Python SDK 文档

SDK说明

对象存储 Python SDK 使用开源的S3 Python SDK boto3。本文档介绍用户如何使用boto3 来使用对象存储服务。更加详细的接口参数说明，请在使用时参照boto3 API官方说明boto3。

环境依赖

此版本的Python SDK适用于Python 2.7

安装 Python S3 SDK

- 通过pip安装， pip install boto3
- 通过源码安装。
  git clone https://github.com/boto/boto3.git && cd boto3 && sudo python setup.py install

卸载 Python S3 SDK

```
pip uninstall boto3
```

快速入门

> 确认您已经理解 `对象存储` 基本概念，如 `Bucket` 、 `Object` 、 `Endpoint` 、 `AccessKey` 和 `SecretKey` 等。

下面介绍如何使用Python S3 boto3 SDK来访问对象存储服务，包括查看Bucket列表，上传文件，下载文件，查看文件列表等。 默认这些程序是写在以py后缀的脚本文件里，通过Python程序可以执行。

> 注意: 请不要用生产Bucket试验本文档中的例子

查看Bucket列表

```
from boto3.session import Session
import boto3
#Client初始化
access_key = "您的AccessKey"
secret_key = "您的SecretKey"
url = "您的Endpoint" #北京5: http://oss-cnbj01.cdsgss.com
session = Session(access_key, secret_key)
s3_client = session.client('s3', endpoint_url=url)
#Client初始化结束
#列出该用户拥有的桶
print [bucket['Name'] for bucket in s3_client.list_buckets()['Buckets']]
```

- 代码中，access_key 与 secret_key 是在对象存储服务中创建用户后，在页面上申请到的给用户用于访问对象存储的认证信息。而url是对象存储服务提供的服务地址及端口。

在初始化之后，就可以利用s3和s3_client进行各种操作了。

- Session对象承载了用户的认证信息
- session.client()对象用于服务相关的操作，目前就是用来列举Bucket；
- s3_client.list_buckets()['Buckets']对象是一个可以遍历用户Bucket信息的迭代器

> 获取用户所拥有的bucket列表。用户必须有有效的Access Key。匿名请求将不允许此操作。

新建Bucket

创建桶。桶用于对象存储用户上传的对象。桶的名字必须是唯一的，如果bucket已经被其他用户使用，则会创建失败。

```
s3_client.create_bucket(Bucket="您的bucket名")  #默认是私有的桶
#创建公开可读的桶
s3_client.create_bucket(Bucket="您的bucket名", ACL='public-read')
#ACL有如下几种"private","public-read","public-read-write","authenticated-read"
```

上传文件

把当前目录下的local.txt文件上传到对象存储服务器

```
resp = s3_client.put_object(Bucket="您的已经存在且配额可用的bucket名", Key="您上传文件后对象存储服务器端处存储的名字", Body=open("local.txt", 'rb').read())
```

下载文件

把对象存储服务器上的Object下载到本地文件`local-backup.bin`：

```
resp = s3_client.get_object(Bucket="您的已经存在的bucket名", Key="您该bucket中的对象名")
with open('local-backup.bin','wb') as f:
    f.write(resp['Body'].read())
```

> 如果桶开启了多版本，s3_client.get_object需要参数`VersionId='对应的版本号'`

列举文件

列举Bucket下的文件：

```
resp = s3_client.list_objects(Bucket="您的已经存在的bucket名")
for obj in resp['Contents']:
    print obj['Key']
```

删除文件

删除对象存储服务器端桶下的对象

```
resp = s3_client.delete_object(Bucket="您的已经存在的bucket名", Key="您要删除的对象名")
```

