阿里云
专有网络VPC

SDK 参考

文档版本：20200612

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。
2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。
3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。
4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。
5. 阿里云文档中所有内容，包括但不限于图片、架构设计、页面布局、文字描述，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。
6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	 **禁止：** 重置操作将丢失用户配置数据。
	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	 **警告：** 重启操作将导致业务中断，恢复业务时间约十分钟。
	用于警示信息、补充说明等，是用户必须了解的内容。	 **注意：** 权重设置为0，该服务器不会再接受新请求。
	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	 **说明：** 您也可以通过按Ctrl + A选中全部文件。
>	多级菜单递进。	单击**设置** > **网络** > **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
Courier字体	命令。	执行cd /d C:/window命令，进入Windows系统文件夹。
斜体	表示参数、变量。	bae log list --instanceid Instance_ID
[]或者[a|b]	表示可选项，至多选择一个。	ipconfig [-all|-t]
{}或者{a|b}	表示必选项，至多选择一个。	switch {active|stand}

目录

法律声明.....I
通用约定.....I
1 SDK下载..... 1
2 Python SDK 示例..... 2
2.1 安装Alibaba Cloud SDK for Python.....2
2.2 准备工作..... 3
2.3 专有网络（VPC）.....3
2.3.1 创建和删除VPC/VSwitch..... 3
2.3.2 创建自定义路由表.....7
2.3.3 创建自定义路由条目.....12
2.4 弹性公网IP（EIP）..... 16
2.4.1 绑定ECS实例.....16
2.4.2 加入共享带宽..... 22
2.4.3 修改带宽峰值..... 27
2.4.4 绑定负载均衡实例/弹性网卡..... 33
2.5 NAT网关.....38
2.5.1 创建NAT网关..... 38
2.5.2 绑定和解绑EIP.....42
2.5.3 创建DNAT条目..... 49
2.5.4 创建SNAT条目.....57
2.6 路由器接口.....65
2.6.1 同账号同地域VPC间互通..... 65
2.6.2 同账号同地域VBR与VPC间互通..... 73

1 SDK下载

专有网络VPC支持Java、Python、Go、.NET和PHP SDK开发。

下表列举了各语言SDK的下载地址和开发指南，更多SDK的信息，请访问阿里云开放平台。

Alibaba Cloud SDK	专有网络SDK	说明文档
Alibaba Cloud SDK for Java	Alibaba Cloud VPC SDK for Java	快速开始
Alibaba Cloud SDK for Python	Alibaba Cloud VPC SDK for Python	快速开始
Alibaba Cloud SDK for Go	Alibaba Cloud VPC SDK for Go	快速开始
Alibaba Cloud SDK for .NET	Alibaba Cloud VPC SDK for .NET	快速开始
Alibaba Cloud SDK for PHP	Alibaba Cloud VPC SDK for PHP	快速开始

2 Python SDK 示例

2.1 安装Alibaba Cloud SDK for Python

本文为您介绍如何在本地搭建可以运行专有网络Python SDK示例的Python开发环境，Alibaba Cloud SDK for Python支持Python 2.7，要运行专有网络的Python SDK示例，您需要安装Alibaba Cloud SDK for Python的核心库和VPC Python SDK。Python SDK支持Windows、Linux和Mac操作系统，所有专有网络Python SDK示例均在Windows系统下运行。

操作步骤

1. 安装Python。
   a) 登录Python下载地址。
   b) 下载2.7版本的Python。

      在下载列表中选择平台安装包，包格式为：python-XYZ.msi 文件 ， XYZ 为您要安装的版本号。
   c) 下载后，双击下载包，进入Python安装向导，使用默认设置即可。
   d) 设置环境变量。

      在**Path**行添加python安装路径和pip命令运行程序所在的目录，目录之间以**;**分隔。

      ```
      C:\Python27;C:\Python27\Scripts
      ```
   e) 在**cmd**命令行，执行python命令，显示类似如下，进入Python交互式环境，表示Python安装成功。

      ```
      Python 2.7.15 (
      Type "help", "copyright", "credits" or "license" for more information.
      >>>
         v2.7.15:ca079a3ea3, Apr 30 2018, 16:30:26) [MSC v.1500 64 bit (AM
      D64)] on win32
      ```
2. 执行如下命令，安装Alibaba Cloud SDK for Python核心库。

   ```
   pip install aliyun-python-sdk-core
   ```

   关于Python及PIP的使用说明，请参见Python文档和PIP文档。

3. 执行如下命令，安装aliyun-python-sdk-vpc。

```
pip install aliyun-python-sdk-vpc
```

2.2 准备工作

在运行专有网络VPC场景功能的SDK文件前，您需要完成公共配置。

在使用Alibaba Cloud SDK for Python前，您需要完成以下准备工作：

- 使用Alibaba Cloud SDK for Python，您需要一个阿里云账号和访问密钥（AccessKey）。 请在阿里云控制台中的AccessKey管理页面上创建和查看您的AccessKey。
- 确保您已经安装了Alibaba Cloud SDK for Python，请参见aliyun-python-sdk-vpc 3.0.8。
- 下载阿里云专有网络Python SDK场景示例的VPC Python Example库。

  进入setup.py所在的目录，执行如下命令，完成环境初始化配置。

```
python setup.py install
```

2.3 专有网络（VPC）

2.3.1 创建和删除VPC/VSwitch

本文操作示例介绍如何使用Alibaba Cloud SDK for Python创建和删除专有网络（VPC）和交换机（VSwitch）。

前提条件

在使用Alibaba Cloud SDK for Python前，您需要完成以下准备工作：

- 使用Alibaba Cloud SDK for Python，您需要一个阿里云账号和访问密钥（AccessKey）。 请在阿里云控制台中的AccessKey管理页面上创建和查看您的AccessKey。
- 确保您已经安装了Alibaba Cloud SDK for Python，请参见aliyun-python-sdk-vpc 3.0.8。
- 下载阿里云专有网络Python SDK场景示例的VPC Python Example库。

  进入setup.py所在的目录，执行如下命令，完成环境初始化配置。

```
python setup.py install
```

背景信息

在华北3张家口地域创建一个VPC，并在该VPC下创建一个VSwitch。VPC和VSwitch创建成功后，删除VPC和VSwitch。

操作步骤

1. 在下载的SDK目录中，打开aliyun-openapi-python-sdk-examples\sdk_examples\examples\vpc文件夹。
2. 使用代码编辑工具打开vpc_quick_start.py文件，根据实际情况配置相关参数，保存退出。

   完整代码示例如下：

```
#encoding=utf-8
import sys
import json
import time

from aliyunsdkcore.acs_exception.exceptions import ServerException, ClientException
from aliyunsdkvpc.request.v20160428 import CreateVpcRequest
from aliyunsdkvpc.request.v20160428 import CreateVSwitchRequest
from aliyunsdkvpc.request.v20160428 import DeleteVSwitchRequest
from aliyunsdkvpc.request.v20160428 import DeleteVpcRequest
from aliyunsdkvpc.request.v20160428 import DescribeVSwitchAttributesRequest
from aliyunsdkvpc.request.v20160428 import DescribeVpcAttributeRequest
from aliyunsdkcore.client import AcsClient

class VpcQuickStart(object):
    def __init__(self, client):
        self.client = client
        self.TIME_DEFAULT_OUT = 15
        self.DEFAULT_TIME = 1

    def check_status(self,time_default_out, default_time, func, check_status, id):
        for i in range(time_default_out):
            time.sleep(default_time)
            status = func(id)
            if status == check_status:
                return True
        return False

    def create_vpc(self):
        try:
            request = CreateVpcRequest.CreateVpcRequest()
            response = self.client.do_action_with_exception(request)
            response_json = json.loads(response)
            # 判断VPC状态是否可用
            if self.check_status(self.TIME_DEFAULT_OUT, self.DEFAULT_TIME,
                                 self.describe_vpc_status,
                                 "Available", response_json['VpcId']):
                return response_json
        except ServerException as e:
            print(e)
        except ClientException as e:
            print(e)

    def delete_vpc(self, params):
        try:
            request = DeleteVpcRequest.DeleteVpcRequest()
            # 要删除的VPC的ID
            request.set_VpcId(params['vpc_id'])
            response = self.client.do_action_with_exception(request)
            response_json = json.loads(response)
            return response_json
        except ServerException as e:
            print(e)
        except ClientException as e:
```

```
            print(e)

    def describe_vpc_attribute(self, vpc_id):
        try:
            request = DescribeVpcAttributeRequest.DescribeVpcAttributeRequest()
            # 要查询的VPC ID
            request.set_VpcId(vpc_id)
            response = self.client.do_action_with_exception(request)
            response_json = json.loads(response)
            return response_json
        except ServerException as e:
            print(e)
        except ClientException as e:
            print(e)

    def describe_vpc_status(self, vpc_id):
        response = self.describe_vpc_attribute(vpc_id)
        return response["Status"]

    def create_vswitch(self, params):
        try:
            request = CreateVSwitchRequest.CreateVSwitchRequest()
            # 交换机所属区的ID，您可以通过调用DescribeZones接口获取地域ID
            request.set_ZoneId(params['zone_id'])
            # 交换机所属的VPC ID
            request.set_VpcId(params['vpc_id'])
            # 交换机的网段
            request.set_CidrBlock(params['cidr_block'])
            response = self.client.do_action_with_exception(request)
            response_json = json.loads(response)
            # 判断VSwitch状态是否可用
            if self.check_status(self.TIME_DEFAULT_OUT, self.DEFAULT_TIME,
                                 self.describe_vswitch_status,
                                 "Available", response_json['VSwitchId']):
                return response_json
        except ServerException as e:
            print(e)
        except ClientException as e:
            print(e)

    def describe_vswitch_attribute(self, vswitch_id):
        try:
            request = DescribeVSwitchAttributesRequest.DescribeVSwitchAttributesRequest
()
            request.set_VSwitchId(vswitch_id)
            response = self.client.do_action_with_exception(request)
            response_json = json.loads(response)
            return response_json
        except ServerException as e:
            print(e)
        except ClientException as e:
            print(e)

    def describe_vswitch_status(self, vswitch_id):
        response = self.describe_vswitch_attribute(vswitch_id)
        return response["Status"]

    def delete_vswitch(self, params):
        try:
            request = DeleteVSwitchRequest.DeleteVSwitchRequest()
            # 要删除的交换机的ID
            request.set_VSwitchId(params['vswitch_id'])
            response = self.client.do_action_with_exception(request)
            response_json = json.loads(response)
```

```
            # 判断VSwitch是否被删除成功
            if self.check_status(self.TIME_DEFAULT_OUT, self.DEFAULT_TIME * 5,
                                 self.describe_vswitch_status,
                                 '', params['vswitch_id']):
                return response_json
        except ServerException as e:
            print(e)
        except ClientException as e:
            print(e)

if __name__ == "__main__":
    client = AcsClient('accessKeyId','accessSecret','cn-shanghai',timeout = 35)
    vpc_quick_start = VpcQuickStart(client)

    params = {}
    params['zone_id'] = "cn-zhangjiakou-b"
    params['cidr_block'] = "172.16.0.0/16"

    # 创建vpc
    vpc_json = vpc_quick_start.create_vpc()
    print("--------------------------create_vpc--------------------------")
    print(vpc_json)

    # 创建vswitch
    params['vpc_id'] = vpc_json['VpcId']
    vswitch_json = vpc_quick_start.create_vswitch(params)
    print("-------------------------create_vswitch-------------------------")
    print(vswitch_json)

    # 删除vswitch
    params['vswitch_id'] = vswitch_json['VSwitchId']
    vswitch_json = vpc_quick_start.delete_vswitch(params)
    print("-------------------------delete_vswitch-------------------------")
    print(vswitch_json)

    # 删除vpc
    vpc_json = vpc_quick_start.delete_vpc(params)
    print("--------------------------delete_vpc--------------------------")
    print(vpc_json)
```

3. 进入vpc_quick_start.py所在的目录，执行如下命令，运行创建和删除VPC/VSwitch示例。

```
python vpc_quick_start.py
```

预期结果

系统显示类似如下：

```
--------------------------create_vpc--------------------------
{
  "ResourceGroupId": "rg-acfmxazxxxxxxxx",
  "RouteTableId": "vtb-8vbf9ud7xrcn9xxxxxxxx",
  "VRouterId": "vrt-8vb1qjnxcm03nxxxxxxxx",
  "VpcId": "vpc-8vb67v4ozd8wfxxxxxxxx",
  "RequestId": "5052F988-75CC-46AD-A1A6-0E9E445BD0D5"
}

-------------------------create_vswitch-------------------------
{
  "VSwitchId": "vsw-8vbqn2at0kljjxxxxxxxx",
  "RequestId": "0BA1ABF7-21CF-4460-9A86-0BB783886E58"
}
```

```
--------------------------delete_vswitch--------------------------
{
  "RequestId": "D691F04B-A6EE-49A7-A434-4A45DD3AA0B8"
}

--------------------------delete_vpc--------------------------
{
  "RequestId": "4570F816-AB8D-45EA-8913-6AE787C1632C"
}
```

2.3.2 创建自定义路由表

本文介绍如何使用Alibaba Cloud SDK for Python创建自定义路由表。

前提条件

在使用Alibaba Cloud SDK for Python前，您需要完成以下准备工作：

- 使用Alibaba Cloud SDK for Python，您需要一个阿里云账号和访问密钥（AccessKey）。 请在阿里云控制台中的AccessKey管理页面上创建和查看您的AccessKey。
- 确保您已经安装了Alibaba Cloud SDK for Python，请参见aliyun-python-sdk-vpc 3.0.8。
- 下载阿里云专有网络Python SDK场景示例的VPC Python Example库。

  进入setup.py所在的目录，执行如下命令，完成环境初始化配置。

  ```
  python setup.py install
  ```

背景信息

本文档中的代码示例包含以下操作：

1. 在华北3张家口地域创建一个VPC。
2. 在新建的VPC下创建一个VSwitch。
3. 创建一个名为sdk_route_table的自定义路由表。
4. 查询新创建的VSwitch。
5. 将新创建的路由表与和同一VPC内的VSwitch进行绑定。
6. 将新创建的路由表与和同一VPC内的VSwitch进行解绑。
7. 删除新创建的自定义路由表。
8. 删除新创建的VSwitch。
9. 删除新创建的VPC。

操作步骤

1. 在下载的SDK目录中，打开$aliyun-openapi-python-sdk-examples\sdk_examples\examples\vpc文件夹。

2. 使用代码编辑工具打开vpc_route_table.py文件，根据实际情况配置相关参数，保存退出。

完整代码示例如下：

```
#encoding=utf-8
import sys
import json
import time

from aliyunsdkcore.acs_exception.exceptions import ServerException, ClientException
from aliyunsdkvpc.request.v20160428 import CreateRouteEntryRequest
from aliyunsdkvpc.request.v20160428 import DeleteRouteEntryRequest
from aliyunsdkvpc.request.v20160428 import DescribeRouteTablesRequest
from sdk_lib.exception import ExceptionHandler
from sdk_lib.check_status import CheckStatus
from sdk_lib.common_util import CommonUtil
from sdk_lib.sdk_vswitch import VSwitch
from sdk_lib.sdk_route_table import RouteTable
from sdk_lib.consts import *

class RouteEntry(object):
    def __init__(self, client):
        self.client = client

    def create_route_entry(self, params):
        """
        create_route_entry: 创建route_entry路由条目

        """
        try:
            request = CreateRouteEntryRequest.CreateRouteEntryRequest()
            # 路由表ID
            request.set_RouteTableId(params['route_table_id'])
            # 自定义路由条目的目标网段
            request.set_DestinationCidrBlock(params['destination_cidr_block'])
            # 下一跳的类型
            request.set_NextHopType(params['nexthop_type'])
            # 下一跳实例的ID
            request.set_NextHopId(params['nexthop_id'])
            response = self.client.do_action_with_exception(request)
            response_json = json.loads(response)
            # 判断router entry状态是否可用
            if CheckStatus.check_status(TIME_DEFAULT_OUT, DEFAULT_TIME,
                                        self.describe_route_entry_status,
                                        AVAILABLE, params['route_table_id']):
                return response_json
        except ServerException as e:
            ExceptionHandler.server_exception(e)
        except ClientException as e:
            ExceptionHandler.client_exception(e)

    def delete_route_entry(self, params):
        """
        delete_route_entry: 删除route_entry路由条目

        """
        try:
            request = DeleteRouteEntryRequest.DeleteRouteEntryRequest()
            # 路由条目所在的路由表的ID
            request.set_RouteTableId(params['route_table_id'])
            # 路由条目的目标网段
            request.set_DestinationCidrBlock(params['destination_cidr_block'])
            # 下一跳实例的ID
```

```
            request.set_NextHopId(params['nexthop_id'])
            response = self.client.do_action_with_exception(request)
            response_json = json.loads(response)
            time.sleep(DEFAULT_TIME)
            return response_json
        except ServerException as e:
            ExceptionHandler.server_exception(e)
        except ClientException as e:
            ExceptionHandler.client_exception(e)

    def describe_route_entry_vrouter(self, params):
        """
        describe_route_entry_vrouter: 查询route_entry路由条目

        """
        try:
            request = DescribeRouteTablesRequest.DescribeRouteTablesRequest()
            # 路由表所属的VPC路由器或边界路由器的ID
            request.set_VRouterId(params['vrouter_id'])
            # 路由表的ID
            request.set_RouteTableId(params['route_table_id'])
            response = self.client.do_action_with_exception(request)
            response_json = json.loads(response)
            return response_json
        except ServerException as e:
            ExceptionHandler.server_exception(e)
        except ClientException as e:
            ExceptionHandler.client_exception(e)

    def describe_route_entry(self, route_table_id):
        """
        describe_route_entry: 查询route_entry路由条目

        """
        try:
            request = DescribeRouteTablesRequest.DescribeRouteTablesRequest()
            request.set_RouteTableId(route_table_id)
            response = self.client.do_action_with_exception(request)
            response_json = json.loads(response)
            return response_json
        except ServerException as e:
            ExceptionHandler.server_exception(e)
        except ClientException as e:
            ExceptionHandler.client_exception(e)

    def describe_route_entry_status(self, route_table_id):
        """
        describe_route_entry_status: 查询route_entry路由条目当前状态

        """
        response = self.describe_route_entry(route_table_id)
        return response["RouteTables"]["RouteTable"][0]["RouteEntrys"]["RouteEntry"][0]["
Status"]

def main():
    #client参数配置
    client = AcsClient(
            '<your-access-key-id>',
            '<your-access-key-secret>',
            '<your-region-id>')

    vswitch = VSwitch(client)
    route_table = RouteTable(client)
    route_entry = RouteEntry(client)
```

```
    params = {}
    params['route_table_name'] = "sdk_route_table"
    params['destination_cidr_block'] = "0.0.0.0/0"
    params['nexthop_id'] = "i-xxx"
    params['nexthop_type'] = "Instance"

    params['vpc_id'] = "vpc-xxx"
    params['vswitch_id'] = "vsw-xxx"

    #创建route table
    route_table_json = route_table.create_route_table(params)
    CommonUtil.log("create_route_table", route_table_json)

    #查询vswitch
    vswitch_json = vswitch.describe_vswitch_attribute(params)
    CommonUtil.log("describe_vswitch_attribute", vswitch_json)

    #route table绑定vswitch
    params['route_table_id'] = route_table_json['RouteTableId']
    associate_json = route_table.associate_route_table(params)
    CommonUtil.log("associate_route_table", associate_json)

    #创建路由条目
    create_route_entry_json = route_entry.create_route_entry(params)
    CommonUtil.log("create_route_entry", create_route_entry_json)

    #删除路由条目
    delete_route_entry_json = route_entry.delete_route_entry(params)
    CommonUtil.log("delete_route_entry", delete_route_entry_json)

    #route table解绑vswitch
    unassociate_json = route_table.unassociate_route_table(params)
    CommonUtil.log("unassociate_route_table", unassociate_json)

    #删除route table
    delete_route_table_json = route_table.delete_route_table(params)
    CommonUtil.log("delete_route_table", delete_route_table_json)


if __name__ == "__main__":
    sys.exit(main())
```

3. 进入vpc_route_table.py所在的目录，执行如下命令，运行创建自定义路由表示例。

```
python vpc_route_table.py
```

预期结果

系统显示类似如下：

```
--------------------------create_vpc--------------------------
{
  "ResourceGroupId": "rg-acfmxazxxxxxxxx",
  "RouteTableId": "vtb-8vb65a5hqy8pcxxxxxxxx",
  "VRouterId": "vrt-8vbbbiftzizc3xxxxxxxx",
  "VpcId": "vpc-8vbebihln001gxxxxxxxx",
  "RequestId": "862F279B-4A27-4300-87A1-047FB9961AF2"
}

--------------------------create_vswitch--------------------------
{
```

```
  "VSwitchId": "vsw-8vb30klhn2is5xxxxxxxx",
  "RequestId": "1DA17173-CB61-4DCE-9C29-AABFDF3001A6"
}

--------------------------create_route_table--------------------------
{
  "RouteTableId": "vtb-8vbc4iwpo13apxxxxxxxx",
  "RequestId": "01E66E67-7801-4705-A02A-853BA7EEA89F"
}

--------------------------describe_vswitch_attribute--------------------------

{
  "Status": "Available",
  "NetworkAclId": "",
  "VpcId": "vpc-8vbebihln001gxxxxxxxx",
  "Description": "",
  "RouteTable": {
    "RouteTableId": "vtb-8vb65a5hqy8pcxxxxxxxx",
    "RouteTableType": "System"
  },
  "CidrBlock": "172.16.0.0/16",
  "CreationTime": "2019-04-12T03:08:43Z",
  "CloudResources": {
    "CloudResourceSetType": []
  },
  "ZoneId": "cn-zhangjiakou-b",
  "ResourceGroupId": "rg-acfmxazbxxxxxxxx",
  "VSwitchId": "vsw-8vb30klhn2is5xxxxxxxx",
  "RequestId": "C5A20BA3-E998-498D-8900-35AE5FDFFB77",
  "Ipv6CidrBlock": "",
  "VSwitchName": "",
  "AvailableIpAddressCount": 252,
  "IsDefault": false
}

--------------------------associate_route_table--------------------------
{
  "RequestId": "5FC0143B-D34B-47DC-8D49-AFD222EA5876"
}

--------------------------unassociate_route_table--------------------------
{
  "RequestId": "F0194718-6E4C-496C-9DA8-1B88DF1D6FAD"
}

--------------------------delete_route_table--------------------------
{
  "RequestId": "B5C068A6-137C-4337-8E3A-9E30E1726703"
}

--------------------------delete_vswitch--------------------------
{
  "RequestId": "26DEDBF8-2F0D-4A13-8CB3-23A84C947704"
}

--------------------------delete_vpc--------------------------
{
  "RequestId": "E1B2641F-5911-40E4-9F36-CC0B2EDD1747"
```

```
}
```

