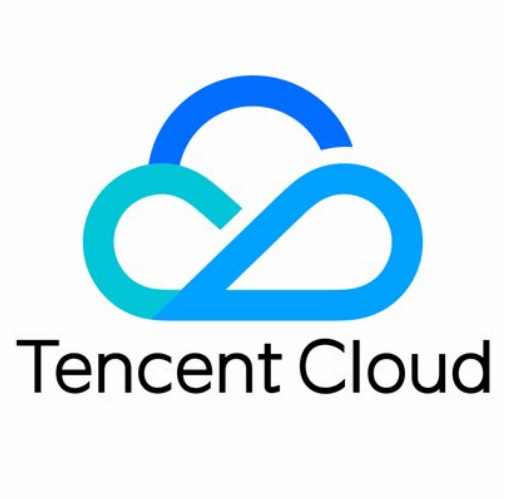
Batch Compute

Command Line Interface

Product Documentation



Contents

Command Line Interface Preparation

Last updated : 2020-10-28 11:15:00

Before installing Tencent Cloud's command line tool TCCLI, ensure that a Python environment has been installed. For more information, see Prerequisites.

Step 1. Installing TCCLI

Installing TCCLI

Run commands based on the actual situation.

- **TCCLI not installed**
  Run the following command to install TCCLI through pip. For more information, see Installing TCCLI.

```
$ sudo pip install tccli
```

- **TCCLI installed**
  Run the following command to quickly upgrade TCCLI through pip:

```
$ sudo pip install --upgrade tccli
```

Verifying Installation

Run the following command to check whether TCCLI is successfully installed and has Batch-related capabilities:

```
tccli batch help
```

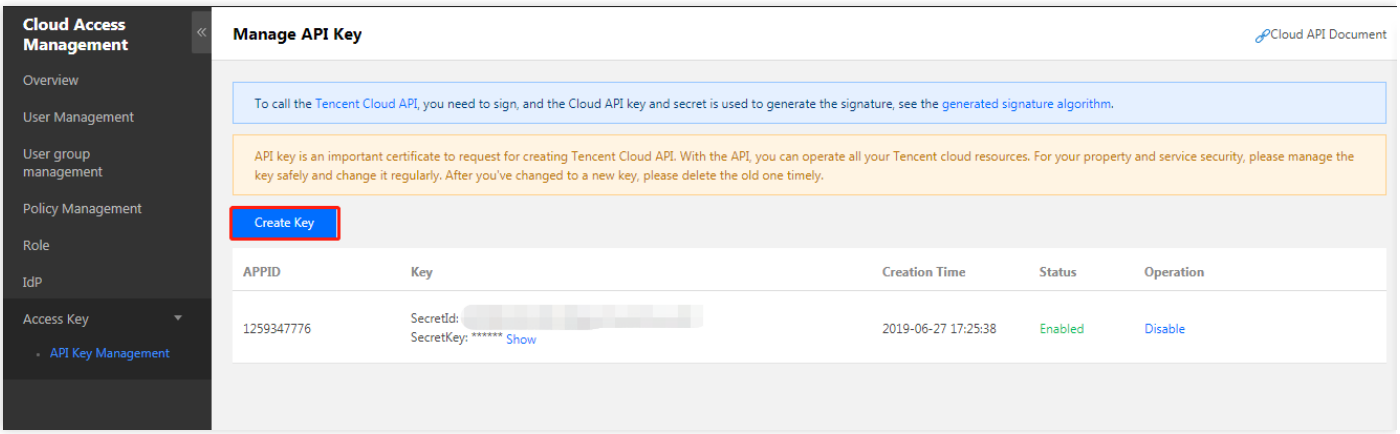
The returned result is as follows, indicating that TCCLI is successfully installed:

```
NAME
batch
DESCRIPTION
batch-2017-03-12
USEAGE
tccli batch <action> [--param...]
OPTIONS
help
```

```
show the tccli batch help info
--version
specify a batch api version
AVAILABLE ACTION
DescribeComputeEnv
Used to query details of the computing environment
CreateTaskTemplate
Used to create a task template
```

Step 2. Configuring TCCLI

1. Log in to the API Key Management.
2. Click **Create Key** or use an available key to record `SecretID` and `SecretKey`. See the figure below:



3. Run the `tccli configure` command and enter the TCCLI configuration information. For more information, see Configuring TCCLI.

