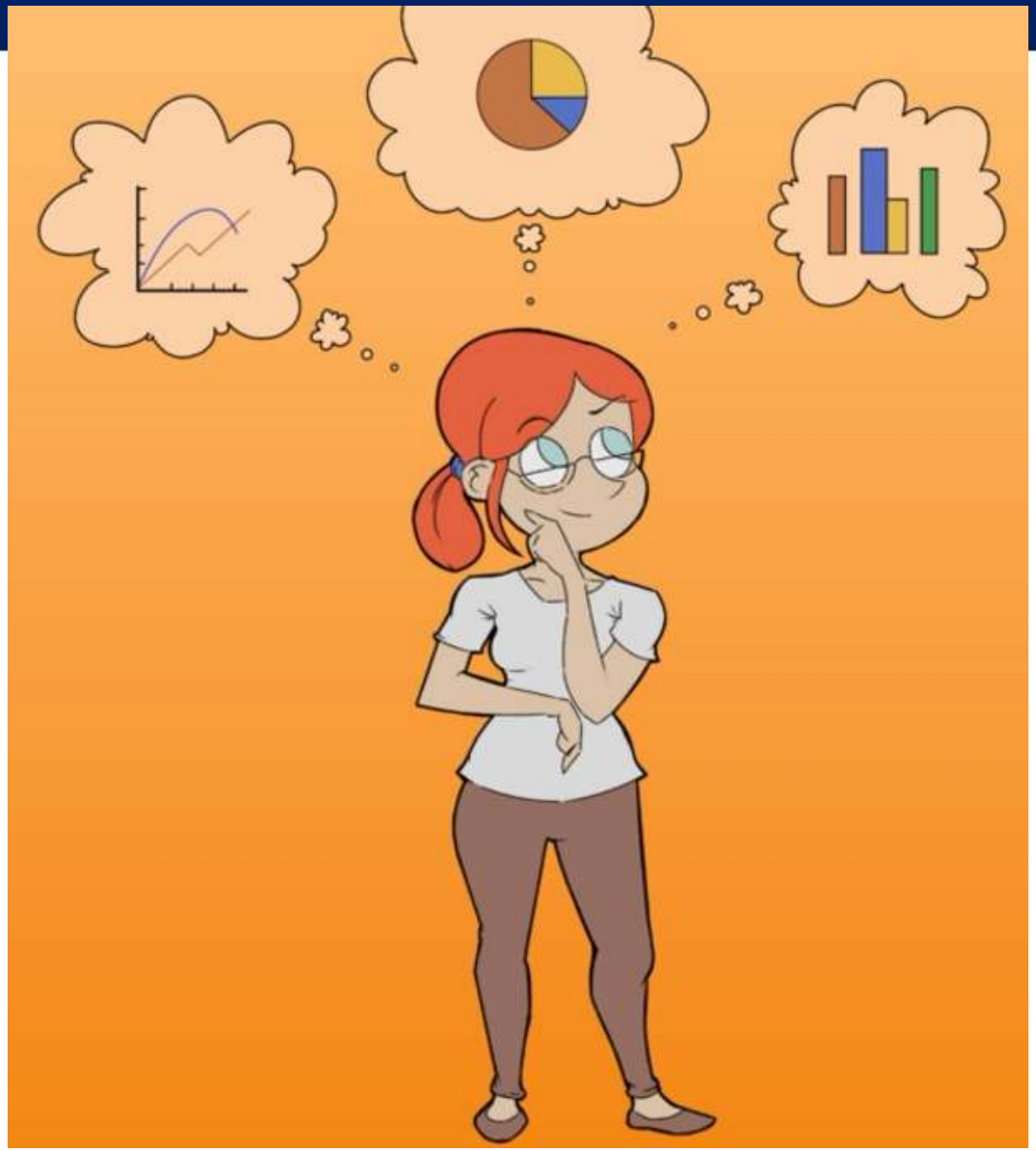
What is S3?

S3 stand for Simple Storage Service.

It is a software offering by Amazon Web Services (AWS) that essentially acts like **your own personal storage space in the cloud**.

You can think of S3 the same way you would an external hard drive.



What is the application for S3?

Like hard disk storage, there are numerous applications for S3.

S3 offers file versioning (i.e. allows for multiple versions of a file to exists in a storage location, also referred to as buckets)

You can scale your storage, in other words it's not a fixed size like an external hard drive would be.

It's cost effective. Depending on how frequent you will need to access your files, you can choose various types of storage types and further reduce the amount you pay.