初始化

Python SDK几乎所有的操作都是通过session.client()进行的。这里，我们会详细说明如何初始化上述类。

确定EndPoint

请先阅读理解对象存储中的AccessKey，SecretKey，Endpoint相关的概念。

下面的代码设置对象存储的访问域名为Endpoint参数：

```
from boto3.session import Session
import boto3
import botocore
access_key = "您的AccessKey"
secret_key = "您的SecretKey"
url = "http://oss-cnbj01.cdsgss.com" #北京5
session = Session(access_key, secret_key)
s3_client = session.client('s3',
endpoint_url=url)
```

管理存储空间

存储空间（Bucket）是对象存储服务器上的命名空间，也是计费、权限控制、日志记录等高级功能的管理实体。

查看所有Bucket

s3_client.list_buckets()['Buckets'] 是可以遍历所有的Bucket的对象：

```
from boto3.session import Session
import boto3
session = Session("您的AccessKey", "您的SecretKey")
s3_client = session.client('s3', endpoint_url="您的Endpoint")
print [bucket['Name'] for bucket in s3_client.list_buckets()['Buckets']]
```

创建Bucket

```
s3_client.create_bucket(Bucket="您的bucket名")  #默认是私有的桶
#创建公开可读的桶
s3_client.create_bucket(Bucket="您的bucket名", ACL='public-read')
#ACL有如下几种"private","public-read","public-read-write","authenticated-read"
```

删除Bucket

删除指定的桶。只有该桶中所有的对象被删除了，该桶才能被删除。另外，只有该桶的拥有者才能删除它，与桶的访问控制权限无关。

```
from botocore.exceptions import ClientError
try:
    resp = s3_client.delete_bucket(Bucket="要删除的桶名")
except ClientError as e:
    print e.response['Error']['Code']
```

查看Bucket访问权限

```
resp = s3_client.get_bucket_acl(Bucket="要查询访问权限的桶")
print resp['Grants']
```

设置Bucket访问权限

```
resp = s3_client.put_bucket_acl(Bucket="要设置访问权限的桶", ACL='public-read')
```

> ACL有如下几种"private","public-read","public-read-write","authenticated-read"

上传文件

对象存储有多种上传方式，不同的上传方式能够上传的数据大小也不一样。普通上传（PutObject）最多只能上传小于或等于5GB的文件；而分片上传（MultipartUpload）每个分片可以达到5GB，合并后的文件能够达到5TB。

首先介绍普通上传，我们会详细展示提供数据的各种方式，即方法中的 data 参数。其他上传接口有类似的data参数，不再赘述。

上传字符串

上传内存中的字符串：

```
resp = s3_client.put_object(Bucket="您的已经存在且配额可用的bucket名", Key="您上传文件后对象存储服务器端处存储的名字", Body="上传的字符串内容")
```

也可以指定上传的是bytes：

```
resp = s3_client.put_object(Bucket="您的已经存在且配额可用的bucket名", Key="您上传文件后对象存储服务器端处存储的名字", Body=b"上传的bytes")
```

或是指定为unicode：

```
resp = s3_client.put_object(Bucket="您的已经存在且配额可用的bucket名", Key="您上传文件后对象存储服务器端处存储的名字", Body=u"上传的unicode")
```

上传本地文件

把当前目录下的local.txt文件上传到对象存储服务器

```
resp = s3_client.put_object(Bucket="您的已经存在且配额可用的bucket名", Key="您上传文件后对象存储服务器端处存储的名字", Body=open("local.txt", 'rb').read())
```

断点续传

当需要上传的本地文件很大，或网络状况不够理想，往往会出现上传到中途就失败了。此时，如果对已经上传的数据重新上传，既浪费时间，又占用了网络资源。Python SDK提供了分片上传接口，用于断点续传本地文件或者并发上传文件不同的区域块来加速上传。

分片上传

采用分片上传，用户可以对上传做更为精细的控制。这适用于诸如预先不知道文件大小、并发上传、自定义断点续传等场景。一次分片上传可以分为三个步骤：

1. 初始化（createMultipartUpload）：获得Upload ID
2. 上传分片（uploadPart）：这一步可以并发进行
3. 完成上传（completeMultipartUpload）：合并分片，生成文件方式1:

