under_interface(cli)
        self.interfacesConnect = interfaceConnect
        ssh.close()

    # ********************************************************
    # Function: to obtain all interfaces and their MAC addresses of the device and assign values to
self.interfaces
    # and self.interfaceMac.
    # Parameters:
    # cli - ssh shell object
    # ********************************************************
    def get_interfaces_info(self, cli):
        cli.send('display interface\n')
        time.sleep(15)
        interfaces_info = cli.recv(999999).decode()
        results = re.findall(r'^(\S+) current state.*\(ifindex', interfaces_info, re.M)
```

```
            if len(results) > 0:
                self.interfaces = results
            else:
                print("No interface is obtained.")

            interface_mac = ["", ""]
            lines = interfaces_info.splitlines()
            for i in lines:
                result1 = re.search(r'^(\S+) current state.*\(ifindex', i)
                if result1:
                    interface_mac[0] = result1.group(1)

                result2 = re.search(r'Hardware address is (\S{4}-\S{4}-\S{4})', i)
                if result2:
                    interface_mac[1] = result2.group(1)
                    self.interfaceMac[interface_mac[0]] = interface_mac[1]

    # ************************************************************
    # Function: to obtain the MAC address learned from the device interface and assign the value to
self.macUnderInterface.
    # Parameters:
    # cli - ssh shell object
    # ************************************************************
    def get_mac_under_interface(self, cli):
        cli.send('display mac-address\n')
        time.sleep(3)
        mac_address_info = cli.recv(999999).decode()
        cli.send('display arp\n')
        time.sleep(3)
        arp_info = cli.recv(999999).decode()

        mac_interface = re.findall(r'^(\S{4}-\S{4}-\S{4}) \S+ +(\S+) ', mac_address_info, re.M)
        if len(mac_interface) > 0:
            for i in mac_interface:
                if i[1] not in self.macUnderInterface.keys():
                    self.macUnderInterface[i[1]] = []
                    self.macUnderInterface[i[1]].append(i[0])
                else:
                    if i[0] not in self.macUnderInterface[i[1]]:
                        self.macUnderInterface[i[1]].append(i[0])
        else:
            print("No entry in the MAC address table.")

        arp_interface = re.findall(r' (\S{4}-\S{4}-\S{4}) .*D.* +(\S+)', arp_info, re.M)
        if len(arp_interface) > 0:
            for i in arp_interface:
                if i[1] not in self.macUnderInterface.keys():
                    self.macUnderInterface[i[1]] = []
                    self.macUnderInterface[i[1]].append(i[0])
                else:
                    if i[0] not in self.macUnderInterface[i[1]]:
                        self.macUnderInterface[i[1]].append(i[0])
        else:
            print("No dynamic entries in ARP entries.")
```

```
    # ******************************************************
    # Function: to determine whether packets are sent to the device.
    # Parameters:
    # packet_info - packet information
    # Return value:
    #     True/False
    # ******************************************************
    def is_to_device(self, packet_info):
        for key in self.interfaceMac.keys():
            # The MAC address may be in the 08e8-4f6e-0516 or 08:e8:4f:6e:05:16 format, which has to be converted for a comparison.
            if re.sub(r"[-:]", "", packet_info.get("macDst")) == re.sub(r"[-:]", "", self.interfaceMac.get(key)):
                return True
        return False

    # ******************************************************
    # Function: to determine whether a packet is sent by the local device.
    # Parameters:
    # packet_info - packet information
    # Returned value:
    #     True/False
    # ******************************************************
    def is_from_device(self, packet_info):
        for key in self.interfaceMac.keys():
            # The MAC address may be in the 08e8-4f6e-0516 or 08:e8:4f:6e:05:16 format, which has to be converted for a comparison.
            if re.sub(r"[-:]", "", packet_info.get("macSrc")) == re.sub(r"[-:]", "", self.interfaceMac.get(key)):
                return True
        return False

    # ******************************************************
    # Function: to obtain the inbound interface of the packet and the interface on the peer device, and return the path section.
    # Parameters:
    # packet_info - packet information
    # Returned value:
    #     packet_path
    #     [{"CE1": "interface1"},
    #      {"CE2": "interface2"}]
    # Interface 1 on CE1 sends packets to interface 2 on CE2.
    # ******************************************************
    def in_device_path(self, packet_info):
        # The peer device sends packets along the following path:
        packet_path_out = {}
        # The local device receives packets along the following path:
        packet_path_in = {}
        interface = ""
        for key in self.macUnderInterface.keys():
            macs = self.macUnderInterface.get(key)
            for i in range(len(macs)):
                macs[i] = re.sub(r"[-:]", "", macs[i])

            if re.sub(r"[-:]", "", packet_info.get("macSrc")) in macs:
                interface = key
                break
```

```
            if interface == "":
                raise Exception('No inbound interface on the device is found for the packets. Packet information: \n{}'.format(packet_info))

            if interface not in self.interfacesConnect:
                raise Exception('The interface on device {} does not have information about the peer device and interface'.format(interface))

            for key, value in self.interfacesConnect.get(interface).items():
                packet_path_out[key] = value

            packet_path_in[self.deviceName] = interface
            return [packet_path_out, packet_path_in]

    # ********************************************************
    #Function: to obtain the outbound interface of the packet and the interface on the peer device, and return the path section.
    # Parameters:
    # packet_info - packet information
    # Returned value:
    #     packet_path
    #     [{"CE1": "interface1"},
    #      {"CE2": "interface2"}]
    # Interface 1 on CE1 sends packets to interface 2 on CE2.
    # ********************************************************
    def out_device_path(self, packet_info):
        # The local device sends packets along the following path:
        packet_path_out = {}
        # The peer device receives packets along the following path:
        packet_path_in = {}
        interface = ""
        for key in self.macUnderInterface.keys():
            macs = self.macUnderInterface.get(key)
            for i in range(len(macs)):
                macs[i] = re.sub(r"[-:]", "", macs[i])

            if re.sub(r"[-:]", "", packet_info.get("macDst")) in macs:
                interface = key
                break

        if interface == "":
            raise Exception('No outbound interface on the device is found for the packets. Packet information: \n{}'.format(packet_info))

        if interface not in self.interfacesConnect:
            raise Exception('The interface on device {} does not have information about the peer device and interface'.format(interface))

        for key, value in self.interfacesConnect.get(interface).items():
            packet_path_in[key] = value

        packet_path_out[self.deviceName] = interface
        return [packet_path_out, packet_path_in]
```

```
# ******************************************************
#Function: to obtain the mirrored packets related to the flow for path calculation.
# Parameters:
# flow_info: flow information for path calculation
# mirror_packets - all mirrored packets
# Returned value:
#     flow_packets - mirrored packets of the flow for path calculation
# ******************************************************
def get_flow_packets(flow_info, mirror_packets):
    flow_packets = []
    for packet in mirror_packets:
        if packet["ipSrc"] != flow_info["ipSrc"]:
            continue
        elif packet["ipDst"] != flow_info["ipDst"]:
            continue
        elif packet["ipProto"] != flow_info["ipProto"]:
            continue

        if flow_info["ipProto"] == "icmp":
            flow_packets.append(packet)
            continue
        elif flow_info["ipProto"] == "tcp" or flow_info["ipProto"] == "udp":
            if flow_info["portSrc"] != "any" and packet["portSrc"] != flow_info["portSrc"]:
                continue
            elif flow_info["portDst"] != "any" and packet["portDst"] != flow_info["portDst"]:
                continue
            else:
                flow_packets.append(packet)
    return flow_packets

# ******************************************************
#Function: to determine whether two path sections are equal.
# Parameters:
# path1 - path section 1
# path2 - path section 2
# Returned value:
#     True/False
# ******************************************************
def is_equal_path(path1, path2):
    for i in range(len(path1)):
        if path1[i] != path2[i]:
            return False
    return True

# ******************************************************
# Function: to integrate all path sections to form a complete packet path.
# Parameters:
# path_section: the list of path sections of each remote mirroring device that packets pass through
# Returned value:
# path - packet path
# ******************************************************
def integrate_path(path_section):
    # Delete duplicate path sections from path_section.
    path_section_norepetition = []
    path_section_norepetition.append(path_section[0])
```

```
    for k in range(len(path_section)):
        flag = False
        for h in path_section_norepetition:
            if is_equal_path(h, path_section[k]):
                flag = True
                break
        if not flag:
            path_section_norepetition.append(path_section[k])
    path = []
    path.append(path_section_norepetition[0][0])
    path.append(path_section_norepetition[0][1])
    del path_section_norepetition[0]
    while len(path_section_norepetition) > 0:
        last_device = list(path[len(path)-1].keys())[0]
        for i in range(len(path_section_norepetition)):
            if last_device in path_section_norepetition[i][0]:
                path.append(path_section_norepetition[i][0])
                path.append(path_section_norepetition[i][1])
                del path_section_norepetition[i]
                break

        if len(path_section_norepetition) == 0:
            break

        first_device = list(path[0].keys())[0]
        for j in range(len(path_section_norepetition)):
            if first_device in path_section_norepetition[j][1]:
                path.insert(0, path_section_norepetition[j][1])
                path.insert(0, path_section_norepetition[j][0])
                del path_section_norepetition[j]
                break
    return path

# ********************************************************
#Function: to print the packet path.
# Parameters:
# path: the list with complete packet path
# Returned value:
#     None
# ********************************************************
def print_path(path):
    path_str = ""
    i = 0
    while i < len(path):
        if i == 0:
            for key, value in path[i].items():
                path_str += key + "(" + value + ")"
        elif i == len(path) - 1:
            for key, value in path[i].items():
                path_str += " ---> (" + value + ")" + key
        else:
            for key, value in path[i].items():
                path_str += " ---> (" + value + ")" + key
            if list(path[i].keys())[0] in path[i+1]:
                for key, value in path[i+1].items():
```

```
                        path_str += "(" + value + ")"
                    i += 1
            i += 1
    print(path_str)

# ******************************************************
# Function: to calculate and display the packet path.
# Parameters:
# flow_info: information about the flow for path calculation
# mirror_packets - all mirrored packets
# devices: all devices with remote mirroring enabled
# Returned value:
#     None
# ******************************************************
def path_calc(flow_info, mirror_packets, devices):
    # Save each path section.
    path_section = []
    flow_packets = get_flow_packets(flow_info, mirror_packets)
    for packet in flow_packets:
        for device in devices:
            if device.is_to_device(packet):
                path = device.in_device_path(packet)
                path_section.append(path)
            elif device.is_from_device(packet):
                path = device.out_device_path(packet)
                path_section.append(path)
    path = integrate_path(path_section)
    print_path(path)

# ******************************************************
#Function: to prompt users to enter the 5-tuple information of the flow for path calculation.
# Parameters:
#     None
# Returned value:
# flow_info: flow 5-tuple information in the following format:
#     flow_info = {
#         "ipSrc": "10.1.12.2",
#         "ipDst": "192.168.2.2",
#         "ipProto": "tcp",
#         "portSrc": 23,
#         "portDst": "any",
#     }
# ******************************************************
def input_flow_info():
    flow_info = {} # Save the 5-tuple information of flow for path calculation.
    input_str = input("Enter the 5-tuple information (source IP address, destination IP address, protocol,
source port number, and destination port number) of the flow for path calculation.")
    input_list = input_str.split(",")
    flow_info["ipSrc"] = input_list[0].strip()
    flow_info["ipDst"] = input_list[1].strip()
    flow_info["ipProto"] = input_list[2].strip()
    flow_info["portSrc"] = ""
    flow_info["portDst"] = ""
    if flow_info["ipProto"] == "tcp" or flow_info["ipProto"] == "udp":
        portSrc = input_list[3].strip()
```

```
        if portSrc != "any":
            flow_info["portSrc"] = int(portSrc)
        else:
            flow_info["portSrc"] = "any"

        portDst = input_list[4].strip()
        if portDst != "any":
            flow_info["portDst"] = int(portDst)
        else:
            flow_info["portDst"] = "any"
    return flow_info

# ******************************************************
# Function: to use the scapy module to capture packets and extract information about mirrored service packets.
# Parameters:
#     None
# Returned value:
# packInfoList - list, which saves information about mirrored service packets.
# ******************************************************
def packets_capture():
    # Use sniff to obtain the mirrored flow. The iface parameter of sniff needs to be modified to be consistent with the actual packet capture interface.
    mirrorPackets = sniff(filter = 'ip proto 47', iface = 'Realtek 8822CE Wireless LAN 802.11ac PCI-E NIC', timeout = 30)

    # Define the variable packInfoList to save information about mirrored service packets.
    packInfoList = []
    grePackets = []
    for packet in mirrorPackets:
        if packet.haslayer(GRE):
            # Check whether the GRE passenger protocol is ERSPAN.
            if packet[GRE].proto == 0x88be:
                packInfo = {
                    "macSrc": "",
                    "macDst": "",
                    "ipSrc": "",
                    "ipDst": "",
                    "ipProto": "",
                    "portSrc": "",
                    "portDst": "",
                }
                erspanPack = packet[GRE][1] # Get the ERSPAN packet in the GRE inner layer.
                etherPack = erspanPack[1] # Get the Ethernet packets in the inner layer of ERSPAN packets.
                macSrc = etherPack.src
                macDst = etherPack.dst
                if etherPack.type != 0x0800:
                    continue
                ipPack = etherPack[1] # Get the upper-layer IP packets of Ethernet packets.
                ipSrc = ipPack.src
                ipDst = ipPack.dst
                ipProto = ""
                # If the packets are ICMP packets...
                if ipPack.proto == 1:
                    ipProto = "icmp"
```

```
                elif ipPack.proto == 6:
                    ipProto = "tcp"
                    packInfo["portDst"] = ipPack[1].dport
                    packInfo["portSrc"] = ipPack[1].sport
                else:
                    # Only ICMP and TCP packets are processed, and UDP packets will be processed later.
                    continue
                packInfo["macSrc"] = macSrc
                packInfo["macDst"] = macDst
                packInfo["ipSrc"] = ipSrc
                packInfo["ipDst"] = ipDst
                packInfo["ipProto"] = ipProto
                if packInfo not in packInfoList:
                    packInfoList.append(packInfo)
    return packInfoList

# Main function
def main():
    flow_info = input_flow_info()
    packInfoList = packets_capture()
    devices = []
    CE_B = NetDevice("CE-B", "10.154.186.102", "python", "Huawei12#$", interfaceConnect_CE_B)
    CE_D = NetDevice("CE-D", "10.154.186.104", "python", "Huawei@1234", interfaceConnect_CE_D)
    devices.append(CE_B)
    devices.append(CE_D)
    path_calc(flow_info, packInfoList, devices)

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

8.3.2 Procedure

In this experiment, service packets are simulated using a ping or telnet test. IP address of 10GE 1/0/17 on CE-A: 192.168.2.2; IP address of 10GE 1/0/17 on CE-C: 10.1.14.1

1. On CE-A, start a ping test to CE-C and run the script on the local PC.
2. On CE-A, start a telnet test to CE-C and run the script on the local PC.

8.3.3 Execution Result

1. Flow path for the ping test from CE-A from CE-C

   icmp echo request packet path:

```
C:\Users\Anaconda3\python.exe D:/pyworkspace/scapy_expriment/mirrorAnalyse.py
Enter the 5-tuple information (source IP address, destination IP address, protocol, source port number, and destination port number) of the flow for path calculation: 192.168.2.2, 10.1.14.1, icmp.
CE-A(10GE1/0/17) ---> (10GE1/0/17)CE-B(10GE1/0/16) ---> (10GE1/0/16)CE-D(10GE1/0/17) ---> (10GE1/0/17)CE-C

Process finished with exit code 0
```

   icmp echo relpy packet path:

```
C:\Users\Anaconda3\python.exe D:/pyworkspace/scapy_expriment/mirrorAnalyse.py
```

```
Enter the 5-tuple information (source IP address, destination IP address, protocol, source port number, and destination port number) of the flow for path calculation: 10.1.14.1, 192.168.2.2, icmp.
CE-C(10GE1/0/17) ---> (10GE1/0/17)CE-D(10GE1/0/16) ---> (10GE1/0/16)CE-B(10GE1/0/17) ---> (10GE1/0/17)CE-A

Process finished with exit code 0
```

2. Flow path for the telnet test from CE-A to CE-C

   Client-to-server packet path:

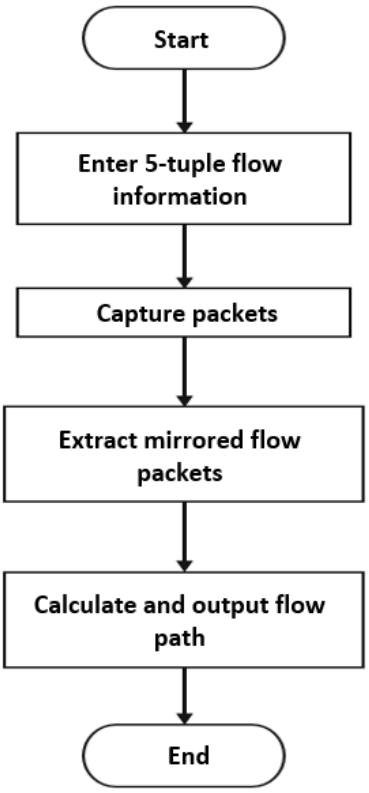
```
C:\Users\Anaconda3\python.exe D:/pyworkspace/scapy_expriment/mirrorAnalyse.py
Enter the 5-tuple information (source IP address, destination IP address, protocol, source port number, and destination port number) of the flow for path calculation: 192.168.2.2, 10.1.14.1, tcp, any, 23
CE-A(10GE1/0/17) ---> (10GE1/0/17)CE-B(10GE1/0/16) ---> (10GE1/0/16)CE-D(10GE1/0/17) ---> (10GE1/0/17)CE-C

Process finished with exit code 0
```

   Server-to-client packet path:

```
C:\Users\Anaconda3\python.exe D:/pyworkspace/scapy_expriment/mirrorAnalyse.py
Enter the 5-tuple information (source IP address, destination IP address, protocol, source port number, and destination port number) of the flow for path calculation: 10.1.14.1, 192.168.2.2, tcp, 23, any
CE-C(10GE1/0/17) ---> (10GE1/0/17)CE-D(10GE1/0/16) ---> (10GE1/0/16)CE-B(10GE1/0/17) ---> (10GE1/0/17)CE-A

Process finished with exit code 0
```

8.4 Code Explaination

The overall logic of the script is as follows:



Step 1 Import the modules.

```
from scapy.all import *
from scapy.contrib.erspan import *
import re, paramiko
```

Import the modules required by the script. In the script, you need to log in to the network device in SSH mode to collect and extract related information. Therefore, the **re** and **paramiko** modules need to be imported.

Step 2 Enter networking information.

```
# Create a dictionary to save networking information about CE-B and CE-D.
interfaceConnect_CE_B = { # The peer of 10GE1/0/17 on CE-B is...
    "10GE1/0/17": {"CE-A": "10GE1/0/17"},   # 10GE1/0/17 on CE-A
    "10GE1/0/14": {"CE-C": "10GE1/0/14"},
    "10GE1/0/16": {"CE-D": "10GE1/0/16"},
}

interfaceConnect_CE_D = {
    "10GE1/0/16": {"CE-B": "10GE1/0/16"},
    "10GE1/0/17": {"CE-C": "10GE1/0/17"},
}
```

The networking information about devices needs to be filled in the script in advance for flow path calculation.

Step 3 Create a **NetDevice** class.

The **NetDevice** class defines six device-related methods: **get_interfaces_info**, **get_mac_under_interface**, **is_to_device**, **is_from_device**, **in_device_path**, and **out_device_path**.

1. **NetDevice** class. It abstracts network devices.

```
# Create the NetDevice class to abstract a device and get its information.
class NetDevice:

def __init__(self, deviceName, deviceIp, username, password, interfaceConnect):
    self.deviceName = deviceName
        self.deviceIp = deviceIp
        self.username = username
        self.password = password
        self.interfaces = []
        self.interfacesConnect = {}
        self.macUnderInterface = {}
        self.interfaceMac = {}
        try:
            ssh = paramiko.client.SSHClient()
            ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
            ssh.connect(hostname=deviceIp, username=username, password=password)
            cli = ssh.invoke_shell()
            cli.send('N\n')
            time.sleep(0.5)
            # Disable split-screen display temporarily so that the command output can be displayed
on one screen.
            cli.send('screen-length 0 temporary\n')
            time.sleep(1)
        except:
            print("An exception occurred during SSH login to device {}.".format(deviceIp))
            raise

        self.get_interfaces_info(cli)
        self.get_mac_under_interface(cli)
        self.interfacesConnect = interfaceConnect
        ssh.close()
```

2. get_interfaces_info method. This method gets the MAC address of a network device interface.

```
    # ******************************************************
    # Function: to obtain all interfaces and their MAC addresses of the device and assign values to
self.interfaces
    # and self.interfaceMac.
    # Parameters:
    # cli - ssh shell object
    # ******************************************************
    def get_interfaces_info(self, cli):
        cli.send('display interface\n')
        time.sleep(15)
        interfaces_info = cli.recv(999999).decode()
        results = re.findall(r'^(\S+) current state.*\(ifindex', interfaces_info, re.M)
        if len(results) > 0:
```

```
            self.interfaces = results
        else:
            print("No interface is obtained.")

        interface_mac = ["", ""]
        lines = interfaces_info.splitlines()
        for i in lines:
            result1 = re.search(r'^(\S+) current state.*\(ifindex', i)
            if result1:
                interface_mac[0] = result1.group(1)

            result2 = re.search(r'Hardware address is (\S{4}-\S{4}-\S{4})', i)
            if result2:
                interface_mac[1] = result2.group(1)
                self.interfaceMac[interface_mac[0]] = interface_mac[1]
```

3. get_mac_under_interface method. This method gets the MAC address learned from the network device interface.

```
    # ************************************************************
    # Function: to obtain the MAC address learned from the device interface and assign a value to
self.macUnderInterface.
    # Parameters:
    # cli - ssh shell object
    # ************************************************************
    def get_mac_under_interface(self, cli):
        cli.send('display mac-address\n')
        time.sleep(3)
        mac_address_info = cli.recv(999999).decode()
        cli.send('display arp\n')
        time.sleep(3)
        arp_info = cli.recv(999999).decode()

        mac_interface = re.findall(r'^(\S{4}-\S{4}-\S{4}) \S+ +(\S+) ', mac_address_info, re.M)
        if len(mac_interface) > 0:
            for i in mac_interface:
                if i[1] not in self.macUnderInterface.keys():
                    self.macUnderInterface[i[1]] = []
                    self.macUnderInterface[i[1]].append(i[0])
                else:
                    if i[0] not in self.macUnderInterface[i[1]]:
                        self.macUnderInterface[i[1]].append(i[0])
        else:
            print("No entry in the MAC address table.")

        arp_interface = re.findall(r' (\S{4}-\S{4}-\S{4}) .*D.* +(\S+)', arp_info, re.M)
        if len(arp_interface) > 0:
            for i in arp_interface:
                if i[1] not in self.macUnderInterface.keys():
                    self.macUnderInterface[i[1]] = []
                    self.macUnderInterface[i[1]].append(i[0])
                else:
                    if i[0] not in self.macUnderInterface[i[1]]:
                        self.macUnderInterface[i[1]].append(i[0])
        else:
            print("No dynamic entries in ARP entries.")
```