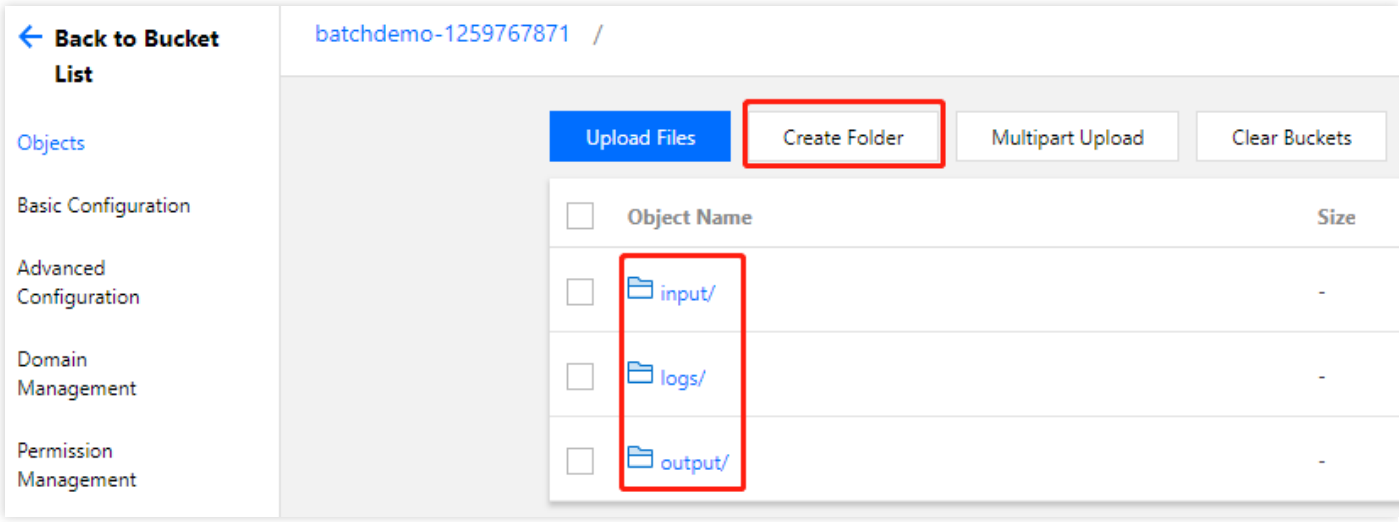
```
$ tccli configure
TencentCloud API secretId[None]:
TencentCloud API secretKey[None]:
region[None]:
output[json]:
```

Step 3. Preparing the COS Directory

Creating a Bucket and Subfolders

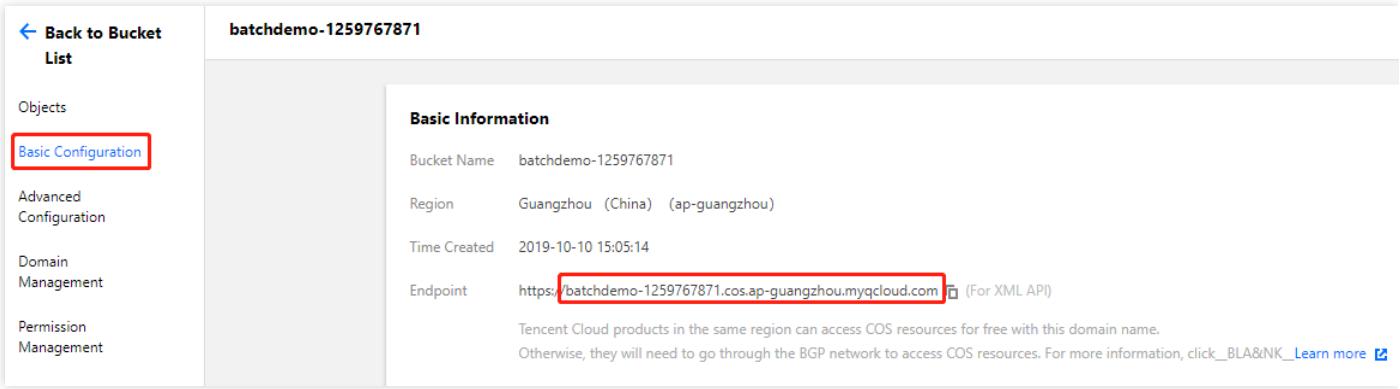
1. Log in to the COS console and choose **Bucket List** in the left sidebar.