2.3.3 创建自定义路由条目

本文介绍如何使用Alibaba Cloud SDK for Python在VPC路由器或边界路由器（VBR）上创建自定义路由条目。

前提条件

在使用Alibaba Cloud SDK for Python前，您需要完成以下准备工作：

- 使用Alibaba Cloud SDK for Python，您需要一个阿里云账号和访问密钥（AccessKey）。 请在阿里云控制台中的AccessKey管理页面上创建和查看您的AccessKey。
- 确保您已经安装了Alibaba Cloud SDK for Python，请参见aliyun-python-sdk-vpc 3.0.8。
- 下载阿里云专有网络Python SDK场景示例的VPC Python Example库。

  进入setup.py所在的目录，执行如下命令，完成环境初始化配置。

  ```
  python setup.py install
  ```

背景信息

本文档中的代码示例包含以下操作：

1. 在华北3张家口地域创建一个名为sdk_route_table的自定义路由表。
2. 查询专有网络vpc-8vb7ztbjqomi9xxxxxxxx下的VSwitch。
3. 将新创建的自定义路由表与交换机vsw-8vbfqpcijj0d1xxxxxxxx进行绑定。
4. 为新建的自定义路由表创建自定义路由条目，目的网络为168.168.0.0/16，下一跳类型为ECS实例，下一跳ID为i-8vbgsnt7046axxxxxxxx。
5. 删除新创建的路由条目。
6. 将新创建的自定义路由表与VSwitch解绑。
7. 删除新创建的自定义路由表。

操作步骤

1. 在下载的SDK目录中，打开$aliyun-openapi-python-sdk-examples\sdk_examples\examples\vpc文件夹。
2. 使用编辑器打开vpc_route_entry.py文件，根据实际情况配置相关参数，保存退出。

   完整代码示例如下：

   ```
   #encoding=utf-8
   import sys
   import json
   import time

   from aliyunsdkcore.acs_exception.exceptions import ServerException, ClientException
   ```

```
from aliyunsdkvpc.request.v20160428 import CreateRouteEntryRequest
from aliyunsdkvpc.request.v20160428 import DeleteRouteEntryRequest
from aliyunsdkvpc.request.v20160428 import DescribeRouteTablesRequest
from sdk_lib.exception import ExceptionHandler
from sdk_lib.check_status import CheckStatus
from sdk_lib.common_util import CommonUtil
from sdk_lib.sdk_vswitch import VSwitch
from sdk_lib.sdk_route_table import RouteTable
from sdk_lib.consts import *

class RouteEntry(object):
    def __init__(self, client):
        self.client = client

    def create_route_entry(self, params):
        """
        create_route_entry: 创建route_entry路由条目

        """
        try:
            request = CreateRouteEntryRequest.CreateRouteEntryRequest()
            # 路由表ID
            request.set_RouteTableId(params['route_table_id'])
            # 自定义路由条目的目标网段
            request.set_DestinationCidrBlock(params['destination_cidr_block'])
            # 下一跳的类型
            request.set_NextHopType(params['nexthop_type'])
            # 下一跳实例的ID
            request.set_NextHopId(params['nexthop_id'])
            response = self.client.do_action_with_exception(request)
            response_json = json.loads(response)
            # 判断router entry状态是否可用
            if CheckStatus.check_status(TIME_DEFAULT_OUT, DEFAULT_TIME,
                                        self.describe_route_entry_status,
                                        AVAILABLE, params['route_table_id']):
                return response_json
        except ServerException as e:
            ExceptionHandler.server_exception(e)
        except ClientException as e:
            ExceptionHandler.client_exception(e)

    def delete_route_entry(self, params):
        """
        delete_route_entry: 删除route_entry路由条目

        """
        try:
            request = DeleteRouteEntryRequest.DeleteRouteEntryRequest()
            # 路由条目所在的路由表的ID
            request.set_RouteTableId(params['route_table_id'])
            # 路由条目的目标网段
            request.set_DestinationCidrBlock(params['destination_cidr_block'])
            # 下一跳实例的ID
            request.set_NextHopId(params['nexthop_id'])
            response = self.client.do_action_with_exception(request)
            response_json = json.loads(response)
            time.sleep(DEFAULT_TIME)
            return response_json
        except ServerException as e:
            ExceptionHandler.server_exception(e)
        except ClientException as e:
            ExceptionHandler.client_exception(e)

    def describe_route_entry_vrouter(self, params):
```

```
        """
        describe_route_entry_vrouter: 查询route_entry路由条目

        """
        try:
            request = DescribeRouteTablesRequest.DescribeRouteTablesRequest()
            # 路由表所属的VPC路由器或边界路由器的ID
            request.set_VRouterId(params['vrouter_id'])
            # 路由表的ID
            request.set_RouteTableId(params['route_table_id'])
            response = self.client.do_action_with_exception(request)
            response_json = json.loads(response)
            return response_json
        except ServerException as e:
            ExceptionHandler.server_exception(e)
        except ClientException as e:
            ExceptionHandler.client_exception(e)

    def describe_route_entry(self, route_table_id):
        """
        describe_route_entry: 查询route_entry路由条目

        """
        try:
            request = DescribeRouteTablesRequest.DescribeRouteTablesRequest()
            request.set_RouteTableId(route_table_id)
            response = self.client.do_action_with_exception(request)
            response_json = json.loads(response)
            return response_json
        except ServerException as e:
            ExceptionHandler.server_exception(e)
        except ClientException as e:
            ExceptionHandler.client_exception(e)

    def describe_route_entry_status(self, route_table_id):
        """
        describe_route_entry_status: 查询route_entry路由条目当前状态

        """
        response = self.describe_route_entry(route_table_id)
        return response["RouteTables"]["RouteTable"][0]["RouteEntrys"]["RouteEntry"][0]["
Status"]

def main():
    client = AcsClient(
            '<your-access-key-id>',
            '<your-access-key-secret>',
            '<your-region-id>')
    vswitch = VSwitch(client)
    route_table = RouteTable(client)
    route_entry = RouteEntry(client)

    params = {}
    params['route_table_name'] = "sdk_route_table"
    params['destination_cidr_block'] = "0.0.0.0/0"
    params['nexthop_id'] = "i-xxx"
    params['nexthop_type'] = "Instance"

    params['vpc_id'] = "vpc-xxx"
    params['vswitch_id'] = "vsw-xxx"

    #创建route table
    route_table_json = route_table.create_route_table(params)
    CommonUtil.log("create_route_table", route_table_json)
```

```
    #查询vswitch
    vswitch_json = vswitch.describe_vswitch_attribute(params)
    CommonUtil.log("describe_vswitch_attribute", vswitch_json)

    #route table绑定vswitch
    params['route_table_id'] = route_table_json['RouteTableId']
    associate_json = route_table.associate_route_table(params)
    CommonUtil.log("associate_route_table", associate_json)

    #创建路由条目
    create_route_entry_json = route_entry.create_route_entry(params)
    CommonUtil.log("create_route_entry", create_route_entry_json)

    #删除路由条目
    delete_route_entry_json = route_entry.delete_route_entry(params)
    CommonUtil.log("delete_route_entry", delete_route_entry_json)

    #route table解绑vswitch
    unassociate_json = route_table.unassociate_route_table(params)
    CommonUtil.log("unassociate_route_table", unassociate_json)

    #删除route table
    delete_route_table_json = route_table.delete_route_table(params)
    CommonUtil.log("delete_route_table", delete_route_table_json)


if __name__ == "__main__":
    sys.exit(main())
```

3. 进入vpc_route_entry.py所在的目录，执行如下命令，运行创建自定义路由条目示例。

```
python vpc_route_entry.py
```

预期结果

系统显示类似如下：

```
--------------------------create_route_table--------------------------
{
  "RouteTableId": "vtb-8vbn7px9zxwr2xxxxxxxx",
  "RequestId": "8B351EE1-614F-44E4-93AF-1CADA4BF02E8"
}

--------------------------describe_vswitch_attribute--------------------------

{
  "Status": "",
  "NetworkAclId": "",
  "VpcId": "",
  "Description": "",
  "Ipv6CidrBlock": "",
  "CreationTime": "",
  "CloudResources": {
    "CloudResourceSetType": []
  },
  "ZoneId": "",
  "ResourceGroupId": "",
  "VSwitchId": "",
  "RequestId": "5E199415-BBA3-443D-B1EC-06341FE267F4",
  "VSwitchName": "",
  "CidrBlock": ""
```

```
}

--------------------------associate_route_table--------------------------
{
  "RequestId": "5F33E444-5CCD-4677-91AB-3E234A9A64E4"
}

--------------------------create_route_entry--------------------------
{
  "RequestId": "D6035ECA-DD81-4FAB-B084-55BE60FB18ED"
}

--------------------------delete_route_entry--------------------------
{
  "RequestId": "54108FD7-8609-4111-919D-B2983466F480"
}

--------------------------unassociate_route_table--------------------------
{
  "RequestId": "0F36A76A-1E54-41DC-852E-1D970FDE8F3F"
}

--------------------------delete_route_table--------------------------
{
  "RequestId": "F3151A59-4F90-4531-AFDC-B7B7CF70A8C1"
}
```

2.4 弹性公网IP（EIP）

2.4.1 绑定ECS实例

本文介绍如何使用Alibaba Cloud SDK for Python为一个专有网络类型的ECS实例绑定一个弹性公网IP（EIP）。

前提条件

在使用Alibaba Cloud SDK for Python前，您需要完成以下准备工作：

- 使用Alibaba Cloud SDK for Python，您需要一个阿里云账号和访问密钥（AccessKey）。 请在阿里云控制台中的AccessKey管理页面上创建和查看您的AccessKey。
- 确保您已经安装了Alibaba Cloud SDK for Python，请参见aliyun-python-sdk-vpc 3.0.8。
- 下载阿里云专有网络Python SDK场景示例的VPC Python Example库。

  进入setup.py所在的目录，执行如下命令，完成环境初始化配置。

  ```
  python setup.py install
  ```

背景信息

本文代码示例中包含以下操作：

1. 在华东1杭州地域创建一个EIP。
2. 将创建的EIP绑定到ECS。

3. 查询绑定到ECS上的EIP。
4. 修改EIP的带宽峰值和名称。
5. 查询修改后的EIP。
6. 将EIP与ECS解绑。
7. 释放EIP。

操作步骤

1. 在下载的SDK目录中，打开$aliyun-openapi-python-sdk-examples\sdk_examples\examples\eip文件夹。
2. 使用编辑器打开eip_quick_start.py文件，根据实际情况配置相关参数，保存退出。

   完整代码示例如下：

```
#encoding=utf-8
import sys
import json
from aliyunsdkcore.acs_exception.exceptions import ServerException, ClientException
from aliyunsdkvpc.request.v20160428 import AllocateEipAddressRequest
from aliyunsdkvpc.request.v20160428 import AssociateEipAddressRequest
from aliyunsdkvpc.request.v20160428 import DescribeEipAddressesRequest
from aliyunsdkvpc.request.v20160428 import UnassociateEipAddressRequest
from aliyunsdkvpc.request.v20160428 import ModifyEipAddressAttributeRequest
from aliyunsdkvpc.request.v20160428 import ReleaseEipAddressRequest
from sdk_lib.exception import ExceptionHandler
from sdk_lib.check_status import CheckStatus
from sdk_lib.consts import *
from sdk_lib.common_util import CommonUtil

"""
创建EIP->绑定EIP到ECS->查询EIP->修改EIP配置和名字->查询EIP->解绑EIP->释放EIP
"""
class Eip(object):
    def __init__(self, client):
        self.client = client

    def allocate_eip_address(self, params):
        """
        allocate_eip_address: 申请弹性公网IP（EIP)

        """
        try:
            request = AllocateEipAddressRequest.AllocateEipAddressRequest()
            response = self.client.do_action_with_exception(request)
            response_json = json.loads(response)
            if CheckStatus.check_status(TIME_DEFAULT_OUT, DEFAULT_TIME,
                                        self.describe_eip_status,
                                        AVAILABLE, response_json["AllocationId"]):
                return response_json
        except ServerException as e:
            ExceptionHandler.server_exception(e)
        except ClientException as e:
            ExceptionHandler.client_exception(e)

    def associate_eip_address(self, params):
        """
        associate_eip_address: 将EIP绑定到同地域的云产品实例上
```

```
    """
    try:
        request = AssociateEipAddressRequest.AssociateEipAddressRequest()
        # EIP的ID
        request.set_AllocationId(params['allocation_id'])
        # 要绑定的云产品实例的类型
        request.set_InstanceType(params['instance_type'])
        # 要绑定的实例ID
        request.set_InstanceId(params['instance_id'])
        response = self.client.do_action_with_exception(request)
        response_json = json.loads(response)
        if CheckStatus.check_status(TIME_DEFAULT_OUT, DEFAULT_TIME,
                                    self.describe_eip_status,
                                    InUse, params['allocation_id']):
            return response_json
    except ServerException as e:
        ExceptionHandler.server_exception(e)
    except ClientException as e:
        ExceptionHandler.client_exception(e)

def describe_eip_address(self, allocation_id):
    """
    describe_eip_status: 查询指定地域已创建的EIP。

    """
    try:
        request = DescribeEipAddressesRequest.DescribeEipAddressesRequest()
        # EIP的ID
        request.set_AllocationId(allocation_id)
        response = self.client.do_action_with_exception(request)
        response_json = json.loads(response)
        return response_json
    except ServerException as e:
        ExceptionHandler.server_exception(e)
    except ClientException as e:
        ExceptionHandler.client_exception(e)

def describe_eip_status(self, allocation_id):
    """
    describe_eip_status: 查询指定地域已创建的EIP的状态

    """
    # EIP的ID
    response = self.describe_eip_address(allocation_id)
    return response["EipAddresses"]["EipAddress"][0]["Status"]


def unassociate_eip_address(self, params):
    """
    unassociate_eip_address: 将EIP从绑定的云资源上解绑。

    """
    try:
        request = UnassociateEipAddressRequest.UnassociateEipAddressRequest()
        # EIP的ID
        request.set_AllocationId(params['allocation_id'])
        # 要解绑的资源类型
        request.set_InstanceType(params['instance_type'])
        # 要解绑的云产品的实例ID
        request.set_InstanceId(params['instance_id'])
        response = self.client.do_action_with_exception(request)
        response_json = json.loads(response)
        if CheckStatus.check_status(TIME_DEFAULT_OUT, DEFAULT_TIME,
```

```
                                self.describe_eip_status,
                                AVAILABLE, params['allocation_id']):
                    return response_json
                return response_json
            except ServerException as e:
                ExceptionHandler.server_exception(e)
            except ClientException as e:
                ExceptionHandler.client_exception(e)

    def modify_eip_address(self, params):
        """
        modify_eip_address: 修改指定EIP的名称、描述信息和带宽峰值

        """
        try:
            request = ModifyEipAddressAttributeRequest.ModifyEipAddressAttributeRequest
()
            # 弹性公网IP的ID
            request.set_AllocationId(params['allocation_id'])
            # EIP的带宽峰值，单位为Mbps
            request.set_Bandwidth(params['bandwidth'])
            # EIP的名称
            request.set_Name(params['name'])
            response = self.client.do_action_with_exception(request)
            response_json = json.loads(response)
            return response_json
        except ServerException as e:
            ExceptionHandler.server_exception(e)
        except ClientException as e:
            ExceptionHandler.client_exception(e)

    def release_eip_address(self, params):
        """
        release_eip_address: 释放指定的EIP。

        """
        try:
            request = ReleaseEipAddressRequest.ReleaseEipAddressRequest()
            # 要释放的弹性公网IP的ID
            request.set_AllocationId(params['allocation_id'])
            response = self.client.do_action_with_exception(request)
            response_json = json.loads(response)
            return response_json
        except ServerException as e:
            ExceptionHandler.server_exception(e)
        except ClientException as e:
            ExceptionHandler.client_exception(e)

def main():

    client = AcsClient(
            '<your-access-key-id>',    # 您的可用区ID
            '<your-access-key-secret>', # 您的AccessKey ID
            '<your-region-id>')        # 您的AccessKey Secret

    params = {}
    # 创建EIP
    eip_response_json = eip.allocate_eip_address(params)
    CommonUtil.log("allocate_eip_address", eip_response_json)

    # 绑定EIP到ECS
    params['allocation_id'] = eip_response_json["AllocationId"]
    params['instance_id'] = ECS_INSTANCE_ID
    params['instance_type'] = 'EcsInstance'
```

```
    eip_response_json = eip.associate_eip_address(params)
    CommonUtil.log("associate_eip_address", eip_response_json)

    # 查询EIP
    eip_response_json = eip.describe_eip_address(params['allocation_id'])
    CommonUtil.log("describe_eip_address", eip_response_json)

    # 修改EIP配置和名字
    params['bandwidth'] = BANDWIDTH_50
    params['name'] = EIP_NEW_NAME
    eip_response_json = eip.modify_eip_address(params)
    CommonUtil.log("modify_eip_address", eip_response_json)

    # 查询EIP
    eip_response_json = eip.describe_eip_address(params['allocation_id'])
    CommonUtil.log("describe_eip_address", eip_response_json)

    # 解绑EIP
    eip_response_json = eip.unassociate_eip_address(params)
    CommonUtil.log("unassociate_eip_address", eip_response_json)

    # 释放EIP
    eip_response_json = eip.release_eip_address(params)
    CommonUtil.log("release_eip_address", eip_response_json)

if __name__ == '__main__':
    sys.exit(main())
```

3. 进入eip_quick_start.py所在的目录，执行如下命令，将EIP绑定ECS实例。

```
python eip_quick_start.py
```

预期结果

系统显示类似如下：

```
-------------------------allocate_eip_address-------------------------
{
 "EipAddress": "47.xx.xx.23",
 "ResourceGroupId": "rg-acfm4odxxxxxxxx",
 "RequestId": "C438312E-F7A4-4A04-901F-D22FE23EDB4D",
 "AllocationId": "eip-bp1wybucvhhx5xxxxxxxx"
}
-------------------------associate_eip_address-------------------------
{
 "RequestId": "6EC6605E-3D2B-4EE8-BD13-F1964CD1EAB1"
}
-------------------------describe_eip_address-------------------------
{
 "TotalCount": 1,
 "PageNumber": 1,
 "PageSize": 10,
 "EipAddresses": {
  "EipAddress": [
   {
    "ISP": "BGP",
    "ExpiredTime": "",
    "InternetChargeType": "PayByBandwidth",
    "IpAddress": "47.xx.xx.23",
    "AllocationId": "eip-bp1wybucvhhx5xxxxxxxx",
```

```
      "PrivateIpAddress": "",
      "Status": "InUse",
      "BandwidthPackageId": "",
      "InstanceId": "i-bp1e82xlhob2xxxxxxxx",
      "InstanceRegionId": "cn-hangzhou",
      "RegionId": "cn-hangzhou",
      "AvailableRegions": {
       "AvailableRegion": [
        "cn-hangzhou"
       ]
      },
      "ResourceGroupId": "rg-acfm4odxxxxxxxx",
      "HasReservationData": false,
      "InstanceType": "EcsInstance",
      "AllocationTime": "2019-04-17T11:57:43Z",
      "Name": "",
      "OperationLocks": {
       "LockReason": []
      },
      "Mode": "NAT",
      "BandwidthPackageType": "",
      "BandwidthPackageBandwidth": "",
      "Bandwidth": "5",
      "HDMonitorStatus": "OFF",
      "ChargeType": "PostPaid",
      "SecondLimited": false,
      "Descritpion": ""
     }
    ]
  },
  "RequestId": "8715A878-A808-4CC4-AAD5-E414FDAB5B0E"
}
----------------------------modify_eip_address----------------------------
{
  "RequestId": "2108AE1C-94FB-475D-BFEE-EC88598BF6A6"
}
----------------------------describe_eip_address----------------------------
{
  "TotalCount": 1,
  "PageNumber": 1,
  "PageSize": 10,
  "EipAddresses": {
   "EipAddress": [
    {
      "ISP": "BGP",
      "ExpiredTime": "",
      "InternetChargeType": "PayByBandwidth",
      "IpAddress": "47.xx.xx.23",
      "AllocationId": "eip-bp1wybucvhhx5xxxxxxxx",
      "PrivateIpAddress": "",
      "Status": "InUse",
      "BandwidthPackageId": "",
      "InstanceId": "i-bp1e82xlhob2xxxxxxxx",
      "InstanceRegionId": "cn-hangzhou",
      "RegionId": "cn-hangzhou",
      "AvailableRegions": {
       "AvailableRegion": [
        "cn-hangzhou"
       ]
      },
      "ResourceGroupId": "rg-acfm4odxxxxxxxx",
      "HasReservationData": false,
      "InstanceType": "EcsInstance",
      "AllocationTime": "2019-04-17T11:57:43Z",
```

```
      "Name": "EIP_NEW_NAME",
      "OperationLocks": {
       "LockReason": []
      },
      "Mode": "NAT",
      "BandwidthPackageType": "",
      "BandwidthPackageBandwidth": "",
      "Bandwidth": "50",
      "HDMonitorStatus": "OFF",
      "ChargeType": "PostPaid",
      "SecondLimited": false,
      "Descritpion": ""
     }
    ]
  },
  "RequestId": "6694D35B-B5DD-4506-8AB1-2D16477646DE"
}
--------------------------unassociate_eip_address--------------------------
{
  "RequestId": "EDE86CF6-EE68-4922-B919-85A4F11BF668"
}
--------------------------release_eip_address--------------------------
{
  "RequestId": "53FEE062-B595-4D64-AB47-834015D32888"
}
```

2.4.2 加入共享带宽

本文介绍如何使用Alibaba Cloud SDK for Python将一个弹性公网IP添加到共享带宽。

前提条件

在使用Alibaba Cloud SDK for Python前，您需要完成以下准备工作：

- 使用Alibaba Cloud SDK for Python，您需要一个阿里云账号和访问密钥（AccessKey）。 请在阿里云控制台中的AccessKey管理页面上创建和查看您的AccessKey。
- 确保您已经安装了Alibaba Cloud SDK for Python，请参见aliyun-python-sdk-vpc 3.0.8。
- 下载阿里云专有网络Python SDK场景示例的VPC Python Example库。

  进入setup.py所在的目录，执行如下命令，完成环境初始化配置。

  ```
  python setup.py install
  ```

背景信息

本文代码示例中包含以下操作：

1. 在华东1杭州地域创建一个EIP。
2. 在华东1杭州地域创建共享带宽。
3. 将新创建的EIP加入共享带宽。
4. 查询共享带宽。
5. 将EIP移出共享带宽。
6. 删除共享带宽。

