Importing IBM database

```
import ibm_db
```

Initializiing the values of the databse

```
# dsn_hostname = "8e359033-a1c9-4643-82ef-
8ac06f5107eb.bs2io90l08kqb1od8lcg.databases.appdomain.cloud"
# dsn_uid = "gcn28434"
# dsn_pwd = "jjXTxanatTQvE11F"
# dsn_driver = "{IBM DB2 ODBC DRIVER}"
# dsn_database = "bludb"
# dsn_port = "31249"
# dsn_protocol = "TCPIP"

conn = ibm_db.connect("DATABASE=bludb;HOSTNAME=8e359033-a1c9-4643-
82ef-
8ac06f5107eb.bs2io90l08kqb1od8lcg.databases.appdomain.cloud;PORT=30120
;SECURITY=SSL;SSLServerCertificate=DigiCertGlobalRootCA.crt;UID=vrk690
34;PWD=sHi69YPZ3KjFmkuD",'','')
print(conn)
print("Connection Successful")
```

```
<ibm_db.IBM_DBConnection object at 0x000001CC32B383B0>
Connection Successful
```

Initializing configuration

```
import ibm_boto3
from ibm_botocore.client import Config, ClientError

# Constants for IBM COS values
COS_ENDPOINT =
"https://control.cloud-object-storage.cloud.ibm.com/v2/endpoints" #
Current list avaiable at https://control.cloud-object-
storage.cloud.ibm.com/v2/endpoints
COS_API_KEY_ID = "OXFtQiq3DXAYPoAeug9Sy6xKv_6EtnxvV9A1Sw9OF-eT" # eg
"W00YixxxxxxxxxMB-odB-2ySfTrFBIQQWanc--P3byk"
COS_INSTANCE_CRN = "crn:v1:bluemix:public:cloud-object-
storage:global:a/aae2d6e4cb8b434781d4640a06b81495:f85d7d56-3a8a-466c-
bc6e-a46fafde85fd::" # eg "crn:v1:bluemix:public:cloud-object-
storage:global:a/3bf0d9003xxxxxxxxxx1c3e97696b71c:d6f04d83-6c4f-4a62-
a165-696756d63903::"

# Create resource
cos = ibm_boto3.resource("s3",
    ibm_api_key_id=COS_API_KEY_ID,
    ibm_service_instance_id=COS_INSTANCE_CRN,
    config=Config(signature_version="oauth"),
```

```
    endpoint_url=COS_ENDPOINT
)
```

Creating a new bucket

```
def create_bucket(bucket_name):
    print("Creating new bucket: {0}".format(bucket_name))
    try:
        cos.Bucket(bucket_name).create(
            CreateBucketConfiguration={
                "LocationConstraint":COS_BUCKET_LOCATION
            }
        )
        print("Bucket: {0} created!".format(bucket_name))
    except ClientError as be:
        print("CLIENT ERROR: {0}\n".format(be))
    except Exception as e:
        print("Unable to create bucket: {0}".format(e))
```

Creating a new text file

```
def create_text_file(bucket_name, item_name, file_text):
    print("Creating new item: {0}".format(item_name))
    try:
        cos.Object(bucket_name, item_name).put(
            Body=file_text
        )
        print("Item: {0} created!".format(item_name))
    except ClientError as be:
        print("CLIENT ERROR: {0}\n".format(be))
    except Exception as e:
        print("Unable to create text file: {0}".format(e))
```

List available buckets

```
def get_buckets():
    print("Retrieving list of buckets")
    try:
        buckets = cos.buckets.all()
        for bucket in buckets:
            print("Bucket Name: {0}".format(bucket.name))
    except ClientError as be:
        print("CLIENT ERROR: {0}\n".format(be))
    except Exception as e:
        print("Unable to retrieve list buckets: {0}".format(e))
```

List items in bucket

```
def get_bucket_contents(bucket_name):
    print("Retrieving bucket contents from: {0}".format(bucket_name))
    try:
        files = cos.Bucket(bucket_name).objects.all()
```

```
        for file in files:
            print("Item: {0} ({1} bytes).".format(file.key,
file.size))
    except ClientError as be:
        print("CLIENT ERROR: {0}\n".format(be))
    except Exception as e:
        print("Unable to retrieve bucket contents: {0}".format(e))
```

Get file contents of particular item

```
def get_item(bucket_name, item_name):
    print("Retrieving item from bucket: {0}, key:
{1}".format(bucket_name, item_name))
    try:
        file = cos.Object(bucket_name, item_name).get()
        print("File Contents: {0}".format(file["Body"].read()))
    except ClientError as be:
        print("CLIENT ERROR: {0}\n".format(be))
    except Exception as e:
        print("Unable to retrieve file contents: {0}".format(e))
```