2. Create a bucket and create 3 folders in the bucket. See the figure below:



Acquiring COS

1. Click **Basic Configuration** on the left to view the endpoint in **Basic Information**. See the figure below:



2. Acquire the endpoints of subfolders in the a COS bucket.

> Acquire COS-related endpoints based on the actual situation.

The acquired COS bucket endpoint is `https://batchdemo-xxxxxxxxx.cos.ap-guangzhou.myqcloud.com`. The endpoints of the three folders created in Creating a Bucket and Subfolders can be acquired by combining `domain and subfolder names` as follows:

- `cos://batchdemo-xxxxxxxxx.cos.ap-guangzhou.myqcloud.com/logs/`
- `cos://batchdemo-xxxxxxxxx.cos.ap-guangzhou.myqcloud.com/input/`
- `cos://batchdemo-xxxxxxxxx.cos.ap-guangzhou.myqcloud.com/output/`

Step 4. Downloading the Demo File

Access the Batch demo, download the test package, and decompress it.

> The demo is provided in the format of the Python+Batch command line tool. Since Batch has many capabilities and configuration items, you can work with it more conveniently by using Python scripts.

Step 5. Modifying Demo Custom Information

> Change the general part of the custom information about the Batch demo. Modify all files in Demo as follows:

Use the following custom information in `1_SimpleStart.py` as an example:

```
# custom (Change to your info)
imageId = "img-m4q71qnf"
Application = {
"DeliveryForm": "LOCAL",
"Command": " python -c ¥"fib=lambda n:1 if n<=2 else fib(n-1)+fib(n-2); print(fib(20))¥" "
}
StdoutRedirectPath = "cos://batchdemo-xxxxxxxxxx.cos.ap-guangzhou.myqcloud.com/logs/"
StderrRedirectPath = "cos://batchdemo-xxxxxxxxxx.cos.ap-guangzhou.myqcloud.com/logs/"
```

The following table lists the information to be modified.

Configuration Item	Description
imageId	• The image containing the Cloud-init service is required. • Custom images are created based on this image.
StdoutRedirectPath	Enter the complete log folder endpoint acquired in Acquiring COS-related endpoints.
StderrRedirectPath	
Application	The startup command. Use the default setting.

```
cmd = "tccli batch SubmitJob ¥
--version 2017-03-12 ¥
--Placement '{¥"Zone¥": ¥"ap-guangzhou-2¥"}' ¥
--Job ' %s ' "%(json.dumps(testJob))
```

The demo specifies Guangzhou Zone 2 for resource application. You can select the corresponding availability zone to apply for resources based on the default region configured in TCCLI.
For more information about regions and availability zones, see Regions and Availability Zones.

Step 6 Performing a Test

Experience the Batch usage methods and computing capability in the following sequence according to the reference course.

1. 1_SimpleStart.py: Quick Start
2. 2_RemoteCodePkg.py: Running Remote Package
3. 3_StoreMapping.py: Mapping Remote Storage

Quick Start

Last updated : 2020-08-19 15:15:17

Scenario

This document describes the usage methods and computing capability of Batch.

Prerequisites

Complete preparations based on the instructions in Preparation, and learn how to configure the general part of the custom information.

Viewing the Demo

> Modify the general part of the custom information in `1_SimpleStart.py` based on the instructions in Preparation.

Open `1_SimpleStart.py` in an editor.

```
# custom (Change to your info)
imageId = "img-m4q71qnf"
Application = {
"DeliveryForm": "LOCAL",
"Command": " python -c ¥"fib=lambda n:1 if n<=2 else fib(n-1)+fib(n-2); print(fib(20))¥" "
}
StdoutRedirectPath = "your cos path"
StderrRedirectPath = "your cos path"
```

All information except `Application` in the custom part is described in Preparation. Set the parameters of `Application` based on the following table.

Configuration Item	Description
DeliveryForm	Three delivery methods of applications are available: software packaging, container image, and direct running within the CVM. In this case, LOCAL indicates direct running within the CVM.