7. 释放EIP。

操作步骤

1. 在下载的SDK目录中，打开$aliyun-openapi-python-sdk-examples\sdk_examples\examples\eip文件夹。
2. 使用编辑器打开eip_add_cbwp.py文件，根据实际情况配置相关参数，保存退出。

   完整代码示例如下：

```
#encoding=utf-8
import sys
from aliyunsdkvpc.request.v20160428 import CreateCommonBandwidthPackageRequest
from aliyunsdkvpc.request.v20160428 import AddCommonBandwidthPackageIpRequest
from aliyunsdkvpc.request.v20160428 import DescribeCommonBandwidthPackagesRequest
from aliyunsdkvpc.request.v20160428 import RemoveCommonBandwidthPackageIpRequest
from aliyunsdkvpc.request.v20160428 import DeleteCommonBandwidthPackageRequest
from sdk_lib.common_util import CommonUtil
from sdk_lib.sdk_eip import *

"""
创建EIP->创建带宽包->通用带宽包id->添加EIP到共享带宽包中->查询共享带宽包->移除共享带宽包中的EIP->删除共享带宽包->释放EIP
"""
class CommonBandwidthPackage(object):
    def __init__(self, client):
        self.client = client

    def create_common_bandwidth_package(self, params):
        """
        create_common_bandwidth_package: 创建共享带宽

        """
        try:
            request = CreateCommonBandwidthPackageRequest.CreateCommonBandwidthPackageRequest()
            # 共享带宽的带宽峰值，单位为Mbps
            request.set_Bandwidth(params['bandwidth'])
            response = self.client.do_action_with_exception(request)
            response_json = json.loads(response)
            if CheckStatus.check_status(TIME_DEFAULT_OUT, DEFAULT_TIME,
                                        self.describe_cbwp_status,
                                        AVAILABLE, response_json["BandwidthPackageId"]):
                return response_json
        except ServerException as e:
            ExceptionHandler.server_exception(e)
        except ClientException as e:
            ExceptionHandler.client_exception(e)

    def add_common_bandwidth_packageIp(self, params):
        """
        add_common_bandwidth_packageIp: 添加EIP到共享带宽中

        """
        try:
```

```
            request = AddCommonBandwidthPackageIpRequest.AddCommonBandwidthPa
ckageIpRequest()
            # EIP实例的ID
            request.set_IpInstanceId(params['ip_instance_id'])
            # 共享带宽的ID
            request.set_BandwidthPackageId(params['bandwidth_package_id'])
            response = self.client.do_action_with_exception(request)
            response_json = json.loads(response)
            return response_json
        except ServerException as e:
            ExceptionHandler.server_exception(e)
        except ClientException as e:
            ExceptionHandler.client_exception(e)

    def describe_cbwp(self, cbwp_id):
        """
        describe_cbwp: 查询共享带宽实例信息

        """
        try:
            request = DescribeCommonBandwidthPackagesRequest.DescribeCo
mmonBandwidthPackagesRequest()
            # 共享带宽实例的ID
            request.set_BandwidthPackageId(cbwp_id)
            response = self.client.do_action_with_exception(request)
            response_json = json.loads(response)
            return response_json
        except ServerException as e:
            ExceptionHandler.server_exception(e)
        except ClientException as e:
            ExceptionHandler.client_exception(e)

    def describe_cbwp_status(self, cbwp_id):
        """
        describe_cbwp_status: 查询共享带宽实例状态

        """
        # 共享带宽实例的ID
        response = self.describe_cbwp(cbwp_id)
        return response["CommonBandwidthPackages"]["CommonBandwidthPackage"][0
]["Status"]


    def remove_common_bandwidth_packageIp(self, params):
        """
        remove_common_bandwidth_packageIp: 移除共享带宽中的EIP

        """
        try:
            request = RemoveCommonBandwidthPackageIpRequest.RemoveComm
onBandwidthPackageIpRequest()
            # EIP实例的ID
            request.set_IpInstanceId(params['ip_instance_id'])
            # 共享带宽实例的ID
            request.set_BandwidthPackageId(params['bandwidth_package_id'])
            response = self.client.do_action_with_exception(request)
            response_json = json.loads(response)
            return response_json
        except ServerException as e:
            ExceptionHandler.server_exception(e)
        except ClientException as e:
            ExceptionHandler.client_exception(e)
```

```
    def delete_common_bandwidth_package(self, params):
        """
        delete_common_bandwidth_package: 删除共享带宽包

        """
        try:
            request = DeleteCommonBandwidthPackageRequest.DeleteCommonBandwidt
hPackageRequest()
            # 共享带宽实例的ID
            request.set_BandwidthPackageId(params['bandwidth_package_id'])
            # 是否强制删除共享带宽实例
            request.set_Force(params['force'])
            response = self.client.do_action_with_exception(request)
            response_json = json.loads(response)
            return response_json
        except ServerException as e:
            ExceptionHandler.server_exception(e)
        except ClientException as e:
            ExceptionHandler.client_exception(e)

def main():

    client = AcsClient(
            '<your-access-key-id>',     # 您的可用区ID
            '<your-access-key-secret>', # 您的AccessKey ID
            '<your-region-id>')         # 您的AccessKey Secret

    params = {}

    # 创建EIP
    eip_response_json = eip.allocate_eip_address(params)
    CommonUtil.log("allocate_eip_address", eip_response_json)
    params['allocation_id'] = eip_response_json["AllocationId"]

    # 创建带宽包
    params['allocation_id'] = eip_response_json["AllocationId"]
    params['ip_instance_id'] = eip_response_json["AllocationId"]
    params['bandwidth'] = BANDWIDTH_10
    cbwp_repsonse_json = cbwp.create_common_bandwidth_package(params)
    CommonUtil.log("create_common_bandwidth_package", cbwp_repsonse_json)

    # 添加EIP到共享带宽包中
    params['bandwidth_package_id'] = cbwp_repsonse_json['BandwidthPackageId']
    cbwp_repsonse_json = cbwp.add_common_bandwidth_packageIp(params)
    CommonUtil.log("add_common_bandwidth_packageIp", cbwp_repsonse_json)

    # 查询共享带宽包
    cbwp_repsonse_json = cbwp.describe_cbwp(params['bandwidth_package_id'])
    CommonUtil.log("add_common_bandwidth_packageIp", cbwp_repsonse_json)

    # 移除共享带宽包中的EIP（
    cbwp_repsonse_json = cbwp.remove_common_bandwidth_packageIp(params)
    CommonUtil.log("remove_common_bandwidth_packageIp", cbwp_repsonse_json)

    # 删除共享带宽包
    params['force'] = True
    cbwp_repsonse_json = cbwp.delete_common_bandwidth_package(params)
    CommonUtil.log("delete_common_bandwidth_package", cbwp_repsonse_json)

    # 释放EIP
    eip_response_json = eip.release_eip_address(params)
    CommonUtil.log("release_eip_address", eip_response_json)

if __name__ == '__main__':
```

```
    sys.exit(main())
```

3. 进入eip_add_cbwp.py所在的目录，执行如下命令，实现EIP加入共享带宽。

```
python eip_add_cbwp.py
```

预期结果

系统显示类似如下：

```
--------------------------allocate_eip_address--------------------------
{
 "EipAddress": "118.xx.xx.198",
 "ResourceGroupId": "rg-acfm4odxxxxxxxx",
 "RequestId": "A830A607-B7C4-49FE-A6EE-7237D64CDE2D",
 "AllocationId": "eip-bp1mdyvr22qvgxxxxxxxx"
}
--------------------------create_common_bandwidth_package----------------------
-----
{
 "ResourceGroupId": "rg-acfm4odxxxxxxxx",
 "BandwidthPackageId": "cbwp-bp12k058pjjiexxxxxxxx",
 "RequestId": "93127320-DD79-4F83-A3B9-DC99D0597B0C"
}
--------------------------add_common_bandwidth_packageIp----------------------
----
{
 "RequestId": "7F314AFE-B398-4348-AF61-B7D27B731286"
}
--------------------------add_common_bandwidth_packageIp----------------------
----
{
 "TotalCount": 1,
 "CommonBandwidthPackages": {
  "CommonBandwidthPackage": [
   {
    "Status": "Available",
    "PublicIpAddresses": {
     "PublicIpAddresse": [
      {
       "IpAddress": "118.xx.xx.198",
       "AllocationId": "eip-bp1mdyvr22qvgxxxxxxxx"
      }
     ]
    },
    "BusinessStatus": "Normal",
    "RegionId": "cn-hangzhou",
    "BandwidthPackageId": "cbwp-bp12k058pjjiexxxxxxxx",
    "Name": "",
    "ISP": "BGP",
    "CreationTime": "2019-04-18T01:46:17Z",
    "ResourceGroupId": "rg-acfm4odxxxxxxxx",
    "Bandwidth": "10",
    "InstanceChargeType": "PostPaid",
    "HasReservationData": false,
    "InternetChargeType": "PayByBandwidth",
    "ExpiredTime": "",
    "Ratio": 100,
    "Description": ""
   }
  ]
 },
```

```
  "PageNumber": 1,
  "RequestId": "015DD0FA-742B-4431-92EA-E3F03FDEB8CD",
  "PageSize": 10
}
--------------------------remove_common_bandwidth_packageIp-------------------
-------
{
  "RequestId": "A49C9126-B703-4D34-B552-A7FE283FB5DD"
}
--------------------------delete_common_bandwidth_package---------------------
-----
{
  "RequestId": "E423F648-C169-4B63-A2CF-5E6C8E441DE1"
}
--------------------------release_eip_address--------------------------
{
  "RequestId": "7E0D34AE-58C3-468A-B021-378F8938AE6B"
}
```

2.4.3 修改带宽峰值

本文介绍如何使用Alibaba Cloud SDK for Python修改EIP的带宽峰值。

前提条件

在使用Alibaba Cloud SDK for Python前，您需要完成以下准备工作：

- 使用Alibaba Cloud SDK for Python，您需要一个阿里云账号和访问密钥（AccessKey）。 请在阿里云控制台中的AccessKey管理页面上创建和查看您的AccessKey。
- 确保您已经安装了Alibaba Cloud SDK for Python，请参见aliyun-python-sdk-vpc 3.0.8。
- 下载阿里云专有网络Python SDK场景示例的VPC Python Example库。

  进入setup.py所在的目录，执行如下命令，完成环境初始化配置。

  ```
  python setup.py install
  ```

背景信息

本文代码示例中包含以下操作：

1. 在华东1杭州地域创建一个EIP。
2. 修改EIP的带宽峰值为50M。
3. 查询修改带宽峰值后的EIP。
4. 修改EIP的带宽峰值为10M。
5. 查询修改带宽峰值后的EIP。
6. 释放EIP。

操作步骤

1. 在下载的SDK目录中，打开$aliyun-openapi-python-sdk-examples\sdk_examples\examples\eip文件夹。

2. 使用编辑器打开eip_modify_attribute.py文件，根据实际情况配置相关参数，保存退出。

完整代码示例如下：

```
#encoding=utf-8
import sys
import json

from aliyunsdkcore.acs_exception.exceptions import ServerException, ClientException
from aliyunsdkvpc.request.v20160428 import AllocateEipAddressRequest
from aliyunsdkvpc.request.v20160428 import AssociateEipAddressRequest
from aliyunsdkvpc.request.v20160428 import DescribeEipAddressesRequest
from aliyunsdkvpc.request.v20160428 import UnassociateEipAddressRequest
from aliyunsdkvpc.request.v20160428 import ModifyEipAddressAttributeRequest
from aliyunsdkvpc.request.v20160428 import ReleaseEipAddressRequest
from sdk_lib.exception import ExceptionHandler
from sdk_lib.check_status import CheckStatus
from sdk_lib.consts import *
from sdk_lib.common_util import CommonUtil

"""
创建EIP->修改EIP带宽到50>查询EIP->修改EIP带宽到10->查询EIP->释放EIP
"""
class Eip(object):
    def __init__(self, client):
        self.client = client

    def allocate_eip_address(self, params):
        """
        allocate_eip_address: 申请弹性公网IP（EIP)
        """
        try:
            request = AllocateEipAddressRequest.AllocateEipAddressRequest()
            response = self.client.do_action_with_exception(request)
            response_json = json.loads(response)
            if CheckStatus.check_status(TIME_DEFAULT_OUT, DEFAULT_TIME,
                                        self.describe_eip_status,
                                        AVAILABLE, response_json["AllocationId"]):
                return response_json
        except ServerException as e:
            ExceptionHandler.server_exception(e)
        except ClientException as e:
            ExceptionHandler.client_exception(e)

    def associate_eip_address(self, params):
        """
        associate_eip_address: 将EIP绑定到同地域的云产品实例上
        """
        try:
            request = AssociateEipAddressRequest.AssociateEipAddressRequest()
            # EIP的ID
            request.set_AllocationId(params['allocation_id'])
            # 要绑定的云产品实例的类型
            request.set_InstanceType(params['instance_type'])
            # 要绑定的实例ID
            request.set_InstanceId(params['instance_id'])
            response = self.client.do_action_with_exception(request)
            response_json = json.loads(response)
            if CheckStatus.check_status(TIME_DEFAULT_OUT, DEFAULT_TIME,
                                        self.describe_eip_status,
                                        InUse, params['allocation_id']):
                return response_json
        except ServerException as e:
```

```
            ExceptionHandler.server_exception(e)
        except ClientException as e:
            ExceptionHandler.client_exception(e)

    def describe_eip_address(self, allocation_id):
        """
        describe_eip_status: 查询指定地域已创建的EIP。
        """
        try:
            request = DescribeEipAddressesRequest.DescribeEipAddressesRequest()
            # EIP的ID
            request.set_AllocationId(allocation_id)
            response = self.client.do_action_with_exception(request)
            response_json = json.loads(response)
            return response_json
        except ServerException as e:
            ExceptionHandler.server_exception(e)
        except ClientException as e:
            ExceptionHandler.client_exception(e)

    def describe_eip_status(self, allocation_id):
        """
        describe_eip_status: 查询指定地域已创建的EIP的状态
        """
        # EIP的ID
        response = self.describe_eip_address(allocation_id)
        return response["EipAddresses"]["EipAddress"][0]["Status"]


    def unassociate_eip_address(self, params):
        """
        unassociate_eip_address: 将EIP从绑定的云资源上解绑。
        """
        try:
            request = UnassociateEipAddressRequest.UnassociateEipAddressRequest()
            # EIP的ID
            request.set_AllocationId(params['allocation_id'])
            # 要解绑的资源类型
            request.set_InstanceType(params['instance_type'])
            # 要解绑的云产品的实例ID
            request.set_InstanceId(params['instance_id'])
            response = self.client.do_action_with_exception(request)
            response_json = json.loads(response)
            if CheckStatus.check_status(TIME_DEFAULT_OUT, DEFAULT_TIME,
                                        self.describe_eip_status,
                                        AVAILABLE, params['allocation_id']):
                return response_json
            return response_json
        except ServerException as e:
            ExceptionHandler.server_exception(e)
        except ClientException as e:
            ExceptionHandler.client_exception(e)

    def modify_eip_address(self, params):
        """
        modify_eip_address: 修改指定EIP的名称、描述信息和带宽峰值
        """
        try:
            request = ModifyEipAddressAttributeRequest.ModifyEipAddressAttributeRequest
()
            # 弹性公网IP的ID
            request.set_AllocationId(params['allocation_id'])
            # EIP的带宽峰值，单位为Mbps
            request.set_Bandwidth(params['bandwidth'])
```

```
            # EIP的名称
            request.set_Name(params['name'])
            response = self.client.do_action_with_exception(request)
            response_json = json.loads(response)
            return response_json
        except ServerException as e:
            ExceptionHandler.server_exception(e)
        except ClientException as e:
            ExceptionHandler.client_exception(e)

    def release_eip_address(self, params):
        """
        release_eip_address: 释放指定的EIP。
        """
        try:
            request = ReleaseEipAddressRequest.ReleaseEipAddressRequest()
            # 要释放的弹性公网IP的ID
            request.set_AllocationId(params['allocation_id'])
            response = self.client.do_action_with_exception(request)
            response_json = json.loads(response)
            return response_json
        except ServerException as e:
            ExceptionHandler.server_exception(e)
        except ClientException as e:
            ExceptionHandler.client_exception(e)

def main():

    client = AcsClient(
        '<your-access-key-id>',     # 您的可用区ID
        '<your-access-key-secret>', # 您的AccessKey ID
        '<your-region-id>')         # 您的AccessKey Secret
    eip = Eip(client)

    params = {}

    # 创建EIP
    eip_response_json = eip.allocate_eip_address(params)
    CommonUtil.log("allocate_eip_address", eip_response_json)

    params['allocation_id'] = eip_response_json["AllocationId"]

    #修改EIP带宽到50
    params['name'] = EIP_NEW_NAME
    params['bandwidth'] = BANDWIDTH_50
    eip_response_json = eip.modify_eip_address(params)
    CommonUtil.log("modify_eip_address", eip_response_json)

    # 查询EIP
    eip_response_json = eip.describe_eip_address(params['allocation_id'])
    CommonUtil.log("describe_eip_address", eip_response_json)

    # 修改EIP带宽到10
    params['bandwidth'] = BANDWIDTH_10
    eip_response_json = eip.modify_eip_address(params)
    CommonUtil.log("modify_eip_address", eip_response_json)

    # 查询EIP
    eip_response_json = eip.describe_eip_address(params['allocation_id'])
    CommonUtil.log("describe_eip_address", eip_response_json)

    # 释放EIP
    eip_response_json = eip.release_eip_address(params)
    CommonUtil.log("release_eip_address", eip_response_json)
```

```
if __name__ == '__main__':
    sys.exit(main())
```

3. 进入eip_modify_attribute.py所在的目录，执行如下命令，修改EIP的带宽峰值。

```
python eip_modify_attribute.py
```

预期结果

系统显示类似如下：

```
--------------------------allocate_eip_address--------------------------
{
 "EipAddress": "47.xx.xx.225",
 "ResourceGroupId": "rg-acfm4odxxxxxxxx",
 "RequestId": "9318DD7A-F065-4EA6-9EA0-20A9C46EDADC",
 "AllocationId": "eip-bp15bzjk5djcsxxxxxxxx"
}
--------------------------modify_eip_address--------------------------
{
 "RequestId": "C39D55A1-6B47-489B-8614-FDB9736EDE73"
}
--------------------------describe_eip_address--------------------------
{
 "TotalCount": 1,
 "PageNumber": 1,
 "PageSize": 10,
 "EipAddresses": {
  "EipAddress": [
   {
    "ISP": "BGP",
    "ExpiredTime": "",
    "InternetChargeType": "PayByBandwidth",
    "IpAddress": "47.xx.xx.225",
    "AllocationId": "eip-bp15bzjk5djcsxxxxxxxx",
    "PrivateIpAddress": "",
    "Status": "Available",
    "BandwidthPackageId": "",
    "InstanceId": "",
    "InstanceRegionId": "",
    "RegionId": "cn-hangzhou",
    "AvailableRegions": {
     "AvailableRegion": [
      "cn-hangzhou"
     ]
    },
    "ResourceGroupId": "rg-acfm4odxxxxxxxx",
    "HasReservationData": false,
    "InstanceType": "",
    "AllocationTime": "2019-04-18T04:01:28Z",
    "Name": "EIP_NEW_NAME",
    "OperationLocks": {
     "LockReason": []
    },
    "Mode": "NAT",
    "BandwidthPackageType": "",
    "BandwidthPackageBandwidth": "",
    "Bandwidth": "50",
    "HDMonitorStatus": "OFF",
    "ChargeType": "PostPaid",
    "SecondLimited": false,
```

```
      "Descritpion": ""
    }
   ]
  },
  "RequestId": "51ECAB45-2518-4A46-89DC-8ADEE1AFDBE9"
}
--------------------------modify_eip_address--------------------------
{
  "RequestId": "ACAB5724-D05C-46A4-8C2B-6064AEEC792B"
}
--------------------------describe_eip_address--------------------------
{
  "TotalCount": 1,
  "PageNumber": 1,
  "PageSize": 10,
  "EipAddresses": {
   "EipAddress": [
    {
     "ISP": "BGP",
     "ExpiredTime": "",
     "InternetChargeType": "PayByBandwidth",
     "IpAddress": "47.xx.xx.225",
     "AllocationId": "eip-bp15bzjk5djcsxxxxxxxx",
     "PrivateIpAddress": "",
     "Status": "Available",
     "BandwidthPackageId": "",
     "InstanceId": "",
     "InstanceRegionId": "",
     "RegionId": "cn-hangzhou",
     "AvailableRegions": {
      "AvailableRegion": [
       "cn-hangzhou"
      ]
     },
     "ResourceGroupId": "rg-acfm4odxxxxxxxx",
     "HasReservationData": false,
     "InstanceType": "",
     "AllocationTime": "2019-04-18T04:01:28Z",
     "Name": "EIP_NEW_NAME",
     "OperationLocks": {
      "LockReason": []
     },
     "Mode": "NAT",
     "BandwidthPackageType": "",
     "BandwidthPackageBandwidth": "",
     "Bandwidth": "10",
     "HDMonitorStatus": "OFF",
     "ChargeType": "PostPaid",
     "SecondLimited": false,
     "Descritpion": ""
    }
   ]
  },
  "RequestId": "6F653A80-AB28-4842-84A8-CD444EB81A29"
}
--------------------------release_eip_address--------------------------
{
  "RequestId": "C407633A-5658-482F-AB5E-069028C3B06C"
```

```
}
```

2.4.4 绑定负载均衡实例/弹性网卡

本文介绍如何使用Alibaba Cloud SDK for Python将弹性公网IP绑定到负载均衡实例和弹性网卡。

前提条件

在使用Alibaba Cloud SDK for Python前，您需要完成以下准备工作：

- 使用Alibaba Cloud SDK for Python，您需要一个阿里云账号和访问密钥（AccessKey）。 请在阿里云控制台中的AccessKey管理页面上创建和查看您的AccessKey。
- 确保您已经安装了Alibaba Cloud SDK for Python，请参见aliyun-python-sdk-vpc 3.0.8。
- 下载阿里云专有网络Python SDK场景示例的VPC Python Example库。

  进入setup.py所在的目录，执行如下命令，完成环境初始化配置。

  ```
  python setup.py install
  ```

背景信息

本文代码示例中包含以下操作：

1. 在华东1杭州地域创建负载均衡实例。
2. 在华东1杭州地域创建EIP。
3. 将EIP绑定到负载均衡实例。
4. 将EIP与负载均衡实例解绑。
5. 将解绑后的EIP绑定到弹性网卡。
6. 将EIP与弹性网卡解绑。
7. 删除负载均衡实例。