4. is_to_device method. This method determines whether the service packet is sent to the local network device according to the information collected in the previous method. If the destination MAC address of the service packet is the MAC address of the local network device, the packet is sent to the local network device.

```
    # ******************************************************
    # Function: to determine whether packets are sent to the device.
    # Parameters:
    # packet_info - packet information
    # Returned value:
    #     True/False
    # ******************************************************
    def is_to_device(self, packet_info):
        for key in self.interfaceMac.keys():
            # The MAC address may be in the 08e8-4f6e-0516 or 08:e8:4f:6e:05:16 format, which has
to be converted for a comparison.
            if re.sub(r"[-:]", "", packet_info.get("macDst")) == re.sub(r"[-:]", "",
self.interfaceMac.get(key)):
                return True
        return False
```

5. is_from_device method. This method determines whether the service packet is sent from the local network device according to the information collected in the previous method. If the source MAC address of the service packet is the MAC address of the local network device, the packet is sent from the local network device.

```
    # ******************************************************
    # Function: to determine whether a packet is sent by the local device.
    # Parameters:
    # packet_info - packet information
    # Returned value:
    #     True/False
    # ******************************************************
    def is_from_device(self, packet_info):
        for key in self.interfaceMac.keys():
            # The MAC address may be in the 08e8-4f6e-0516 or 08:e8:4f:6e:05:16 format, which has
to be converted for a comparison.
            if re.sub(r"[-:]", "", packet_info.get("macSrc")) == re.sub(r"[-:]", "",
self.interfaceMac.get(key)):
                return True
        return False
```

6. in_device_path method. This method calculates the path along which a service packet is sent to the local network device. Specifically, this method checks the interface through which the packet enters the network device by searching the MAC address table and ARP table by the source MAC address. Then, it identifies the peer device and interface according to the networking information. If the local network device is CE2 and service packets are sent from GE 1/0/0 on CE1 to GE 1/0/2 on CE2, then the calculated path is CE1(GE1/0/0) ---> (GE1/0/2)CE2.

```
# ******************************************************
```

```
    # Function: to obtain the inbound interface of the packet and the interface on the peer device, and
return the path section.
    # Parameters:
    # packet_info - packet information
    # Returned value:
    #     packet_path
    #     [{"CE1": "interface1"},
    #      {"CE2": "interface2"}]
    # Interface 1 on CE1 sends packets to interface 2 on CE2.
    # ********************************************************
    def in_device_path(self, packet_info):
        # The peer device sends packets along the following path:
        packet_path_out = {}
        # The local device receives packets along the following path:
        packet_path_in = {}
        interface = ""
        for key in self.macUnderInterface.keys():
            macs = self.macUnderInterface.get(key)
            for i in range(len(macs)):
                macs[i] = re.sub(r"[-:]", "", macs[i])

            if re.sub(r"[-:]", "", packet_info.get("macSrc")) in macs:
                interface = key
                break

        if interface == "":
            raise Exception('No inbound interface on the device is found for the packets. Packet
information: \n{}.format(packet_info))

        if interface not in self.interfacesConnect:
            raise Exception('The interface on device {} does not have information about the peer
device and interface'. format(interface))

        for key, value in self.interfacesConnect.get(interface).items():
            packet_path_out[key] = value

        packet_path_in[self.deviceName] = interface
        return [packet_path_out, packet_path_in]
```

7. out_device_path method. This method calculates the path of the service packet sent from the local network device. Specifically, this method checks the interface through which the packet is sent by searching the MAC address table and ARP table by the destination MAC address. Then, it identifies the peer device and interface according to the networking information. If the local network device is CE1 and service packets are sent from GE 1/0/0 on CE1 to GE 1/0/2 on CE2, the calculated path is CE1(GE1/0/0) ---> (GE1/0/2)CE2.

```
    # ********************************************************
    #Function: to get the outbound interface of the packet and the interface of the peer device, and
return the path section.
    # Parameters:
    # packet_info - packet information
    # Returned value:
    #     packet_path
```

```
    #     [{"CE1": "interface1"},
    #      {"CE2": "interface2"}]
    # Interface 1 on CE1 sends packets to interface 2 on CE2.
    # ******************************************************
    def out_device_path(self, packet_info):
        # The local device sends packets along the following path:
        packet_path_out = {}
        # The peer device receives packets along the following path:
        packet_path_in = {}
        interface = ""
        for key in self.macUnderInterface.keys():
            macs = self.macUnderInterface.get(key)
            for i in range(len(macs)):
                macs[i] = re.sub(r"[-:]", "", macs[i])

            if re.sub(r"[-:]", "", packet_info.get("macDst")) in macs:
                interface = key
                break

        if interface == "":
            raise Exception('No outbound interface on the device is found for the packets. Packet
information: \n{}.format(packet_info))

        if interface not in self.interfacesConnect:
            raise Exception('The interface on device {} does not have information about the peer
device and interface'. format(interface))

        for key, value in self.interfacesConnect.get(interface).items():
            packet_path_in[key] = value

        packet_path_out[self.deviceName] = interface
        return [packet_path_out, packet_path_in]
```

Step 4 Enter 5-tuple information of the flow.

The **input_flow_info** method prompts the user to enter 5-tuple information of the flow for path calculation, saves the 5-tuple information to the **flow_info** variable, and returns the value.

```
# ******************************************************
#Function: to prompt users to enter the 5-tuple information of the flow for path calculation.
# Parameters:
#     None
# Returned value:
# flow_info: flow 5-tuple information in the following format:
#     flow_info = {
#           "ipSrc": "10.1.12.2",
#           "ipDst": "192.168.2.2",
#           "ipProto": "tcp",
#           "portSrc": 23,
#           "portDst": "any",
#     }
# ******************************************************
def input_flow_info():
    flow_info = {} # Save the 5-tuple information of flow for path calculation.
```

```
    input_str = input("Enter the 5-tuple information (source IP address, destination IP address, protocol, source port number, and destination port number) of the flow for path calculation."
    input_list = input_str.split(",")
    flow_info["ipSrc"] = input_list[0].strip()
    flow_info["ipDst"] = input_list[1].strip()
    flow_info["ipProto"] = input_list[2].strip()
    flow_info["portSrc"] = ""
    flow_info["portDst"] = ""
    if flow_info["ipProto"] == "tcp" or flow_info["ipProto"] == "udp":
        portSrc = input_list[3].strip()
        if portSrc != "any":
            flow_info["portSrc"] = int(portSrc)
        else:
            flow_info["portSrc"] = "any"

        portDst = input_list[4].strip()
        if portDst != "any":
            flow_info["portDst"] = int(portDst)
        else:
            flow_info["portDst"] = "any"
    return flow_info
```

Step 5 Capture packets using **Scapy**.

The **packets_capture** method uses the **sniff** function of the **Scapy** module to capture mirrored packets, extracts the source and destination MAC addresses, source and destination IP addresses, protocol, and source and destination port numbers from the encapsulated service packets, and saves them to the **packInfoList** variable.

For details about how to use **Scapy**, see the official document at https://scapy.readthedocs.io/en/latest/introduction.html.

```
# ****************************************************
# Function: to use the scapy module to capture packets and extract information about mirrored service packets.
# Parameters:
#     None
# Returned value:
# packInfoList - list, which saves information about mirrored service packets.
# ****************************************************
def packets_capture():
    # Use sniff to obtain mirrored flow. The iface parameter of sniff needs to be modified to be consistent with the actual packet capture interface.
    mirrorPackets = sniff(filter = 'ip proto 47', iface = 'Realtek 8822CE Wireless LAN 802.11ac PCI-E NIC', timeout = 30)

    # Define the variable packInfoList to save information about mirrored service packets.
    packInfoList = []
    grePackets = []
    for packet in mirrorPackets:
        if packet.haslayer(GRE):
            # Check whether the GRE passenger protocol is ERSPAN.
            if packet[GRE].proto == 0x88be:
                packInfo = {
```

```
                    "macSrc": "",
                    "macDst": "",
                    "ipSrc": "",
                    "ipDst": "",
                    "ipProto": "",
                    "portSrc": "",
                    "portDst": "",
                }
                erspanPack = packet[GRE][1] # Get the ERSPAN packet in the GRE inner layer.
                etherPack = erspanPack[1] # Get the Ethernet packets in the inner layer of ERSPAN packets.
                macSrc = etherPack.src
                macDst = etherPack.dst
                if etherPack.type != 0x0800:
                    continue
                ipPack = etherPack[1] # Get the upper-layer IP packets of Ethernet packets.
                ipSrc = ipPack.src
                ipDst = ipPack.dst
                ipProto = ""
                # If the packets are ICMP packets...
                if ipPack.proto == 1:
                    ipProto = "icmp"
                elif ipPack.proto == 6:
                    ipProto = "tcp"
                    packInfo["portDst"] = ipPack[1].dport
                    packInfo["portSrc"] = ipPack[1].sport
                else:
                    # Only ICMP and TCP packets are processed, and UDP packets will be processed later.
                    continue
                packInfo["macSrc"] = macSrc
                packInfo["macDst"] = macDst
                packInfo["ipSrc"] = ipSrc
                packInfo["ipDst"] = ipDst
                packInfo["ipProto"] = ipProto
                if packInfo not in packInfoList:
                    packInfoList.append(packInfo)
    return packInfoList
```

Step 6 Use the **main** method.

The **main** method calculates and outputs the flow path. The code prompts the user to enter the 5-tuple information of the flow for path calculation and then uses the **Scapy** to capture packets. Instantiate each device with remote mirroring enabled on the network and add all **NetDevice** class instances to the **devices** variable. The **path_calc** method uses the 5-tuple flow information, mirrored service packet information, and devices with remote mirroring enabled as parameters to calculate and output the flow path.

```
def main():
    flow_info = input_flow_info()
    packInfoList = packets_capture()
    devices = []
    CE_B = NetDevice("CE-B", "10.154.186.102", "python", "Huawei12#$", interfaceConnect_CE_B)
    CE_D = NetDevice("CE-D", "10.154.186.104", "python", "Huawei@1234", interfaceConnect_CE_D)
    devices.append(CE_B)
```

```
    devices.append(CE_D)
    path_calc(flow_info, packInfoList, devices)

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

----End

8.5 Quiz

In this experiment, the source and destination MAC addresses of service packets are compared with the MAC addresses of network devices to determine the network devices and interfaces through which the packets pass. However, the inbound and outbound interfaces may be incorrectly identified by searching the MAC address table and ARP table, because the MAC address table and ARP table of network devices are obtained only after packets are captured while they may change in the course. Is there any simpler and more accurate method of determining the network devices and interfaces that service packets pass through?

Reference Answers to Quiz

- Reference answers to quiz in *SSH Experiment*:

  You can use SSH to log in to multiple devices to view configurations in a cyclic manner.

- Reference answers to quiz in *Automatic SNMP Configuration Experiment:*

1. User-defined function.
2. Multiple threads can be used. For details, see the *Python Programming Basics*.

- Reference answers to quiz in *NETCONF Configuration Experiment:*

1. The **if __name__ == '__main__'** code block is not executed when it is imported as a module.
2. The NETCONF query object is YANG, and the SNMP query object is MIB.

- Reference answers to quiz in *Configuration File Comparison Experiment:*

1. N/A
2. Some code for logging in to the device can be encapsulated as a function, which can be invoked by the functions to write and get configurations.
3. The time module is introduced to implement the timing function, and NETCONF is used to obtain device configuration files more efficiently.

- Reference answers to quiz in *gRPC Remote Query Configuration Experiment:*

  N/A

- Reference answers to quiz in *Telemetry Configuration Experiment:*

1. gRPC compression implements more efficient transmission when the data volume is excessively large.
2. According to the network application requirements, data can be collected and parsed in a synchronous or asynchronous manner. In synchronous manner, data is parsed while being collected; in asynchronous manner, all data is collected and then parsed.

- Reference answers to quiz in *OPS Experiment*:

  In the ZTP process, the device (without a configuration file and without a USB flash drive inserted) obtains a temporary IP address and file server address

from the DHCP server, obtains the configuration script (**boot.py**) from the file server, and runs the script. The script implements functions related to device initialization, such as obtaining the version, viewing the configuration file, and configuring the IP address.

- Reference answer to quiz in *Network Flow Analysis Experiment*:

  When configuring Layer 3 remote mirroring on a network device, you can specify ERSPAN ID. The command is as follows: observe-port [observe-port-index] [interface interface-type interface-number] destination-ip dest-ip-address source-ip source-ip-address [dscp dscp-value] [erspan-id erspan-id].

  The **erspan-id** parameter is encapsulated in the mirrored packet to differentiate mirrored packets, and its value ranges from 0 to 1023. You can use **erspan-id** to determine the network devices and interfaces through which the mirrored service packets pass. In this experiment, you can configure remote mirroring on CE-B as follows:

```
<CE-B>system-view immediately
Enter system view, return user view with return command.
[CE-B]observe-port 1 destination-ip 10.166.231.109 source-ip 10.154.186.102 erspan-id 1
[CE-B]observe-port 2 destination-ip 10.166.231.109 source-ip 10.154.186.102 erspan-id 2
[CE-B]observe-port 3 destination-ip 10.166.231.109 source-ip 10.154.186.102 erspan-id 3
[CE-B]observe-port 4 destination-ip 10.166.231.109 source-ip 10.154.186.102 erspan-id 4
[CE-B]interface 10GE 1/0/17
[CE-B-10GE1/0/17] port-mirroring observe-port 1 inbound
[CE-B-10GE1/0/17] port-mirroring observe-port 2 outbound
[CE-B-10GE1/0/17]interface 10GE 1/0/16
[CE-B-10GE1/0/16] port-mirroring observe-port 3 inbound
[CE-B-10GE1/0/16] port-mirroring observe-port 4 outbound
[CE-B-10GE1/0/16]quit
```

  If the value of **erspan-id** in the remote mirrored packet is 2, service packets are sent out from 10GE1/0/17 on CE-B in the outbound direction.

  For CE-D remote mirroring, the value of **erspan-id** ranges from 5 to 8. The **erspan-id** values of the two devices are different, which can be used to determine the network devices that packets pass through.

Huawei Certification Training

HCIP-Datacom-Network Automation Developer NCE Northbound Openness Lab Guide

V1.0

Huawei Technologies CO., LTD.

Trademarks and Permissions

and other Huawei trademarks are trademarks of Huawei Technologies Co., Ltd.

All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The purchased products, services and features are stipulated by the contract made between Huawei and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

Huawei Technologies Co., Ltd.

Address: Huawei Industrial Base
Bantian, Longgang
Shenzhen 518129
People's Republic of China

Website: http://e.huawei.com

Huawei Certification System

Huawei Certification follows the "platform + ecosystem" development strategy, which is a new collaborative architecture of ICT infrastructure based on "Cloud-Pipe-Terminal". Huawei has set up a complete certification system consisting of three categories: ICT infrastructure certification, platform and service certification, and ICT vertical certification. It is the only certification system that covers all ICT technical fields in the industry.

Huawei offers three levels of certification: Huawei Certified ICT Associate (HCIA), Huawei Certified ICT Professional (HCIP), and Huawei Certified ICT Expert (HCIE). Huawei Certification covers all ICT fields and adapts to the industry trend of ICT convergence. With its leading talent development system and certification standards, it is committed to fostering new ICT talent in the digital era, and building a sound ICT talent ecosystem.

HCIP-Datacom-Network Automation Developer is designed for senior engineers with professional knowledge and skills in network automation development in the datacom field. Passing the HCIP-Datacom-Network Automation Developer certification proves that you are competent for the position of enterprise network automation development engineer and have the capability of using Huawei datacom devices for automatic deployment, development, and O&M of enterprise networks.

The Huawei certification system introduces the industry, fosters innovation, and imparts cutting-edge datacom knowledge.

Huawei Certification
ICT Vertical Certification
行业ICT认证
Finance
Public Safety
Platform and Service Certification
平台与服务认证
Big Data
AI
IoT
Intelligent Video Surveillance
Enterprise Communication
GaussDB
Cloud Computing
Cloud Service
Kunpeng Application Developer
ICT Infrastructure Certification
ICT技术架构认证
Data Center
Storage
Intelligent Computing
Datacom
WLAN
SDN
Transmission
Access
LTE
5G
Security
HCIE
Huawei Certified ICT Expert
HCIP
Huawei Certified ICT Professional
HCIA
Huawei Certified ICT Associate

About This Document

Introduction

This document is designed for the HCIP-Datacom-Network Automation Developer certification training course and is intended for trainees who are going to take the HCIP-Datacom-Network Automation Developer exam or readers who want to understand the basic knowledge and practices of Huawei iMaster NCE northbound openness.

Background Knowledge Required

This document is intended for senior network automation engineers. To better understand this course, familiarize yourself with the following requirements:

- Python programming basics
- Basic understanding of HUAWEI CLOUD code hosting
- RESTful fundamentals and practices
- Basic understanding of Huawei iMaster NCE products

Lab Environment Description

Environment Description

This lab environment is based on the Intent-Driven Network (IDN) developer community.

The IDN developer community is an open platform that provides one-stop support and services (including learning, development, testing, and communication) for developers and partners in the datacom field based on Huawei DevCloud. Currently, this community provides online experiment environments for the CloudCampus Solution, CloudFabric Solution, and CloudWAN Solution. In addition, it provides API Explorer, API Studio, sandboxes, DevOps IDE and software development kit (SDK), and other resources to help developers quickly develop, integrate, and roll out industry applications. The IDN developer community is now available only in Chinese, and its English version is coming soon.

This document describes how to call open northbound APIs in the CloudCampus Solution and CloudFabric Solution in the IDN developer community.

Instructions

- Register a HUAWEI CLOUD account and complete real-name authentication at https://www.huaweicloud.com/en-us/.
- For more information, visit the IDN developer community(CloudCampus Solution) at http://intl.devzone.huawei.com/en/network/cloudcampus.html.

Contents

1 RESTful API Calling Practice

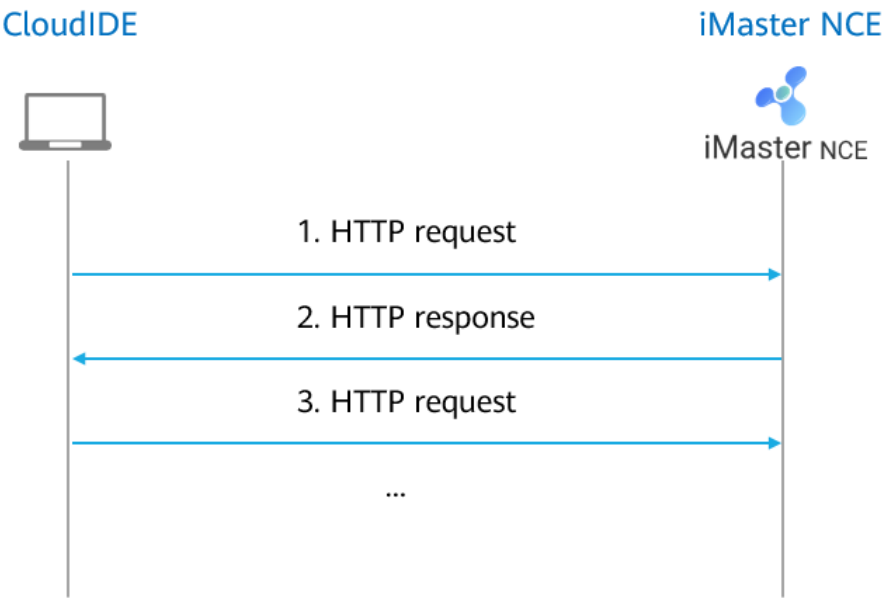
1.1 Background

Huawei iMaster NCE is a network automation and intelligence platform that integrates management, control, analysis, and AI functions. Under a software-defined networking (SDN) architecture, an SDN controller manages network devices in a unified manner. You can set up intent-driven networks using open northbound APIs provided by the SDN controller.

This experiment aims to help you quickly master how to use the Python requests module to log in to iMaster NCE and call its service APIs.

1.2 Introduction

1.2.1 Networking



This experiment involves two objects: CloudIDE and iMaster NCE.

- CloudIDE is a cloud-based development environment provided by HUAWEI CLOUD. You can also compile code in your local environment.
- A sandbox environment for iMaster NCE is provided in the IDN developer community.

1.2.2 Procedure

The experiment procedure is as follows:

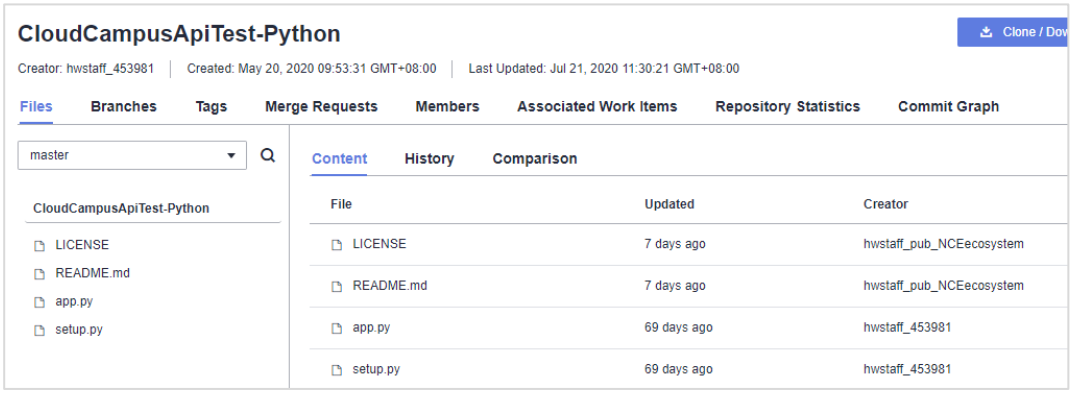
1. Prepare the environment: Prepare an integrated development environment (IDE) and a lab sandbox environment.
2. Compile code using Python for calling RESTful APIs.
3. Verify the configuration.

1.3 Environment Preparations

1.3.1 Preparing an IDE

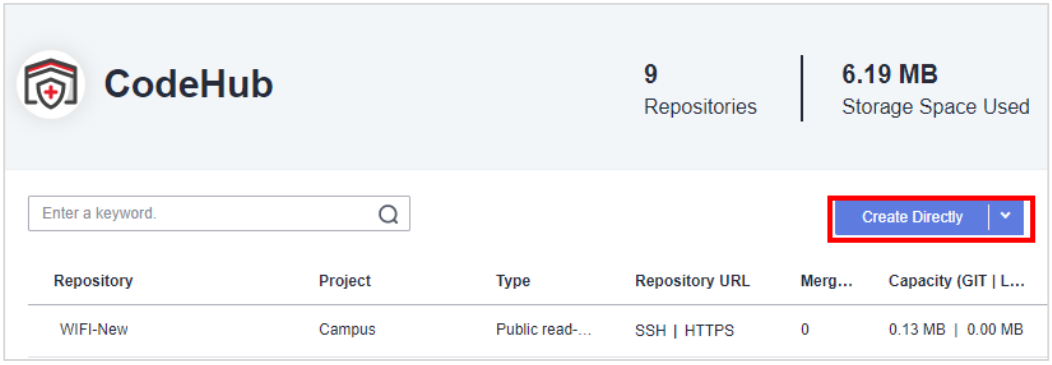
Complete code files involved in this experiment have been uploaded to DevCloud on HUAWEI CLOUD. You can use Huawei CloudIDE for online programming or download the code files to your local PC from https://devcloud.cn-north-4.huaweicloud.com/codehub/project/68494a8ad06b4eea9a1c3f18be115161/codehub/620653/home.

Currently, DevCloud is available only at the China site. You can select the China site and switch the language to English or download the code to your local host and run it on the local IDE. The following uses DevCloud as an example.



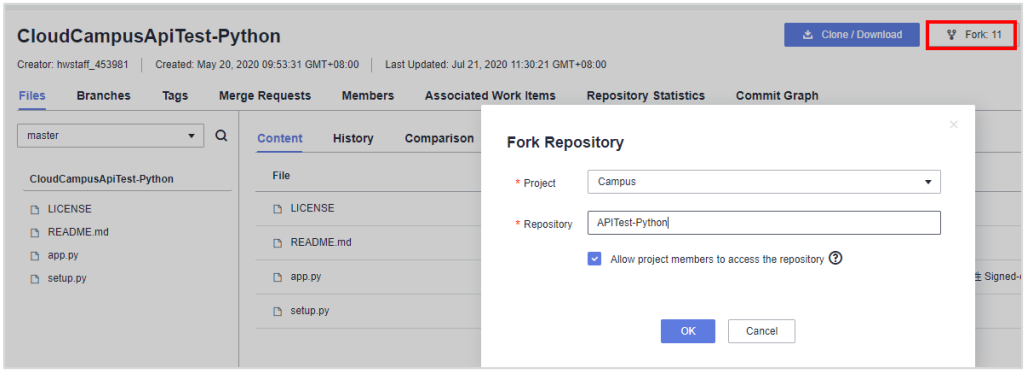
If not, you can also download codes from the following link: https://intl.devzone.huawei.com/en/enterprise/campus/sdkList.html.

1. Access the CodeHub console and create a project and a cloud-based code repository.

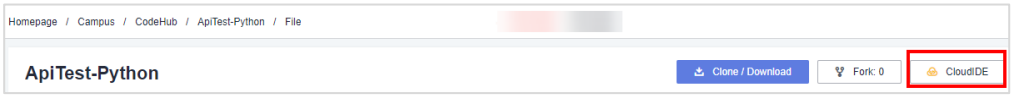


For more operations, see the CodeHub guide at https://support.huaweicloud.com/en-us/usermanual-cts/cts_1229.html

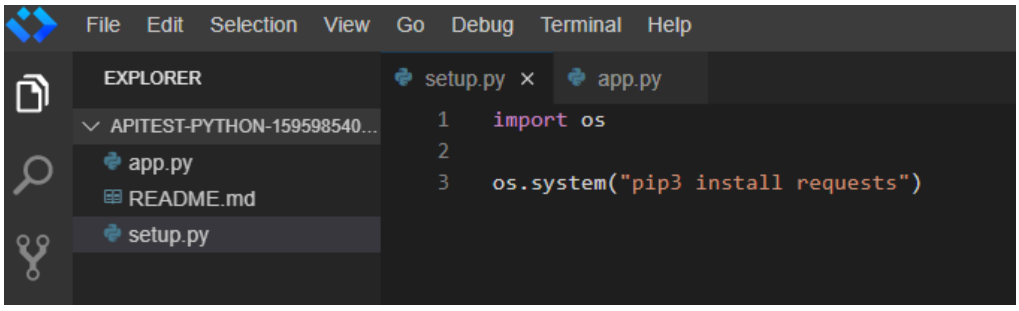
2. Fork complete code files to the cloud-based code repository on CodeHub.



3. Click **CloudIDE** to open the code files.



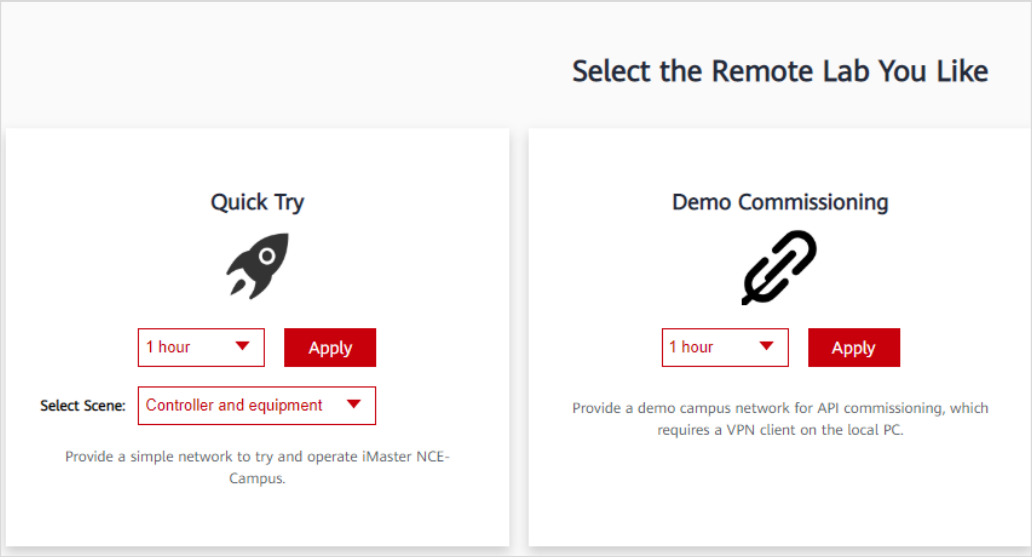
4. The CloudIDE window is displayed as follows.



1.3.2 Reserving a Sandbox Environment

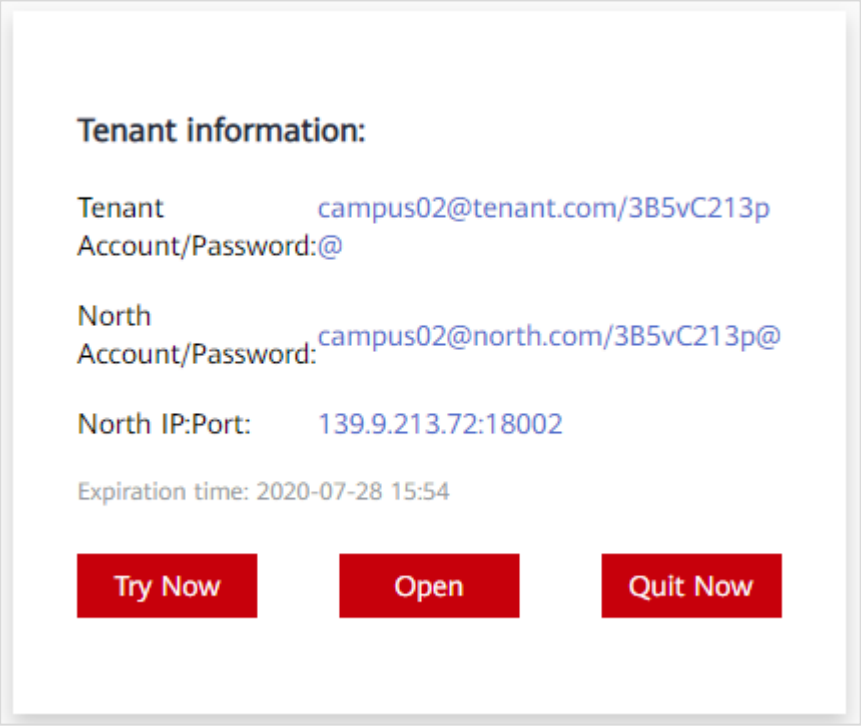
The IDE developer community provides a sandbox environment for Huawei datacom solutions. In this experiment, you need to apply for a CloudCampus sandbox environment.

You can experience the sandbox environment in two ways: quick try and demo commissioning.



- Quick try: Lab access through a public network, which is applicable to code development in CloudIDE
- Demo commissioning: Lab access through a VPN, which is applicable to code development in the local IDE

To access the sandbox environment in quick try mode, select **Controller and equipment** from the **Select Scene** drop-down list, select an experience duration, and click **Apply**. Then, the information for logging in to the sandbox environment is provided.



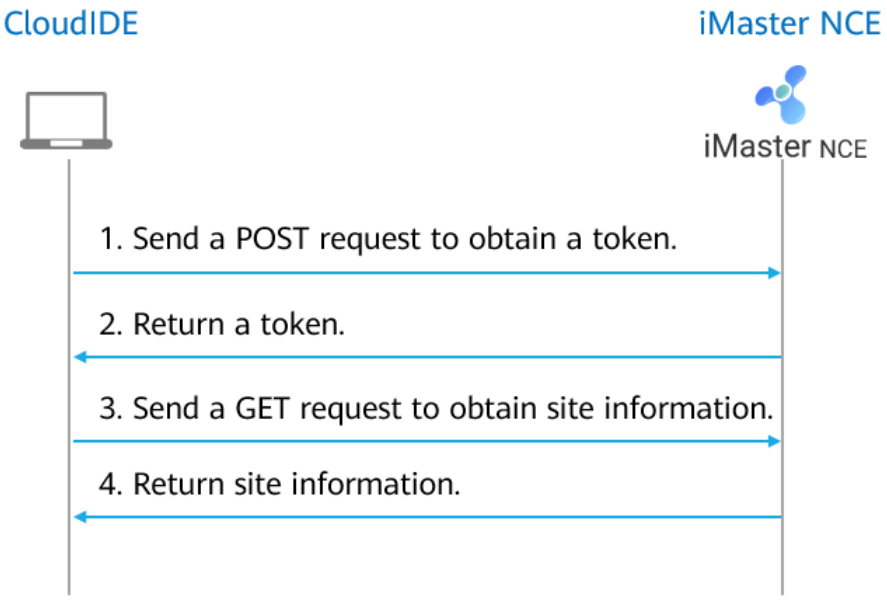
Tenant Account/Password indicates the account and password for logging in to the iMaster NCE-Campus sandbox environment.

North Account/Password indicates the account and password used by the Location-based service (LBS) application to interact with iMaster NCE-Campus, as shown in steps 1 to 3 in 1.2.1 Networking.

North IP:Port indicates the login address of iMaster NCE-Campus. You can click **Try Now** to go to the iMaster NCE login page.

So far, the environment is prepared.

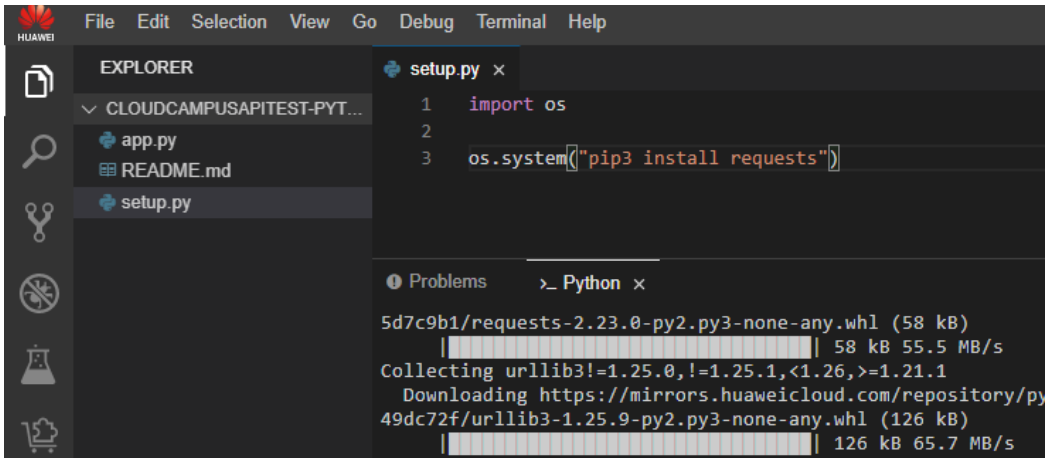
1.4 Python Code Compilation



The preceding figure shows the interaction between Python code and iMaster NCE. You need to compile code to initiate a POST request to obtain an authentication token and then initiate a GET request (carrying the token) to obtain site information.

1.4.1 Installing the Requests Module

In this experiment, the Python requests module is required. Right-click the **setup.py** file and choose **Run Python File in Terminal** to install the requests module.



1.4.2 Configuring Interconnection with iMaster NCE

Configure iMaster NCE information.

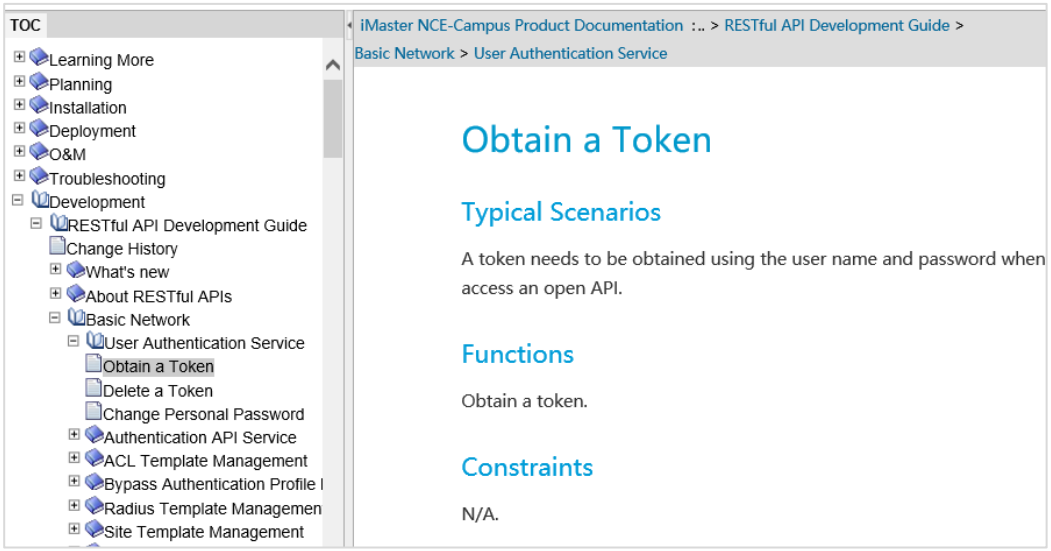
Set the parameter values obtained on the sandbox environment reservation page, including the northbound username, password, IP address, and port number.

```
import requests

# Configure the northbound user and northbound login address.
nbi_name = "campus02@north.com"
nbi_pwd = "B5vC213p@"
host = "139.9.213.72"
port = "18002"
```

1.4.3 Obtaining Login Authentication Information

iMaster NCE determines a user's permissions based on the tenant account that the user uses to log in to iMaster NCE. When the client (Python code) sends a login authentication request to the server, the server will return a token and its expiration time.

For the Uniform Resource Identifier (URI), request header, and request body of the API used to obtain a token, see "RESTful API Development Guide" in the *iMaster NCE-Campus Product Documentation*, as shown in the following figure.



1. Send a POST request.

   In this example, the URI is **/controller/v2/tokens**, the request header contains **Content-Type** and **Accept**, and the body contains **userName** and **password**.

```
# Define the URI of the API for obtaining a token.
POST_TOKEN_URL = "/controller/v2/tokens"

# Configure the URL and header.
post_token_url = "https://" + host + ":" + port + POST_TOKEN_URL
headers_post = {'Content-Type': 'application/json', 'Accept': 'application/json'}
# Initiate a request with data in JSON format.
r = requests.post(post_token_url, headers=headers_post, json={"userName": nbi_name, "password":
nbi_pwd}, verify=False)
```

   The POST method is required for this HTTP operation according to the development guide. In this example, the requests.post method is used to initiate a request.

   For more requests methods, visit https://requests.readthedocs.io/en/master/.

2. Obtain a token and display it.

   Parse the response returned by iMaster NCE.

```
# Parse token_id.
token_id = r.json()['data']['token_id']
print("1.【Get Token Id】")
```

```
print("【post_token_url】: "+post_token_url)
print("【token_id】: "+token_id)
```

The command output is as follows:

```
1.【Get Token Id】
【post_token_url】: https://139.9.213.72:18002/controller/v2/tokens
【token_id】:
x-rs5jvy7z47vsum042r2qc9vtaq44o9hgqohf3uca7z9clio5vvvz86al3s5dvs0aphpck5defw45uqpi3xdgddmn
482kgajutf7vlfs649rw6k5c6r6l0a5i1g4b5j9c
```

1.4.4 Obtaining Site Information

Call the site query API provided by iMaster NCE-Campus to initiate a GET request.

1. Obtain the URI of the site query API.

   Obtain the URI of the site query API from the API documentation.

   ```
   # Define the URI of the site query API.

   GET_SITES_URL = "/controller/campus/v3/sites"
   ```

2. Configure attributes in the GET request.

   Configure the URL and header. Note that **token_id** must be placed in the header.

   ```
   # Configure the URL and header.
   get_sites_url = "https://" + host + ":" + port + GET_SITES_URL