Command	Task startup command. In this case, a Python script is executed. Starting from 1, add up the first 20 numbers of the Fibonacci sequence and output the result to StdOutput.

Submitting a Job

Run the following command to run the Python script.

The demo encapsulates the job submission process by using Python scripts and the Batch command line tool.

```
python 1_SimpleStart.py
```

The returned result is as follows, indicating that the job is successfully submitted:

```
{
"RequestId": "393292f4-5583-48ad-a9f5-f673138ea637",
"JobId": "job-o0xxxxxq7"
}
```

If the submit operation fails, check the returned value or Contact Us.

Viewing State

- Run the following command to view the execution status through DescribeJob:

```
$ tccli batch DescribeJob --version 2017-03-12 --JobId job-xxx
```

> --Replace job-xxx with the JobId acquired after Submitting a Job.

The returned result is as follows (some information is omitted):

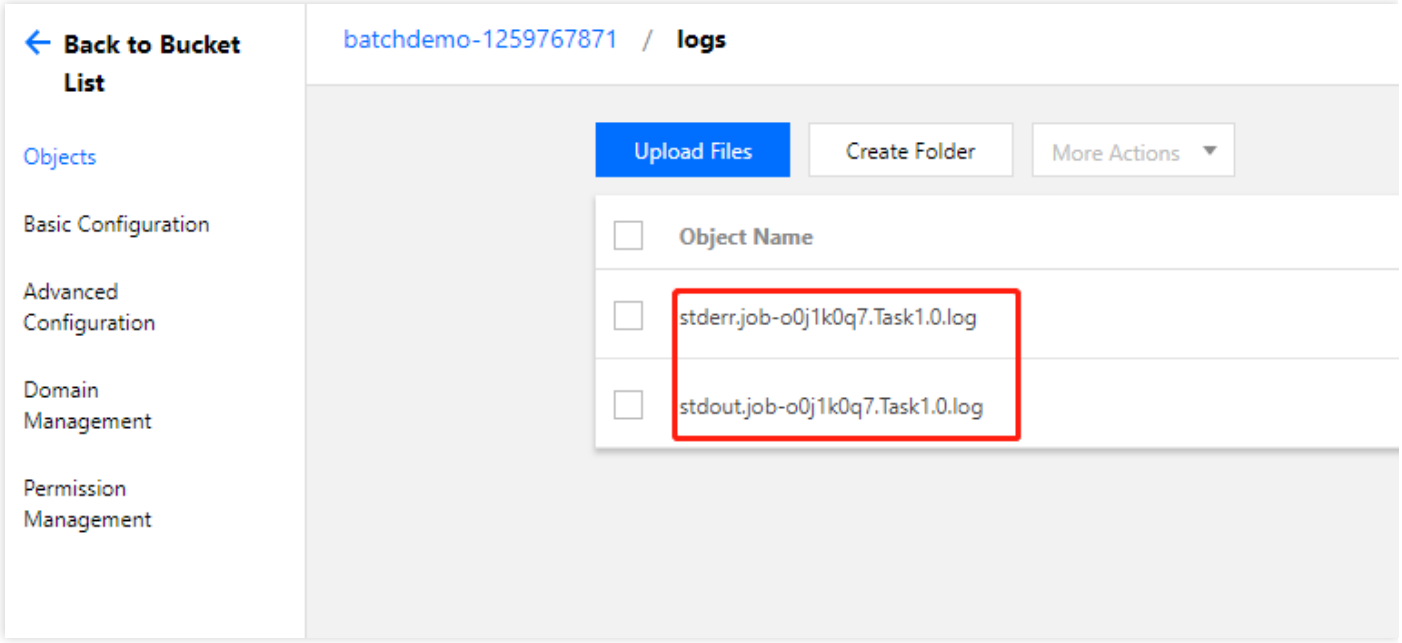
```
{
"EndTime": "2019-10-08T04:06:58Z",
"JobState": "SUCCEED",
"TaskInstanceMetrics": {
...
},
"Zone": "ap-guangzhou-2",
"TaskMetrics": {
...
},
"JobName": "TestJob",
"Priority": 1,
```

```
"RequestId": "7a5f4c94-1357-486c-9c48-8286ba01b5b2",
"TaskSet": [
...
],
"StateReason": null,
"JobId": "job-o0xxxxxq7",
"DependenceSet": [],
"CreateTime": "2019-10-08T04:05:54Z"
}
```