操作步骤

1. 在下载的SDK目录中，打开$aliyun-openapi-python-sdk-examples\sdk_examples\examples\eip文件夹。
2. 使用编辑器打开eip_associate_slb.py文件，根据实际情况配置相关参数，保存退出。

   完整代码示例如下：

   ```
   #encoding=utf-8
   import sys
   import json
   from aliyunsdkcore.acs_exception.exceptions import ServerException, ClientException
   from aliyunsdkvpc.request.v20160428 import AllocateEipAddressRequest
   from aliyunsdkvpc.request.v20160428 import AssociateEipAddressRequest
   from aliyunsdkvpc.request.v20160428 import DescribeEipAddressesRequest
   from aliyunsdkvpc.request.v20160428 import UnassociateEipAddressRequest
   from aliyunsdkvpc.request.v20160428 import ModifyEipAddressAttributeRequest
   from aliyunsdkvpc.request.v20160428 import ReleaseEipAddressRequest
   ```

```
from sdk_lib.exception import ExceptionHandler
from sdk_lib.check_status import CheckStatus
from sdk_lib.consts import *
from sdk_lib.common_util import CommonUtil
from sdk_lib.sdk_load_balancer import LoadBalancer

"""
创建VPC->创建VSWITCH->以及调用ECS API生成 ENI(这几步不在以下代码中)
创建负载均衡->创建EIP->绑定EIP到SLB->解绑EIP->绑定EIP到ENI->解绑EIP->删除负载均衡
"""
class Eip(object):
    def __init__(self, client):
        self.client = client

    def allocate_eip_address(self, params):
        """
        allocate_eip_address: 申请弹性公网IP（EIP)
        """
        try:
            request = AllocateEipAddressRequest.AllocateEipAddressRequest()
            response = self.client.do_action_with_exception(request)
            response_json = json.loads(response)
            if CheckStatus.check_status(TIME_DEFAULT_OUT, DEFAULT_TIME,
                                        self.describe_eip_status,
                                        AVAILABLE, response_json["AllocationId"]):
                return response_json
        except ServerException as e:
            ExceptionHandler.server_exception(e)
        except ClientException as e:
            ExceptionHandler.client_exception(e)

    def associate_eip_address(self, params):
        """
        associate_eip_address: 将EIP绑定到同地域的云产品实例上
        """
        try:
            request = AssociateEipAddressRequest.AssociateEipAddressRequest()
            # EIP的ID
            request.set_AllocationId(params['allocation_id'])
            # 要绑定的云产品实例的类型
            request.set_InstanceType(params['instance_type'])
            # 要绑定的实例ID
            request.set_InstanceId(params['instance_id'])
            response = self.client.do_action_with_exception(request)
            response_json = json.loads(response)
            if CheckStatus.check_status(TIME_DEFAULT_OUT, DEFAULT_TIME,
                                        self.describe_eip_status,
                                        InUse, params['allocation_id']):
                return response_json
        except ServerException as e:
            ExceptionHandler.server_exception(e)
        except ClientException as e:
            ExceptionHandler.client_exception(e)

    def describe_eip_address(self, allocation_id):
        """
        describe_eip_status: 查询指定地域已创建的EIP。
        """
        try:
            request = DescribeEipAddressesRequest.DescribeEipAddressesRequest()
            # EIP的ID
            request.set_AllocationId(allocation_id)
            response = self.client.do_action_with_exception(request)
            response_json = json.loads(response)
```

```
            return response_json
        except ServerException as e:
            ExceptionHandler.server_exception(e)
        except ClientException as e:
            ExceptionHandler.client_exception(e)

    def describe_eip_status(self, allocation_id):
        """
        describe_eip_status: 查询指定地域已创建的EIP的状态
        """
        # EIP的ID
        response = self.describe_eip_address(allocation_id)
        return response["EipAddresses"]["EipAddress"][0]["Status"]


    def unassociate_eip_address(self, params):
        """
        unassociate_eip_address: 将EIP从绑定的云资源上解绑。
        """
        try:
            request = UnassociateEipAddressRequest.UnassociateEipAddressRequest()
            # EIP的ID
            request.set_AllocationId(params['allocation_id'])
            # 要解绑的资源类型
            request.set_InstanceType(params['instance_type'])
            # 要解绑的云产品的实例ID
            request.set_InstanceId(params['instance_id'])
            response = self.client.do_action_with_exception(request)
            response_json = json.loads(response)
            if CheckStatus.check_status(TIME_DEFAULT_OUT, DEFAULT_TIME,
                                        self.describe_eip_status,
                                        AVAILABLE, params['allocation_id']):
                return response_json
            return response_json
        except ServerException as e:
            ExceptionHandler.server_exception(e)
        except ClientException as e:
            ExceptionHandler.client_exception(e)

    def modify_eip_address(self, params):
        """
        modify_eip_address: 修改指定EIP的名称、描述信息和带宽峰值
        """
        try:
            request = ModifyEipAddressAttributeRequest.ModifyEipAddressAttributeRequest
()
            # 弹性公网IP的ID
            request.set_AllocationId(params['allocation_id'])
            # EIP的带宽峰值，单位为Mbps
            request.set_Bandwidth(params['bandwidth'])
            # EIP的名称
            request.set_Name(params['name'])
            response = self.client.do_action_with_exception(request)
            response_json = json.loads(response)
            return response_json
        except ServerException as e:
            ExceptionHandler.server_exception(e)
        except ClientException as e:
            ExceptionHandler.client_exception(e)

    def release_eip_address(self, params):
        """
        release_eip_address: 释放指定的EIP。
        """
```

```
        try:
            request = ReleaseEipAddressRequest.ReleaseEipAddressRequest()
            # 要释放的弹性公网IP的ID
            request.set_AllocationId(params['allocation_id'])
            response = self.client.do_action_with_exception(request)
            response_json = json.loads(response)
            return response_json
        except ServerException as e:
            ExceptionHandler.server_exception(e)
        except ClientException as e:
            ExceptionHandler.client_exception(e)
def main():

    client = AcsClient(
        '<your-access-key-id>',    # 您的可用区ID
        '<your-access-key-secret>', # 您的AccessKey ID
        '<your-region-id>')        # 您的AccessKey Secret
    eip = Eip(client)
    load_balancer = LoadBalancer(client)
    #创建VPC, 创建VSWITCH, 以及调用ECS API生成 ENI
    params = {}
    params['vpc_id'] = VPC_ID
    params['vswitch_id'] = VSWITCH_ID

    #创建负载均衡
    load_balancer_repsonse_json = load_balancer.create_load_balancer(params)
    CommonUtil.log("create_load_balancer", load_balancer_repsonse_json)

    # 创建EIP
    params['load_balancer_id'] = load_balancer_repsonse_json['LoadBalancerId']
    params['instance_id'] = load_balancer_repsonse_json['LoadBalancerId']
    eip_response_json = eip.allocate_eip_address(params)
    CommonUtil.log("allocate_eip_address slb", eip_response_json)

    # 绑定EIP到SLB
    params['allocation_id'] = eip_response_json["AllocationId"]
    params['instance_type'] = 'SlbInstance'
    eip_response_json = eip.associate_eip_address(params)
    CommonUtil.log("associate_eip_address eip", eip_response_json)

    # 解绑EIP
    eip_response_json = eip.unassociate_eip_address(params)
    CommonUtil.log("unassociate_eip_address slb", eip_response_json)

    # 绑定EIP到ENI
    params['instance_id'] = ENI_ID
    params['instance_type'] = 'NetworkInterface'
    eip_response_json = eip.associate_eip_address(params)
    CommonUtil.log("associate_eip_address eni", eip_response_json)

    # 解绑EIP
    eip_response_json = eip.unassociate_eip_address(params)
    CommonUtil.log("unassociate_eip_address eni", eip_response_json)

    # 删除负载均衡实例
    load_balancer_repsonse_json = load_balancer.delete_load_balancer(params)
    CommonUtil.log("delete_load_balancer", load_balancer_repsonse_json)

if __name__ == '__main__':
```

```
    sys.exit(main())
```

3. 进入eip_associate_slb.py所在的目录，执行如下命令，实现EIP绑定负载均衡实例/弹性网卡。

```
Python eip_associate_slb.py
```

预期结果

系统显示类似如下：

```
--------------------------create_load_balancer--------------------------
{
 "VpcId": "vpc-bp15opprpg0rgxxxxxxxx",
 "AddressIPVersion": "ipv4",
 "LoadBalancerName": "auto_named_slb",
 "ResourceGroupId": "rg-acfm4odxxxxxxxx",
 "VSwitchId": "vsw-bp1e67w26n2sjxxxxxxxx",
 "RequestId": "D3651A96-008C-4B35-A36E-54C2902535C5",
 "Address": "172.xx.xx.146",
 "NetworkType": "vpc",
 "LoadBalancerId": "lb-bp15u6kumammdxxxxxxxx"
}
--------------------------allocate_eip_address slb--------------------------
{
 "EipAddress": "47.xx.xx.76",
 "ResourceGroupId": "rg-acfm4odxxxxxxxx",
 "RequestId": "15FD58CD-B186-4E2C-B4B3-74F712168832",
 "AllocationId": "eip-bp1ofhmmep6rkxxxxxxxx"
}
--------------------------associate_eip_address eip--------------------------
{
 "RequestId": "5EDABF0B-A067-474E-9556-0A3AA870960A"
}
--------------------------unassociate_eip_address slb--------------------------
-
{
 "RequestId": "89556510-D726-490A-9092-9BEA0644CC43"
}
--------------------------associate_eip_address eni--------------------------
{
 "RequestId": "FAE87FDD-232A-4859-803B-F9B57508AEDC"
}
--------------------------unassociate_eip_address eni--------------------------
-
{
 "RequestId": "7DF556E8-12BE-481A-B83D-B3D9E8836534"
}
--------------------------delete_load_balancer--------------------------
{
 "RequestId": "2B70C01A-A440-40A5-A98B-83A687537CCE"
```

```
}
```

2.5 NAT网关

2.5.1 创建NAT网关

本文介绍如何使用Alibaba Cloud SDK for Python创建NAT网关。

前提条件

在使用Alibaba Cloud SDK for Python前，您需要完成以下准备工作：

- 使用Alibaba Cloud SDK for Python，您需要一个阿里云账号和访问密钥（AccessKey）。 请在阿里云控制台中的AccessKey管理页面上创建和查看您的AccessKey。
- 确保您已经安装了Alibaba Cloud SDK for Python，请参见aliyun-python-sdk-vpc 3.0.8。
- 下载阿里云专有网络Python SDK场景示例的VPC Python Example库。

  进入setup.py所在的目录，执行如下命令，完成环境初始化配置。

  ```
  python setup.py install
  ```

背景信息

本文代码示例中包含以下操作：

1. 在华东2上海地域创建一个VPC。
2. 在新建的VPC下创建一个VSwitch。
3. 在新建的VPC下创建一个NAT网关。
4. 查询新创建的NAT网关。
5. 删除NAT网关。
6. 删除VSwitch。
7. 删除VPC。

操作步骤

1. 在下载的SDK目录中，打开$aliyun-openapi-python-sdk-examples\sdk_examples\examples\natgw文件夹。
2. 使用编辑器打开natgw_quick_start.py文件，根据实际情况配置相关参数，保存退出。

   完整代码示例如下：

   ```
   #encoding=utf-8
   import sys
   import json
   import time

   from aliyunsdkcore.acs_exception.exceptions import ServerException, ClientException
   ```

```
from aliyunsdkvpc.request.v20160428 import CreateNatGatewayRequest
from aliyunsdkvpc.request.v20160428 import DeleteNatGatewayRequest
from aliyunsdkvpc.request.v20160428 import DescribeNatGatewaysRequest
from sdk_lib.sdk_vpc import Vpc
from sdk_lib.sdk_vswitch import VSwitch
from sdk_lib.common_util import CommonUtil
from sdk_lib.check_status import CheckStatus
from sdk_lib.exception import ExceptionHandler
from sdk_lib.consts import *

client = AcsClient(
        '<your-access-key-id>',     # 您的可用区ID
        '<your-access-key-secret>', # 您的AccessKey ID
        '<your-region-id>')         # 您的AccessKey Secret


class NatGateway(object):
    def __init__(self, client):
        self.client = client

    def create_nat_gateway(self, params):
        """
        create_nat_gateway: 创建nat gateway
        """
        try:
            request = CreateNatGatewayRequest.CreateNatGatewayRequest()
            request.set_VpcId(params['vpc_id'])
            response = client.do_action_with_exception(request)
            response_json = json.loads(response)
            # 判断Nat Gateway状态是否可用
            if CheckStatus.check_status(TIME_DEFAULT_OUT, DEFAULT_TIME,
                                        self.describe_nat_gateway_status,
                                        AVAILABLE, response_json['NatGatewayId']):
                return response_json
        except ServerException as e:
            ExceptionHandler.server_exception(e)
        except ClientException as e:
            ExceptionHandler.client_exception(e)

    def describe_nat_gateway(self, nat_gateway_id):
        """
        describe_nat_gateway: 查询指定地域已创建的nat gateway的信息
        """
        try:
            request = DescribeNatGatewaysRequest.DescribeNatGatewaysRequest()
            request.set_NatGatewayId(nat_gateway_id)
            response = client.do_action_with_exception(request)
            response_json = json.loads(response)
            return response_json
        except ServerException as e:
            ExceptionHandler.server_exception(e)
        except ClientException as e:
            ExceptionHandler.client_exception(e)

    def delete_nat_gateway(self, params):
        """
        delete_nat_gateway: 删除nat gateway
        """
        try:
            request = DeleteNatGatewayRequest.DeleteNatGatewayRequest()
            request.set_NatGatewayId(params['nat_gateway_id'])
            response = client.do_action_with_exception(request)
            response_json = json.loads(response)
            # 判断Nat Gateway状态是否可用
```

```
            if CheckStatus.check_status(TIME_DEFAULT_OUT, DEFAULT_TIME * 5,
                                        self.describe_nat_gateway_status,
                                        '', params['nat_gateway_id']):
                return response_json
        except ServerException as e:
            ExceptionHandler.server_exception(e)
        except ClientException as e:
            ExceptionHandler.client_exception(e)

    def describe_nat_gateway_status(self, nat_gateway_id):
        """
        describe_nat_gateway_status: 查询指定地域已创建的nat gateway的状态
        """
        response = self.describe_nat_gateway(nat_gateway_id)
        if len(response["NatGateways"]["NatGateway"]) == 0:
            return ''
        return response["NatGateways"]["NatGateway"][0]['Status']


def main():
    vpc = Vpc(client)
    vswitch = VSwitch(client)
    nat_gateway = NatGateway(client)

    params = {}

    # 创建vpc
    vpc_json = vpc.create_vpc()
    CommonUtil.log("create_vpc", vpc_json)

    # 创建vswitch
    params['vpc_id'] = vpc_json['VpcId']
    params['zone_id'] = "cn-hangzhou-d"
    params['cidr_block'] = "172.16.1.0/24"
    vswitch_json = vswitch.create_vswitch(params)
    CommonUtil.log("create_vswitch", vswitch_json)

    # 创建natgw
    nat_gateway_json = nat_gateway.create_nat_gateway(params)
    CommonUtil.log("create_nat_gateway", nat_gateway_json)

    # 查询natgw
    params['nat_gateway_id'] = nat_gateway_json['NatGatewayId']
    nat_gateway_json = nat_gateway.describe_nat_gateway(params['nat_gateway_id'])
    CommonUtil.log("describe_nat_gateway", nat_gateway_json)

    # 删除natgw
    nat_gateway_json = nat_gateway.delete_nat_gateway(params)
    CommonUtil.log("delete_nat_gateway", nat_gateway_json)

    # 删除vswitch
    params['vswitch_id'] = vswitch_json['VSwitchId']
    vswitch_json = vswitch.delete_vswitch(params)
    CommonUtil.log("delete_vswitch", vswitch_json)

    # 删除vpc
    vpc_json = vpc.delete_vpc(params)
    CommonUtil.log("delete_vpc", vpc_json)


if __name__ == "__main__":
```

```
    sys.exit(main())
```

3. 进入natgw_quick_start.py所在的目录，执行如下命令，创建NAT网关。

```
python natgw_quick_start.py
```

预期结果

系统显示类似如下：

```
-------------------------create_vpc-------------------------
{
 "ResourceGroupId": "rg-acfmxazxxxxxxxx",
 "RouteTableId": "vtb-uf6wzp25d8lkbxxxxxxxx",
 "VRouterId": "vrt-uf6di7voecmyqxxxxxxxx",
 "VpcId": "vpc-uf63cqupghmk1xxxxxxxx",
 "RequestId": "97D36E19-F789-424F-A473-660D63EF8CF9"
}

-------------------------create_vswitch-------------------------
{
 "VSwitchId": "vsw-uf6fovepnk4yexxxxxxxx",
 "RequestId": "18DA0E81-34A6-4877-9771-E2C4EEEBADD1"
}

-------------------------create_nat_gateway-------------------------
{
 "NatGatewayId": "ngw-uf6mfrcmzktstxxxxxxxx",
 "BandwidthPackageIds": {
  "BandwidthPackageId": []
 },
 "ForwardTableIds": {
  "ForwardTableId": [
   "ftb-uf6411str8n9sxxxxxxxx"
  ]
 },
 "RequestId": "B1F791C8-73B1-46C5-8A20-726A615BC627",
 "SnatTableIds": {
  "SnatTableId": [
   "stb-uf6t4eijvq3aexxxxxxxx"
  ]
 }
}

-------------------------describe_nat_gateway-------------------------
{
 "TotalCount": 1,
 "PageNumber": 1,
 "RequestId": "1F9303B1-4024-4A92-B67E-FB6BE1DC76D1",
 "PageSize": 10,
 "NatGateways": {
  "NatGateway": [
   {
    "Status": "Available",
    "BandwidthPackageIds": {
     "BandwidthPackageId": []
    },
    "VpcId": "vpc-uf63cqupghmk1xxxxxxxx",
    "Description": "",
    "ForwardTableIds": {
     "ForwardTableId": [
      "ftb-uf6411str8n9sxxxxxxxx"
```

```
        ]
      },
      "IpLists": {
        "IpList": []
      },
      "BusinessStatus": "Normal",
      "RegionId": "cn-shanghai",
      "CreationTime": "2019-04-24T09:09:12Z",
      "NatGatewayId": "ngw-uf6mfrcmzktstxxxxxxxx",
      "SnatTableIds": {
        "SnatTableId": [
          "stb-uf6t4eijvq3aexxxxxxxx"
        ]
      },
      "AutoPay": false,
      "InstanceChargeType": "PostPaid",
      "ExpiredTime": "",
      "Spec": "Small",
      "Name": ""
    }
  ]
 }
}

--------------------------delete_nat_gateway--------------------------
{
 "RequestId": "A0B71FE4-4756-4D91-899E-1DFA52D8615E"
}

--------------------------delete_vswitch--------------------------
{
 "RequestId": "F224307E-3DE4-4415-AE19-DDCF24695462"
}

--------------------------delete_vpc--------------------------
{
 "RequestId": "1BFFCBC3-7F83-436C-96E9-CA4A620072DA"
}
```

2.5.2 绑定和解绑EIP

本文介绍如何使用Alibaba Cloud SDK for Python从NAT网关上绑定或解绑一个弹性公网IP（EIP）。

前提条件

在使用Alibaba Cloud SDK for Python前，您需要完成以下准备工作：

- 使用Alibaba Cloud SDK for Python，您需要一个阿里云账号和访问密钥（AccessKey）。 请在阿里云控制台中的AccessKey管理页面上创建和查看您的AccessKey。
- 确保您已经安装了Alibaba Cloud SDK for Python，请参见aliyun-python-sdk-vpc 3.0.8。

- 下载阿里云专有网络Python SDK场景示例的VPC Python Example库。

  进入setup.py所在的目录，执行如下命令，完成环境初始化配置。

  ```
  python setup.py install
  ```

背景信息

本文代码示例中包含以下操作：

1. 在华东2上海地域创建一个VPC。
2. 在新建的VPC下创建一个VSwitch。
3. 在新建的VPC下创建一个NAT网关。
4. 在华东2上海地域创建一个EIP。
5. 将创建的EIP绑定到NAT网关。
6. 查询绑定到NAT网关的EIP。
7. 在华东2上海地域创建共享带宽实例。
8. 添加EIP到共享带宽实例。
9. 查询已创建的NAT网关。
10. 将EIP与NAT网关解绑。
11. 将EIP从共享带宽实例中移出。
12. 删除共享带宽实例。
13. 删除NAT网关。
14. 释放EIP。
15. 删除VSwitch。
16. 删除VPC。

操作步骤

1. 在下载的SDK目录中，打开$aliyun-openapi-python-sdk-examples\sdk_examples\examples\natgw文件夹。
2. 使用编辑器打开natgw_associate_eip.py文件，根据实际情况配置相关参数，保存退出。