方式2:

```
import boto3
s3 = boto3.resource('s3', endpoint_url="您的Endpoint", aws_access_key_id='您的AccessKey', aws_secret_access_key='您的SecretKey')
bucket = s3.Bucket('您的已经存在且配额可用的bucket名')
mpu = bucket.Object('您上传文件后对象存储服务器端处存储的名字').initiate_multipart_upload() #step1.初始化
part_info = {
    'Parts': []
}
i = 1
file = open('10GiB.bin', 'rb')
while 1:
    data = file.read(10 * 1024 * 1024)#每个分块10MiB大小，可调整
    if data == b'':
        break
    part = mpu.Part(i)
    response = part.upload(Body=data)#step2.上传分片 #可改用多线程
    part_info['Parts'].append({
        'PartNumber': i,
        'ETag': response['ETag']
    })
    i += 1
mpu.complete(MultipartUpload=part_info)#step3.完成上传
```

```
mpu = s3_client.create_multipart_upload(Bucket="您的已经存在且配额可用的bucket名",Key="您上传文件后对象存储服务器端处存储的名字")#step1.初始化
part_info = {
    'Parts': []
}
i = 1
while 1:
    data = file.read(10 * 1024 * 1024)#每个分块10MiB大小，可调整
    if data == b'':
        break
    response = s3_client.upload_part(Bucket="您的已经存在且配额可用的bucket名", Key="您上传文件后对象存储服务器端处存储的名字", PartNumber=i, UploadId=mpu["UploadId"],Body=data)#step2.上传分片 #可改用多线程
    part_info['Parts'].append({
         'PartNumber': i,
        'ETag': response['ETag']
    })
    i += 1
s3_client.complete_multipart_upload(Bucket="您的已经存在且配额可用的bucket名",Key="您上传文件后对象存储服务器端处存储的名字",UploadId=mpu["UploadId"],MultipartUpload=part_info)#step3.完成上传
```

下载文件

把对象存储服务器上的Object下载到本地文件`local-backup.bin`：

```
resp = s3_client.get_object(Bucket="您的已经存在的bucket名", Key="您该bucket中的对象名")
with open('local-backup.bin','wb') as f:
    f.write(resp['Body'].read())
```

下载文件

下载对象存储服务器上的Object的前10个字节：

```
resp = s3_client.get_object(Bucket="您的已经存在的bucket名",Key="您该bucket中的对象名", Range="bytes=0-10")
print resp['Body'].read()
```

管理文件

通过Python SDK，用户可以罗列、删除、拷贝文件，也可以查看文件信息，更改文件元信息等。

罗列文件

Python SDK提供了一系列的迭代器，用于列举文件、分片上传等。

简单罗列

列举Bucket里的文件：

```
resp = s3_client.list_objects(Bucket="您的已经存在的bucket名")
for obj in resp['Contents']:
    print obj['Key']
```

按前缀罗列

只列举前缀为”img-“的所有文件：

```
resp = s3_client.list_objects(Bucket="您的已经存在的bucket名",Prefix='img-')
for obj in resp['Contents']:
    print obj['Key']
```