   headers_get = {'Content-Type': 'application/json', 'Accept': 'application/json', 'X-AUTH-TOKEN':token_id}
   ```

   The **X-AUTH-TOKEN field** carries **token_id**.

3. Send a GET request.

   ```
   # Initiate a request.
   r = requests.get(get_sites_url, headers=headers_get, verify=False)
   ```

4. Parse the response and output the result.

   ```
   # Parse site information.
   print("2.【Get Sites Info】")
   print("【get_sites_url】: "+get_sites_url)
   total_records = r.json()['totalRecords']
   print("【total_records】: "+str(total_records))
   print(r.text)
   ```

   The command output is as follows:

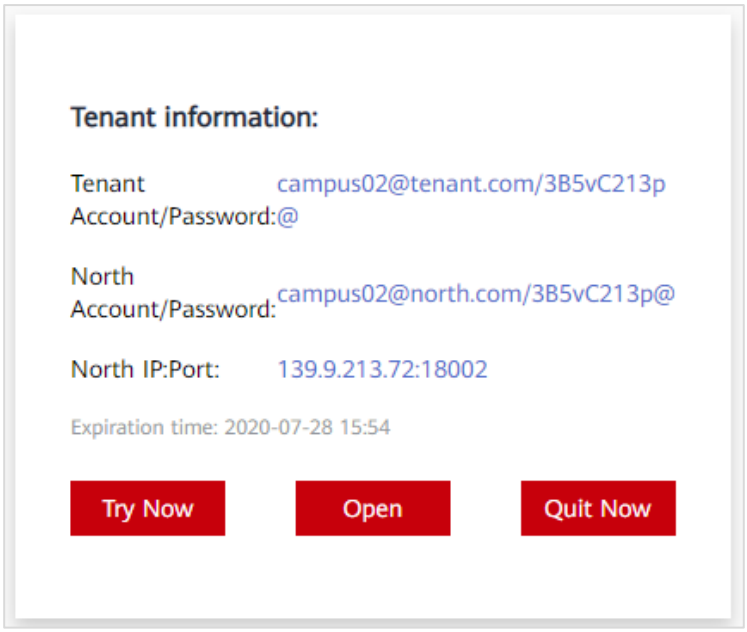
   ```
   2.【Get Sites Info】
   【get_sites_url】: https://139.9.213.72:18002/controller/campus/v3/sites
   【total_records】: 1
   {"errcode":"0","errmsg":"","totalRecords":1,"pageIndex":1,"pageSize":20,"data":[{"id":"95e61c0c-ed50-46
   83-9fb3-30dcbfca3c69","tenantId":"687fc863-8aaa-4c28-ba3a-e912a6b52e77","name":"site01","descript
   ion":"","type":["AP"]}]}
   ```

   According to the output, a site named **site01** of the AP type has been configured on iMaster NCE-Campus.

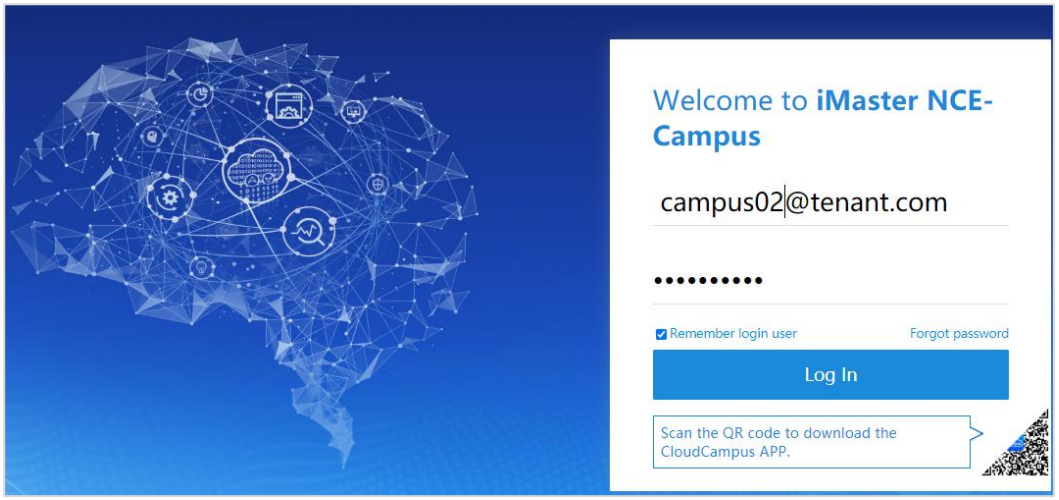
1.5 Verification

Log in to iMaster NCE-Campus to verify the configuration.

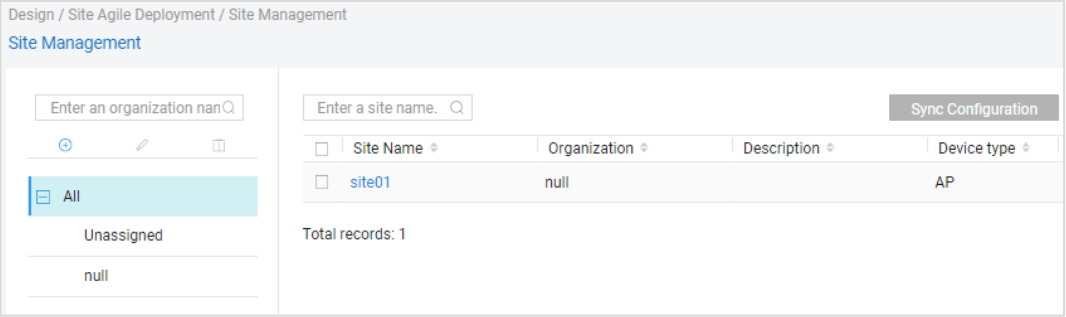
Click **Try Now** on the sandbox environment reservation page.



Enter the tenant account and password to log in to iMaster NCE-Campus.



Choose **Design** > **Site Agile Deployment** > **Site Management** to view information about sites.



The practice is complete.

Developers can call APIs provided by iMaster NCE-Campus to provision services as needed.

1.6 Quiz

How many times does the Python code interact with iMaster NCE-Campus in this example? What is the difference?

2 Wi-Fi Terminal Location Practice in Huawei CloudCampus Solution

2.1 Background

Huawei CloudCampus Solution combines cutting-edge wired and wireless technologies with big data, AI, and cloud technologies to build service-centric campus networks that feature ubiquitous connections, worry-free services, and smooth evolution, enabling digital transformation in the industry.

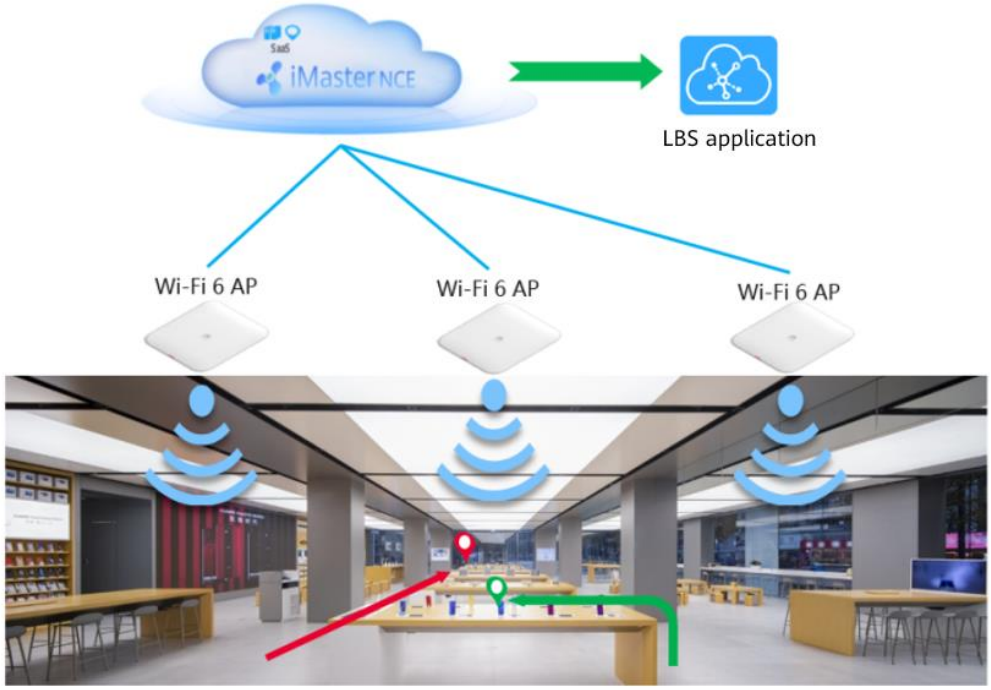
This experiment guides you through the development of a location-based service (LBS) and the calling of LBS and value-added service (VAS) APIs provided by iMaster NCE-Campus to collect locations of wireless users. In this course, you will learn to:

- Call the LBS APIs provided by iMaster NCE-Campus.
- Call the value-added service (VAS) APIs provided by iMaster NCE-Campus.
- Parse the terminal location data reported to LBS applications by iMaster NCE-Campus.

2.1.1 Introduction to LBS

LBS uses various locating technologies to obtain the current locations of devices and provide information and basic service for these devices through mobile Internet.

Huawei CloudCampus Solution provides an LBS which obtains location data based on Wi-Fi and reports the data to developers' LBS applications. Developers then can compute the locations of terminals (associated or not).

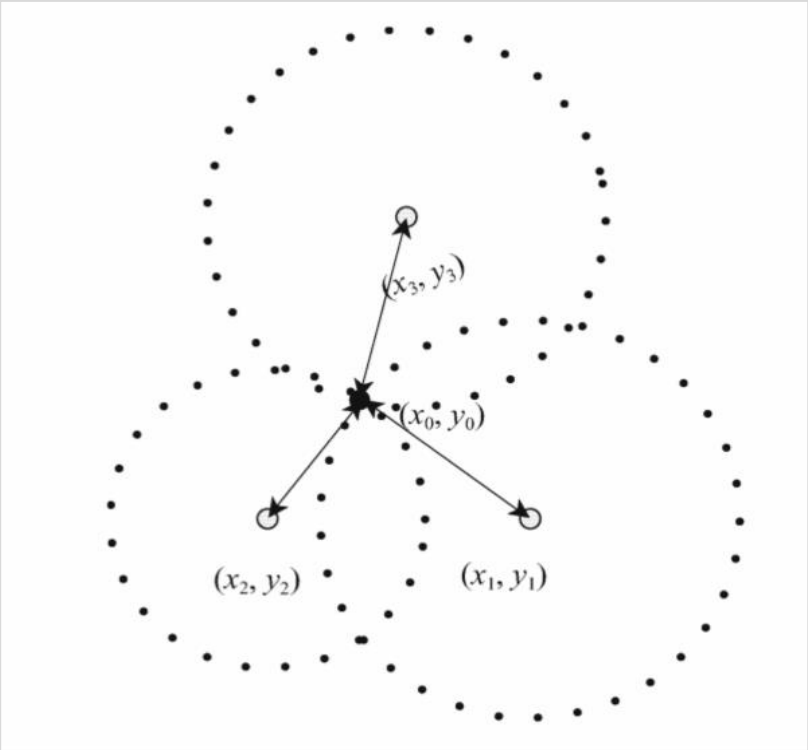


LBS and VAS can be used together to provide the following functions: real-time locating, indoor navigation, customer flow analysis, user profiling, asset monitoring and management, promotion push, location heatmap, and tracking.

2.1.2 Introduction to Wi-Fi Locating

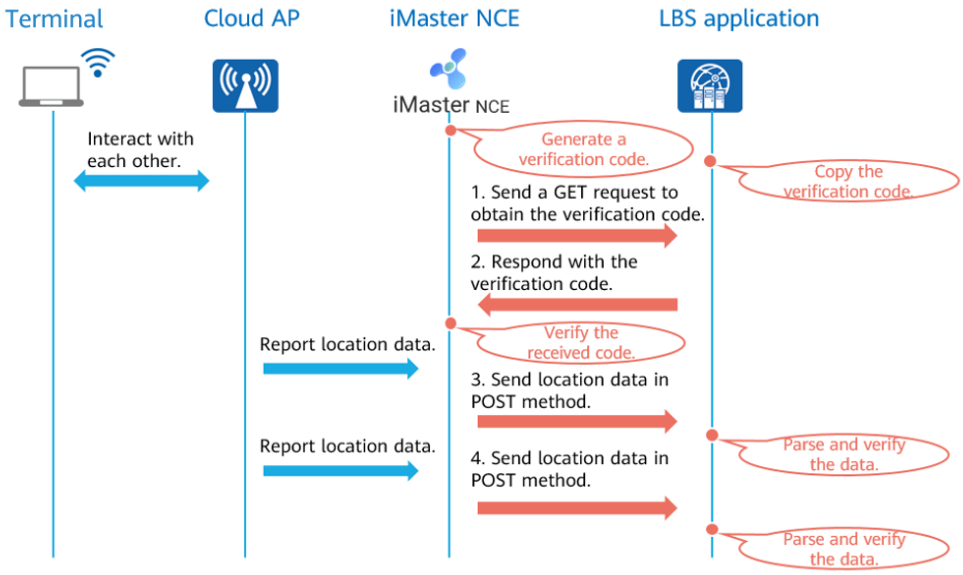
LBS over Wi-Fi is a popular indoor locating technology. It adopts the trilateration method based on the received signal strength indicator (RSSI) model.

It first calculates the distance between terminals and APs based on RSSIs. When more than three APs receive RSSI at the same time, take each AP as a circular center and the terminals are at the points of interactions. The following algorithm model can be extracted: figure out the coordinates of an unknown point (x0,y0) based on the locations of more than three fixed points (x1,y1), (x2,y2), and (x3,y3), and their distance to the unknown point (l1, l2, and l3).



2.2 Introduction

2.2.1 Networking



This lab network consists of four objects: a terminal, a Cloud AP, iMaster NCE-Campus, and an LBS application. The IDE developer community can provide a sandbox environment with the terminal, Cloud AP, and iMaster NCE-Campus. You need to develop an LBS application, which is recommended to be run in CloudIDE.

2.2.2 Procedure

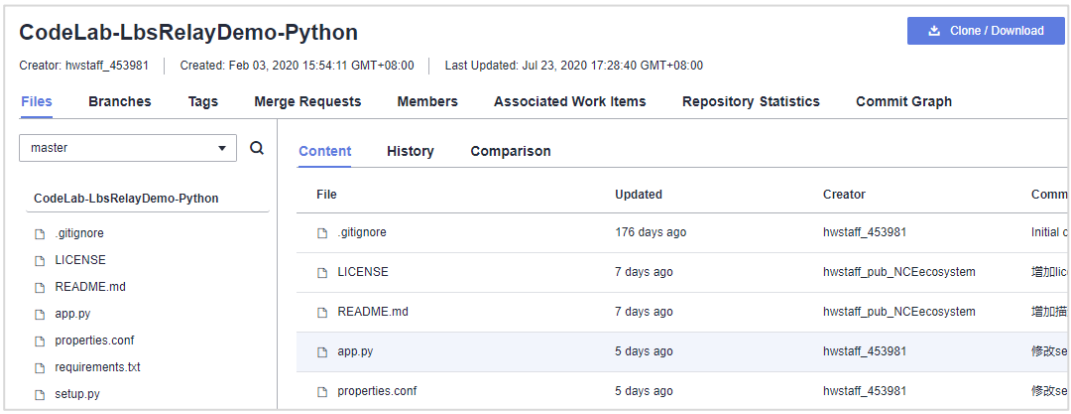
The experiment procedure is as follows:

1. Prepare the environment: Prepare an IDE and a lab sandbox environment.
2. Develop an LBS application: Compile code for the LBS application.
3. Verify the configuration: Interconnect the LBS application with iMaster NCE-Campus and verify the interconnection result.

2.3 Environment Preparations

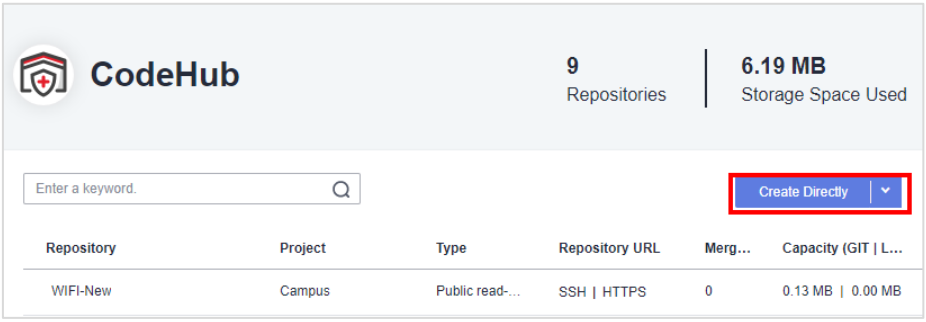
2.3.1 Preparing an IDE

Complete code files involved in this experiment have been uploaded to DevCloud on HUAWEI CLOUD. You can use Huawei CloudIDE for online programming or download the code files to your local PC from https://devcloud.cn-north-4.huaweicloud.com/codehub/project/68494a8ad06b4eea9a1c3f18be115161/codehub/577707/home.



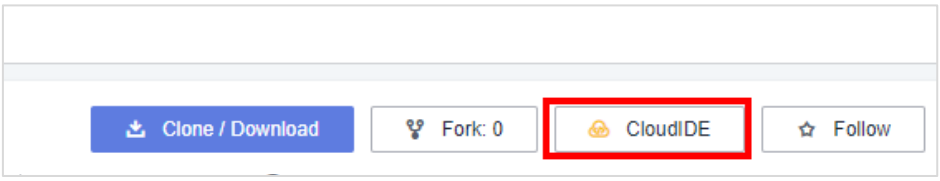
If not, you can also download codes from the following link: https://intl.devzone.huawei.com/en/enterprise/campus/sdkList.html.

1. Access the CodeHub console and create a project and a cloud-based code repository.

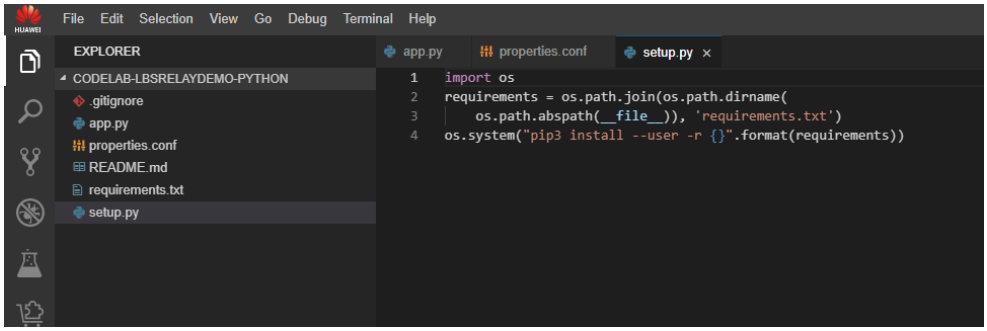


For more operations, see the CodeHub guide at https://support.huaweicloud.com/en-us/usermanual-cts/cts_1229.html

2. Folk complete code files to the cloud-based code repository and click **CloudIDE** to open the code files.



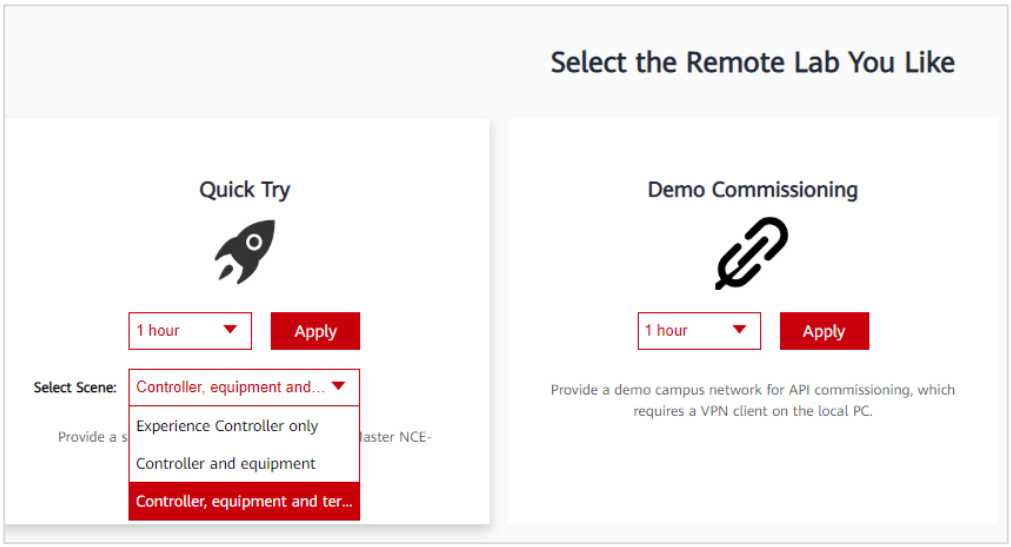
3. The CloudIDE window is displayed as follows.



2.3.2 Reserving a Sandbox Environment

The IDE developer community provides a sandbox environment for Huawei datacom solutions. In this experiment, you need to apply for a CloudCampus sandbox environment.

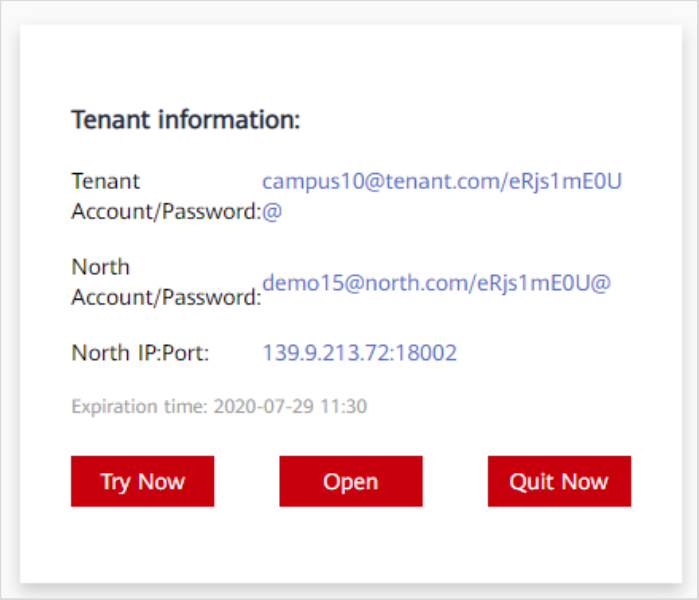
You can experience the sandbox environment in two ways: quick try and demo commissioning.



- Quick try: Lab access through a public network, which is applicable to code development in CloudIDE
- Demo commissioning: Lab access through a VPN, which is applicable to code development in the local IDE

To access the sandbox environment in quick experience mode, select **Controller and equipment** from the **Select Scene** drop-down list, select an experience

duration, and click **Apply**. Then, the information for logging in to the sandbox environment is provided.



Tenant Account/Password indicates the account and password for logging in to the iMaster NCE-Campus sandbox environment.

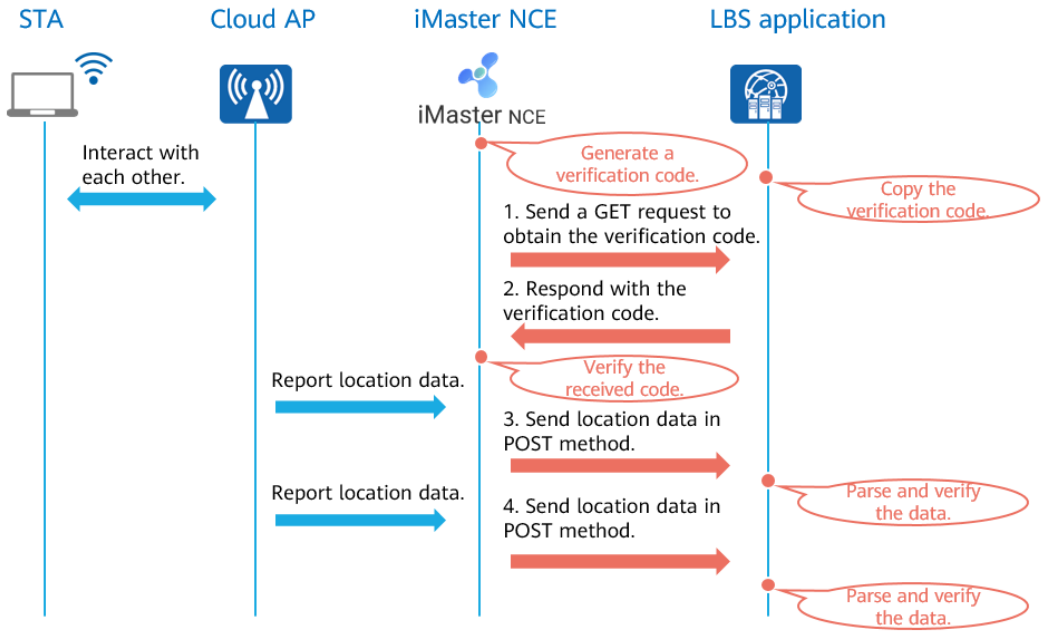
North Account/Password indicates the account and password used by the LBS application to interact with iMaster NCE-Campus, as shown in steps 1 to 3 in 1.2.1 Networking.

North IP:Port indicates the login address of iMaster NCE-Campus. You can click **Try Now** to go to the iMaster NCE login page.

So far, the environment is prepared.

2.4 LBS Application Development

This section describes how to develop an LBS application, how to call the LBS API provided by iMaster NCE-Campus to enable terminal location reporting, and how to call VAS APIs provided by iMaster NCE-Campus to query user information and traffic statistics.



The preceding flowchart shows the interaction between the LBS application and iMaster NCE-Campus.

- iMaster NCE-Campus generates a verification code. You need to copy this code to the LBS application.
- iMaster NCE-Campus sends a GET request to the LBS application to obtain a verification code. The verification succeeds if iMaster NCE receives a response that matches the verification code it sent to the LBS location.
- After the verification is successful, iMaster NCE-Campus sends terminal location data to the LBS application.
- The LBS application parses and verifies the data.

To develop an LBS application, perform the following operations:

1. Install iMaster NCE-Campus Python SDK.
2. Compile code for responding to the verification request sent from iMaster NCE-Campus.
3. Compile code for parsing terminal location data sent from iMaster NCE-Campus.
4. Compile code for querying user information and traffic statistics.

2.4.1 Installing the iMaster NCE-Campus SDK

The dependency package has been written into the **requirements.txt** file in the code repository. The file content is as follows:

```
certifi >= 14.05.14
six >= 1.10
python_dateutil >= 2.5.3
setuptools >= 21.0.0
```

```
urllib3 >= 1.15.1
flask >= 1.1.1
requests >=2.22.0

-e
git+https://codehub.devcloud.cn-north-4.huaweicloud.com/campus00001/CloudCampusPythonSDK.git@master#egg=cloudcampus
```

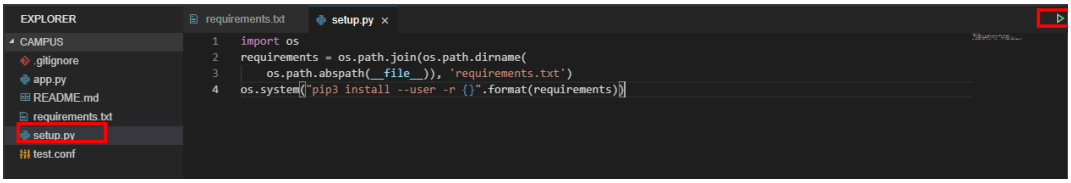
For more resources applicable to Huawei CloudCampus Solution, visit http://intl.devzone.huawei.com/en/apistudio/sample/campus/apiSdk.html?language=us.

The **setup.py** file contains the code for installing the iMaster NCE-Campus SDK.

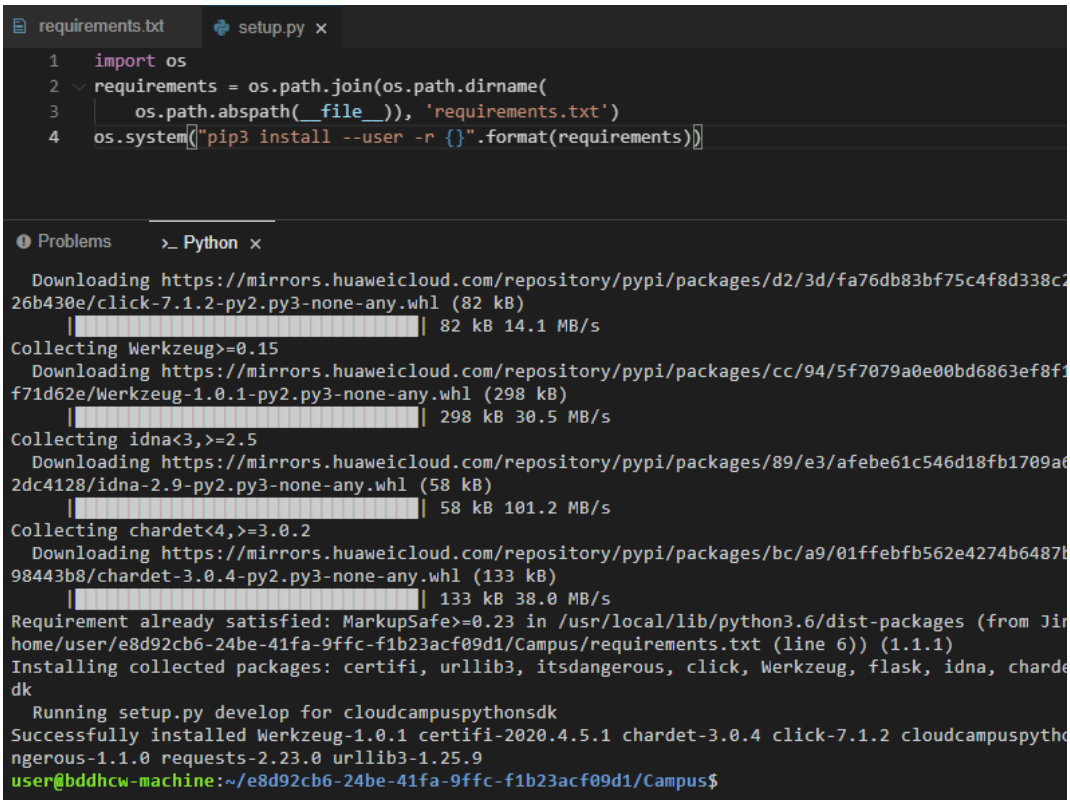
```
import os
requirements = os.path.join(os.path.dirname(
    os.path.abspath(__file__)), 'requirements.txt')
os.system("pip3 install --user -r {}".format(requirements))
```

Run **setup.py** to install the SDK.

Right-click **setup.py** in CloudIDE and choose **Run Python File in Terminal**. Alternatively, click the  button in the upper right corner of the CloudIDE window.



The installation succeeds.



For details about how to obtain the SDK, visit CloudCampus > Resource Download.

2.4.2 Compiling Code for Responding to a Verification Request

iMaster NCE-Campus sends a verification request to the LBS application, and the LBS application needs to send a correct response to iMaster NCE-Campus.

1. Create variables in the **properties.conf** file.