   完整代码示例如下：

   ```
   #encoding=utf-8
   import sys
   import json
   import time

   from aliyunsdkcore.acs_exception.exceptions import ServerException, ClientException
   from aliyunsdkvpc.request.v20160428 import CreateNatGatewayRequest
   from aliyunsdkvpc.request.v20160428 import DeleteNatGatewayRequest
   from aliyunsdkvpc.request.v20160428 import DescribeNatGatewaysRequest
   from sdk_lib.sdk_vpc import Vpc
   ```

```
from sdk_lib.sdk_vswitch import VSwitch
from sdk_lib.sdk_eip import Eip
from sdk_lib.sdk_cbwp import CommonBandwidthPackage
from sdk_lib.common_util import CommonUtil
from sdk_lib.check_status import CheckStatus
from sdk_lib.exception import ExceptionHandler
from sdk_lib.consts import *

client = AcsClient(
    'accessKeyId',
    'accessSecret',
    'RegionId'
    #timeout = 35
)


class NatGateway(object):
    def __init__(self, client):
        self.client = client

    def create_nat_gateway(self, params):
        """
        create_nat_gateway: 创建nat gateway
        """
        try:
            request = CreateNatGatewayRequest.CreateNatGatewayRequest()
            request.set_VpcId(params['vpc_id'])
            response = client.do_action_with_exception(request)
            response_json = json.loads(response)
            # 判断Nat Gateway状态是否可用
            if CheckStatus.check_status(TIME_DEFAULT_OUT, DEFAULT_TIME,
                                        self.describe_nat_gateway_status,
                                        AVAILABLE, response_json['NatGatewayId']):
                return response_json
        except ServerException as e:
            ExceptionHandler.server_exception(e)
        except ClientException as e:
            ExceptionHandler.client_exception(e)

    def describe_nat_gateway(self, nat_gateway_id):
        """
        describe_nat_gateway: 查询指定地域已创建的nat gateway的信息
        """
        try:
            request = DescribeNatGatewaysRequest.DescribeNatGatewaysRequest()
            request.set_NatGatewayId(nat_gateway_id)
            response = client.do_action_with_exception(request)
            response_json = json.loads(response)
            return response_json
        except ServerException as e:
            ExceptionHandler.server_exception(e)
        except ClientException as e:
            ExceptionHandler.client_exception(e)

    def delete_nat_gateway(self, params):
        """
        delete_nat_gateway: 删除nat gateway
        """
        try:
            request = DeleteNatGatewayRequest.DeleteNatGatewayRequest()
            request.set_NatGatewayId(params['nat_gateway_id'])
            response = client.do_action_with_exception(request)
            response_json = json.loads(response)
            # 判断Nat Gateway状态是否可用
```

```
            if CheckStatus.check_status(TIME_DEFAULT_OUT, DEFAULT_TIME * 5,
                                        self.describe_nat_gateway_status,
                                        '', params['nat_gateway_id']):
                return response_json
        except ServerException as e:
            ExceptionHandler.server_exception(e)
        except ClientException as e:
            ExceptionHandler.client_exception(e)

    def describe_nat_gateway_status(self, nat_gateway_id):
        """
        describe_nat_gateway_status: 查询指定地域已创建的nat gateway的状态
        """
        response = self.describe_nat_gateway(nat_gateway_id)
        if len(response["NatGateways"]["NatGateway"]) == 0:
            return ''
        return response["NatGateways"]["NatGateway"][0]['Status']


def main():
    vpc = Vpc(client)
    vswitch = VSwitch(client)
    eip = Eip(client)
    cbwp = CommonBandwidthPackage(client)
    nat_gateway = NatGateway(client)

    params = {}

    # 创建vpc
    vpc_json = vpc.create_vpc()
    CommonUtil.log("create_vpc", vpc_json)

    # 创建vswitch
    params['vpc_id'] = vpc_json['VpcId']
    params['zone_id'] = "cn-shanghai-d"
    params['cidr_block'] = "172.16.1.0/24"
    vswitch_json = vswitch.create_vswitch(params)
    CommonUtil.log("create_vswitch", vswitch_json)

    # 创建natgw
    nat_gateway_json = nat_gateway.create_nat_gateway(params)
    CommonUtil.log("create_nat_gateway", nat_gateway_json)

    # 创建EIP
    eip_response_json = eip.allocate_eip_address(params)
    CommonUtil.log("allocate_eip_address", eip_response_json)
    params['allocation_id'] = eip_response_json["AllocationId"]

    # 绑定EIP到NAT网关
    params['instance_id'] = nat_gateway_json['NatGatewayId']
    params['allocation_id'] = eip_response_json["AllocationId"]
    params['instance_type'] = 'Nat'
    eip_response_json = eip.associate_eip_address(params)
    CommonUtil.log("associate_eip_address eip", eip_response_json)

    # 查询EIP
    eip_response_json = eip.describe_eip_address(params['allocation_id'])
    CommonUtil.log("describe_eip_address", eip_response_json)

    # 创建带宽包
    params['bandwidth'] = BANDWIDTH_10
    cbwp_repsonse_json = cbwp.create_common_bandwidth_package(params)
    CommonUtil.log("create_common_bandwidth_package", cbwp_repsonse_json)
```

```
    # 添加EIP到共享带宽包中
    params['ip_instance_id'] = params['allocation_id']
    params['bandwidth_package_id'] = cbwp_repsonse_json['BandwidthPackageId']
    cbwp_repsonse_json = cbwp.add_common_bandwidth_packageIp(params)
    CommonUtil.log("add_common_bandwidth_packageIp", cbwp_repsonse_json)

    # 查询natgw
    params['nat_gateway_id'] = nat_gateway_json['NatGatewayId']
    nat_gateway_json = nat_gateway.describe_nat_gateway(params['nat_gateway_id'])
    CommonUtil.log("describe_nat_gateway", nat_gateway_json)

    # 解绑EIP
    eip_response_json = eip.unassociate_eip_address(params)
    CommonUtil.log("unassociate_eip_address nat", eip_response_json)

    # 移除共享带宽包中的EIP（
    cbwp_repsonse_json = cbwp.remove_common_bandwidth_packageIp(params)
    CommonUtil.log("remove_common_bandwidth_packageIp", cbwp_repsonse_json)

    # 删除共享带宽包
    params['force'] = True
    cbwp_repsonse_json = cbwp.delete_common_bandwidth_package(params)
    CommonUtil.log("delete_common_bandwidth_package", cbwp_repsonse_json)

    # 删除natgw
    nat_gateway_json = nat_gateway.delete_nat_gateway(params)
    CommonUtil.log("delete_nat_gateway", nat_gateway_json)

    # 释放EIP
    eip_response_json = eip.release_eip_address(params)
    CommonUtil.log("release_eip_address", eip_response_json)

    # 删除vswitch
    params['vswitch_id'] = vswitch_json['VSwitchId']
    vswitch_json = vswitch.delete_vswitch(params)
    CommonUtil.log("delete_vswitch", vswitch_json)

    # 删除vpc
    vpc_json = vpc.delete_vpc(params)
    CommonUtil.log("delete_vpc", vpc_json)


if __name__ == "__main__":
    sys.exit(main())
```

3. 进入natgw_associate_eip.py所在的目录，执行如下命令，绑定和解绑EIP。

```
python natgw_associate_eip.py
```

预期结果

系统显示类似如下：

```
--------------------------create_vpc--------------------------
{
  "ResourceGroupId": "rg-acfmxazxxxxxxxx",
  "RouteTableId": "vtb-uf6agemvkcmd8xxxxxxxx",
  "VRouterId": "vrt-uf6r7lqtsv65dxxxxxxxx",
  "VpcId": "vpc-uf6mqfqx8vjmoxxxxxxxx",
  "RequestId": "ADF806C6-FCD6-4E46-B8E3-72C2BE895344"
}
```

```
--------------------------create_vswitch--------------------------
{
 "VSwitchId": "vsw-uf6rm6add6w89xxxxxxxx",
 "RequestId": "897FFCC1-E6BA-484E-A245-C5DAEBBA269C"
}

-------------------------create_nat_gateway-------------------------
{
 "NatGatewayId": "ngw-uf681h38pbvlyxxxxxxxx",
 "BandwidthPackageIds": {
  "BandwidthPackageId": []
 },
 "ForwardTableIds": {
  "ForwardTableId": [
   "ftb-uf6jd0vbyao2dxxxxxxxx"
  ]
 },
 "RequestId": "7C7CD3CB-041A-4B80-80B4-8BF8D5EF0D26",
 "SnatTableIds": {
  "SnatTableId": [
   "stb-uf6uj997htg3uxxxxxxxx"
  ]
 }
}

-------------------------allocate_eip_address-------------------------
{
 "EipAddress": "106.xx.xx.129",
 "ResourceGroupId": "rg-acfmxazxxxxxxxx",
 "RequestId": "DB795B99-1CEA-4FC1-9CE3-9DE2B977BF02",
 "AllocationId": "eip-uf62tf8y4uyacxxxxxxxx"
}

-------------------------associate_eip_address eip-------------------------
{
 "RequestId": "443D7060-B716-4193-A44D-23FE762004F8"
}

-------------------------describe_eip_address-------------------------
{
 "TotalCount": 1,
 "PageNumber": 1,
 "PageSize": 10,
 "EipAddresses": {
  "EipAddress": [
   {
    "ISP": "BGP",
    "ExpiredTime": "",
    "InternetChargeType": "PayByBandwidth",
    "IpAddress": "106.xx.xx.129",
    "AllocationId": "eip-uf62tf8y4uyacxxxxxxxx",
    "PrivateIpAddress": "",
    "Status": "InUse",
    "BandwidthPackageId": "",
    "InstanceId": "ngw-uf681h38pbvlyxxxxxxxx",
    "InstanceRegionId": "cn-shanghai",
    "RegionId": "cn-shanghai",
    "AvailableRegions": {
     "AvailableRegion": [
      "cn-shanghai"
     ]
    },
    "ResourceGroupId": "rg-acfmxazxxxxxxxx",
    "HasReservationData": false,
```

```
      "InstanceType": "Nat",
      "AllocationTime": "2019-04-24T10:03:08Z",
      "Name": "",
      "OperationLocks": {
       "LockReason": []
      },
      "Mode": "NAT",
      "BandwidthPackageType": "",
      "BandwidthPackageBandwidth": "",
      "Bandwidth": "5",
      "HDMonitorStatus": "OFF",
      "ChargeType": "PostPaid",
      "SecondLimited": false,
      "Descritpion": ""
    }
   ]
 },
 "RequestId": "F0AEE605-14AD-4ADD-980C-4B508CE7EE4B"
}

--------------------------create_common_bandwidth_package----------------------
-----
{
 "ResourceGroupId": "rg-acfmxazxxxxxxxx",
 "BandwidthPackageId": "cbwp-uf6dmfvq0gzzgxxxxxxxx",
 "RequestId": "D5E02777-2A72-42EC-8308-5D4B6E56D900"
}

--------------------------add_common_bandwidth_packageIp-----------------------
----
{
 "RequestId": "3B4CD99C-6E59-4DA2-9256-EACF3F699412"
}

--------------------------describe_nat_gateway---------------------------
{
 "TotalCount": 1,
 "PageNumber": 1,
 "RequestId": "A07498A0-4D60-4E0F-A7DE-3A832B174D59",
 "PageSize": 10,
 "NatGateways": {
  "NatGateway": [
   {
    "Status": "Available",
    "BandwidthPackageIds": {
     "BandwidthPackageId": []
    },
    "VpcId": "vpc-uf6mqfqx8vjmoxxxxxxxx",
    "Description": "",
    "ForwardTableIds": {
     "ForwardTableId": [
      "ftb-uf6jd0vbyao2dxxxxxxxx"
     ]
    },
    "IpLists": {
     "IpList": [
      {
       "UsingStatus": "Idle",
       "IpAddress": "106.xx.xx.129",
       "AllocationId": "eip-uf62tf8y4uyacxxxxxxxx"
      }
     ]
    },
    "BusinessStatus": "Normal",
```

```
      "RegionId": "cn-shanghai",
      "CreationTime": "2019-04-24T10:03:05Z",
      "NatGatewayId": "ngw-uf681h38pbvlyxxxxxxxx",
      "SnatTableIds": {
        "SnatTableId": [
          "stb-uf6uj997htg3uxxxxxxxx"
        ]
      },
      "AutoPay": false,
      "InstanceChargeType": "PostPaid",
      "ExpiredTime": "",
      "Spec": "Small",
      "Name": ""
    }
  ]
 }
}

---------------------------unassociate_eip_address nat--------------------------
-
{
 "RequestId": "92CB670E-239D-4659-B91F-E0565D5C0F2D"
}

---------------------------remove_common_bandwidth_packageIp--------------------
-------
{
 "RequestId": "A58E9647-6761-4CA3-8786-3CD4E7D2A7AB"
}

---------------------------delete_common_bandwidth_package----------------------
-----
{
 "RequestId": "4AA428BC-B72F-4567-94B8-AC2398EF7529"
}

---------------------------delete_nat_gateway---------------------------
{
 "RequestId": "EC6C5D04-AF7D-4560-A30E-80EC141D174D"
}

---------------------------release_eip_address---------------------------
{
 "RequestId": "9B1380B3-EE97-49BD-88FE-DBF356304208"
}

---------------------------delete_vswitch---------------------------
{
 "RequestId": "A9A1D63E-5709-4B98-90BF-9069AA264230"
}

---------------------------delete_vpc---------------------------
{
 "RequestId": "3B687C37-5315-4E0B-BE13-103BB287A80D"
}
```

2.5.3 创建DNAT条目

本文介绍如何使用Alibaba Cloud SDK for Python创建DNAT条目。

前提条件

在使用Alibaba Cloud SDK for Python前，您需要完成以下准备工作：

- 使用Alibaba Cloud SDK for Python，您需要一个阿里云账号和访问密钥（AccessKey）。 请在阿里云控制台中的AccessKey管理页面上创建和查看您的AccessKey。
- 确保您已经安装了Alibaba Cloud SDK for Python，请参见aliyun-python-sdk-vpc 3.0.8。
- 下载阿里云专有网络Python SDK场景示例的VPC Python Example库。

  进入setup.py所在的目录，执行如下命令，完成环境初始化配置。

```
python setup.py install
```

背景信息

本文代码示例中包含以下操作：

1. 在华东2上海地域创建一个VPC。
2. 在新建的VPC下创建一个VSwitch。
3. 在新建的VPC下创建一个NAT网关。
4. 在华东2上海地域创建一个EIP。
5. 将创建的EIP绑定到NAT网关。
6. 创建DNAT条目。
7. 查询绑定到NAT网关的EIP。
8. 查询NAT网关。
9. 删除DNAT条目。
10. 将EIP与NAT网关解绑。
11. 删除NAT网关。
12. 释放EIP。
13. 删除VSwitch。
14. 删除VPC。

操作步骤

1. 在下载的SDK目录中，打开$aliyun-openapi-python-sdk-examples\sdk_examples\examples\natgw文件夹。
2. 使用编辑器打开natgw_dnat.py文件，根据实际情况配置相关参数，保存退出。

   完整代码示例如下：

```
#encoding=utf-8
import sys
import json
import time

from aliyunsdkcore.acs_exception.exceptions import ServerException, ClientException
```

```
from aliyunsdkvpc.request.v20160428 import CreateNatGatewayRequest
from aliyunsdkvpc.request.v20160428 import DeleteNatGatewayRequest
from aliyunsdkvpc.request.v20160428 import DescribeNatGatewaysRequest
from aliyunsdkvpc.request.v20160428 import CreateForwardEntryRequest
from aliyunsdkvpc.request.v20160428 import DescribeForwardTableEntriesRequest
from aliyunsdkvpc.request.v20160428 import DeleteForwardEntryRequest
from sdk_lib.sdk_vpc import Vpc
from sdk_lib.sdk_vswitch import VSwitch
from sdk_lib.sdk_eip import Eip
from sdk_lib.sdk_cbwp import CommonBandwidthPackage
from sdk_lib.common_util import CommonUtil
from sdk_lib.check_status import CheckStatus
from sdk_lib.exception import ExceptionHandler
from sdk_lib.consts import *

client = AcsClient(
    'accessKeyId',
    'accessSecret',
    'RegionId'
    #timeout = 35
)


class NatGateway(object):
    def __init__(self, client):
        self.client = client

    def create_nat_gateway(self, params):
        """
        create_nat_gateway: 创建nat gateway
        """
        try:
            request = CreateNatGatewayRequest.CreateNatGatewayRequest()
            request.set_VpcId(params['vpc_id'])
            response = client.do_action_with_exception(request)
            response_json = json.loads(response)
            # 判断Nat Gateway状态是否可用
            if CheckStatus.check_status(TIME_DEFAULT_OUT, DEFAULT_TIME,
                                        self.describe_nat_gateway_status,
                                        AVAILABLE, response_json['NatGatewayId']):
                return response_json
        except ServerException as e:
            ExceptionHandler.server_exception(e)
        except ClientException as e:
            ExceptionHandler.client_exception(e)

    def describe_nat_gateway(self, nat_gateway_id):
        """
        describe_nat_gateway: 查询指定地域已创建的nat gateway的信息
        """
        try:
            request = DescribeNatGatewaysRequest.DescribeNatGatewaysRequest()
            request.set_NatGatewayId(nat_gateway_id)
            response = client.do_action_with_exception(request)
            response_json = json.loads(response)
            return response_json
        except ServerException as e:
            ExceptionHandler.server_exception(e)
        except ClientException as e:
            ExceptionHandler.client_exception(e)

    def delete_nat_gateway(self, params):
        """
        delete_nat_gateway: 删除nat gateway
```

```
        """
        try:
            request = DeleteNatGatewayRequest.DeleteNatGatewayRequest()
            request.set_NatGatewayId(params['nat_gateway_id'])
            response = client.do_action_with_exception(request)
            response_json = json.loads(response)
            # 判断Nat Gateway状态是否可用
            if CheckStatus.check_status(TIME_DEFAULT_OUT, DEFAULT_TIME * 5,
                                        self.describe_nat_gateway_status,
                                        '', params['nat_gateway_id']):
                return response_json
        except ServerException as e:
            ExceptionHandler.server_exception(e)
        except ClientException as e:
            ExceptionHandler.client_exception(e)

    def describe_nat_gateway_status(self, nat_gateway_id):
        """
        describe_nat_gateway_status: 查询指定地域已创建的nat gateway的状态
        """
        response = self.describe_nat_gateway(nat_gateway_id)
        if len(response["NatGateways"]["NatGateway"]) == 0:
            return ''
        return response["NatGateways"]["NatGateway"][0]['Status']

    def create_forward_entry(self, params):
        """
        create_forward_entry: 创建forward entry
        """
        try:
            request = CreateForwardEntryRequest.CreateForwardEntryRequest()
            request.set_ForwardTableId(params['forward_table_id'])
            request.set_ExternalIp(params['external_ip'])
            request.set_IpProtocol(params['ip_protocol'])
            request.set_ExternalPort(params['external_port'])
            request.set_InternalIp(params['internal_ip'])
            request.set_InternalPort(params['internal_port'])
            response = client.do_action_with_exception(request)
            response_json = json.loads(response)
            # 判断Forward Entry状态是否可用
            if CheckStatus.check_status(TIME_DEFAULT_OUT, DEFAULT_TIME,
                                        self.describe_forward_status,
                                        AVAILABLE, params['forward_table_id']):
                return response_json
        except ServerException as e:
            ExceptionHandler.server_exception(e)
        except ClientException as e:
            ExceptionHandler.client_exception(e)

    def describe_forward(self, forward_table_id):
        """
        describe_forward: 查询指定地域已创建的dnat的信息
        """
        try:
            request = DescribeForwardTableEntriesRequest.DescribeForwardTable
EntriesRequest()
            request.set_ForwardTableId(forward_table_id)
            response = client.do_action_with_exception(request)
            response_json = json.loads(response)
            return response_json
        except ServerException as e:
            ExceptionHandler.server_exception(e)
        except ClientException as e:
            ExceptionHandler.client_exception(e)
```

```
    def describe_forward_status(self, forward_table_id):
        """
        describe_forward_status: 查询指定地域已创建的dnat的状态
        """
        response = self.describe_forward(forward_table_id)
        if len(response["ForwardTableEntries"]["ForwardTableEntry"]) == 0:
            return ''
        return response["ForwardTableEntries"]["ForwardTableEntry"][0]['Status']

    def delete_forward_entry(self, params):
        """
        delete_forward_entry: 删除forward entry
        """
        try:
            request = DeleteForwardEntryRequest.DeleteForwardEntryRequest()
            request.set_ForwardTableId(params['forward_table_id'])
            request.set_ForwardEntryId(params['forward_entry_id'])
            response = client.do_action_with_exception(request)
            response_json = json.loads(response)
            # 判断Forward Entry状态是否可用
            if CheckStatus.check_status(TIME_DEFAULT_OUT, DEFAULT_TIME * 5,
                                        self.describe_forward_status,
                                        '', params['forward_table_id']):
                return response_json
        except ServerException as e:
            ExceptionHandler.server_exception(e)
        except ClientException as e:
            ExceptionHandler.client_exception(e)


def main():
    vpc = Vpc(client)
    vswitch = VSwitch(client)
    eip = Eip(client)
    cbwp = CommonBandwidthPackage(client)
    nat_gateway = NatGateway(client)

    params = {}

    # 创建vpc
    vpc_json = vpc.create_vpc()
    CommonUtil.log("create_vpc", vpc_json)

    # 创建vswitch
    params['vpc_id'] = vpc_json['VpcId']
    params['zone_id'] = "cn-hangzhou-d"
    params['cidr_block'] = "172.16.1.0/24"
    vswitch_json = vswitch.create_vswitch(params)
    CommonUtil.log("create_vswitch", vswitch_json)
    params['vswitch_id'] = vswitch_json['VSwitchId']

    # 创建natgw
    nat_gateway_json = nat_gateway.create_nat_gateway(params)
    CommonUtil.log("create_nat_gateway", nat_gateway_json)

    # 创建EIP
    eip_response_json = eip.allocate_eip_address(params)
    CommonUtil.log("allocate_eip_address", eip_response_json)
    params['allocation_id'] = eip_response_json["AllocationId"]
    params['external_ip'] = eip_response_json['EipAddress']

    # 绑定EIP到NAT网关
    params['instance_id'] = nat_gateway_json['NatGatewayId']
```

```
    params['allocation_id'] = eip_response_json["AllocationId"]
    params['instance_type'] = 'Nat'
    eip_response_json = eip.associate_eip_address(params)
    CommonUtil.log("associate_eip_address eip", eip_response_json)

    # 创建forward entry
    params['forward_table_id'] = nat_gateway_json['ForwardTableIds']['ForwardTableId
']][0]
    params['ip_protocol'] = 'tcp'
    params['external_port'] = '8080'
    params['internal_port'] = '80'
    params['internal_ip'] = '172.16.1.0'
    forward_entry_json = nat_gateway.create_forward_entry(params)
    CommonUtil.log("create_forward_entry", forward_entry_json)

    # 查询EIP
    eip_response_json = eip.describe_eip_address(params['allocation_id'])
    CommonUtil.log("describe_eip_address", eip_response_json)

    # 查询natgw
    params['nat_gateway_id'] = nat_gateway_json['NatGatewayId']
    nat_gateway_json = nat_gateway.describe_nat_gateway(params['nat_gateway_id'])
    CommonUtil.log("describe_nat_gateway", nat_gateway_json)

    # 删除forward entry
    params['forward_entry_id'] = forward_entry_json['ForwardEntryId']
    forward_entry_json = nat_gateway.delete_forward_entry(params)
    CommonUtil.log("delete_forward_entry", forward_entry_json)

    # 解绑EIP
    eip_response_json = eip.unassociate_eip_address(params)
    CommonUtil.log("unassociate_eip_address nat", eip_response_json)

    # 删除natgw
    nat_gateway_json = nat_gateway.delete_nat_gateway(params)
    CommonUtil.log("delete_nat_gateway", nat_gateway_json)

    # 释放EIP
    eip_response_json = eip.release_eip_address(params)
    CommonUtil.log("release_eip_address", eip_response_json)

    # 删除vswitch
    params['vswitch_id'] = vswitch_json['VSwitchId']
    vswitch_json = vswitch.delete_vswitch(params)
    CommonUtil.log("delete_vswitch", vswitch_json)

    # 删除vpc
    vpc_json = vpc.delete_vpc(params)
    CommonUtil.log("delete_vpc", vpc_json)


if __name__ == "__main__":
    sys.exit(main())
```