- The returned result includes the following common execution statuses:
  - STARTING: Launching
  - RUNNING: Running
  - SUCCEED: Running successfully
  - FAILED: Failed to run

Viewing the Result

1. Log in to the COS console and click **Bucket List** in the left sidebar.
2. Select the ID of the created bucket, click **Objects**, and select the **logs**, which stores the execution results. See the figure below:



  - View the standard output in stdout.job-xxx.xxxx.0.log for a successful operation. The content is as follows:

    ```
    6765
    ```

  - View the standard error in stderr.job-xxx.xxxx.0.log in case of a failure. The content may be:

```
/bin/sh: -c: line 0: syntax error near unexpected token `('
/bin/sh: -c: line 0: ` python -c ¥"fib=lambda n:1 if n<=2 else fib(n-1)+fib(n-2); print(fib
(20))¥" '
```

Running Remote Package

Last updated : 2020-10-28 11:19:35

Scenario

Batch allows you to acquire a code package from a .tgz file via HTTP. You can compress the code and upload it to COS. This helps you organize the code more conveniently than using LOCAL mode.

Prerequisites

Complete preparations based on the instructions in Preparation, and learn how to configure the general part of the custom information.

Steps

Viewing the Demo

> Modify the general part of the custom information in `2_RemoteCodePkg.py` based on the instructions in Preparation.

Open the `2_RemoteCodePkg.py` file in an editor.

```
# custom (Change to your info)
imageId = "img-m4q71qnf"
Application = {
"DeliveryForm": "PACKAGE",
"Command": "python ./codepkg/fib.py",
"PackagePath": "http://batchdemo-xxxxxxxxx.cos.ap-guangzhou.myqcloud.com/codepkg/codepkg.tgz"
}
StdoutRedirectPath = "your cos path"
StderrRedirectPath = "your cos path"
```

All information except `Application` in the general part is described in Preparation. Set the parameters of `Application` based on the following table.

Configuration	Description

Item	
DeliveryForm	Three delivery methods of applications are available: software packaging, container image, and direct running within the CVM. In this case, PACKAGE indicates software packaging.
PackagePath	Address of the .tgz software package in HTTP. Batch downloads the software package to a directory of the scheduled CVM and runs `Command` in the directory.
Command	Task startup command. In this case, a Python script file in the software package is directly called. You can download the package, and view the file structure and content in it.

`fib.py` is composed as below:

```
fib = lambda n:1 if n<=2 else fib(n-1)+fib(n-2)
print("Remote Code Package : %d"%(fib(20)))
```

Submitting a Job

Run the following command to run the Python script.
The demo encapsulates the job submission process by using Python scripts and the Batch command line tool.

```
python 2_RemoteCodePkg.py
```

The returned result is as follows, indicating that the job is successfully submitted:

```
{
"RequestId": "c09e9291-2661-xxxx-8783-72d36f91ec8a",
"JobId": "job-7xxxx26l"
}
```

If the submit operation fails, check the returned value or Contact Us.

Viewing State

See Viewing State in Quick Start.

Viewing the Result

1. See Viewing the Result in Quick Start.
2. The execution result of `2_RemoteCodePkg.py` is as follows:

```
Remote Code Package : 6765
```

Mapping Remote Storage

Last updated : 2020-12-22 14:23:00

Operation Scenarios

Remote mapping is used as an auxiliary feature by BatchCompute for storage services to map remote storage, such as COS and CFS, to a local folder.

Prerequisites

Complete the preparations as instructed in Preparations and learn how to configure the general part of the custom information.