```
[LBS]
validator =
secret = Codelab
```

The value of **validator** is the verification code generated by iMaster NCE-Campus.

The value of **secret** is the password for interconnecting iMaster NCE-Campus with the LBS application. In this example, the value is **Codelab**.

2. Create a web-based LBS application. Add the following code to the **app.py** file. In this example, the application uses the Flask framework.

```
app = Flask(__name__)

@app.route('/', methods=['GET', 'POST'])
def main():
    #
    #

if __name__ == "__main__":
```

```
# Start the Flask app.
app.run(
        host = "0.0.0.0",
        port = 8899
    )
```

3. Create the ValidatorData class. Add the following code to the **app.py** file.

```
class ValidatorData:
# Save the verification code generated by iMaster NCE-Campus.
    validator = ""

    def __init__(self, validator):
        self.validator = validator
```

4. Configure the LBS application to respond to the verification request sent from iMaster NCE-Campus. Add the following code to the **app.py** file.

```
@app.route('/', methods=['GET', 'POST'])
def main():
# Read the properties.conf file.
    parser = configparser.ConfigParser()
    parser.read("properties.conf")
    if request.method == 'GET':
# Return the verification code saved in the file to interconnect with iMaster NCE-Campus.
        return json.dumps(ValidatorData(parser.get("LBS", "validator")).__dict__)
```

For more information about Flask, visit https://flask.palletsprojects.com/en/1.1.x/.

2.4.3 Compiling Code for Parsing Terminal Location Data

When receiving terminal location data sent by iMaster NCE-Campus in POST method, the LBS application stores and outputs the parsed data.

1. Format of terminal location data

   Terminal location data in the request body is in JSON format:

```
{
    "data": [
      {
        "apMac": "4C:FA:CA:D8:23:A0",
        "terminalList": [
         {
            "terminalMac": "88:19:08:F1:88:45",
            "rssi": -68,
            "timestamp": 1557460789000
          },
         {
           "terminalMac": "90:2E:1C:6A:2A:57",
           "rssi": -57,
           "timestamp": 1557460789000
          }
        ]
      }
    ],
    "secret": "Codelab",
```

```
    "type": "ApLocation"
}
```

JSON Field	Description
apMac	MAC address of the AP associated with the terminal.
terminalMac	MAC address of the terminal.
rssi	Strength (in dBm) of the Wi-Fi signals that the terminal receives from the AP.
timestamp	Timestamp when the AP collects terminal data.
secret	Credential, which is provided by the developer for data verification.
Type	Type of data to report. Currently, only **ApLocation** is supported.

For details about the interconnection fundamentals, visit CloudCampus > Study Tutorial.

2. Create a .csv file for saving terminal location data. Add the following code to the **app.py** file.

```
# Obtain the response to the POST request.
        body = request.get_json()
print(body)# dict type
# Create a .csv file of the current time.
        path = str(time.strftime("%Y%m%d-%H%M%S", time.localtime())) + ".csv"
        with open(path, 'a', newline='') as f:
            csv_write = csv.writer(f)
            csv_head = ["terminalMac", "rssi", "timestamp", "apMac", "secret", "type"]
            csv_write.writerow(csv_head)
```

3. Parse the data sent by iMaster NCE-Campus and save the data to a .csv file. Add the following code to the **app.py** file.

```
        path = str(time.strftime("%Y%m%d-%H%M%S", time.localtime())) + ".csv"
        with open(path, 'a', newline='') as f:
            csv_write = csv.writer(f)
# Generate column names.
            csv_head = ["terminalMac", "rssi", "timestamp", "apMac", "secret", "type"]
            csv_write.writerow(csv_head)
# Verify the secret field.
            secret = body['secret']
            if secret != parser.get("LBS", "secret"):
                return ""

            type = body['type']
            # print(type)

            data_list = body['data']
            # print(data_list)
```

```
# Cyclically parse JSON data.
                for data in data_list:
                    ap_mac = data['apMac']
                    # print(ap_mac)
                    terminal_list = data['terminalList']
                    # print(terminal_list)
                    for terminal in terminal_list:
                        rssi = terminal['rssi']
                        # print(rssi)
                        terminal_mac = terminal['terminalMac']
                        # print(terminal_mac)
                        timestamp = terminal['timestamp']
                        # print(timestamp)
                        csv_write.writerow([terminal_mac, rssi, timestamp, ap_mac, secret, type])
```

2.4.4 Compiling Code for Querying User Information and Traffic Statistics

iMaster NCE-Campus provides flexible VAS APIs for user profiling and analysis to ensure personalized Wi-Fi services for users. Access the VAS page of the IDN developer community to obtain more information.

1. Create an API client. Add the following code to the **app.py** file.

```
    # init api
    tenantName = 'demo15@north.com'
    tenantPwd = 'eRjs1mE0U@'
    host = '139.9.213.72'
    port = '18002'
    config = Configuration(host, port, tenantName, tenantPwd)
    api_client = ApiClient(config)
```

   Set **tenantName** to the northbound account in Reserving a Sandbox Environment, and set other parameters to the values set when reserving a sandbox environment.

2. Query a specific site. Add the following code to the **app.py** file.

   A site is a collection of network devices that can be managed and maintained in a unified manner.

   Set the site name based on the actual value in the sandbox environment. Log in to iMaster NCE-Campus as a tenant and check site information. In this example, you need to query the site named **site01**.

```
# Query the site ID based on the site name.
site_api = SiteManagerApi(api_client)
        page_index = 1
        page_size = 20
        name = 'site01'
        site_id = site_api.query_sites(page_index=page_index, page_size=page_size,
name=name).data[0].id
        print('siteId:   ' + site_id)
```

3. Query user information and traffic statistics at a specific site. Add the following code to the **app.py** file.

```
# Display user information and traffic statistics.
station_api = StationOpenApiApi(api_client)
        try:
            model =    station_api.query_site_station_info(site_id,1,20)
        except ValueError as e :
            print('EXCEPT :          ' + str(e))
        else:
            print('QUERY STATION INFO : SUCCESS')
            print('response body :          ' + str(model))
        finally:
            print('QUERY STATION INFO END')
```

2.5 Verification

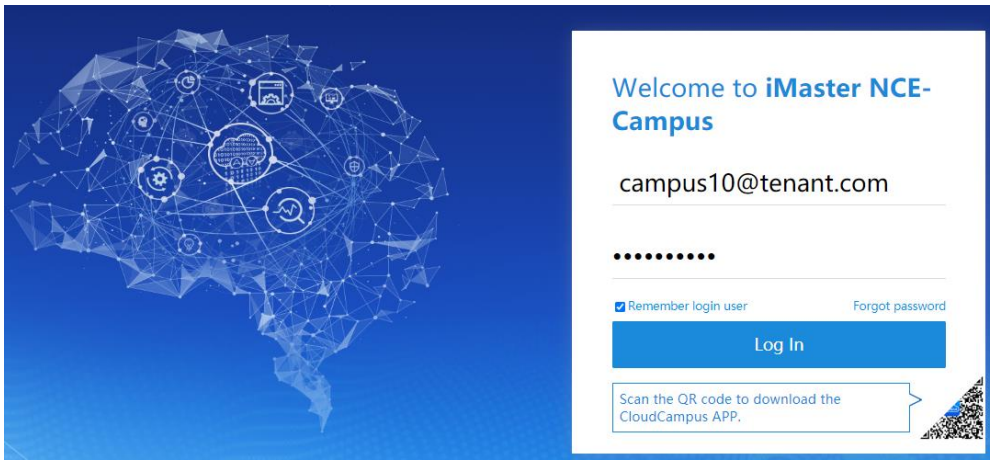
You need to verify whether the interconnection between iMaster NCE-Campus and the LBS application succeeds, whether terminal location data can be parsed and stored, and whether Wi-Fi user information and traffic statistics can be collected.

2.5.1 Configuring User Access

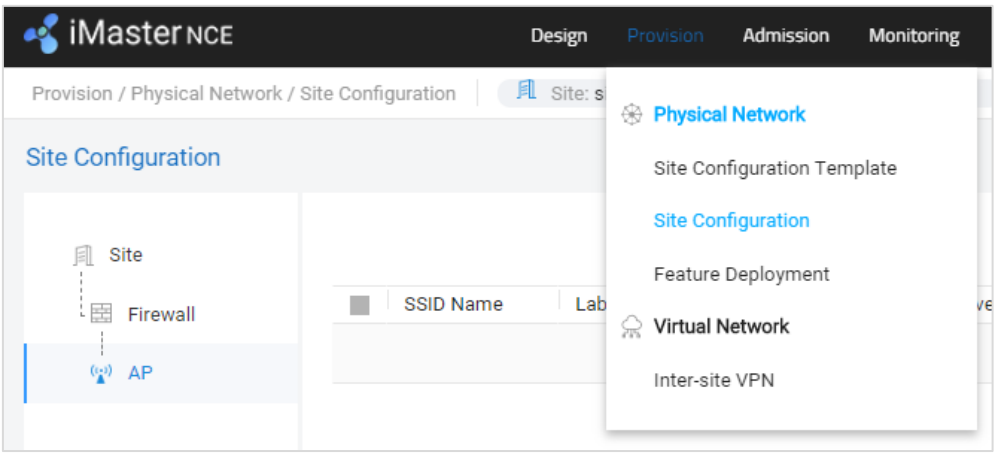
2.5.1.1 Configuring an SSID

1. Log in to iMaster-NCE-Campus as a tenant.

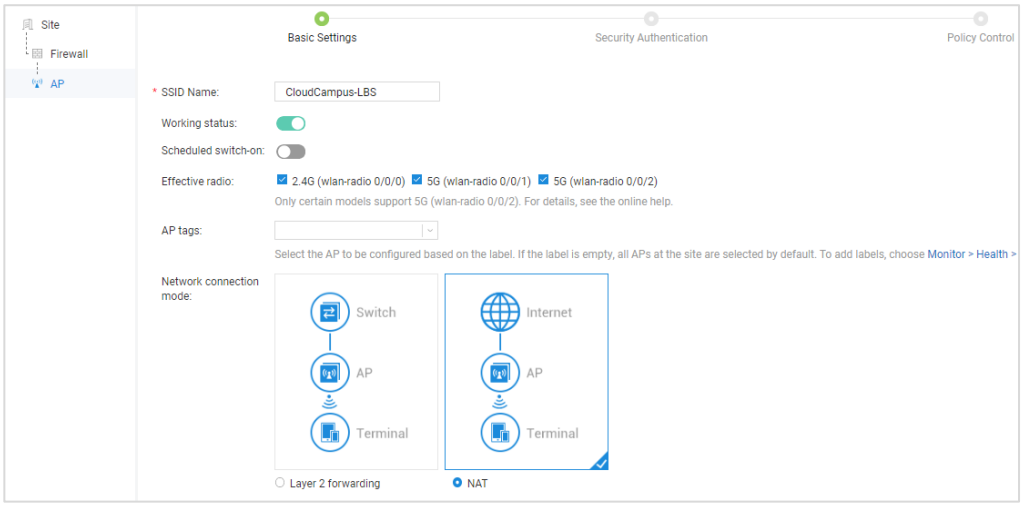
   For the login URL, see Reserving a Sandbox Environment.



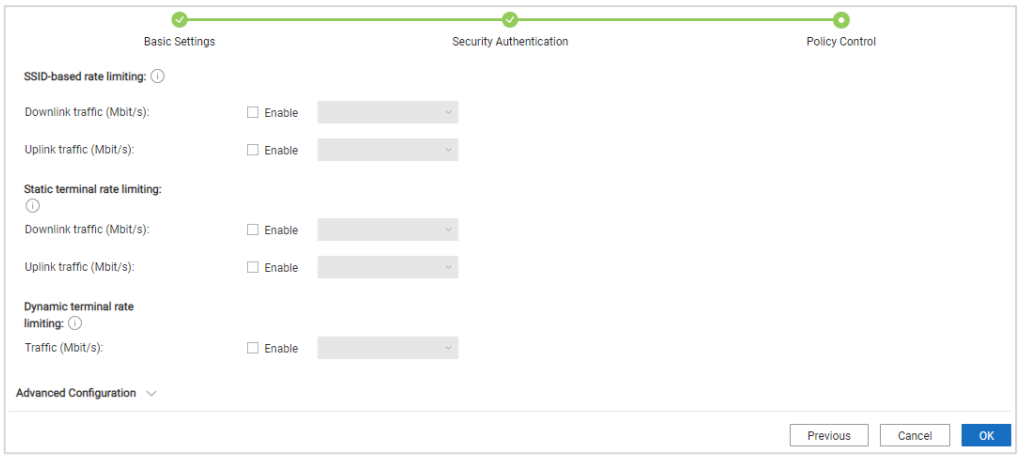
2. Choose **Provision** > **Physical Network** > **Site Configuration** from the main menu, and choose **AP** > **SSID** from the navigation pane.



3. Click **Create**, set the SSID name, set **Network connection mode** to **NAT**, and click **Next**.



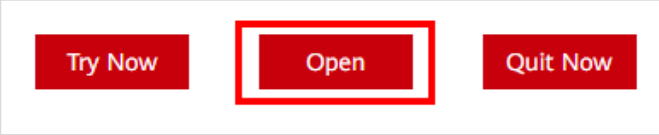
4. Retain the default security authentication settings and click **Next**. Then, retain the default policy control settings and click **OK**.



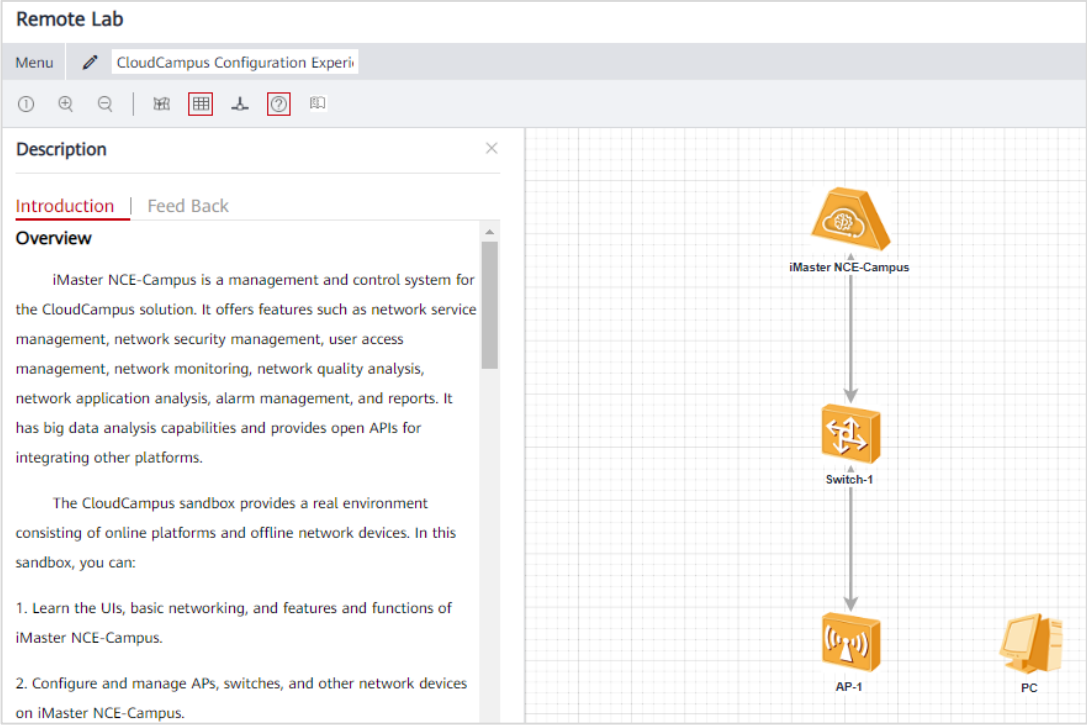
The SSID is configured. terminals can connect to this SSID to access the Internet.

2.5.1.2 Configuring Terminal Access

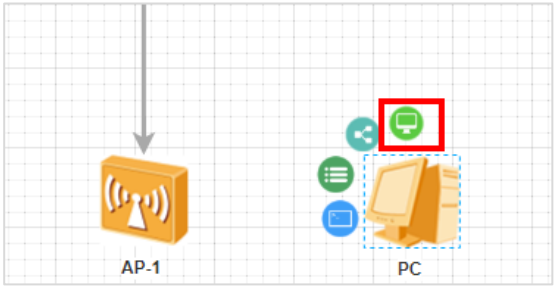
Click **Open** on the sandbox environment reservation success page.



The topology page is displayed.



Select a terminal to connect to the AP.



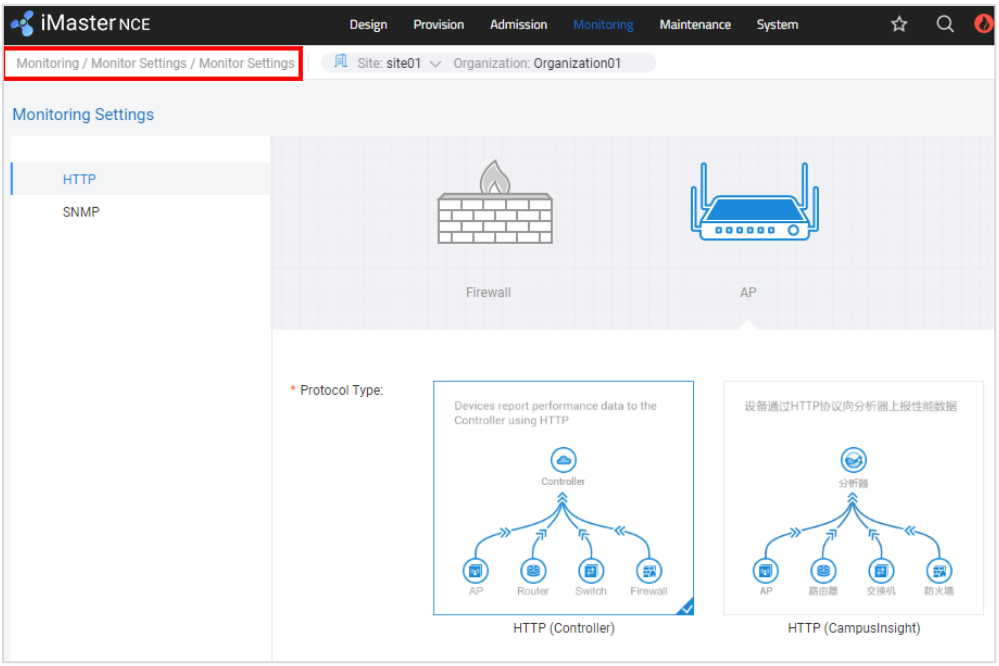
You can log in to the terminal interface to connect it to the configured SSID and then access the Internet.

2.5.2 Configuring Interconnection Between the LBS Application with iMaster NCE-Campus

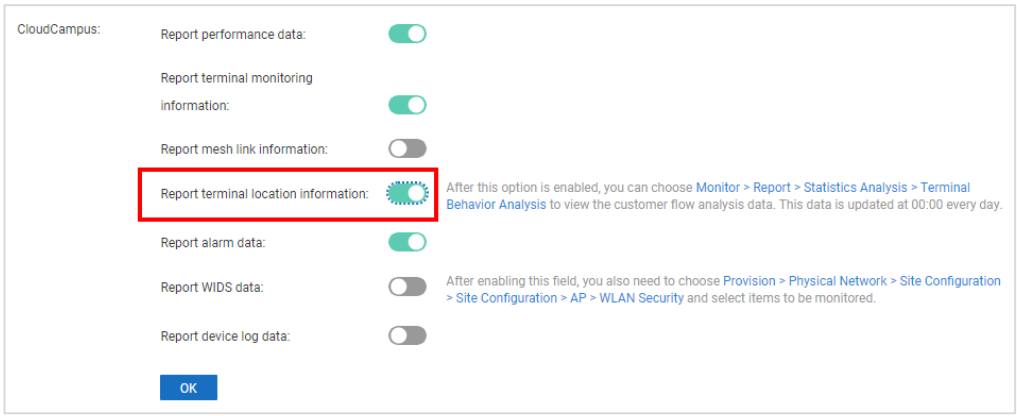
2.5.2.1 Enabling Terminal Location Data Reporting

Before interconnecting iMaster NCE-Campus with the LBS application, enable terminal location data reporting on iMaster NCE-Campus.

1. On iMaster NCE-Campus, choose **Monitoring** > **Monitor Settings** > **Monitor Settings** from the main menu, choose **HTTP** from the navigation pane, and click **AP**.



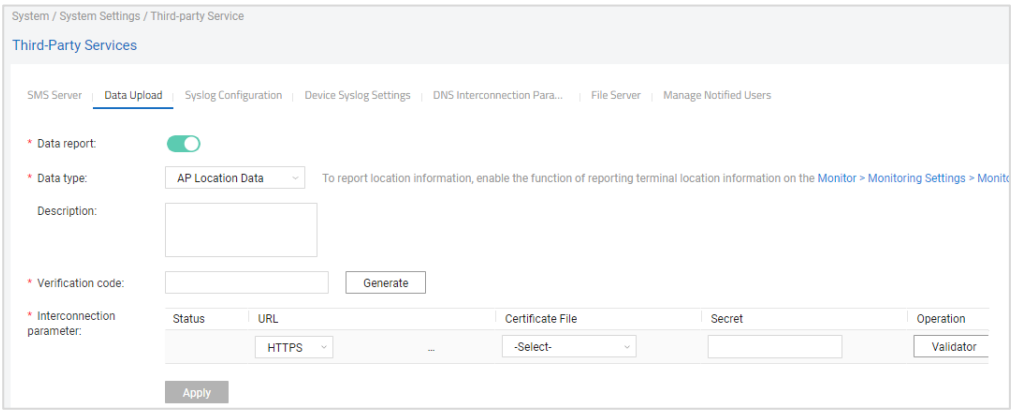
2. Enable **Report terminal location information** and click **OK**.



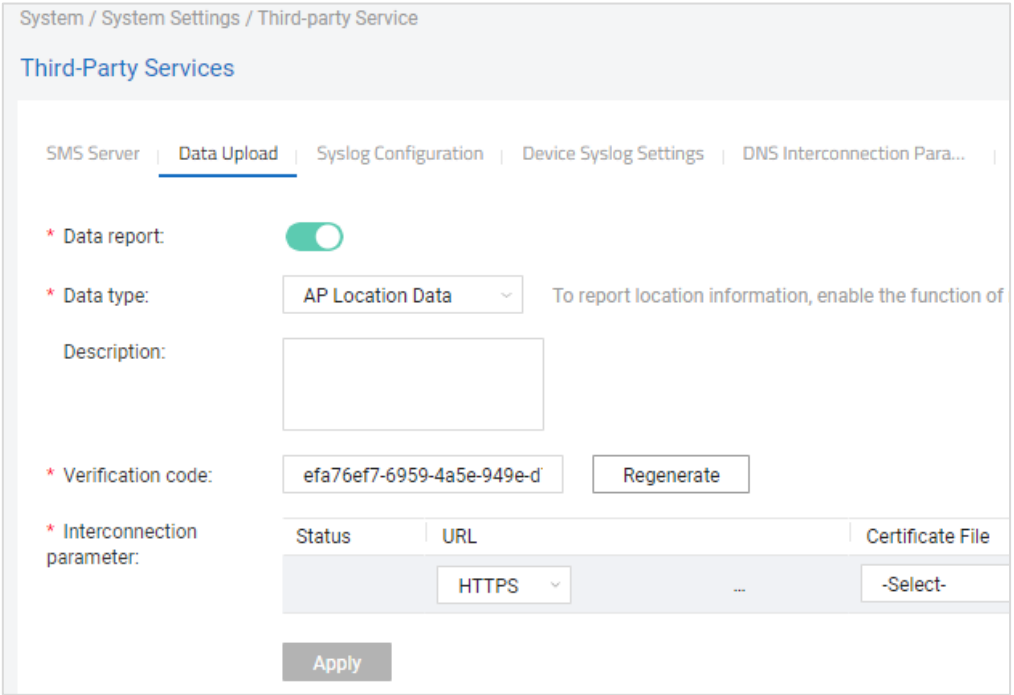
2.5.2.2 Configuring Interconnection Between the LBS Application with iMaster NCE-Campus and Starting the LBS Application

On iMaster NCE-Campus, choose **System** > **Third-Party Service** > **Data Upload** to configure interconnection with the LBS application.

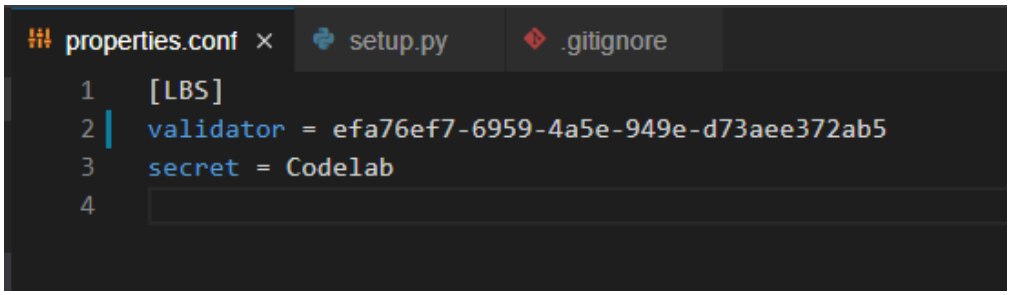
1. Set **Data type** to **AP Location Data**.



2. Click **Regenerate** next to **Verification code**. A verification code in UUID format is generated in the **Verification code** text box.

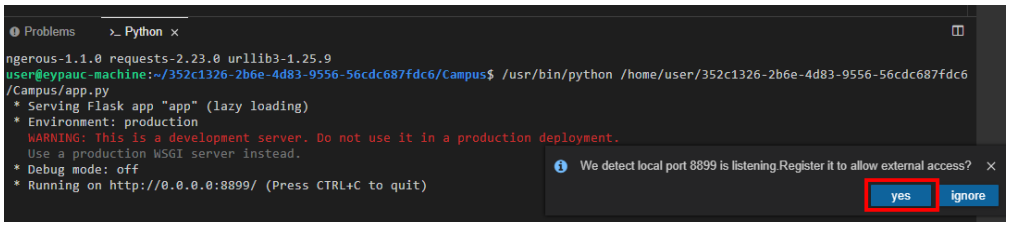


3. Copy the verification code to the **properties.conf** file in CloudIDE.



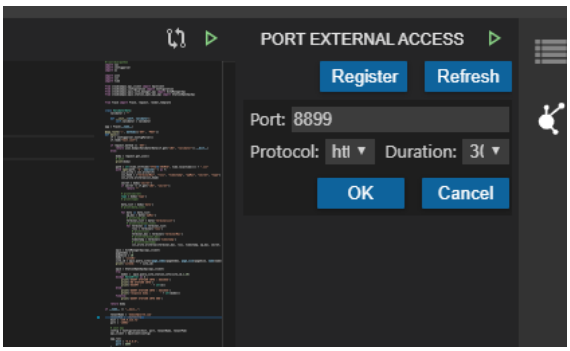
4. Start the LBS application.

   Run the **app.py** file in CloudIDE.

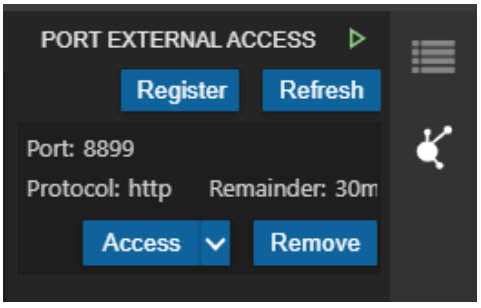


   The preceding command output shows that the Flask-based LBS application is running properly. To log in to the LBS application web page, you need to allow external access to this application. A dialog box will be displayed in CloudIDE, as shown in the preceding figure.

   Click **yes**. The port external access settings are displayed in the upper right corner of the CloudIDE window.

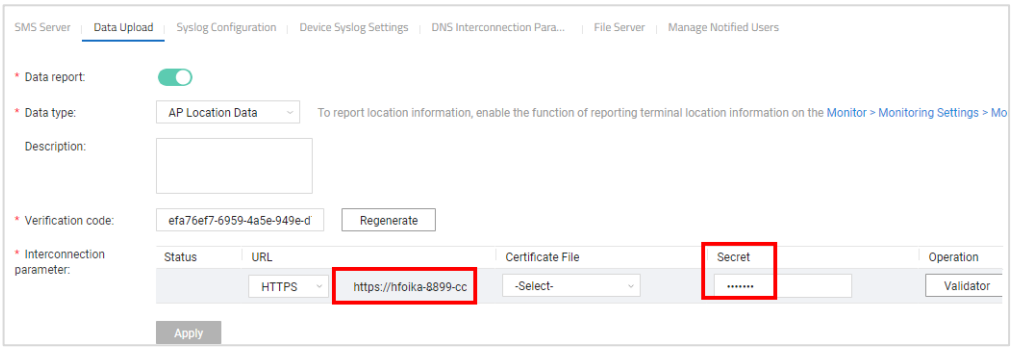


   Click **OK**. Then, the button will change to **Access**.

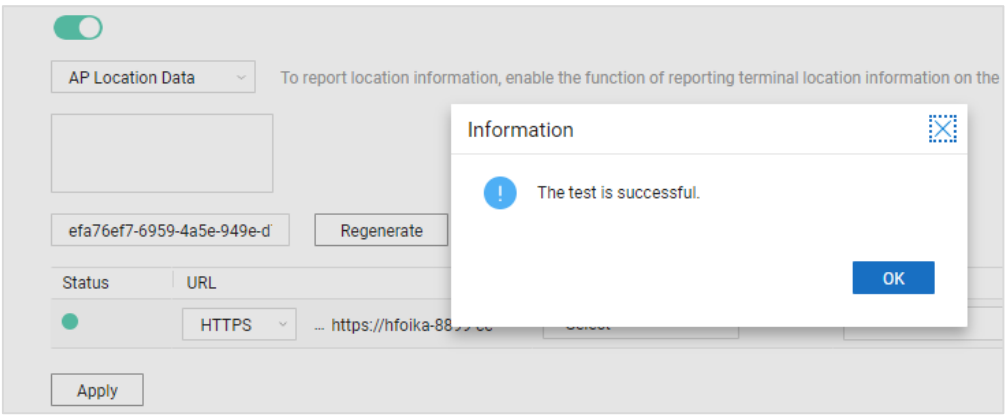


Click **Access**. The LBS application web page is displayed.

5. On iMaster NCE-Campus, copy the URL of the web page to the **URL** text box in the **Interconnection parameter** area, enter the password **Codelab** in the **Secret** text box, and click **Validator**.



If the test succeeds, the interconnection verification succeeds.



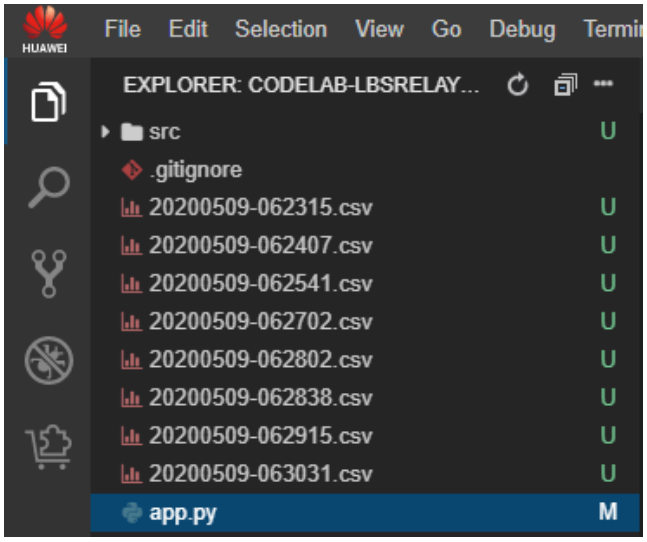
Click **Apply**.

2.5.3 Verifying Terminal Location Data

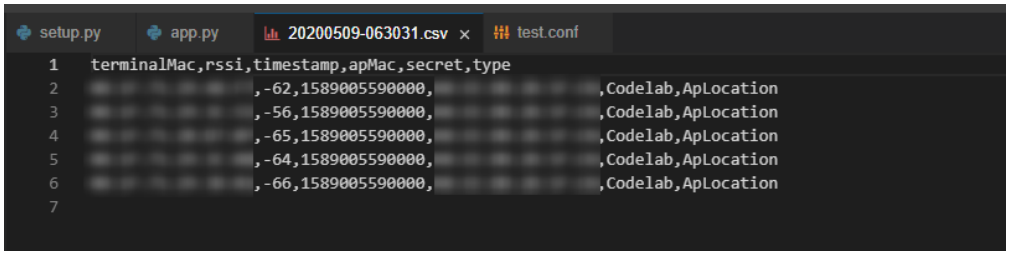
After iMaster NCE-Campus is interconnected with the LBS application successfully, iMaster NCE-Campus reports terminal locations identified by cloud APs at sites to the LBS application.

1. The LBS application parses the terminal location data.

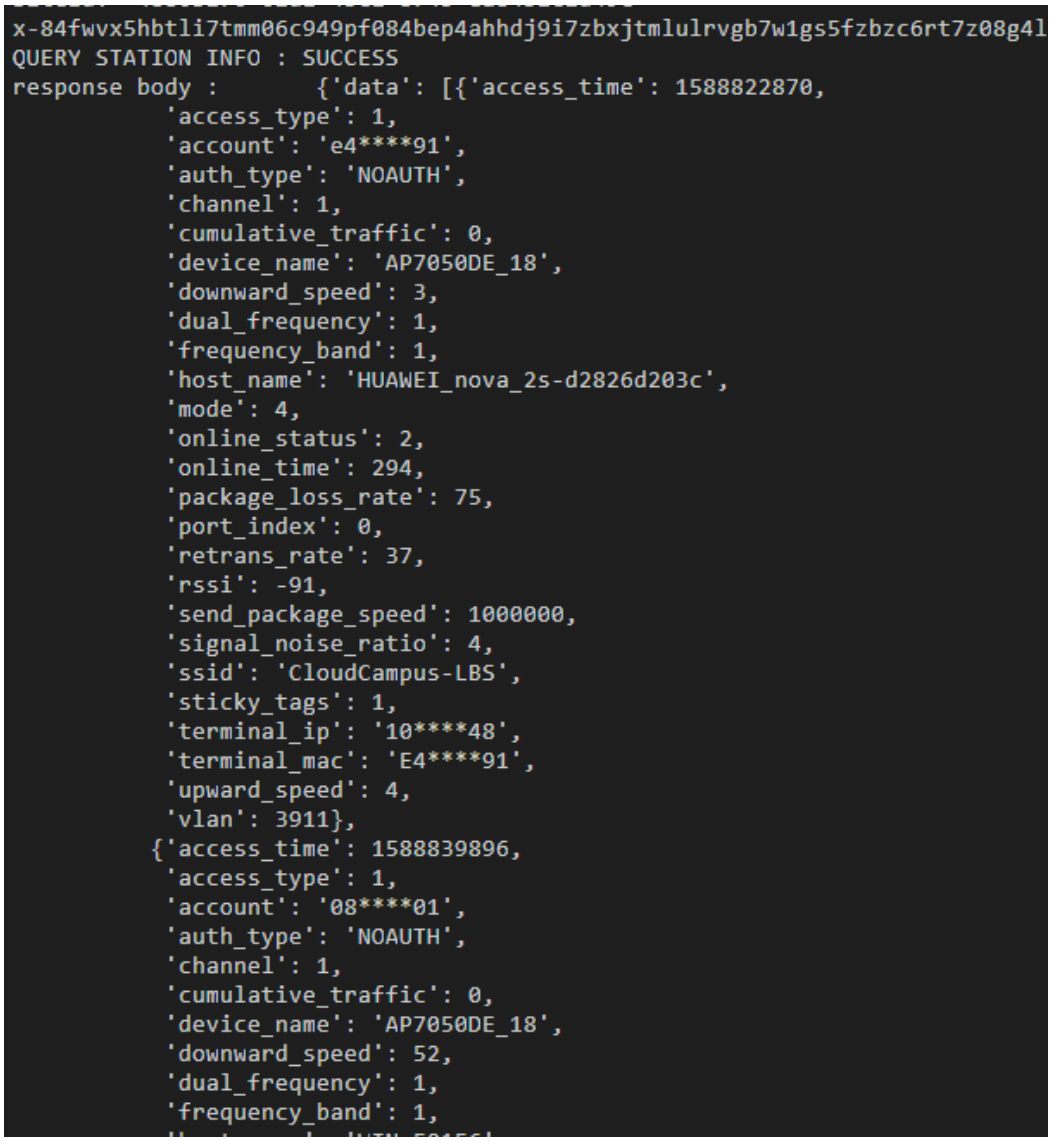
The LBS application parses the terminal location data, displays the data on the console, and saves it to .csv files.



The content of a .csv file is as follows.



2. Check user information and traffic statistics.