3. 进入natgw_dnat.py所在的目录，执行如下命令，创建DNAT条目。

```
python natgw_dnat.py
```

预期结果

系统显示类似如下：

```
--------------------------create_vpc--------------------------
{
 "ResourceGroupId": "rg-acfmxazxxxxxxxx",
 "RouteTableId": "vtb-uf63rln6gbb50xxxxxxxx",
 "VRouterId": "vrt-uf6p1hfo0ho8gxxxxxxxx",
 "VpcId": "vpc-uf6c3r8yca7dhxxxxxxxx",
 "RequestId": "1F97FC59-77DF-4D76-BE62-0A13EB4E614C"
}

--------------------------create_vswitch--------------------------
{
 "VSwitchId": "vsw-uf6liy66d9ssuxxxxxxxx",
 "RequestId": "88CCCFED-1448-49D2-8550-71952981A47A"
}

--------------------------create_nat_gateway--------------------------
{
 "NatGatewayId": "ngw-uf6aolgwhssvsxxxxxxxx",
 "BandwidthPackageIds": {
  "BandwidthPackageId": []
 },
 "ForwardTableIds": {
  "ForwardTableId": [
   "ftb-uf6unjiun4i12xxxxxxxx"
  ]
 },
 "RequestId": "62A58351-D608-43A4-849E-1E177E917BEA",
 "SnatTableIds": {
  "SnatTableId": [
   "stb-uf65utljwcdkpxxxxxxxx"
  ]
 }
}

--------------------------allocate_eip_address--------------------------
{
 "EipAddress": "101.xx.xx.110",
 "ResourceGroupId": "rg-acfmxazxxxxxxxx",
 "RequestId": "0565295E-2F49-4511-93BC-747A2D19A6BD",
 "AllocationId": "eip-uf683xrl32ge8xxxxxxxx"
}

--------------------------associate_eip_address eip--------------------------
{
 "RequestId": "8759FCE8-F8C2-4372-91D5-7A25D43FD78C"
}

--------------------------create_forward_entry--------------------------
{
 "ForwardEntryId": "fwd-uf6ng3wt8sfwmxxxxxxxx",
 "RequestId": "CC81BCF6-2F64-40CF-85B0-676A83AC3902"
}

--------------------------describe_eip_address--------------------------
{
 "TotalCount": 1,
 "PageNumber": 1,
 "PageSize": 10,
 "EipAddresses": {
  "EipAddress": [
   {
```

```
      "ISP": "BGP",
      "ExpiredTime": "",
      "InternetChargeType": "PayByBandwidth",
      "IpAddress": "101.xx.xx.110",
      "AllocationId": "eip-uf683xrl32ge8xxxxxxxx",
      "PrivateIpAddress": "",
      "Status": "InUse",
      "BandwidthPackageId": "",
      "InstanceId": "ngw-uf6aolgwhssvsxxxxxxxxx",
      "InstanceRegionId": "cn-shanghai",
      "RegionId": "cn-shanghai",
      "AvailableRegions": {
       "AvailableRegion": [
        "cn-shanghai"
       ]
      },
      "ResourceGroupId": "rg-acfmxazxxxxxxxx",
      "HasReservationData": false,
      "InstanceType": "Nat",
      "AllocationTime": "2019-04-24T10:56:53Z",
      "Name": "",
      "OperationLocks": {
       "LockReason": []
      },
      "Mode": "NAT",
      "BandwidthPackageType": "",
      "BandwidthPackageBandwidth": "",
      "Bandwidth": "5",
      "HDMonitorStatus": "OFF",
      "ChargeType": "PostPaid",
      "SecondLimited": false,
      "Descritpion": ""
     }
    ]
  },
  "RequestId": "CD2B3613-2A99-4687-9C23-A8E9F1F03048"
}

--------------------------describe_nat_gateway--------------------------
{
  "TotalCount": 1,
  "PageNumber": 1,
  "RequestId": "D7519663-8D3B-4CC5-894F-A6798C89688D",
  "PageSize": 10,
  "NatGateways": {
   "NatGateway": [
    {
     "Status": "Available",
     "BandwidthPackageIds": {
      "BandwidthPackageId": []
     },
     "VpcId": "vpc-uf6c3r8yca7dhxxxxxxxx",
     "Description": "",
     "ForwardTableIds": {
      "ForwardTableId": [
       "ftb-uf6unjiun4i12xxxxxxxx"
      ]
     },
     "IpLists": {
      "IpList": [
       {
        "UsingStatus": "UsedByForwardTable",
        "IpAddress": "101.xx.xx.110",
        "AllocationId": "eip-uf683xrl32ge8xxxxxxxx"
```

```
        }
       ]
      },
      "BusinessStatus": "Normal",
      "RegionId": "cn-shanghai",
      "CreationTime": "2019-04-24T10:56:50Z",
      "NatGatewayId": "ngw-uf6aolgwhssvsxxxxxxxx",
      "SnatTableIds": {
       "SnatTableId": [
        "stb-uf65utljwcdkpxxxxxxxx"
       ]
      },
      "AutoPay": false,
      "InstanceChargeType": "PostPaid",
      "ExpiredTime": "",
      "Spec": "Small",
      "Name": ""
     }
    ]
   }
}

--------------------------delete_forward_entry--------------------------
{
 "RequestId": "32C76D08-5738-4B07-A638-ACE5F5F5220E"
}

--------------------------unassociate_eip_address nat-------------------------
-
{
 "RequestId": "AE686920-2CD1-4850-AADC-C249484D4B1A"
}

--------------------------delete_nat_gateway--------------------------
{
 "RequestId": "FEBB1E7A-BA5B-4445-B2AB-5B828C17BBE6"
}

--------------------------release_eip_address--------------------------
{
 "RequestId": "812D5E78-5113-4B92-892D-0B293BAD66F6"
}

--------------------------delete_vswitch--------------------------
{
 "RequestId": "8E13EEE4-21B5-4280-B46B-5C168736DC3A"
}

--------------------------delete_vpc--------------------------
{
 "RequestId": "DCBA91E7-F355-4EB6-83E3-27F2E68A8435"
}
```

2.5.4 创建SNAT条目

本文介绍如何使用Alibaba Cloud SDK for Python创建SNAT条目。

前提条件

在使用Alibaba Cloud SDK for Python前，您需要完成以下准备工作：

- 使用Alibaba Cloud SDK for Python，您需要一个阿里云账号和访问密钥（AccessKey）。 请在阿里云控制台中的AccessKey管理页面上创建和查看您的AccessKey。
- 确保您已经安装了Alibaba Cloud SDK for Python，请参见aliyun-python-sdk-vpc 3.0.8。
- 下载阿里云专有网络Python SDK场景示例的VPC Python Example库。

  进入setup.py所在的目录，执行如下命令，完成环境初始化配置。

```
python setup.py install
```

背景信息

本文代码示例中包含以下操作：

1. 在华东2上海地域创建一个VPC。
2. 在新建的VPC下创建一个VSwitch。
3. 在新建的VPC下创建一个NAT网关。
4. 在华东2上海地域创建一个EIP。
5. 将创建的EIP绑定到NAT网关。
6. 创建SNAT条目。
7. 查询绑定到NAT网关的EIP。
8. 查询NAT网关。
9. 删除SNAT条目。
10. 将EIP与NAT网关解绑。
11. 删除NAT网关。
12. 释放EIP。
13. 删除VSwitch。
14. 删除VPC。

操作步骤

1. 在下载的SDK目录中，打开$aliyun-openapi-python-sdk-examples\sdk_examples\examples\natgw文件夹。
2. 使用编辑器打开natgw_snat.py文件，根据实际情况配置相关参数，保存退出。

   完整代码示例如下：

```
#encoding=utf-8
import sys
import json
import time

from aliyunsdkcore.acs_exception.exceptions import ServerException, ClientException
from aliyunsdkvpc.request.v20160428 import CreateNatGatewayRequest
from aliyunsdkvpc.request.v20160428 import DeleteNatGatewayRequest
```

```
from aliyunsdkvpc.request.v20160428 import DescribeNatGatewaysRequest
from aliyunsdkvpc.request.v20160428 import CreateSnatEntryRequest
from aliyunsdkvpc.request.v20160428 import DescribeSnatTableEntriesRequest
from aliyunsdkvpc.request.v20160428 import DeleteSnatEntryRequest
from sdk_lib.sdk_vpc import Vpc
from sdk_lib.sdk_vswitch import VSwitch
from sdk_lib.sdk_eip import Eip
from sdk_lib.sdk_cbwp import CommonBandwidthPackage
from sdk_lib.common_util import CommonUtil
from sdk_lib.check_status import CheckStatus
from sdk_lib.exception import ExceptionHandler
from sdk_lib.consts import *

client = ACS_CLIENT = AcsClient(
    'accessKeyId',
    'accessSecret',
    'RegionId'
    #timeout = 35
)


class NatGateway(object):
    def __init__(self, client):
        self.client = client

    def create_nat_gateway(self, params):
        """
        create_nat_gateway: 创建nat gateway
        """
        try:
            request = CreateNatGatewayRequest.CreateNatGatewayRequest()
            request.set_VpcId(params['vpc_id'])
            response = client.do_action_with_exception(request)
            response_json = json.loads(response)
            # 判断Nat Gateway状态是否可用
            if CheckStatus.check_status(TIME_DEFAULT_OUT, DEFAULT_TIME,
                                        self.describe_nat_gateway_status,
                                        AVAILABLE, response_json['NatGatewayId']):
                return response_json
        except ServerException as e:
            ExceptionHandler.server_exception(e)
        except ClientException as e:
            ExceptionHandler.client_exception(e)

    def describe_nat_gateway(self, nat_gateway_id):
        """
        describe_nat_gateway: 查询指定地域已创建的nat gateway的信息
        """
        try:
            request = DescribeNatGatewaysRequest.DescribeNatGatewaysRequest()
            request.set_NatGatewayId(nat_gateway_id)
            response = client.do_action_with_exception(request)
            response_json = json.loads(response)
            return response_json
        except ServerException as e:
            ExceptionHandler.server_exception(e)
        except ClientException as e:
            ExceptionHandler.client_exception(e)

    def delete_nat_gateway(self, params):
        """
        delete_nat_gateway: 删除nat gateway
        """
        try:
```

```
        request = DeleteNatGatewayRequest.DeleteNatGatewayRequest()
        request.set_NatGatewayId(params['nat_gateway_id'])
        response = client.do_action_with_exception(request)
        response_json = json.loads(response)
        # 判断Nat Gateway状态是否可用
        if CheckStatus.check_status(TIME_DEFAULT_OUT, DEFAULT_TIME * 5,
                                    self.describe_nat_gateway_status,
                                    '', params['nat_gateway_id']):
            return response_json
    except ServerException as e:
        ExceptionHandler.server_exception(e)
    except ClientException as e:
        ExceptionHandler.client_exception(e)

def describe_nat_gateway_status(self, nat_gateway_id):
    """
    describe_nat_gateway_status: 查询指定地域已创建的nat gateway的状态
    """
    response = self.describe_nat_gateway(nat_gateway_id)
    if len(response["NatGateways"]["NatGateway"]) == 0:
        return ''
    return response["NatGateways"]["NatGateway"][0]['Status']

def create_snat_entry(self, params):
    """
    describe_snat: 创建snat entry
    """
    try:
        request = CreateSnatEntryRequest.CreateSnatEntryRequest()
        request.set_SnatTableId(params['snat_table_id'])
        request.set_SourceVSwitchId(params['vswitch_id'])
        request.set_SnatIp(params['snat_ip'])
        response = client.do_action_with_exception(request)
        response_json = json.loads(response)
        # 判断Snat Entry状态是否可用
        if CheckStatus.check_status(TIME_DEFAULT_OUT, DEFAULT_TIME,
                                    self.describe_snat_status,
                                    AVAILABLE, params['snat_table_id']):
            return response_json
    except ServerException as e:
        ExceptionHandler.server_exception(e)
    except ClientException as e:
        ExceptionHandler.client_exception(e)

def describe_snat(self, snat_table_id):
    """
    describe_snat: 查询指定地域已创建的snat的信息l
    """
    try:
        request = DescribeSnatTableEntriesRequest.DescribeSnatTableEntriesRequest()
        request.set_SnatTableId(snat_table_id)
        response = client.do_action_with_exception(request)
        response_json = json.loads(response)
        return response_json
    except ServerException as e:
        ExceptionHandler.server_exception(e)
    except ClientException as e:
        ExceptionHandler.client_exception(e)

def describe_snat_status(self, snat_table_id):
    """
    describe_snat_status: 查询指定地域已创建的snat的状态
    """
    response = self.describe_snat(snat_table_id)
```

```
        if len(response["SnatTableEntries"]["SnatTableEntry"]) == 0:
            return ''
        return response["SnatTableEntries"]["SnatTableEntry"][0]['Status']

    def delete_snat_entry(self, params):
        """
        delete_snat_entry: 删除snat entry
        """
        try:
            request = DeleteSnatEntryRequest.DeleteSnatEntryRequest()
            request.set_SnatTableId(params['snat_table_id'])
            request.set_SnatEntryId(params['snat_entry_id'])
            response = client.do_action_with_exception(request)
            response_json = json.loads(response)
            # 判断Snat Entry状态是否可用
            if CheckStatus.check_status(TIME_DEFAULT_OUT, DEFAULT_TIME * 5,
                                        self.describe_snat_status,
                                        '', params['snat_table_id']):
                return response_json
        except ServerException as e:
            ExceptionHandler.server_exception(e)
        except ClientException as e:
            ExceptionHandler.client_exception(e)


def main():
    vpc = Vpc(client)
    vswitch = VSwitch(client)
    eip = Eip(client)
    cbwp = CommonBandwidthPackage(client)
    nat_gateway = NatGateway(client)

    params = {}

    # 创建vpc
    vpc_json = vpc.create_vpc()
    CommonUtil.log("create_vpc", vpc_json)

    # 创建vswitch
    params['vpc_id'] = vpc_json['VpcId']
    params['zone_id'] = "cn-hangzhou-d"
    params['cidr_block'] = "172.16.1.0/24"
    vswitch_json = vswitch.create_vswitch(params)
    CommonUtil.log("create_vswitch", vswitch_json)
    params['vswitch_id'] = vswitch_json['VSwitchId']

    # 创建natgw
    nat_gateway_json = nat_gateway.create_nat_gateway(params)
    CommonUtil.log("create_nat_gateway", nat_gateway_json)

    # 创建EIP
    eip_response_json = eip.allocate_eip_address(params)
    CommonUtil.log("allocate_eip_address", eip_response_json)
    params['allocation_id'] = eip_response_json["AllocationId"]
    params['snat_ip'] = eip_response_json['EipAddress']

    # 绑定EIP到NAT网关
    params['instance_id'] = nat_gateway_json['NatGatewayId']
    params['allocation_id'] = eip_response_json["AllocationId"]
    params['instance_type'] = 'Nat'
    eip_response_json = eip.associate_eip_address(params)
    CommonUtil.log("associate_eip_address eip", eip_response_json)

    # 创建snat entry
```

```
    params['snat_table_id'] = nat_gateway_json['SnatTableIds']['SnatTableId'][0]
    snat_entry_json = nat_gateway.create_snat_entry(params)
    CommonUtil.log("create_snat_entry", snat_entry_json)

    # 查询EIP
    eip_response_json = eip.describe_eip_address(params['allocation_id'])
    CommonUtil.log("describe_eip_address", eip_response_json)

    # 查询natgw
    params['nat_gateway_id'] = nat_gateway_json['NatGatewayId']
    nat_gateway_json = nat_gateway.describe_nat_gateway(params['nat_gateway_id'])
    CommonUtil.log("describe_nat_gateway", nat_gateway_json)

    # 删除snat entry
    params['snat_entry_id'] = snat_entry_json['SnatEntryId']
    snat_entry_json = nat_gateway.delete_snat_entry(params)
    CommonUtil.log("delete_snat_entry", snat_entry_json)

    # 解绑EIP
    eip_response_json = eip.unassociate_eip_address(params)
    CommonUtil.log("unassociate_eip_address nat", eip_response_json)

    # 删除natgw
    nat_gateway_json = nat_gateway.delete_nat_gateway(params)
    CommonUtil.log("delete_nat_gateway", nat_gateway_json)

    # 释放EIP
    eip_response_json = eip.release_eip_address(params)
    CommonUtil.log("release_eip_address", eip_response_json)

    # 删除vswitch
    params['vswitch_id'] = vswitch_json['VSwitchId']
    vswitch_json = vswitch.delete_vswitch(params)
    CommonUtil.log("delete_vswitch", vswitch_json)

    # 删除vpc
    vpc_json = vpc.delete_vpc(params)
    CommonUtil.log("delete_vpc", vpc_json)


if __name__ == "__main__":
    sys.exit(main())
```

3. 进入natgw_snat.py所在的目录，执行如下命令，创建SNAT条目。

```
python natgw_snat.py
```

预期结果

系统显示类似如下：

```
--------------------------create_vpc--------------------------
{
  "ResourceGroupId": "rg-acfmxazxxxxxxxx",
  "RouteTableId": "vtb-uf6a8ccj9ne58xxxxxxxx",
  "VRouterId": "vrt-uf6qqqaf1o1ptxxxxxxxx",
  "VpcId": "vpc-uf6hxer3h07wgxxxxxxxx",
  "RequestId": "8F483A7B-8A38-47ED-85BD-1E83C075AEA4"
}

-------------------------create_vswitch-------------------------
{
```

```
  "VSwitchId": "vsw-uf6lbov9tyetqxxxxxxxx",
  "RequestId": "2EE2E11B-EF60-4C88-BE2A-F45517290B31"
}

--------------------------create_nat_gateway--------------------------
{
  "NatGatewayId": "ngw-uf6l3c3rswubuxxxxxxxx",
  "BandwidthPackageIds": {
    "BandwidthPackageId": []
  },
  "ForwardTableIds": {
    "ForwardTableId": [
      "ftb-uf6086r1hyecbxxxxxxxx"
    ]
  },
  "RequestId": "9037D769-24C8-46AD-83F3-4C0538FA5970",
  "SnatTableIds": {
    "SnatTableId": [
      "stb-uf6ppo11rsecmxxxxxxxx"
    ]
  }
}

--------------------------allocate_eip_address--------------------------
{
  "EipAddress": "101.xx.xx.110",
  "ResourceGroupId": "rg-acfmxazxxxxxxxx",
  "RequestId": "0DE621B4-6BDE-4E17-A294-8F71FBB9F710",
  "AllocationId": "eip-uf6d311cpmr0nxxxxxxxx"
}

--------------------------associate_eip_address eip--------------------------
{
  "RequestId": "C95B2EDC-F081-4784-B60B-2600F60E684D"
}

--------------------------create_snat_entry--------------------------
{
  "SnatEntryId": "snat-uf6ppbwshdu40xxxxxxxx",
  "RequestId": "BB9F8FD2-3CB5-4F84-8006-FE64BF3BEA06"
}

--------------------------describe_eip_address--------------------------
{
  "TotalCount": 1,
  "PageNumber": 1,
  "PageSize": 10,
  "EipAddresses": {
    "EipAddress": [
      {
        "ISP": "BGP",
        "ExpiredTime": "",
        "InternetChargeType": "PayByBandwidth",
        "IpAddress": "101.xx.xx.110",
        "AllocationId": "eip-uf6d311cpmr0nxxxxxxxx",
        "PrivateIpAddress": "",
        "Status": "InUse",
        "BandwidthPackageId": "",
        "InstanceId": "ngw-uf6l3c3rswubuxxxxxxxx",
        "InstanceRegionId": "cn-shanghai",
        "RegionId": "cn-shanghai",
        "AvailableRegions": {
          "AvailableRegion": [
            "cn-shanghai"
```

```
        ]
      },
      "ResourceGroupId": "rg-acfmxazxxxxxxxx",
      "HasReservationData": false,
      "InstanceType": "Nat",
      "AllocationTime": "2019-04-24T11:20:09Z",
      "Name": "",
      "OperationLocks": {
       "LockReason": []
      },
      "Mode": "NAT",
      "BandwidthPackageType": "",
      "BandwidthPackageBandwidth": "",
      "Bandwidth": "5",
      "HDMonitorStatus": "OFF",
      "ChargeType": "PostPaid",
      "SecondLimited": false,
      "Descritpion": ""
     }
    ]
  },
  "RequestId": "19052237-6E84-4258-89B9-05772C33C0DC"
}

--------------------------describe_nat_gateway--------------------------
{
  "TotalCount": 1,
  "PageNumber": 1,
  "RequestId": "26CAA3FE-B400-4522-9582-2DAAF69129AE",
  "PageSize": 10,
  "NatGateways": {
   "NatGateway": [
    {
     "Status": "Available",
     "BandwidthPackageIds": {
      "BandwidthPackageId": []
     },
     "VpcId": "vpc-uf6hxer3h07wgxxxxxxxx",
     "Description": "",
     "ForwardTableIds": {
      "ForwardTableId": [
       "ftb-uf6086r1hyecbxxxxxxxx"
      ]
     },
     "IpLists": {
      "IpList": [
       {
        "UsingStatus": "UsedBySnatTable",
        "IpAddress": "101.xx.xx.110",
        "AllocationId": "eip-uf6d311cpmr0nxxxxxxxx"
       }
      ]
     },
     "BusinessStatus": "Normal",
     "RegionId": "cn-shanghai",
     "CreationTime": "2019-04-24T11:20:06Z",
     "NatGatewayId": "ngw-uf6l3c3rswubuxxxxxxxx",
     "SnatTableIds": {
      "SnatTableId": [
       "stb-uf6ppo11rsecmxxxxxxxx"
      ]
     },
     "AutoPay": false,
     "InstanceChargeType": "PostPaid",
```

```
      "ExpiredTime": "",
      "Spec": "Small",
      "Name": ""
    }
   ]
  }
}

--------------------------delete_snat_entry-------------------------
{
 "RequestId": "CDA82881-ACF0-4DD1-887B-A764F56F180D"
}

--------------------------unassociate_eip_address nat-------------------------
-
{
 "RequestId": "140C46FF-0DB0-47F9-B0F4-3459DF117EAF"
}

--------------------------delete_nat_gateway-------------------------
{
 "RequestId": "8709747C-6786-443C-8AEA-51647AA49769"
}

--------------------------release_eip_address-------------------------
{
 "RequestId": "0BC21C23-0FB3-4594-8B14-6552DF788C93"
}

--------------------------delete_vswitch-------------------------
{
 "RequestId": "16E840EC-E058-40C1-A5BF-8CDF672EA139"
}

--------------------------delete_vpc-------------------------
{
 "RequestId": "B5F18126-FF2A-4005-9973-3984034DF0F4"
}
```