模拟文件夹功能

对象存储的存储空间（Bucket）本身是扁平结构的，并没有文件夹或目录的概念。用户可以通过在文件名里加入”/“来模拟文件夹。在列举的时候，则要设置delimiter参数（目录分隔符）为"/"，并通过是否"在CommonPrefixes"来判断是否为文件夹。
例如：websitebucket1下结构如下：

```
.
├── about.html
├── blog
│   ├── angry-post.html
│   ├── atom.xml
│   ├── excerpts.xml
│   ├── happy-post.html
│   ├── index.html
│   ├── sad-post.html
│   └── tags
│       ├── angry.html
│       ├── happy.html
│       ├── sad.html
│       └── thoughts.html
├── index.html
├── media
│   ├── css
│   │   ├── site.css
│   │   └── syntax.css
│   ├── images
│   │   ├── airport.png
│   │   ├── apple-touch-icon.png
│   │   ├── dark.png
│   │   └── favicon.ico
│   └── js
│       └── libs
│           ├── dd_belatedpng.js
│           ├── jquery-1.5.1.min.js
│           └── modernizr-1.7.min.js
└── portfolio
    └── index.html
```

```
#列举websitebucket1下的文件夹和文件（不递归列出子文件夹下的文件和子子文件夹）
resp = s3_client.list_objects(Bucket="websitebucket1", Delimiter='/')
print "==============DIRS FOLLOWS============"
for o in resp.get('CommonPrefixes'):
    print(o.get('Prefix'))
print "==============FILES FOLLOWS============"
for o in resp.get('Contents'):
    print(o.get('Key'))
```

结果

```
==============DIRS FOLLOWS============
blog/
media/
portfolio/
==============FILES FOLLOWS============
about.html
copy3
data4
index.html
```

```
#列举websitebucket1下的文件夹blog下的文件夹和文件（不递归列出子文件夹下的文件和子子文件夹）
resp = s3_client.list_objects(Bucket="websitebucket1", Delimiter='/',Prefix='blog/')
print "==============DIRS FOLLOWS============"
for o in resp.get('CommonPrefixes'):
    print(o.get('Prefix'))
print "==============FILES FOLLOWS============"
for o in resp.get('Contents'):
    print(o.get('Key'))
```

结果

```
==============DIRS FOLLOWS============
blog/tags/
==============FILES FOLLOWS============
blog/
blog/angry-post.html
blog/atom.xml
blog/excerpts.xml
blog/happy-post.html
blog/index.html
blog/sad-post.html
```

判断文件是否存在

可以使用head接口来判断对象是否存在

```
try:
    s3_client.head_object(Bucket="您的已经存在的bucket名", Key="要查询的对象名字")
    print "EXIST"
except:
    print "NOT FOUND"
```

删除文件

删除单个文件

```
resp = s3_client.delete_object(Bucket="您的已经存在的bucket名", Key="您要删除的对象名")
```

删除多个文件

批量删除bucket1下的以2017-05为前缀的对象

```
import boto3
s3 = boto3.resource('s3', endpoint_url="您的ENDPOINT", aws_access_key_id='您的AccessKey',
aws_secret_access_key='您的SecretKey')
bucket = s3.Bucket('bucket1')
objects_to_delete = []
for obj in bucket.objects.filter(Prefix='2017-05'):
    objects_to_delete.append({'Key': obj.key})
bucket.delete_objects( Delet
    e={
        'Objects': objects_to_delete
    }
)
```

拷贝文件

把Bucket名为src-bucket下的source.txt拷贝到dst_bucket下且命名为的target.txt文件。

```
resp = s3_client.copy_object(Bucket=dst_bucket, Key="target.txt",
CopySource=str(src_bucket+'/'+"source.txt"))
```

查看文件访问权限

查看文件的访问权限

```
resp = s3_client.get_object_acl(Bucket="用户存在的bucket", Key="bucket下的对象名字")
print resp['Grants']
print resp['Owner']
```

设置文件访问权限

设置文件的访问权限

```
resp = s3_client.put_object_acl(Bucket="用户存在的bucket", Key="bucket下的对象名字", ACL='public-
read')
```

使用私有链接下载

对于私有Bucket，可生成私有链接（又称为“签名URL”），下面是生成私有链接下载，该链接在3600 秒后失效

```
print
  s3_client.generate_presigned_url( Cl
  ientMethod = 'get_object',
  Params = {'Bucket' : "您的bucket", 'Key' : "您的bucket下的要生成私有下载链接的对象"},
  ExpiresIn = 3600,
  HttpMethod = 'GET')
```