```
x-84fwvx5hbtli7tmm06c949pf084bep4ahhdj9i7zbxjtmlulrvgb7w1gs5fzbzc6rt7z08g4l
QUERY STATION INFO : SUCCESS
response body :       {'data': [{'access_time': 1588822870,
           'access_type': 1,
           'account': 'e4****91',
           'auth_type': 'NOAUTH',
           'channel': 1,
           'cumulative_traffic': 0,
           'device_name': 'AP7050DE_18',
           'downward_speed': 3,
           'dual_frequency': 1,
           'frequency_band': 1,
           'host_name': 'HUAWEI_nova_2s-d2826d203c',
           'mode': 4,
           'online_status': 2,
           'online_time': 294,
           'package_loss_rate': 75,
           'port_index': 0,
           'retrans_rate': 37,
           'rssi': -91,
           'send_package_speed': 1000000,
           'signal_noise_ratio': 4,
           'ssid': 'CloudCampus-LBS',
           'sticky_tags': 1,
           'terminal_ip': '10****48',
           'terminal_mac': 'E4****91',
           'upward_speed': 4,
           'vlan': 3911},
          {'access_time': 1588839896,
           'access_type': 1,
           'account': '08****01',
           'auth_type': 'NOAUTH',
           'channel': 1,
           'cumulative_traffic': 0,
           'device_name': 'AP7050DE_18',
           'downward_speed': 52,
           'dual_frequency': 1,
           'frequency_band': 1,
```

The terminal locating application practice is complete.

2.6 Quiz

What should I do after I obtain terminal location data?

3 Third-Party Authentication Practice in Huawei CloudCampus Solution

3.1 Background

Huawei CloudCampus Solution combines cutting-edge wired and wireless technologies with big data, AI, and cloud computing technologies to build service-centric campus networks that feature ubiquitous connections, worry-free services, and smooth evolution, enabling digital transformation in the industry.

This experiment guides you through the development of a third-party authentication application and the calling of third-party authentication APIs provided by iMaster NCE-Campus to implement web page redirection, obtain guest information, and manage guest login and logout. In this course, you will learn to:

- Configure interconnection with third-party authentication APIs provided by iMaster NCE-Campus.
- Call APIs to bring users online and offline.

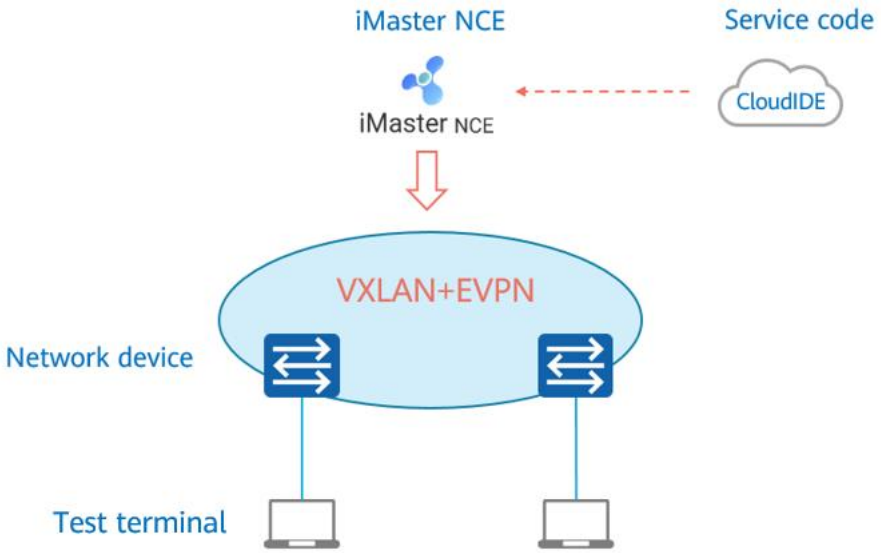
3.1.1 Introduction to Third-Party Authentication

Third-party authentication is mainly used in public places like shopping malls, hotels, airports, subways, and enterprises where guests access the Internet through Wi-Fi. It provides the following functions: guest authentication, promotion, recommendation, and marketing. Only authenticated guests can access the Internet through Wi-Fi.

In this experiment, you need to configure a Portal authentication page for guests and enable the third-party authentication application to call the third-party authentication APIs provided by iMaster NCE-Campus to implement services, including authentication, accounting, user analysis, and marketing.

3.2 Introduction

3.2.1 Networking



This lab network consists of four objects: a terminal, a Cloud AP, iMaster NCE-Campus, and a third-party authentication application. The IDE developer community can provide a sandbox environment with the terminal, Cloud AP, and iMaster NCE-Campus. You need to compile code for the third-party authentication application, which is recommended to be performed in CloudIDE.

3.2.2 Procedure

The experiment procedure is as follows:

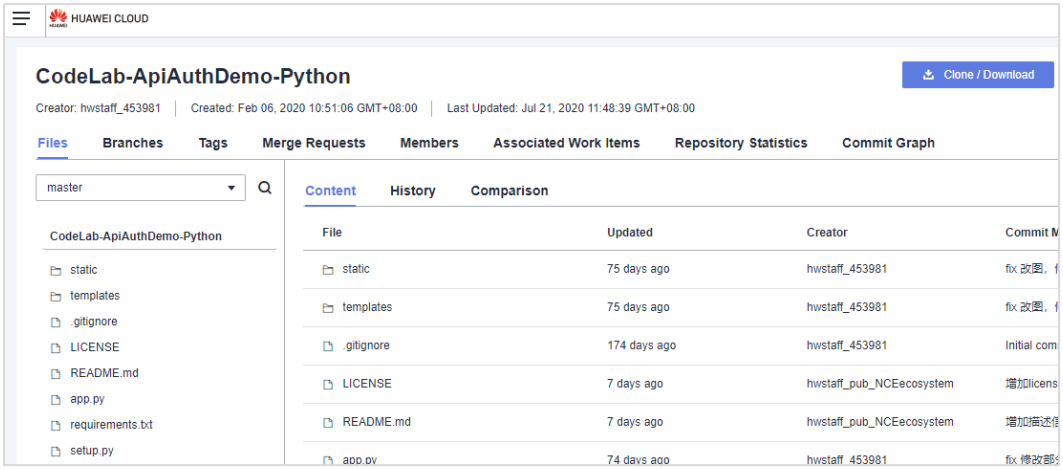
1. Prepare the environment: Prepare an IDE and a lab sandbox environment.
2. Develop a third-party authentication application: Use the Flask framework to develop a third-party authentication application.
3. Verify the configuration: Configure an SSID and user access to verify the configuration.

3.3 Environment Preparations

3.3.1 Preparing an IDE

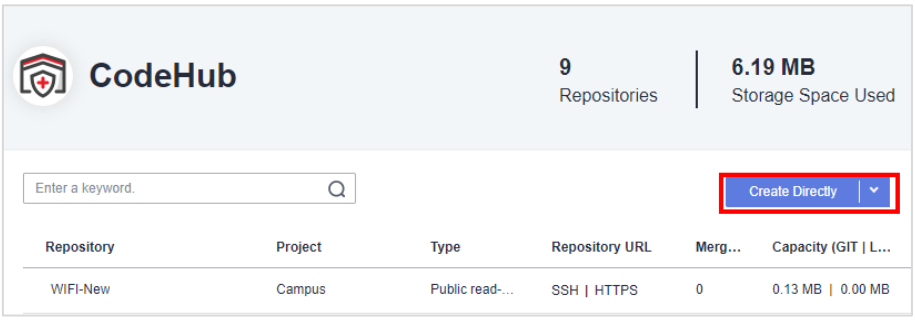
Complete code files involved in this experiment have been uploaded to DevCloud on HUAWEI CLOUD. You can use Huawei CloudIDE for online programming or download the code files to your local PC from

https://devcloud.cn-north-4.huaweicloud.com/codehub/project/68494a8ad06b4eea9a1c3f18be115161/codehub/578588/home.



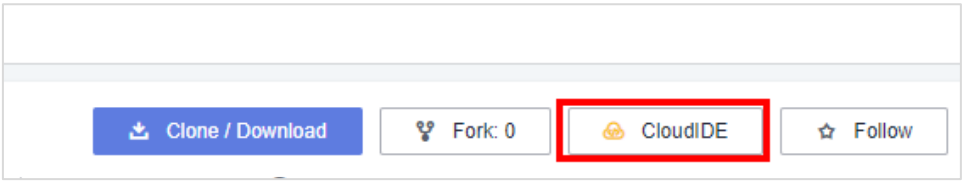
If not, you can also download codes from the following link: https://intl.devzone.huawei.com/en/enterprise/campus/sdkList.html.

1. Access the CodeHub console and create a project and a cloud-based code repository.

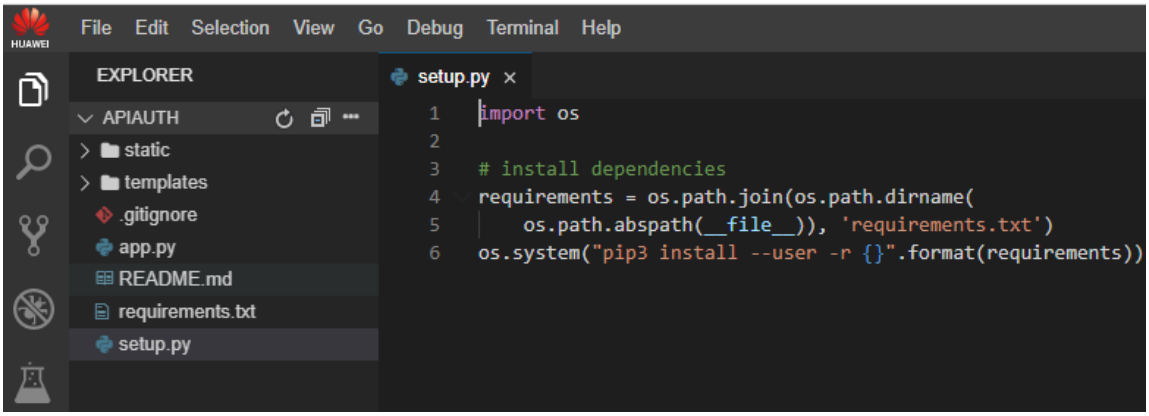


For more operations, see the CodeHub guide at https://support.huaweicloud.com/en-us/usermanual-cts/cts_1229.html.

2. Folk complete code files to the cloud-based code repository and click **CloudIDE** to open the code files.



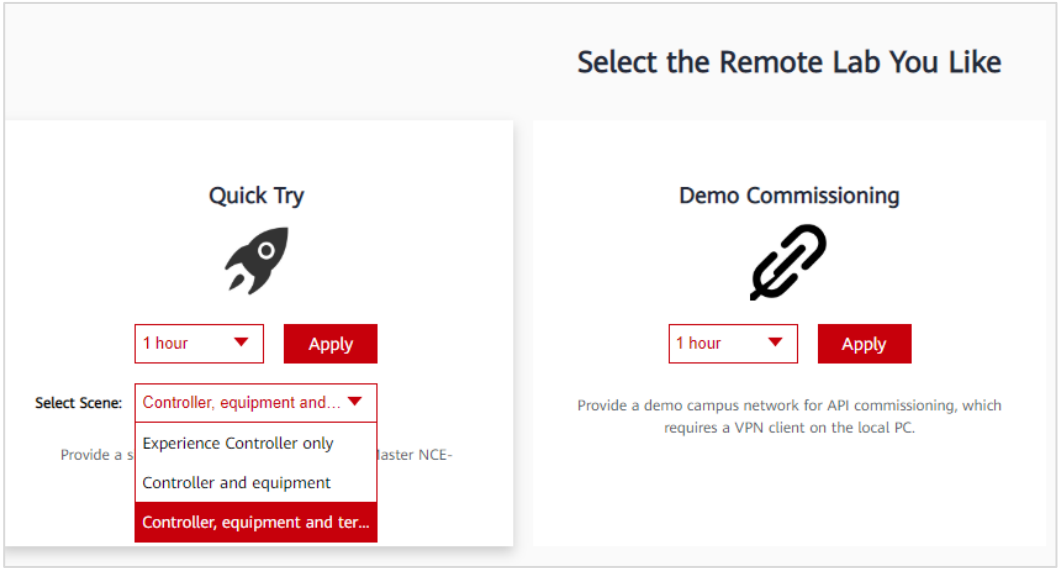
3. The CloudIDE window is displayed as follows.



3.3.2 Reserving a Sandbox Environment

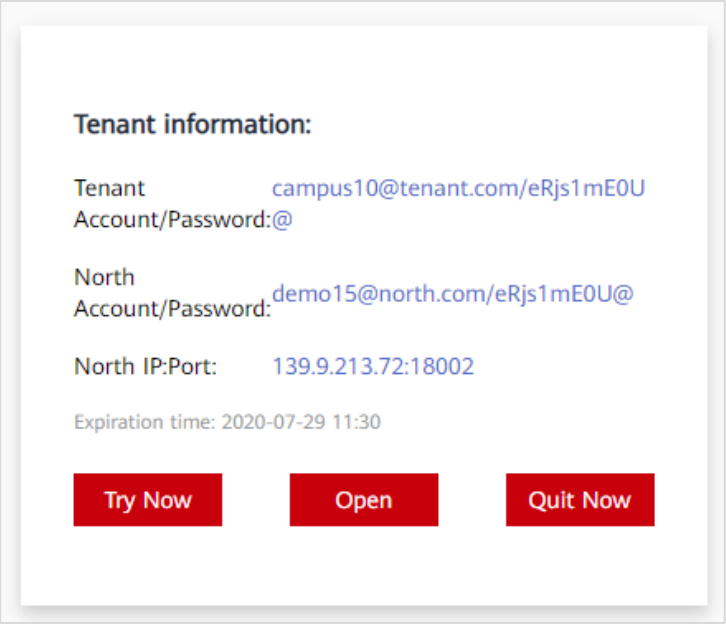
The IDE developer community provides a sandbox environment for Huawei datacom solutions. In this experiment, you need to apply for a CloudCampus sandbox environment.

You can experience the sandbox environment in two ways: quick try and demo commissioning.



- Quick try: Lab access through a public network, which is applicable to code development in CloudIDE
- Demo commissioning: Lab access through a VPN, which is applicable to code development in the local IDE

To access the sandbox environment in quick experience mode, select **Controller and equipment** from the **Select Scene** drop-down list, select an experience duration, and click **Apply**. Then, the information for logging in to the sandbox environment is provided.



Tenant Account/Password indicates the account and password for logging in to the iMaster NCE-Campus sandbox environment.

North Account/Password indicates the account used by the third-party authentication application to interact with iMaster NCE-Campus, as shown in steps 1 to 3 in 1.2.1 Networking.

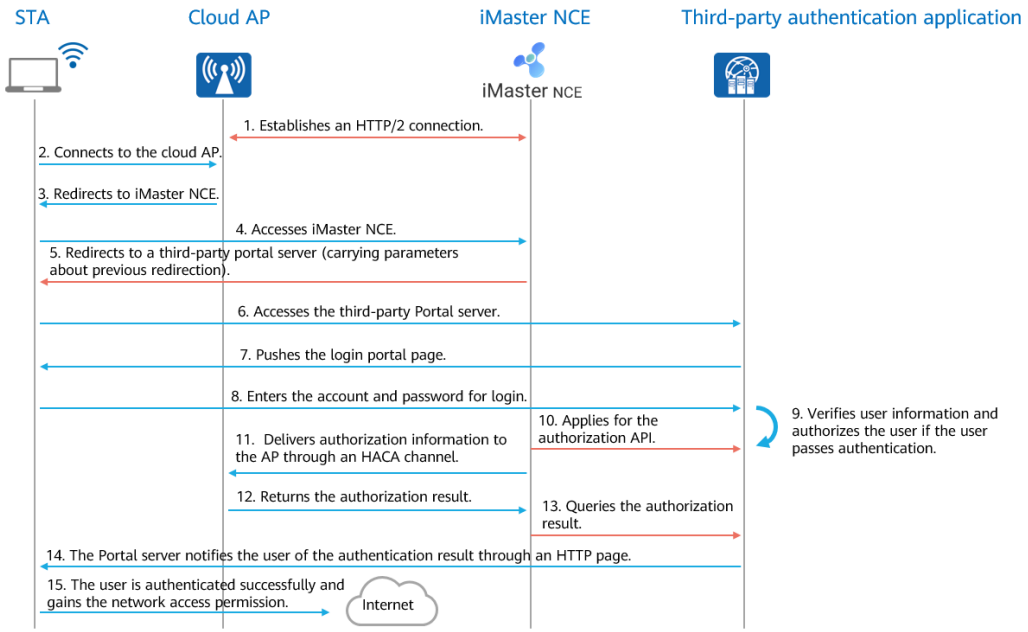
North IP:Port indicates the login address of iMaster NCE-Campus. You can click **Try Now** to go to the iMaster NCE login page.

So far, the environment is prepared.

3.4 Third-Party Authentication Application Development

This section describes how to develop a third-party authentication application demo and how to interconnect iMaster NCE-Campus with this application to perform Portal authentication for wireless users.

The interaction between each component in this experiment is as follows.



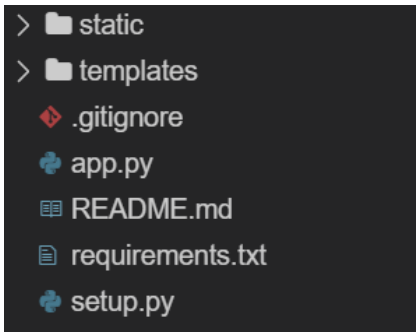
The third-party authentication application needs to provide the following functions:

- Acts as a Portal server to push authentication pages to users, receives user login information, and returns a login result to users.
- Parses, stores, and verifies user information.
- Calls APIs provided by iMaster NCE-Campus to authorize terminals.

3.4.1 Code Directory Structure

The demo in this experiment is developed based on the Python Flask framework.

The code structure is as follows.



- The **static** directory stores images and CSS resources.
- The **templates** directory stores HTML pages.
- The **app.py** file defines the main logic of the third-party authentication application.
- The **setup.py** file contains the dependency package for installing the third-party authentication application.

- The **requirements.txt** file stores the path of the dependency package required in this experiment.

3.4.2 Installing the iMaster NCE-Campus SDK

The dependency package has been written into the **requirements.txt** file in the code repository. The file content is as follows:

```
certifi >= 14.05.14
six >= 1.10
python_dateutil >= 2.5.3
setuptools >= 21.0.0
urllib3 >= 1.15.1
flask >= 1.1.1
requests >=2.22.0

-e
git+https://codehub.devcloud.cn-north-4.huaweicloud.com/campus00001/CloudCampusPythonSDK.git@master#egg=cloudcampus
```

For more resources applicable to Huawei CloudCampus Solution, visit http://intl.devzone.huawei.com/en/apistudio/sample/campus/apiSdk.html?language=us.

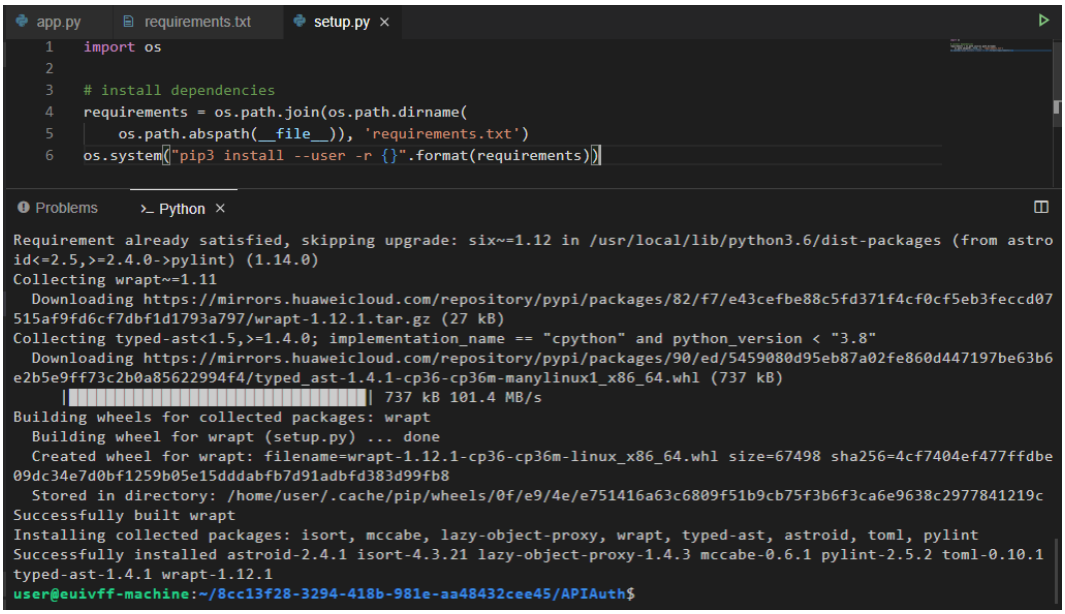
The **setup.py** file contains the code for installing the iMaster NCE-Campus SDK. To download the SDK to your local PC, visit CloudCampus > Resource Download.

```
import os
requirements = os.path.join(os.path.dirname(
    os.path.abspath(__file__)), 'requirements.txt')
os.system("pip3 install --user -r {}".format(requirements))
```

Run **setup.py** to install the SDK.

Right-click **setup.py** in CloudIDE and choose **Run Python File in Terminal**.

Alternatively, click the  button in the right upper corner.



The installation succeeds.

3.4.3 Initializing the API Client

Use the APIClient class to initialize your SDK. You need to configure the port number (18002 in this example) used by northbound APIs, IP address of iMaster NCE-Campus, and northbound account and password.

```
if __name__ == "__main__":

    # init api
    tenantName = ' demo15@north.com'
    tenantPwd = eRjs1mE0U@''
    host = '139.9.213.72'
    port = '18002'
    config = Configuration(host, port, tenantName, tenantPwd)
    api_client = ApiClient(config)
```

The information must be the same as that in the sandbox environment.

3.4.4 Parsing User Information

Configure the Flask application and redirect user requests to this application. In this case, end user information carried in the requests can be parsed.

1. Configure and start the Flask application. Add the following code to the **app.py** file.

```
app = Flask(__name__)

if __name__ == "__main__":

# Start the Flask application.
    app.run(
        host="0.0.0.0",
```

```
        port=8899
    )
```

2. Parse information in user requests. Add the following code to the **app.py** file.

```
@app.route('/', methods=['GET', 'POST']) # Indicates the login main page and HTTP method.
def index():

# Parse the parameters used to call the API.
    uaddress = request.args.get('uaddress')
    umac = request.args.get('umac')
    ssid = request.args.get('ssid')
    apmac = request.args.get('apmac')
    node_ip = request.args.get('nodeIp')
```

The parameters from top to bottom are the terminal IP address, terminal MAC address, SSID to which the terminal connects, MAC address of the AP to which the terminal connects, and IP address of iMaster NCE-Campus.

For detailed information about the parameters, see the Developer Guide.

3.4.5 Bringing a User Online

1. Check the HTTP method on the main page.

   If the GET method is used, the login.html page is returned. If the POST method is used, the authorization process starts.

```
if request.method == 'GET':
return render_template(peoplelogin.htmlpeople) # If the HTTP method is GET, the login.html page is returned.
    else:
        username = request.form.get('username')
        # print(username)
        password = request.form.get('password')
        # print(password)
# Account and password set in this experiment
        if username == '123' and password == '123':
# Authorize the user...
```

   - If the GET method is used on the main page, the **login.html** page in the **template** directory is returned to the user.
   - If the POST method is used and the input account and password are both **123**, the user authorization process starts.

   The default account and password used in this experiment are both **123**. Developers can change the account and password or connect to the database which stores accounts and passwords.

2. Call the API for authorizing end users and send user information to the success page. The user information will be used again when the user is brought offline. Add the following code to the **app.py** file.

```
# Obtain the API for authorizing end users.
    client_user_manager_api = ClientUserManagerApi(api_client)

    try:
# Create user information.
```

```
            user_dto = UserAuthorizationInputDto(device_mac=apmac, terminal_ip_v4=uaddress,
terminal_mac=umac, ssid=ssid, node_ip=node_ip, user_name=username)
        except ValueError as e:
# Capture exceptions.
            print('EXCEPT :     ' + str(e))
            return render_template('login.html', err_msg = 'request parameter')
# Call the user authorization API and print the result.
        auth_output = client_user_manager_api.user_authorization(user_dto)
        print(auth_output)
```

Configure user information in **user_dto**. In this example, this field corresponds to **UserAuthorizationInputDto** in the HTTP request body for end user authorization. For parameters that need to be carried in **UserAuthorizationInputDto**, see the API documentation. The sample code is provided under **API List** > **End User Management** > **Authorize User**.

3. Call the API for querying the authorization result, verify that the authorization succeeds, and redirect the user to the success page. Add the following code to the **app.py** file.

```
        time.sleep(3)    # Wait for 3 seconds and then query the authorization result.
        query_auth_output =
client_user_manager_api.get_authorizationresult(auth_output.psessionid,node_ip)
print(query_auth_output)

# If the query fails, an error message is returned.
        if query_auth_output.errmsg != 'true':
            return render_template('login.html', err_msg = 'query auth')

# If the query is successful, the user is redirected to the success page.
        return redirect(url_for('success', uaddress = uaddress, umac=umac, ssid=ssid, apmac=apmac,
nodeIp=node_ip, psessionid=auth_output.psessionid, username=username))
```

For details about how to call the API for querying the authorization result, see the API documentation.

If the query is successful, the user will be redirected to the success page.

4. Configure the success page.

   The user request sent to the success page contains parameters that specify user information.

```
@app.route('/success', methods=['GET', 'POST'])
def success():
    if request.method == 'GET':
        data = request.url.lstrip(request.base_url)
        print(data)
# The parameters are reserved and will be used again when the user is brought offline.
        return render_template('success.html', querystring = '/logout' + data)
```

The **success.html** file in the **template** directory is returned to the user.

3.4.6 Bringing a User Offline

The API for bring a user offline is called when a user goes offline.

1. Parse the user data stored in the cookie. Add the following code to the **app.py** file.

```
@app.route('/logout', methods=['GET', 'POST'])
def logout():
# Parse the parameters carried in the request.
    uaddress = request.args.get('uaddress')
    umac = request.args.get('umac')
    apmac = request.args.get('apmac')
    node_ip = request.args.get('nodeIp')
    psessionid = request.args.get('psessionid')
    username = request.args.get('username')
    ssid = request.args.get('ssid')
```

2. Call the API for bringing a user offline and redirect the user to the login page. Add the following code to the **app.py** file.

```
# Obtain the API for bring offline a user.
    client_user_manager_api = ClientUserManagerApi(api_client)

# Force a user offline.
    third_user_info = ThirdUserInfoData(device_mac=apmac, terminal_ip_v4=uaddress,
terminal_mac=umac, node_ip=node_ip, psessionid=psessionid, user_name=username)
    print(third_user_info)
    model = client_user_manager_api.cut_user(CutUserInputDto([third_user_info]))