2.6 路由器接口

2.6.1 同账号同地域VPC间互通

本文介绍如何使用Alibaba Cloud SDK for Python实现同账号同地域下的两个VPC互通。

前提条件

在使用Alibaba Cloud SDK for Python前，您需要完成以下准备工作：

- 使用Alibaba Cloud SDK for Python，您需要一个阿里云账号和访问密钥（AccessKey）。 请在阿里云控制台中的AccessKey管理页面上创建和查看您的AccessKey。
- 确保您已经安装了Alibaba Cloud SDK for Python，请参见aliyun-python-sdk-vpc 3.0.8。

- 下载阿里云专有网络Python SDK场景示例的VPC Python Example库。

  进入setup.py所在的目录，执行如下命令，完成环境初始化配置。

  ```
  python setup.py install
  ```

背景信息

创建路由器接口时，请注意：

- 任意两个路由器之间，最多只能存在一对互连的路由器接口。
- 一个路由器上可以最多创建5个路由器接口。
- 如果账户下存在欠费状态的路由器接口，则无法再创建路由器接口。
- 同一路由表下的路由条目的目标网段（DestinationCidrBlock）不允许相同。

本文代码示例中包含以下操作：

1. 创建发起端路由器接口。
2. 创建接收端路由器接口。
3. 修改发起端路由器接口的信息。
4. 修改接收端路由器接口的信息。
5. 查询发起端路由器接口的信息。
6. 查询接收端路由器接口的信息。
7. 连接发起端路由器接口和接收端路由器接口。
8. 创建下一跳为发起端路由器接口的路由条目。
9. 创建下一跳为接收端路由器接口的路由条目。
10. 删除下一跳为接收端路由器接口的路由条目。
11. 删除下一跳为发起端路由器接口的路由条目。
12. 冻结发起端路由器接口。
13. 冻结接收端路由器接口。
14. 删除发起端路由器接口。
15. 删除接收端路由器接口。

操作步骤

1. 在下载的SDK目录中，打开$aliyun-openapi-python-sdk-examples\sdk_examples\examples\ec文件夹。
2. 使用编辑器打开ec_same_account.py文件，根据实际情况配置相关参数，保存退出。

   完整代码示例如下：

   ```
   #encoding=utf-8
   ```

```
import sys
import json
from aliyunsdkcore.acs_exception.exceptions import ServerException, ClientException
from aliyunsdkvpc.request.v20160428 import CreateRouterInterfaceRequest
from aliyunsdkvpc.request.v20160428 import DeleteRouterInterfaceRequest
from aliyunsdkvpc.request.v20160428 import DescribeRouterInterfacesRequest
from aliyunsdkvpc.request.v20160428 import ConnectRouterInterfaceRequest
from aliyunsdkvpc.request.v20160428 import DeactivateRouterInterfaceRequest
from aliyunsdkvpc.request.v20160428 import ModifyRouterInterfaceAttributeRequest
from sdk_lib.sdk_route_entry import RouteEntry
from sdk_lib.exception import ExceptionHandler
from sdk_lib.common_util import CommonUtil
from sdk_lib.check_status import CheckStatus
from sdk_lib.consts import *

client = AcsClient(
        '<your-access-key-id>',    # 您的可用区ID
        '<your-access-key-secret>', # 您的AccessKey ID
        '<your-region-id>')        # 您的AccessKey Secret

class RouterInterface(object):
    def __init__(self, client):
        self.client = client

    def create_router_interface(self, params):
        """
        create_router_interface: 创建路由器接口
        """
        try:
            request = CreateRouterInterfaceRequest.CreateRouterInterfaceRequest()
            # 路由器接口的规格
            request.set_Spec(params['spec'])
            # 路由器接口的角色
            request.set_Role(params['role'])
            # 路由器接口关联的路由器ID
            request.set_RouterId(params['router_id'])
            # 路由器接口关联的路由器类型
            request.set_RouterType(params['router_type'])
            # 连接接收端所在的地域ID
            request.set_OppositeRegionId(params['opposite_region_id'])
            # 对端路由器接口关联的路由器类型
            request.set_OppositeRouterType(params['opposite_router_type'])

            response = self.client.do_action_with_exception(request)
            response_json = json.loads(response)
            # 判断路由器接口状态是否可用
            if CheckStatus.check_status(TIME_DEFAULT_OUT, DEFAULT_TIME,
                                        self.describe_ri_status,
                                        'Idle', response_json['RouterInterfaceId']):
                return response_json
        except ServerException as e:
            ExceptionHandler.server_exception(e)
        except ClientException as e:
            ExceptionHandler.client_exception(e)

    def connect_router_interface(self, params):
        """
        connect_router_interface: 由发起端路由器接口向接收端发起连接
        """
        try:
            request = ConnectRouterInterfaceRequest.ConnectRouterInterfaceRequest()
            # 发起端路由器接口的ID
            request.set_RouterInterfaceId(params['router_interface_id'])
            response = self.client.do_action_with_exception(request)
```

```
            response_json = json.loads(response)
            if CheckStatus.check_status(TIME_DEFAULT_OUT * 5, DEFAULT_TIME * 5,
                                        self.describe_ri_status,
                                        'Active', params['router_interface_id']):
                return response_json
        except ServerException as e:
            ExceptionHandler.server_exception(e)
        except ClientException as e:
            ExceptionHandler.client_exception(e)

    def deactivate_router_interface(self, params):
        """
        deactivate_router_interface: 冻结路由器接口
        """
        try:
            request = DeactivateRouterInterfaceRequest.DeactivateRouterInterfaceRequest
()
            # 路由器接口的ID
            request.set_RouterInterfaceId(params['router_interface_id'])
            response = self.client.do_action_with_exception(request)
            response_json = json.loads(response)
            # 判断路由器接口状态是否可用
            if CheckStatus.check_status(TIME_DEFAULT_OUT * 5, DEFAULT_TIME * 5,
                                        self.describe_ri_status,
                                        'Inactive', params['router_interface_id']):
                return response_json
        except ServerException as e:
            ExceptionHandler.server_exception(e)
        except ClientException as e:
            ExceptionHandler.client_exception(e)

    def modify_router_interface_attribute(self, params):
        """
        modify_router_interface_attribute: 修改路由器接口的配置
        """
        try:
            request = ModifyRouterInterfaceAttributeRequest.ModifyRouterInterfac
eAttributeRequest()
            # 路由器接口的ID
            request.set_RouterInterfaceId(params['router_interface_id'])
            # 对端路由器接口ID
            request.set_OppositeInterfaceId(params['opposite_interface_id'])
            # 对端的路由器的ID
            request.set_OppositeRouterId(params['opposite_router_id'])
            response = self.client.do_action_with_exception(request)
            response_json = json.loads(response)
            # 判断路由器接口状态是否可用
            if CheckStatus.check_status(TIME_DEFAULT_OUT, DEFAULT_TIME,
                                        self.describe_ri_status,
                                        'Idle', params['router_interface_id']):
                return response_json
        except ServerException as e:
            ExceptionHandler.server_exception(e)
        except ClientException as e:
            ExceptionHandler.client_exception(e)



    def describe_router_interface(self, instance_id):
        """
        describe_router_interface: 查询指定地域内的路由器接口
        """
        try:
            request = DescribeRouterInterfacesRequest.DescribeRouterInterfacesRequest()
```

```
            # 查询的过滤类型
            request.add_query_param('Filter.1.Key', "RouterInterfaceId")
            # 查询的实例ID
            request.add_query_param('Filter.1.Value.1', instance_id)
            response = self.client.do_action_with_exception(request)
            response_json = json.loads(response)
            return response_json
        except ServerException as e:
            ExceptionHandler.server_exception(e)
        except ClientException as e:
            ExceptionHandler.client_exception(e)

    def describe_ri_status(self, instance_id):
        """
        describe_ri_status: 查询指定地域内的路由器接口状态
        """
        response = self.describe_router_interface(instance_id)
        if len(response['RouterInterfaceSet']['RouterInterfaceType']) == 0:
            return ''
        return response['RouterInterfaceSet']['RouterInterfaceType'][0]['Status']

    def delete_router_interface(self, params):
        """
        delete_router_interface: 删除路由器接口
        """
        try:
            request = DeleteRouterInterfaceRequest.DeleteRouterInterfaceRequest()
            # 路由器接口的ID
            request.set_RouterInterfaceId(params['instance_id'])
            response = self.client.do_action_with_exception(request)
            response_json = json.loads(response)
            # 判断Router Interface状态是否可用
            if CheckStatus.check_status(TIME_DEFAULT_OUT, DEFAULT_TIME * 5,
                                        self.describe_ri_status,
                                        '', params['instance_id']):
                return response_json
        except ServerException as e:
            ExceptionHandler.server_exception(e)
        except ClientException as e:
            ExceptionHandler.client_exception(e)

def main():
    client = ACS_CLIENT
    router_interface = RouterInterface(client)
    route_entry = RouteEntry(client)

    params = {}
    params['spec'] = "Large.2"
    params['role'] = "InitiatingSide"
    params['router_id'] = ROUTER_ID
    params['router_type'] = "VRouter"
    params['opposite_region_id'] = "cn-hangzhou"
    params['opposite_router_type'] = "VRouter"

    # 创建发起端路由器接口
    router_interface_json = router_interface.create_router_interface(params)
    CommonUtil.log("create_router_interface", router_interface_json)

    # 创建接收端路由器接口
    params['spec'] = "Negative"
    params['role'] = "AcceptingSide"
    params['router_id'] = ROUTER_ID2
    router_interface_json2 = router_interface.create_router_interface(params)
    CommonUtil.log("create_router_interface", router_interface_json2)
```

```
    # 修改发起端路由器接口信息
    params['router_interface_id'] = router_interface_json['RouterInterfaceId']
    params['opposite_interface_id'] = router_interface_json2['RouterInterfaceId']
    params['opposite_router_id'] = ROUTER_ID2
    modify_ri_json = router_interface.modify_router_interface_attribute(params)
    CommonUtil.log("modify_router_interface_attribute", modify_ri_json)

    # 修改接收端路由器接口信息
    params['router_interface_id'] = router_interface_json2['RouterInterfaceId']
    params['opposite_interface_id'] = router_interface_json['RouterInterfaceId']
    params['opposite_router_id'] = ROUTER_ID
    modify_ri_json2 = router_interface.modify_router_interface_attribute(params)
    CommonUtil.log("modify_router_interface_attribute", modify_ri_json2)

    # 查询发起端路由器接口信息
    describe_ri_json = router_interface.describe_router_interface(router_interface_json['
RouterInterfaceId'])
    CommonUtil.log("describe_router_interface", describe_ri_json)

    # 查询接收端路由器接口信息
    describe_ri_json2 = router_interface.describe_router_interface(router_interface_jso
n2['RouterInterfaceId'])
    CommonUtil.log("describe_router_interface", describe_ri_json2)

    # 发起连接
    params['router_interface_id'] = router_interface_json['RouterInterfaceId']
    connect_ri_json = router_interface.connect_router_interface(params)
    CommonUtil.log("connect_router_interface", connect_ri_json)

    # 创建下一跳为发起端路由器接口的路由条目
    params['route_table_id'] = TABLE_ID
    params['destination_cidr_block'] = "0.0.0.0/0"
    params['nexthop_type'] = 'RouterInterface'
    params['nexthop_id'] = router_interface_json['RouterInterfaceId']
    route_entry_json = route_entry.create_route_entry(params)
    CommonUtil.log("create_route_entry", route_entry_json)

    # 创建下一跳为接收端路由器接口的路由条目
    params['route_table_id'] = TABLE_ID2
    params['destination_cidr_block'] = "0.0.0.0/0"
    params['nexthop_type'] = 'RouterInterface'
    params['nexthop_id'] = router_interface_json2['RouterInterfaceId']
    route_entry_json2 = route_entry.create_route_entry(params)
    CommonUtil.log("create_route_entry", route_entry_json2)

    # 删除下一跳为接收端路由器接口的路由条目
    route_entry_json = route_entry.delete_route_entry(params)
    CommonUtil.log("delete_route_entry", route_entry_json)

    # 删除下一跳为发起端路由器接口的路由条目
    params['route_table_id'] = TABLE_ID
    params['nexthop_id'] = router_interface_json['RouterInterfaceId']
    route_entry_json = route_entry.delete_route_entry(params)
    CommonUtil.log("delete_route_entry", route_entry_json)

    # 冻结发起端路由器接口
    params['router_interface_id'] = router_interface_json['RouterInterfaceId']
    deactivate_ri_json = router_interface.deactivate_router_interface(params)
    CommonUtil.log("deactivate_router_interface", deactivate_ri_json)

    # 冻结接收端路由器接口
    params['router_interface_id'] = router_interface_json2['RouterInterfaceId']
    deactivate_ri_json2 = router_interface.deactivate_router_interface(params)
```

```
    CommonUtil.log("deactivate_router_interface", deactivate_ri_json2)

    # 删除发起端路由器接口
    params['instance_id'] = router_interface_json['RouterInterfaceId']
    router_interface_json = router_interface.delete_router_interface(params)
    CommonUtil.log("delete_router_interface", router_interface_json)

    # 删除接收端路由器接口
    params['instance_id'] = router_interface_json2['RouterInterfaceId']
    router_interface_json2 = router_interface.delete_router_interface(params)
    CommonUtil.log("delete_router_interface", router_interface_json2)


if __name__ == '__main__':
    sys.exit(main())
```

3. 进入ec_same_account.py所在的目录，执行如下命令，实现同账号同地域下的两个VPC互通。

```
python ec_same_account.py
```

预期结果

系统显示类似如下：

```
--------------------------create_router_interface--------------------------
{
 "RequestId": "F5493DC1-53B0-4916-874F-87773A54525F",
 "RouterInterfaceId": "ri-bp1ujwb6xsw16xxxxxxxx"
}
--------------------------create_router_interface--------------------------
{
 "RequestId": "E3F5BE99-B2C5-4751-8173-0F590300EE72",
 "RouterInterfaceId": "ri-bp1vze2rusg2cxxxxxxxx"
}
-------------------------modify_router_interface_attribute-------------------------
{
 "RequestId": "8D691507-F31A-41E5-9C72-457EFF1F7727"
}
-------------------------modify_router_interface_attribute-------------------------
{
 "RequestId": "2E664208-8F37-4F19-B719-00B4F1AF03B2"
}
--------------------------describe_router_interface--------------------------
{
 "TotalCount": 1,
 "RouterInterfaceSet": {
  "RouterInterfaceType": [
   {
    "BusinessStatus": "Normal",
    "CreationTime": "2019-04-30T06:09:16Z",
    "Role": "InitiatingSide",
    "OppositeRouterId": "vrt-bp141no9pds2bxxxxxxxx",
    "Spec": "Large.2",
    "Status": "Idle",
    "EndTime": "2999-09-08T16:00:00Z",
    "OppositeInterfaceSpec": "Negative",
    "RouterInterfaceId": "ri-bp1ujwb6xsw16xxxxxxxx",
    "RouterType": "VRouter",
    "OppositeBandwidth": 0,
    "OppositeVpcInstanceId": "vpc-bp1v31by9jix2xxxxxxxx",
    "HasReservationData": false,
```

```
      "OppositeInterfaceBusinessStatus": "Normal",
      "OppositeRouterType": "VRouter",
      "OppositeRegionId": "cn-hangzhou",
      "VpcInstanceId": "vpc-bp15opprpg0rgxxxxxxxx",
      "RouterId": "vrt-bp1ltkytn6lgmxxxxxxxx",
      "CrossBorder": false,
      "OppositeInterfaceOwnerId": "",
      "Bandwidth": 2048,
      "OppositeInterfaceId": "ri-bp1vze2rusg2cxxxxxxxx",
      "ChargeType": "AfterPay"
    }
   ]
 },
 "PageNumber": 1,
 "RequestId": "89EF0631-0A36-41AD-A586-AF4FFDA6E68B",
 "PageSize": 10
}
--------------------------describe_router_interface--------------------------
{
 "TotalCount": 1,
 "RouterInterfaceSet": {
  "RouterInterfaceType": [
   {
    "Status": "Idle",
    "OppositeRegionId": "cn-hangzhou",
    "BusinessStatus": "Normal",
    "OppositeRouterId": "vrt-bp1ltkytn6lgmxxxxxxxx",
    "VpcInstanceId": "vpc-bp1v31by9jix2xxxxxxxx",
    "RouterInterfaceId": "ri-bp1vze2rusg2cxxxxxxxx",
    "CreationTime": "2019-04-30T06:09:18Z",
    "RouterType": "VRouter",
    "OppositeInterfaceOwnerId": "",
    "RouterId": "vrt-bp141no9pds2bxxxxxxxx",
    "Bandwidth": 0,
    "OppositeInterfaceId": "ri-bp1ujwb6xsw16xxxxxxxx",
    "EndTime": "2999-09-08T16:00:00Z",
    "ChargeType": "AfterPay",
    "OppositeVpcInstanceId": "vpc-bp15opprpg0rgxxxxxxxx",
    "HasReservationData": false,
    "CrossBorder": false,
    "OppositeInterfaceBusinessStatus": "Normal",
    "Spec": "Negative",
    "OppositeRouterType": "VRouter",
    "Role": "AcceptingSide"
   }
  ]
 },
 "PageNumber": 1,
 "RequestId": "578448D7-9DCF-4703-8337-EF88DDF2C325",
 "PageSize": 10
}
--------------------------connect_router_interface--------------------------
{
 "RequestId": "A5DBB86B-F6F5-4A53-899E-8CF4FEF510F2"
}
--------------------------create_route_entry--------------------------
{
 "RequestId": "70D896FE-986B-48EF-9734-17D6BDC8327A"
}
--------------------------create_route_entry--------------------------
{
 "RequestId": "A2233E25-4D6B-4713-A96F-E7CA745973CA"
}
--------------------------delete_route_entry--------------------------
```

```
{
  "RequestId": "464C62A4-EE65-4414-AF0A-4984AE6B8696"
}
-------------------------delete_route_entry-------------------------
{
  "RequestId": "0C11A332-969B-47CA-A683-5BFFECA28B3D"
}
-------------------------deactivate_router_interface-------------------------
{
  "RequestId": "018305AD-FB9E-450A-91E8-3830634F5AC2"
}
-------------------------deactivate_router_interface-------------------------
{
  "RequestId": "89B03203-9224-4CA3-8679-E0A49029A2D2"
}
-------------------------delete_router_interface-------------------------
{
  "RequestId": "FB0424CE-D0C7-438B-A3FA-BCF24EE9CC8A"
}
-------------------------delete_router_interface-------------------------
{
  "RequestId": "8A0A0BB6-A69D-461B-A117-14AD6670DECA"
}
```

2.6.2 同账号同地域VBR与VPC间互通

本文介绍如何使用Alibaba Cloud SDK for Python实现同账号同地域下的VBR与VPC互通。

前提条件

在使用Alibaba Cloud SDK for Python前，您需要完成以下准备工作：

- 使用Alibaba Cloud SDK for Python，您需要一个阿里云账号和访问密钥（AccessKey）。 请在阿里云控制台中的AccessKey管理页面上创建和查看您的AccessKey。
- 确保您已经安装了Alibaba Cloud SDK for Python，请参见aliyun-python-sdk-vpc 3.0.8。
- 下载阿里云专有网络Python SDK场景示例的VPC Python Example库。

  进入setup.py所在的目录，执行如下命令，完成环境初始化配置。

  ```
  python setup.py install
  ```

背景信息

创建路由器接口时，请注意：

- 任意两个路由器之间，最多只能存在一对互连的路由器接口。
- 一个路由器上可以最多创建5个路由器接口。
- 如果账户下存在欠费状态的路由器接口，则无法再创建路由器接口。
- 同一路由表下的路由条目的目标网段（DestinationCidrBlock）不允许相同。
- 边界路由器（VBR）只能作为连接发起端，并且处于已激活状态。

本文代码示例中包含以下操作：

1. 创建边界路由器。
2. 创建发起端路由器接口。
3. 创建接收端路由器接口。
4. 修改发起端路由器接口信息。
5. 修改接收端路由器接口信息。
6. 查询发起端路由器接口信息。
7. 查询接收端路由器接口信息。
8. 查询边界路由器的路由表ID。
9. 连接发起端路由器接口和接收端路由器接口。
10. 创建下一跳为发起端路由器接口的路由条目。
11. 创建下一跳为接收端路由器接口的路由条目。
12. 删除下一跳为接收端路由器接口的路由条目。
13. 删除下一跳为发起端路由器接口的路由条目。
14. 冻结发起端路由器接口。
15. 冻结接收端路由器接口。
16. 删除发起端路由器接口。
17. 删除接收端路由器接口。
18. 删除边界路由器。

操作步骤

1. 在下载的SDK目录中，打开$aliyun-openapi-python-sdk-examples\sdk_examples\examples\ec文件夹。
2. 使用编辑器打开ec_vbr_vpc.py文件，根据实际情况配置相关参数，保存退出。