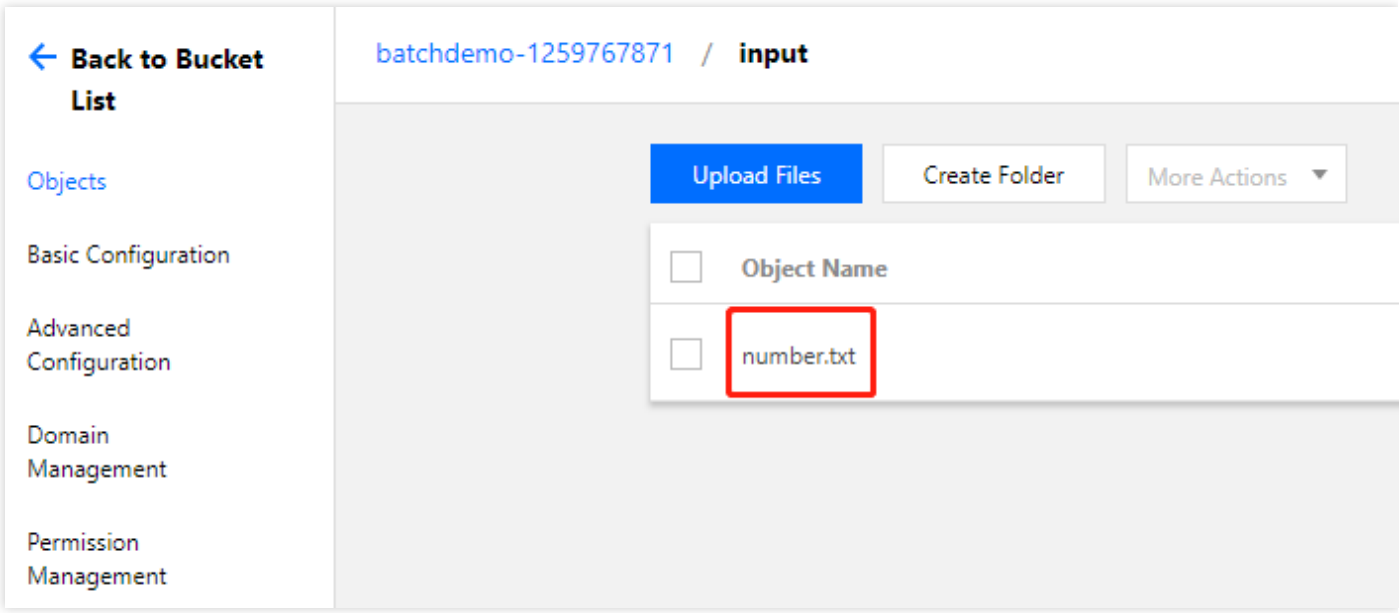
Directions

Uploading input data file

1. Create `number.txt` with the following content:

```
1
2
3
4
5
6
7
8
9
```

2. Log in to the COS Console and click **Bucket List** on the left sidebar.
3. Select the ID of the created bucket, click **Objects**, select **input**, and upload `number.txt` as shown below:



Viewing and modifying demo

Note :

Modify the general part of the custom information in `3_StoreMapping.py` as instructed in Preparations.

Open `3_StoreMapping.py` in an editor.

```
# custom (Change to your info)
imageId = "img-m4q71qnf"
Application = {
"DeliveryForm": "PACKAGE",
"Command": "python ./codepkg/sumnum.py",
"PackagePath": "http://batchdemo-xxxxxxxxx.cos.ap-guangzhou.myqcloud.com/codepkg/codepkg.tgz"
}
StdoutRedirectPath = "your cos path"
StderrRedirectPath = "your cos path"
InputMapping = {
"SourcePath": "cos://batchdemo-xxxxxxxxx.cos.ap-guangzhou.myqcloud.com/input/",
"DestinationPath": "/data/input/"
}
OutputMapping = {
"SourcePath": "/data/output/",
"DestinationPath": "your output remote path"
}
```

Compared to `2_RemoteCodePkg.py`, some of the parameters in the custom part are modified as follows:

Configuration Item	Description
Application	Modify `Command` to `sumnum.py`.
InputMapping	Input mapping. • `SourcePath` remote storage address: modify this address to the path of the **input** folder in "Preparations". For more information, please see Getting COS Endpoints. • `DestinationPath` local directory: retain the original setting for the time being.
OutputMapping	Output mapping. • `SourcePath` local directory: retain the original setting for the moment. • `DestinationPath` remote storage address: modify this address to the path of the **output** folder in "Preparations". For more information, please see Getting COS Endpoints.

sumnum.py is composed as below:

Open `input/number.txt`, add up all numbers in each row, and write the result into `output/result.txt`.

```
import os

inputfile = "/data/input/number.txt"
outputfile = "/data/output/result.txt"

def readFile(filename):
total = 0
fopen = open(filename, 'r')
for eachLine in fopen:
total += int(eachLine)
fopen.close()
print "total = ",total
fwrite = open(outputfile, 'w')
fwrite.write(str(total))
fwrite.close()

print("Local input file is ",inputfile)
readFile(inputfile)
```

Submitting job

Run the following command to run the Python script.
The demo encapsulates the job submission process by using Python scripts and the BatchCompute command line tool.

```
python 3_StoreMapping.py
```

The returned result is as follows, indicating that the job is successfully submitted:

```
{
"RequestId": "8eaeb01e-94a6-41a1-b40f-95f15417c0b4",
"JobId": "job-97smiptb"
}
```

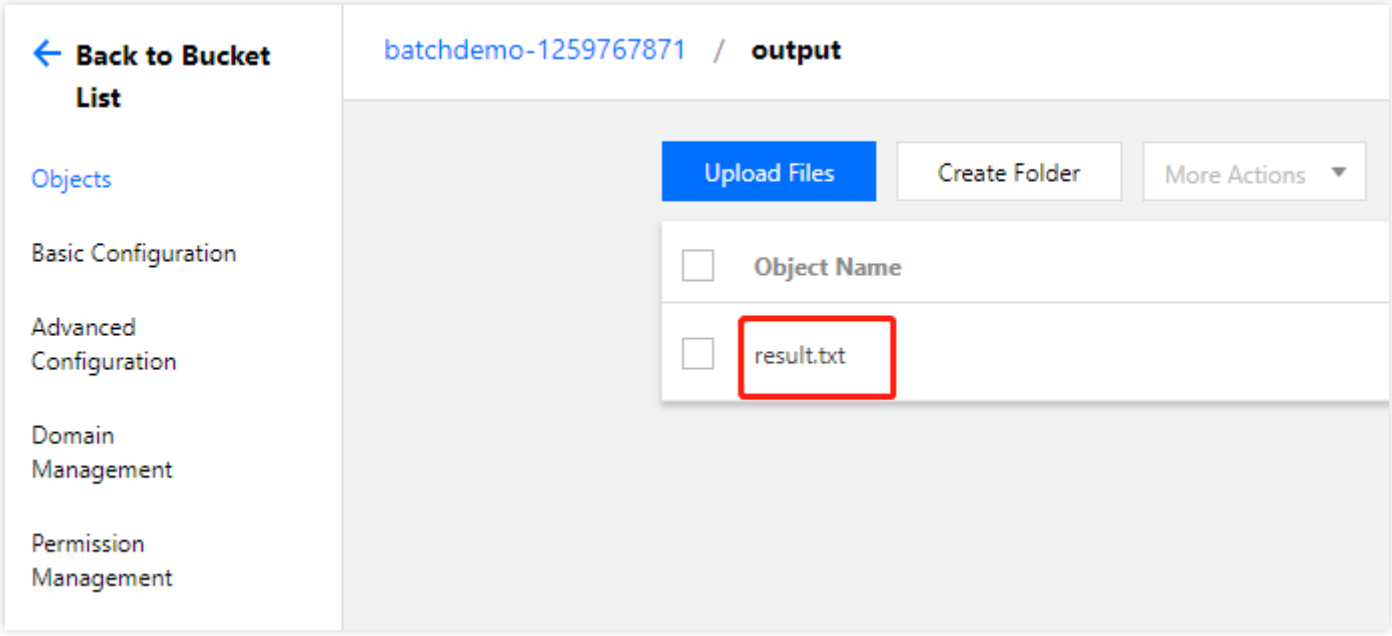
If the submission operation fails, check the returned value or contact us for assistance.

Viewing status

Please see Viewing Status in Quick Start.

Viewing result

1. Log in to the COS Console and click **Bucket List** on the left sidebar.
2. Select the ID of the created bucket, click **Files**, and select **output** as shown below:
   BatchCompute copies the output data from the local directory to the remote directory. The execution results of `3_StoreMapping.py` are stored in `result.txt`, which is automatically synced to COS.



`result.txt` is composed as below:

```
45
```