    print(model)
# Redirect the user to the login page. User parameters are sent to the login page as well.
    return redirect(url_for('index', uaddress = uaddress,
umac=umac,ssid=ssid,apmac=apmac,nodeIp=node_ip))
```

ClientUserManager provides a northbound API for forcing users offline. The cut_user method is used. For details, see the API documentation.

3.5 Verification

To verify whether third-party authentication can be performed successfully, start the developed third-party authentication application, configure Portal authentication in API mode on iMaster NCE-Campus, and then check whether terminals can be authenticated and access the network in wireless mode.

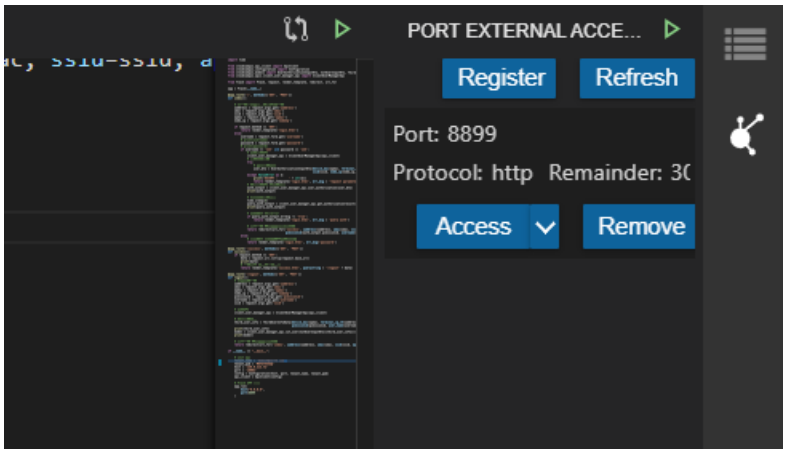
3.5.1 Starting the Third-Party Authentication Application

1. Start the server.

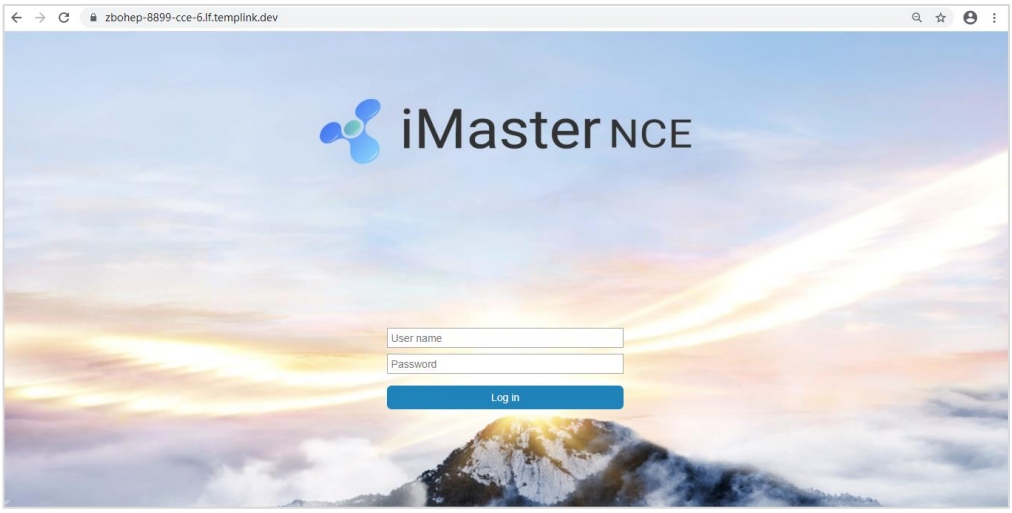
   Run the **app.py** file in CloudIDE.

2. Enable external access to the third-party authentication application. This operation is not required for debugging in the local IDE.

   Click **yes** in the preceding dialog box. The **PORT EXTERNAL ACCESS** page is displayed in the upper right corner of the window. Click **Access**. The login Portal page is displayed. The default registration port number is 8899.



3. Check the login Portal page.



   In this example, the page URL is https://zbohep-8899-cce-6.lf.templink.dev/.

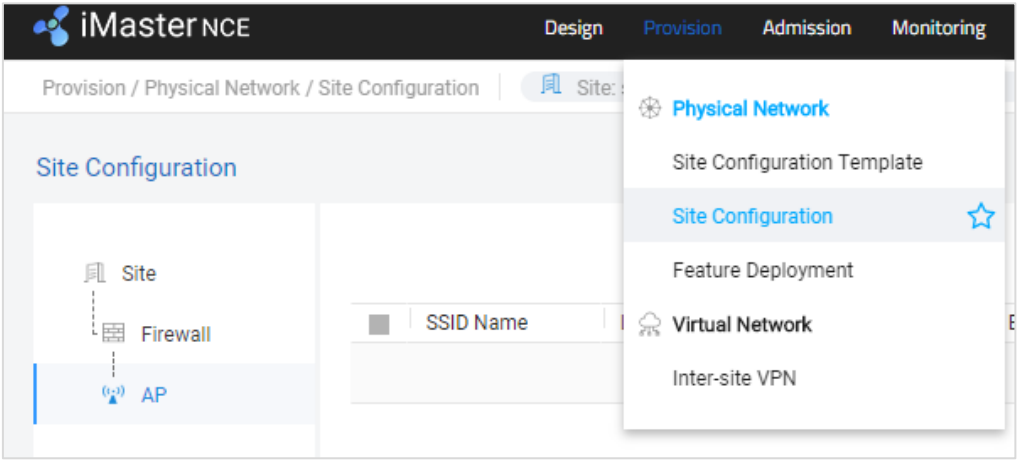
3.5.2 Configuring Interconnection Between iMaster NCE-Campus and the Third-Party Portal Authentication Application

Configure interconnection with the third-party Portal authentication application on iMaster NCE-Campus.

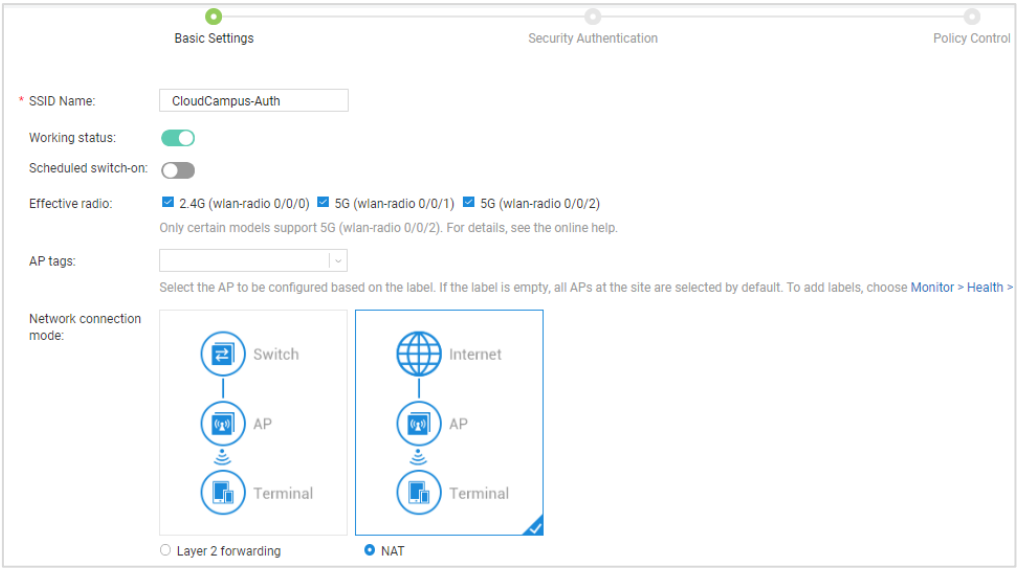
3.5.2.1 Configuring an SSID

1. Create an SSID.

Log in to iMaster NCE-Campus using the account and password in the sandbox information, choose **Provision** > **Physical Network > Site Configuration** from the main menu, and choose **AP** > **SSID** from the navigation pane.

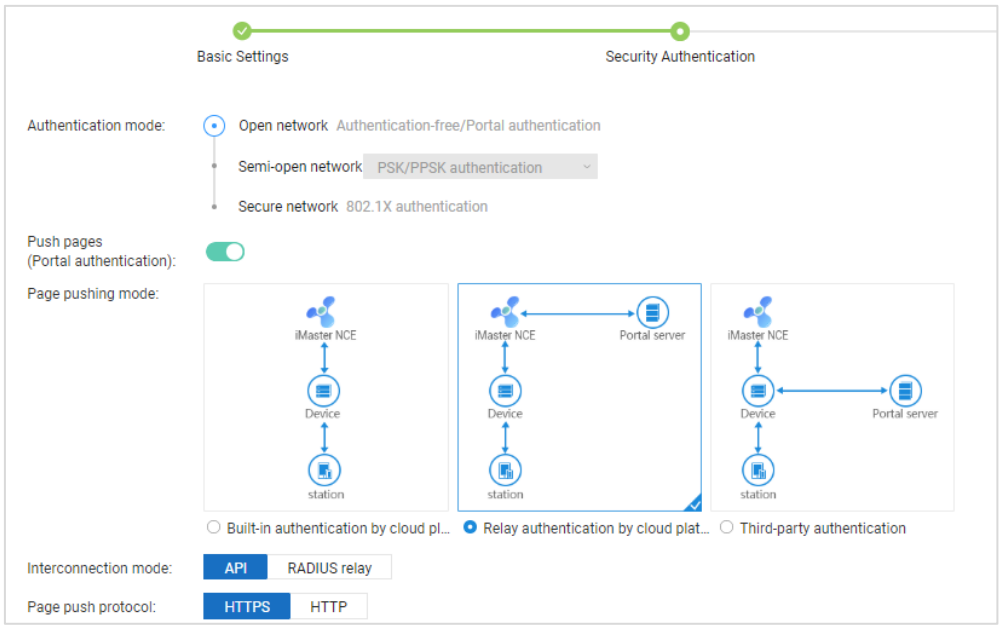


2. Click **Create**, enter the SSID name, set **Network connection mode** to **NAT**, and click **Next**.



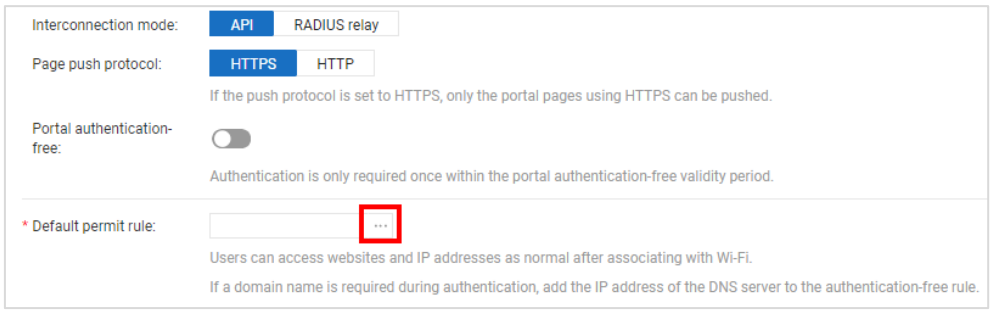
3. Configure the authentication mode used when end users access the network by connecting to the SSID.

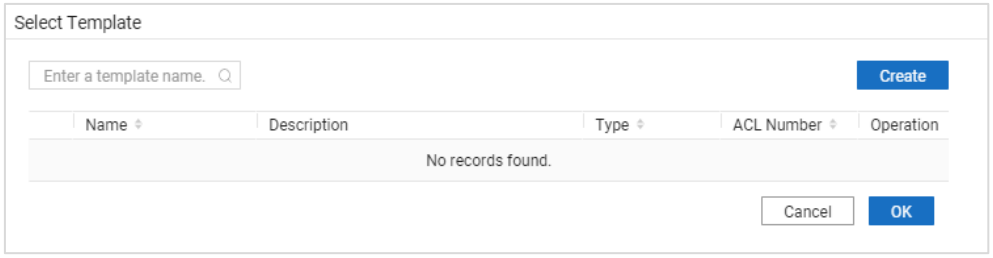
   Set **Authentication mode** to **Open network**, enable **Push pages (Portal authentication)**, set **Page pushing mode** to **Relay authentication by cloud platform**, and set **Interconnection mode** to **API**.



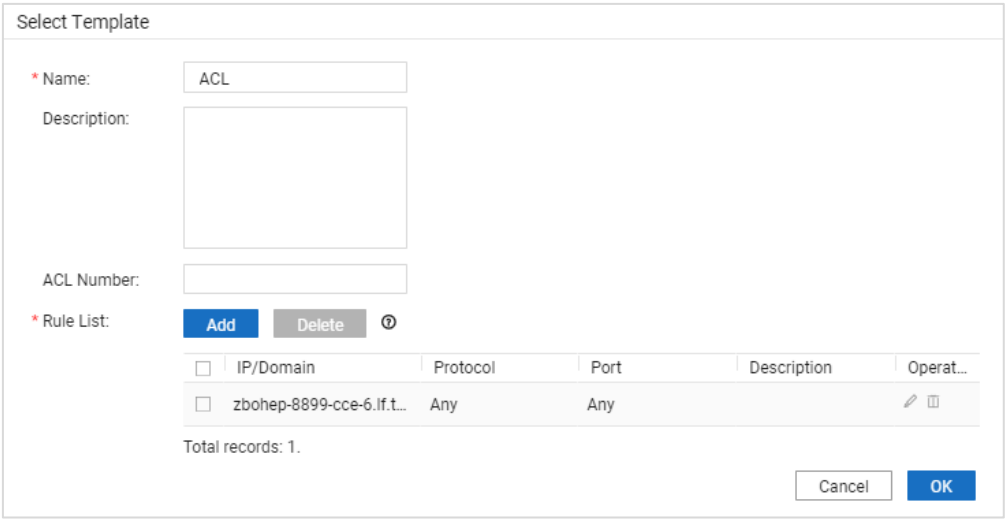
4. Configure a default permit rule.

   Click ... next to **Default permit rule**, and click **Create** in the displayed **Select Template** dialog box to create an ACL template.





   Copy the domain name obtained in 3.5.1 Starting the Third-Party Authentication Application and add the domain name to the rule list. Do not enter **http** or **https** in the rule.



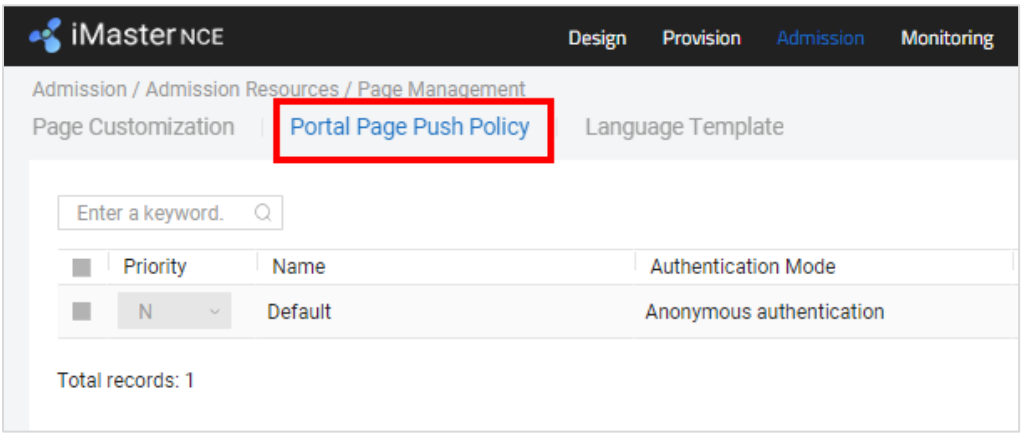
Finally Select a bypass policy, click **Next**, and click **OK**.

3.5.2.2 Configuring a Portal Page Push Policy

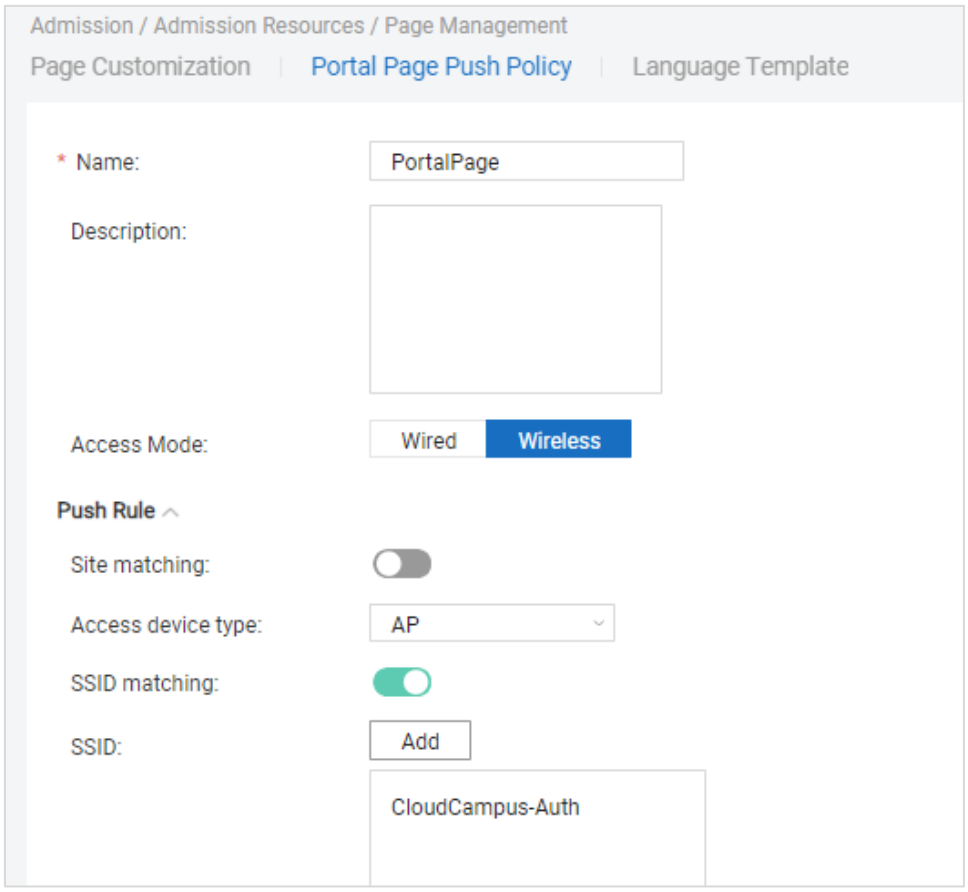
When iMaster NCE-Campus functions as a relay agent for Portal authentication and connects to the Portal server through an API, a Portal page push policy needs to be configured. When an end user is connected to an SSID, iMaster NCE-Campus pushes a specified Portal page to the end user based on Portal page push policies.

1. Set the name for a Portal page push policy and SSIDs associated with this policy.

   Choose **Admission > Admission Resources > Page Management** from the main menu, click the **Portal Page Push Policy** tab, and select the desired site.



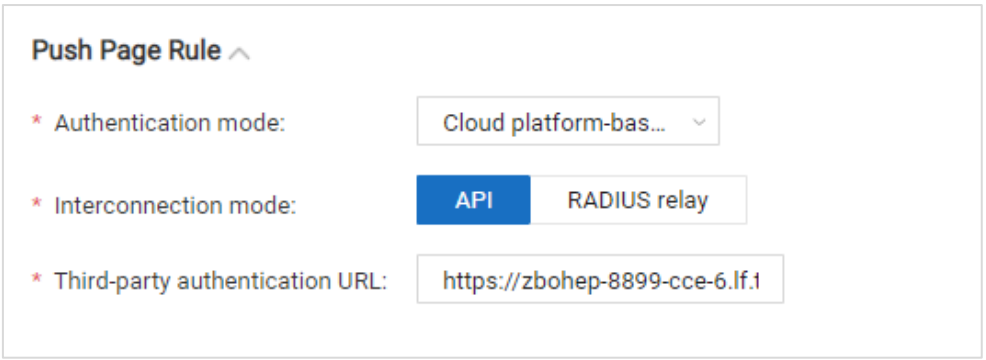
   Click **Create** to create a Portal page push policy. In this policy, set **Access device type** to **AP**.



Associate SSIDs with the policy.

2. Connect iMaster NCE-Campus to the third-party authentication application.

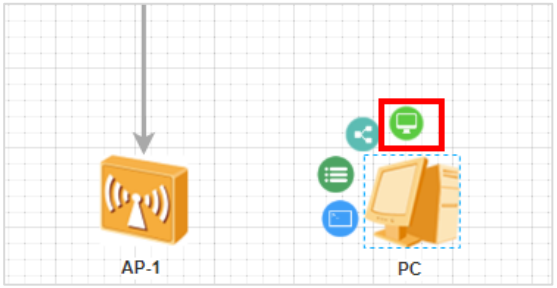
   Select **Cloud platform-based relay authentication** from the **Authentication mode** drop-down list. In the **Third-party authentication URL** text box, enter the URL of the login Portal page displayed after the third-party authentication application is started in CloudIDE.



   Click **Apply**. The interconnection between iMaster NCE-Campus and the third-party authentication application is complete.

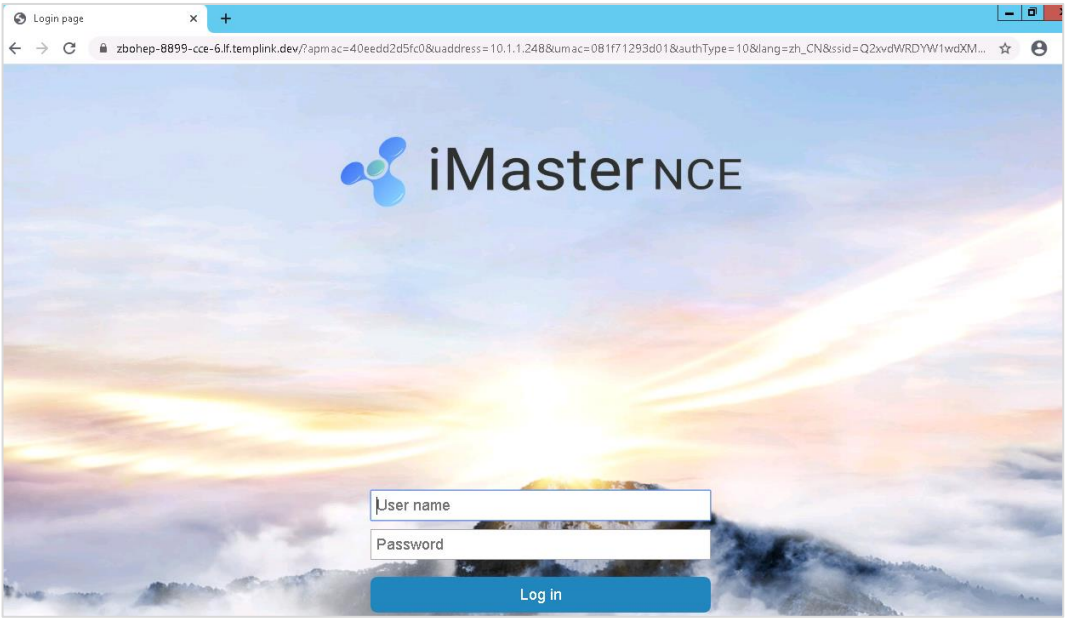
3.5.3 Verifying End User Access

On the sandbox environment reservation page, click **Open** and click  next to a PC to access the PC in the remote lab.

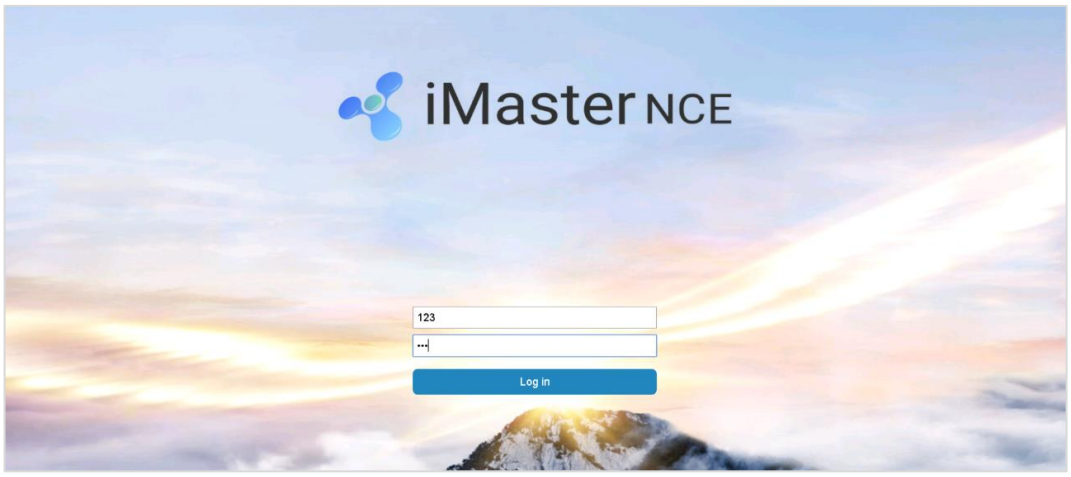


Connect the PC to the configured SSID.

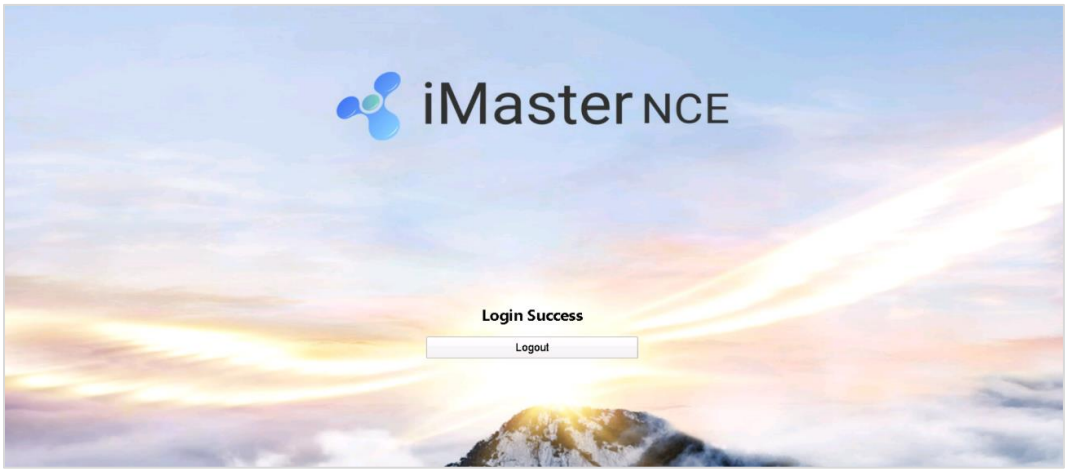
After the PC connects to the SSID, open a browser to access any website. The login Portal page configured in this experiment is displayed.



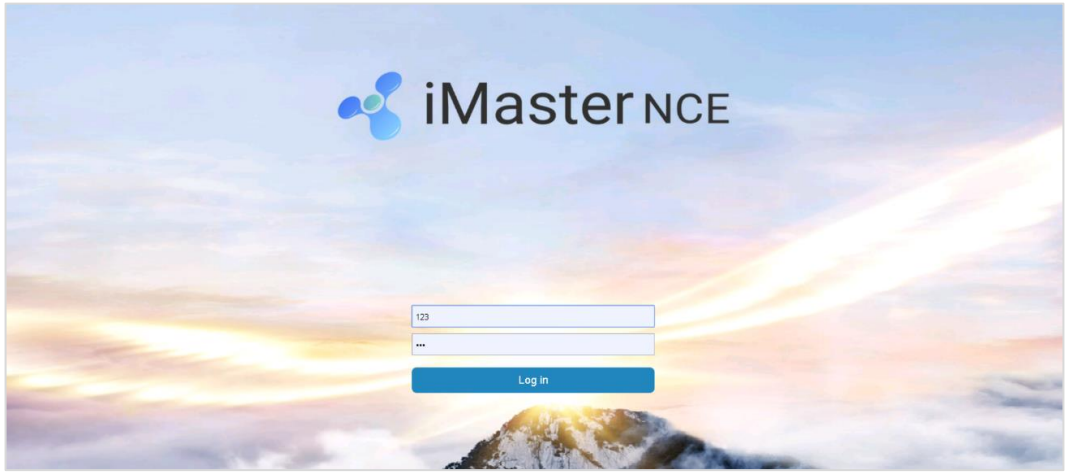
Enter the user name and password, and click **Log in**. The default user name and password in this experiment are both **123**.



The login is successful and the PC can access the Internet.



Click **Logout**. The login page is displayed.



The third-party authentication application is developed successfully.