   完整代码示例如下：

```
#encoding=utf-8
import sys
import json
from aliyunsdkcore.acs_exception.exceptions import ServerException, ClientException
from aliyunsdkvpc.request.v20160428 import CreateRouterInterfaceRequest
from aliyunsdkvpc.request.v20160428 import DeleteRouterInterfaceRequest
from aliyunsdkvpc.request.v20160428 import DescribeRouterInterfacesRequest
from aliyunsdkvpc.request.v20160428 import ConnectRouterInterfaceRequest
from aliyunsdkvpc.request.v20160428 import DeactivateRouterInterfaceRequest
from aliyunsdkvpc.request.v20160428 import ModifyRouterInterfaceAttributeRequest
from aliyunsdkvpc.request.v20160428 import CreateVirtualBorderRouterRequest
from aliyunsdkvpc.request.v20160428 import DescribeVirtualBorderRoutersRequest
from aliyunsdkvpc.request.v20160428 import DescribeRouteTablesRequest
from aliyunsdkvpc.request.v20160428 import DeleteVirtualBorderRouterRequest
from sdk_lib.sdk_route_entry import RouteEntry
from sdk_lib.exception import ExceptionHandler
from sdk_lib.common_util import CommonUtil
```

```
from sdk_lib.check_status import CheckStatus
from sdk_lib.consts import *

client = AcsClient(
        '<your-access-key-id>',    # 您的可用区ID
        '<your-access-key-secret>', # 您的AccessKey ID
        '<your-region-id>')        # 您的AccessKey Secret

class RouterInterface(object):
    def __init__(self, client):
        self.client = client

    def create_router_interface(self, params):
        """
        create_router_interface: 创建路由器接口
        """
        try:
            request = CreateRouterInterfaceRequest.CreateRouterInterfaceRequest()
            # 路由器接口的规格
            request.set_Spec(params['spec'])
            # 路由器接口的角色
            request.set_Role(params['role'])
            # 路由器接口关联的路由器ID
            request.set_RouterId(params['router_id'])
            # 路由器接口关联的路由器类型
            request.set_RouterType(params['router_type'])
            # 连接接收端所在的地域ID
            request.set_OppositeRegionId(params['opposite_region_id'])
            # 对端路由器接口关联的路由器类型
            request.set_OppositeRouterType(params['opposite_router_type'])
            # 对端所属的接入点ID
            request.set_OppositeAccessPointId(params['opposite_access_point_id'])
            # VBR所属的接入点ID
            #request.set_AccessPointId(params['access_point_id'])

            response = self.client.do_action_with_exception(request)
            response_json = json.loads(response)
            # 判断路由器接口状态是否可用
            if CheckStatus.check_status(TIME_DEFAULT_OUT, DEFAULT_TIME,
                                        self.describe_ri_status,
                                        'Idle', response_json['RouterInterfaceId']):
                return response_json
        except ServerException as e:
            ExceptionHandler.server_exception(e)
        except ClientException as e:
            ExceptionHandler.client_exception(e)

    def create_router_interface_vbr(self, params):
        """
        create_router_interface: 创建路由器接口
        """
        try:
            request = CreateRouterInterfaceRequest.CreateRouterInterfaceRequest()
            # 路由器接口的规格
            request.set_Spec(params['spec'])
            # 路由器接口的角色
            request.set_Role(params['role'])
            # 路由器接口关联的路由器ID
            request.set_RouterId(params['router_id'])
            # 路由器接口关联的路由器类型
            request.set_RouterType(params['router_type'])
            # 连接接收端所在的地域ID
            request.set_OppositeRegionId(params['opposite_region_id'])
            # 对端路由器接口关联的路由器类型
```

```
            request.set_OppositeRouterType(params['opposite_router_type'])
            # VBR所属的接入点ID
            request.set_AccessPointId(params['access_point_id'])

            response = self.client.do_action_with_exception(request)
            response_json = json.loads(response)
            # 判断路由器接口状态是否可用
            if CheckStatus.check_status(TIME_DEFAULT_OUT, DEFAULT_TIME,
                                        self.describe_ri_status,
                                        'Idle', response_json['RouterInterfaceId']):
                return response_json
        except ServerException as e:
            ExceptionHandler.server_exception(e)
        except ClientException as e:
            ExceptionHandler.client_exception(e)

    def connect_router_interface(self, params):
        """
        connect_router_interface: 由发起端路由器接口向接收端发起连接
        """
        try:
            request = ConnectRouterInterfaceRequest.ConnectRouterInterfaceRequest()
            # 发起端路由器接口的ID
            request.set_RouterInterfaceId(params['router_interface_id'])
            response = self.client.do_action_with_exception(request)
            response_json = json.loads(response)
            if CheckStatus.check_status(TIME_DEFAULT_OUT * 5, DEFAULT_TIME * 5,
                                        self.describe_ri_status,
                                        'Active', params['router_interface_id']):
                return response_json
        except ServerException as e:
            ExceptionHandler.server_exception(e)
        except ClientException as e:
            ExceptionHandler.client_exception(e)

    def deactivate_router_interface(self, params):
        """
        deactivate_router_interface: 冻结路由器接口
        """
        try:
            request = DeactivateRouterInterfaceRequest.DeactivateRouterInterfaceRequest
()
            # 路由器接口的ID
            request.set_RouterInterfaceId(params['router_interface_id'])
            response = self.client.do_action_with_exception(request)
            response_json = json.loads(response)
            # 判断路由器接口状态是否可用
            if CheckStatus.check_status(TIME_DEFAULT_OUT * 5, DEFAULT_TIME * 5,
                                        self.describe_ri_status,
                                        'Inactive', params['router_interface_id']):
                return response_json
        except ServerException as e:
            ExceptionHandler.server_exception(e)
        except ClientException as e:
            ExceptionHandler.client_exception(e)

    def modify_router_interface_attribute(self, params):
        """
        modify_router_interface_attribute: 修改路由器接口的配置
        """
        try:
            request = ModifyRouterInterfaceAttributeRequest.ModifyRouterInterfac
eAttributeRequest()
            # 路由器接口的ID
```

```
        request.set_RouterInterfaceId(params['router_interface_id'])
        # 对端路由器接口ID
        request.set_OppositeInterfaceId(params['opposite_interface_id'])
        # 对端的路由器的ID
        request.set_OppositeRouterId(params['opposite_router_id'])
        # 对端的路由器的类型
        request.set_OppositeRouterType(params['opposite_router_type'])
        response = self.client.do_action_with_exception(request)
        response_json = json.loads(response)
        # 判断路由器接口状态是否可用
        if CheckStatus.check_status(TIME_DEFAULT_OUT, DEFAULT_TIME,
                                    self.describe_ri_status,
                                    'Idle', params['router_interface_id']):
            return response_json
    except ServerException as e:
        ExceptionHandler.server_exception(e)
    except ClientException as e:
        ExceptionHandler.client_exception(e)



def describe_router_interface(self, instance_id):
    """
    describe_router_interface: 查询指定地域内的路由器接口
    """
    try:
        request = DescribeRouterInterfacesRequest.DescribeRouterInterfacesRequest()
        # 查询的过滤类型
        request.add_query_param('Filter.1.Key', "RouterInterfaceId")
        # 查询的实例ID
        request.add_query_param('Filter.1.Value.1', instance_id)
        response = self.client.do_action_with_exception(request)
        response_json = json.loads(response)
        return response_json
    except ServerException as e:
        ExceptionHandler.server_exception(e)
    except ClientException as e:
        ExceptionHandler.client_exception(e)

def describe_ri_status(self, instance_id):
    """
    describe_ri_status: 查询指定地域内的路由器接口状态
    """
    response = self.describe_router_interface(instance_id)
    if len(response['RouterInterfaceSet']['RouterInterfaceType']) == 0:
        return ''
    return response['RouterInterfaceSet']['RouterInterfaceType'][0]['Status']

def delete_router_interface(self, params):
    """
    delete_router_interface: 删除路由器接口
    """
    try:
        request = DeleteRouterInterfaceRequest.DeleteRouterInterfaceRequest()
        # 路由器接口的ID
        request.set_RouterInterfaceId(params['instance_id'])
        response = self.client.do_action_with_exception(request)
        response_json = json.loads(response)
        # 判断路由器接口状态是否可用
        if CheckStatus.check_status(TIME_DEFAULT_OUT, DEFAULT_TIME * 5,
                                    self.describe_ri_status,
                                    '', params['instance_id']):
            return response_json
    except ServerException as e:
```

```
            ExceptionHandler.server_exception(e)
        except ClientException as e:
            ExceptionHandler.client_exception(e)

    def create_virtual_border_router(self, params):
        """
        create_virtual_border_router: 新建边界路由器
        """
        try:
            request = CreateVirtualBorderRouterRequest.CreateVirtualBorderRouterRequest
()
            # 物理专线的ID
            request.set_PhysicalConnectionId(params['physical_connection_id'])
            # 边界路由器的阿里云侧互联IP
            request.set_LocalGatewayIp(params['local_gateway_ip'])
            # 边界路由器专线侧接口对端的IP地址
            request.set_PeerGatewayIp(params['peer_gateway_ip'])
            # 边界路由器的阿里云侧和客户侧互联IP的子网掩码
            request.set_PeeringSubnetMask(params['peering_subnet_mask'])
            # 边界路由器的VLAN ID
            request.set_VlanId(params['vlan_id'])

            response = self.client.do_action_with_exception(request)
            response_json = json.loads(response)
            # 判断边界路由器状态是否可用
            if CheckStatus.check_status(TIME_DEFAULT_OUT, DEFAULT_TIME,
                                        self.describe_vbr_status,
                                        'active', response_json['VbrId']):
                return response_json
        except ServerException as e:
            ExceptionHandler.server_exception(e)
        except ClientException as e:
            ExceptionHandler.client_exception(e)

    def delete_virtual_border_router(self, params):
        """
        delete_virtual_border_router: 删除边界路由器
        """
        try:
            request = DeleteVirtualBorderRouterRequest.DeleteVirtualBorderRouterRequest
()
            request.set_VbrId(params['vbr_id'])
            response = self.client.do_action_with_exception(request)
            response_json = json.loads(response)
            if CheckStatus.check_status(TIME_DEFAULT_OUT, DEFAULT_TIME,
                                        self.describe_vbr_status,
                                        '', params['vbr_id']):
                return response_json
        except ServerException as e:
            ExceptionHandler.server_exception(e)
        except ClientException as e:
            ExceptionHandler.client_exception(e)

    def describe_virtual_border_router(self, instance_id):
        """
        describe_virtual_border_router: 查询边界路由器
        """
        try:
            request = DescribeVirtualBorderRoutersRequest.DescribeVirtualBorde
rRoutersRequest()
            # 查询的过滤类型
            request.add_query_param('Filter.1.Key', "VbrId")
            # 查询的实例ID
            request.add_query_param('Filter.1.Value.1', instance_id)
```

```
            response = self.client.do_action_with_exception(request)
            response_json = json.loads(response)
            return response_json
        except ServerException as e:
            ExceptionHandler.server_exception(e)
        except ClientException as e:
            ExceptionHandler.client_exception(e)

    def describe_vbr_status(self, instance_id):
        """
        describe_virtual_border_router: 查询边界路由器状态
        """
        response = self.describe_virtual_border_router(instance_id)
        if len(response['VirtualBorderRouterSet']['VirtualBorderRouterType']) == 0:
            return ''
        return response['VirtualBorderRouterSet']['VirtualBorderRouterType'][0]['Status']

    def describe_route_table(self, params):
        """
        describe_route_table: 查询路由表
        """
        try:
            request = DescribeRouteTablesRequest.DescribeRouteTablesRequest()
            # 路由表所属的VPC路由器或边界路由器的ID
            request.set_RouterId(params['router_id'])
            # 路由表所属的路由器类型
            request.set_RouterType(params['router_type'])
            response = self.client.do_action_with_exception(request)
            response_json = json.loads(response)
            return response_json
        except ServerException as e:
            ExceptionHandler.server_exception(e)
        except ClientException as e:
            ExceptionHandler.client_exception(e)


def main():
    router_interface = RouterInterface(client)
    route_entry = RouteEntry(client)

    params = {}
    params['spec'] = "Large.2"
    params['role'] = "InitiatingSide"
    params['router_type'] = "VBR"
    params['opposite_region_id'] = "cn-hangzhou"
    params['opposite_router_type'] = "VRouter"
    params['access_point_id'] = "ap-cn-hangzhou-xx-x"

    # 创建边界路由器
    params['physical_connection_id'] = PC_ID
    params['local_gateway_ip'] = "116.xx.xx.254"
    params['peer_gateway_ip'] = "116.xx.xx.254"
    params['peering_subnet_mask'] = "255.255.255.252"
    params['vlan_id'] = 30
    vbr_json = router_interface.create_virtual_border_router(params)
    CommonUtil.log("create_virtual_border_router", vbr_json)

    # 创建发起端路由器接口
    params['router_id'] = vbr_json['VbrId']
    router_interface_json = router_interface.create_router_interface_vbr(params)
    CommonUtil.log("create_router_interface", router_interface_json)

    # 创建接收端路由器接口
    params['router_type'] = "VRouter"
```

```
    params['spec'] = "Negative"
    params['role'] = "AcceptingSide"
    params['router_id'] = ROUTER_ID2
    params['opposite_router_type'] = 'VBR'
    params['opposite_access_point_id'] = params['access_point_id']
    router_interface_json2 = router_interface.create_router_interface(params)
    CommonUtil.log("create_router_interface", router_interface_json2)

    # 修改发起端路由器接口信息
    params['router_interface_id'] = router_interface_json['RouterInterfaceId']
    params['opposite_interface_id'] = router_interface_json2['RouterInterfaceId']
    params['opposite_router_id'] = ROUTER_ID2
    params['opposite_router_type'] = "VRouter"
    modify_ri_json = router_interface.modify_router_interface_attribute(params)
    CommonUtil.log("modify_router_interface_attribute", modify_ri_json)

    # 修改接收端路由器接口信息
    params['router_interface_id'] = router_interface_json2['RouterInterfaceId']
    params['opposite_interface_id'] = router_interface_json['RouterInterfaceId']
    params['opposite_router_id'] = vbr_json['VbrId']
    params['opposite_router_type'] = "VBR"
    modify_ri_json2 = router_interface.modify_router_interface_attribute(params)
    CommonUtil.log("modify_router_interface_attribute", modify_ri_json2)

    # 查询发起端路由器接口信息
    describe_ri_json = router_interface.describe_router_interface(router_interface_json['
RouterInterfaceId'])
    CommonUtil.log("describe_router_interface", describe_ri_json)

    # 查询接收端路由器接口信息
    describe_ri_json2 = router_interface.describe_router_interface(router_interface_jso
n2['RouterInterfaceId'])
    CommonUtil.log("describe_router_interface", describe_ri_json2)

    # 查询边界路由器的路由表ID
    params['router_id'] = vbr_json['VbrId']
    params['router_type'] = 'VBR'
    route_table_json = router_interface.describe_route_table(params)
    CommonUtil.log("describe_route_table", route_table_json)

    # 发起连接
    params['router_interface_id'] = router_interface_json['RouterInterfaceId']
    connect_ri_json = router_interface.connect_router_interface(params)
    CommonUtil.log("connect_router_interface", connect_ri_json)

    # 创建下一跳为发起端路由器接口的路由条目
    params['route_table_id'] = route_table_json["RouteTables"]["RouteTable"][0]["
RouteTableId"]
    params['destination_cidr_block'] = "0.0.0.0/0"
    params['nexthop_type'] = 'RouterInterface'
    params['nexthop_id'] = router_interface_json['RouterInterfaceId']
    route_entry_json = route_entry.create_route_entry(params)
    CommonUtil.log("create_route_entry", route_entry_json)

    # 创建下一跳为接收端路由器接口的路由条目
    params['route_table_id'] = TABLE_ID2
    params['destination_cidr_block'] = "0.0.0.0/0"
    params['nexthop_type'] = 'RouterInterface'
    params['nexthop_id'] = router_interface_json2['RouterInterfaceId']
    route_entry_json2 = route_entry.create_route_entry(params)
    CommonUtil.log("create_route_entry", route_entry_json2)

    # 删除下一跳为接收端路由器接口的路由条目
    route_entry_json = route_entry.delete_route_entry(params)
```

```
    CommonUtil.log("delete_route_entry", route_entry_json)

    # 删除下一跳为发起端路由器接口的路由条目
    params['route_table_id'] = route_table_json["RouteTables"]["RouteTable"][0]["
RouteTableId"]
    params['nexthop_id'] = router_interface_json['RouterInterfaceId']
    route_entry_json = route_entry.delete_route_entry(params)
    CommonUtil.log("delete_route_entry", route_entry_json)

    # 冻结发起端路由器接口
    params['router_interface_id'] = router_interface_json['RouterInterfaceId']
    deactivate_ri_json = router_interface.deactivate_router_interface(params)
    CommonUtil.log("deactivate_router_interface", deactivate_ri_json)

    # 冻结接收端路由器接口
    params['router_interface_id'] = router_interface_json2['RouterInterfaceId']
    deactivate_ri_json2 = router_interface.deactivate_router_interface(params)
    CommonUtil.log("deactivate_router_interface", deactivate_ri_json2)

    # 删除发起端路由器接口
    params['instance_id'] = router_interface_json['RouterInterfaceId']
    router_interface_json = router_interface.delete_router_interface(params)
    CommonUtil.log("delete_router_interface", router_interface_json)

    # 删除接收端路由器接口
    params['instance_id'] = router_interface_json2['RouterInterfaceId']
    router_interface_json2 = router_interface.delete_router_interface(params)
    CommonUtil.log("delete_router_interface", router_interface_json2)

    # 删除边界路由器
    params['vbr_id'] = vbr_json['VbrId']
    vbr_json = router_interface.delete_virtual_border_router(params)
    CommonUtil.log("delete_virtual_border_router", vbr_json)

if __name__ == '__main__':
    sys.exit(main())
```

3. 进入ec_vbr_vpc.py所在的目录，执行如下命令，实现同账号同地域下的VBR与VPC互通。

```
python ec_vbr_vpc.py
```

预期结果

系统显示类似如下：

```
--------------------------create_virtual_border_router--------------------------
{
  "VbrId": "vbr-bp1vudncgk9jtxxxxxxxx",
  "RequestId": "0C4FABDB-FF18-4E70-9E43-6DC03197F0EA"
}
--------------------------create_router_interface--------------------------
{
  "RequestId": "11F14FA2-ECFA-4B27-A75D-7C6D5FECACDF",
  "RouterInterfaceId": "ri-bp1vabte8nbdexxxxxxxx"
}
--------------------------create_router_interface--------------------------
{
  "RequestId": "72FAB86D-470B-40E4-8476-F9223A911486",
  "RouterInterfaceId": "ri-bp1mxkc61ly0nxxxxxxxx"
}
--------------------------modify_router_interface_attribute--------------------------
```

```
{
  "RequestId": "93D5FFF7-6C05-41B1-92F0-AC8F1809BC98"
}
--------------------------modify_router_interface_attribute--------------------------
{
  "RequestId": "1C461452-42BD-4649-B318-2214222B4FCA"
}
--------------------------describe_router_interface--------------------------
{
  "TotalCount": 1,
  "RouterInterfaceSet": {
    "RouterInterfaceType": [
      {
        "BusinessStatus": "Normal",
        "CreationTime": "2019-04-30T02:54:22Z",
        "AccessPointId": "ap-cn-hangzhou-xx-x",
        "Role": "InitiatingSide",
        "OppositeRouterId": "vrt-bp141no9pds2bxxxxxxxx",
        "Spec": "Large.2",
        "Status": "Idle",
        "EndTime": "2999-09-08T16:00:00Z",
        "OppositeInterfaceSpec": "Negative",
        "RouterInterfaceId": "ri-bp1vabte8nbdexxxxxxxx",
        "RouterType": "VBR",
        "OppositeBandwidth": 0,
        "OppositeVpcInstanceId": "vpc-bp1v31by9jix2xxxxxxxx",
        "HasReservationData": false,
        "OppositeInterfaceBusinessStatus": "Normal",
        "OppositeRouterType": "VRouter",
        "OppositeRegionId": "cn-hangzhou",
        "RouterId": "vbr-bp1vudncgk9jtxxxxxxxx",
        "CrossBorder": false,
        "OppositeInterfaceOwnerId": "",
        "Bandwidth": 2048,
        "OppositeInterfaceId": "ri-bp1mxkc61ly0nxxxxxxxx",
        "ChargeType": "AfterPay"
      }
    ]
  },
  "PageNumber": 1,
  "RequestId": "3553DB38-A4A0-4453-976C-542E96B1B5A9",
  "PageSize": 10
}
--------------------------describe_router_interface--------------------------
{
  "TotalCount": 1,
  "RouterInterfaceSet": {
    "RouterInterfaceType": [
      {
        "Status": "Idle",
        "OppositeRegionId": "cn-hangzhou",
        "BusinessStatus": "Normal",
        "OppositeRouterId": "vbr-bp1vudncgk9jtxxxxxxxx",
        "VpcInstanceId": "vpc-bp1v31by9jix2xxxxxxxx",
        "RouterInterfaceId": "ri-bp1mxkc61ly0nxxxxxxxx",
        "CreationTime": "2019-04-30T02:54:24Z",
        "RouterType": "VRouter",
        "OppositeInterfaceOwnerId": "",
        "RouterId": "vrt-bp141no9pds2bxxxxxxxx",
        "Bandwidth": 0,
        "OppositeInterfaceId": "ri-bp1vabte8nbdexxxxxxxx",
        "EndTime": "2999-09-08T16:00:00Z",
        "ChargeType": "AfterPay",
        "OppositeAccessPointId": "ap-cn-hangzhou-xx-x",
```

```
      "HasReservationData": false,
      "CrossBorder": false,
      "OppositeInterfaceBusinessStatus": "Normal",
      "Spec": "Negative",
      "OppositeRouterType": "VBR",
      "Role": "AcceptingSide"
    }
   ]
  },
  "PageNumber": 1,
  "RequestId": "217D8D94-C508-4285-8101-7DE3B53A88A5",
  "PageSize": 10
}
--------------------------describe_route_table--------------------------
{
  "TotalCount": 1,
  "PageNumber": 1,
  "RequestId": "CA6BBE52-DF5E-496A-93CF-9857AF22D2AC",
  "PageSize": 10,
  "RouteTables": {
   "RouteTable": [
    {
     "RouteTableId": "vtb-bp1s126yz0swpxxxxxxxx",
     "RouteEntrys": {
      "RouteEntry": []
     },
     "CreationTime": "2019-04-30T02:54:19Z",
     "VSwitchIds": {
      "VSwitchId": []
     },
     "ResourceGroupId": "",
     "VRouterId": "vbr-bp1vudncgk9jtxxxxxxxx",
     "RouteTableType": "System"
    }
   ]
  }
}
--------------------------connect_router_interface--------------------------
{
  "RequestId": "93476D4E-6C08-44B8-83E4-5759E1A08F29"
}
--------------------------create_route_entry--------------------------
{
  "RequestId": "66D4F46B-5E0F-4565-8740-845EC26EBE00"
}
--------------------------create_route_entry--------------------------
{
  "RequestId": "E1F99FEF-2499-40A7-84ED-6F9D440D4FF8"
}
--------------------------delete_route_entry--------------------------
{
  "RequestId": "8C983886-F058-4234-A6D7-ECDEE2C2D945"
}
--------------------------delete_route_entry--------------------------
{
  "RequestId": "DDEC56B9-CDD4-4D0A-B460-040B85B97FE9"
}
--------------------------deactivate_router_interface--------------------------
{
  "RequestId": "04069B7C-6A42-43F2-A086-50F802940045"
}
--------------------------deactivate_router_interface--------------------------
{
  "RequestId": "B2EBF829-E1C0-43E5-9BAD-DFFCBFA301F7"
```

```
}
--------------------------delete_router_interface--------------------------
{
  "RequestId": "20EEC4A6-E468-4F3C-A743-89193F7504E1"
}
--------------------------delete_router_interface--------------------------
{
  "RequestId": "59DCE168-B9CC-43A3-A54F-0D8C194BABE0"
}
-------------------------delete_virtual_border_router-------------------------
{
  "RequestId": "E1D71EEC-FE9F-4834-8780-19EBBD91A638"
}
```