Delete an item from a bucket

```
def delete_item(bucket_name, object_name):
    try:
        cos.delete_object(Bucket=bucket_name, Key=object_name)
        print("Item: {0} deleted!\n".format(object_name))
    except ClientError as be:
        print("CLIENT ERROR: {0}\n".format(be))
    except Exception as e:
        print("Unable to delete object: {0}".format(e))
```

Delete multiple items from a bucket

```
def delete_items(bucket_name):
    try:
        delete_request = {
            "Objects": [
                { "Key": "deletetest/testfile1.txt" },
                { "Key": "deletetest/testfile2.txt" },
                { "Key": "deletetest/testfile3.txt" },
                { "Key": "deletetest/testfile4.txt" },
                { "Key": "deletetest/testfile5.txt" }
            ]
        }

        response = cos.delete_objects(
            Bucket=bucket_name,
            Delete=delete_request
        )
```

```
        print("Deleted items for {0}\n".format(bucket_name))
        print(json.dumps(response.get("Deleted"), indent=4))
    except ClientError as be:
        print("CLIENT ERROR: {0}\n".format(be))
    except Exception as e:
        print("Unable to copy item: {0}".format(e))
```

Delete a bucket

```
def delete_bucket(bucket_name):
    print("Deleting bucket: {0}".format(bucket_name))
    try:
        cos.Bucket(bucket_name).delete()
        print("Bucket: {0} deleted!".format(bucket_name))
    except ClientError as be:
        print("CLIENT ERROR: {0}\n".format(be))
    except Exception as e:
        print("Unable to delete bucket: {0}".format(e))
```

Run a multi-part upload

```
def multi_part_upload(bucket_name, item_name, file_path):
    try:
        print("Starting file transfer for {0} to bucket: {1}\
n".format(item_name, bucket_name))
        # set 5 MB chunks
        part_size = 1024 * 1024 * 5

        # set threadhold to 15 MB
        file_threshold = 1024 * 1024 * 15

        # set the transfer threshold and chunk size
        transfer_config = ibm_boto3.s3.transfer.TransferConfig(
            multipart_threshold=file_threshold,
            multipart_chunksize=part_size
        )

        # the upload_fileobj method will automatically execute a
multi-part upload
        # in 5 MB chunks for all files over 15 MB
        with open(file_path, "rb") as file_data:
            cos.Object(bucket_name, item_name).upload_fileobj(
                Fileobj=file_data,
                Config=transfer_config
            )

        print("Transfer for {0} Complete!\n".format(item_name))
    except ClientError as be:
        print("CLIENT ERROR: {0}\n".format(be))
    except Exception as e:
        print("Unable to complete multi-part upload: {0}".format(e))
```

Manually run a multi-part upload

```
def multi_part_upload_manual(bucket_name, item_name, file_path):
    try:
        # create client object
        cos_cli = ibm_boto3.client("s3",
            ibm_api_key_id=COS_API_KEY_ID,
            ibm_service_instance_id=COS_SERVICE_CRN,
            config=Config(signature_version="oauth"),
            endpoint_url=COS_ENDPOINT
        )

        print("Starting multi-part upload for {0} to bucket: {1}\
n".format(item_name, bucket_name))

        # initiate the multi-part upload
        mp = cos_cli.create_multipart_upload(
            Bucket=bucket_name,
            Key=item_name
        )

        upload_id = mp["UploadId"]

        # min 20MB part size
        part_size = 1024 * 1024 * 20
        file_size = os.stat(file_path).st_size
        part_count = int(math.ceil(file_size / float(part_size)))
        data_packs = []
        position = 0
        part_num = 0

        # begin uploading the parts
        with open(file_path, "rb") as file:
            for i in range(part_count):
                part_num = i + 1
                part_size = min(part_size, (file_size - position))

                print("Uploading to {0} (part {1} of
{2})".format(item_name, part_num, part_count))

                file_data = file.read(part_size)

                mp_part = cos_cli.upload_part(
                    Bucket=bucket_name,
                    Key=item_name,
                    PartNumber=part_num,
                    Body=file_data,
                    ContentLength=part_size,
                    UploadId=upload_id
                )
```

```
                data_packs.append({
                    "ETag":mp_part["ETag"],
                    "PartNumber":part_num
                })

                position += part_size

        # complete upload
        cos_cli.complete_multipart_upload(
            Bucket=bucket_name,
            Key=item_name,
            UploadId=upload_id,
            MultipartUpload={
                "Parts": data_packs
            }
        )
        print("Upload for {0} Complete!\n".format(item_name))
    except ClientError as be:
        # abort the upload
        cos_cli.abort_multipart_upload(
            Bucket=bucket_name,
            Key=item_name,
            UploadId=upload_id
        )
        print("Multi-part upload aborted for {0}\n".format(item_name))
        print("CLIENT ERROR: {0}\n".format(be))
    except Exception as e:
        print("Unable to complete multi-part upload: {0}".format(e))
```

Large Object Upload by using TransferManager

```
def upload_large_file(bucket_name, item_name, file_path):
    print("Starting large file upload for {0} to bucket:
{1}".format(item_name, bucket_name))

    # set the chunk size to 5 MB
    part_size = 1024 * 1024 * 5

    # set threadhold to 5 MB
    file_threshold = 1024 * 1024 * 5

    # Create client connection
    cos_cli = ibm_boto3.client("s3",
        ibm_api_key_id=COS_API_KEY_ID,
        ibm_service_instance_id=COS_SERVICE_CRN,
        config=Config(signature_version="oauth"),
        endpoint_url=COS_ENDPOINT
    )
```

```
    # set the transfer threshold and chunk size in config settings
    transfer_config = ibm_boto3.s3.transfer.TransferConfig(
        multipart_threshold=file_threshold,
        multipart_chunksize=part_size
    )

    # create transfer manager
    transfer_mgr = ibm_boto3.s3.transfer.TransferManager(cos_cli,
config=transfer_config)

    try:
        # initiate file upload
        future = transfer_mgr.upload(file_path, bucket_name,
item_name)

        # wait for upload to complete
        future.result()

        print ("Large file upload complete!")
    except Exception as e:
        print("Unable to complete large file upload: {0}".format(e))
    finally:
        transfer_mgr.shutdown()
```

List items in a bucket (v2)

```
def get_bucket_contents_v2(bucket_name, max_keys):
    print("Retrieving bucket contents from: {0}".format(bucket_name))
    try:
        # create client object
        cos_cli = ibm_boto3.client("s3",
            ibm_api_key_id=COS_API_KEY_ID,
            ibm_service_instance_id=COS_SERVICE_CRN,
            config=Config(signature_version="oauth"),
            endpoint_url=COS_ENDPOINT)

        more_results = True
        next_token = ""

        while (more_results):
            response = cos_cli.list_objects_v2(Bucket=bucket_name,
MaxKeys=max_keys, ContinuationToken=next_token)
            files = response["Contents"]
            for file in files:
                print("Item: {0} ({1} bytes).".format(file["Key"],
file["Size"]))

            if (response["IsTruncated"]):
                next_token = response["NextContinuationToken"]
                print("...More results in next batch!\n")
```

```
            else:
                more_results = False
                next_token = ""

    except ClientError as be:
        print("CLIENT ERROR: {0}\n".format(be))
    except Exception as e:
        print("Unable to retrieve bucket contents: {0}".format(e))
```

Creating a bucket with key-protect enabled

```
COS_KP_ALGORITHM = "<algorithm>"
COS_KP_ROOTKEY_CRN = "<root-key-crn>"

# Create a new bucket with key protect (encryption)
def create_bucket_kp(bucket_name):
    print("Creating new encrypted bucket: {0}".format(bucket_name))
    try:
        cos.Bucket(bucket_name).create(
            CreateBucketConfiguration={
                "LocationConstraint":COS_BUCKET_LOCATION
            },
            IBMSSEKPEncryptionAlgorithm=COS_KP_ALGORITHM,
            IBMSSEKPCustomerRootKeyCrn=COS_KP_ROOTKEY_CRN
        )
        print("Encrypted Bucket: {0} created!".format(bucket_name))
    except ClientError as be:
        print("CLIENT ERROR: {0}\n".format(be))
    except Exception as e:
        print("Unable to create encrypted bucket: {0}".format(e))

# import cos_aspera
import ibm_boto3
from ibm_botocore.client import Config
from ibm_s3transfer.aspera.manager import AsperaTransferManager
# from ibm_s3transfer.manager import TransferCoordinatorController

COS_ENDPOINT =
"https://control.cloud-object-storage.cloud.ibm.com/v2/endpoints" #
Current list avaiable at https://control.cloud-object-
storage.cloud.ibm.com/v2/endpoints
COS_API_KEY_ID = "OXFtQiq3DXAYPoAeug9Sy6xKv_6EtnxvV9A1Sw9OF-eT"
COS_RESOURCE_CRN = "crn:v1:bluemix:public:cloud-object-
storage:global:a/aae2d6e4cb8b434781d4640a06b81495:f85d7d56-3a8a-466c-
bc6e-a46fafde85fd::"
COS_BUCKET_LOCATION = "us-south"

# Create resource
cos = ibm_boto3.client("s3",
    ibm_api_key_id=COS_API_KEY_ID,
```

```
    ibm_service_instance_id=COS_RESOURCE_CRN,
    config=Config(signature_version="oauth"),
    endpoint_url=COS_ENDPOINT
)

transfer_manager = AsperaTransferManager(cos)
```