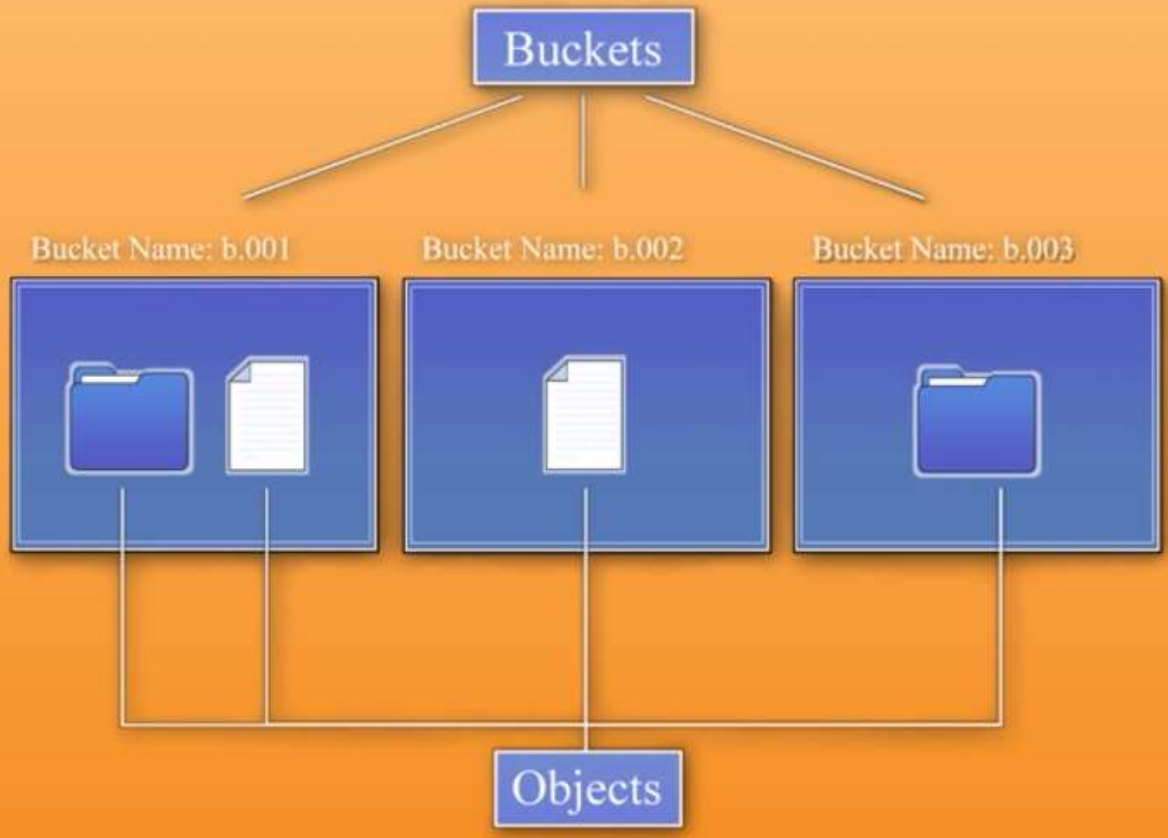
There are analytics use cases that will call for S3 usage and you can even use it to hosts simple HTML sites!

What is an S3 bucket?

An S3 bucket is a storage unit.

It can contain **folders and files, also referred to as objects**

S3 bucket names are unique, so you can't share the same bucket name with anyone, even if they have a different AWS account.



Create a Bucket

```
import boto
def create_bucket(bucket_name):
    """
    Create a bucket.  If the bucket already exists and you have
    access to it, no error will be returned by AWS.
    Note that bucket names are global to S3
    so you need to choose a unique name.
    """
    s3 = boto.connect_s3()
    # First let's see if we already have a bucket of this name.
    # The lookup method will return a Bucket object if the
    # bucket exists and we have access to it or None.
    bucket = s3.lookup(bucket_name)
    if bucket:
        print 'Bucket (%s) already exists' % bucket_name
```

```
 else:
       # Let's try to create the bucket.  This will fail if
       # the bucket has already been created by someone else.
       try:
          bucket = s3.create_bucket(bucket_name)
       except s3.provider.storage_create_error, e:
          print 'Bucket (%s) is owned by another user' %
bucket_name
    return bucket
```

Create a Bucket in a Specific Location

```
import boto
from boto.s3.connection import Location
def create_bucket(bucket_name, location=Location.DEFAULT):
    """
    Create a bucket.  If the bucket already exists and you have
    access to it, no error will be returned by AWS.
    Note that bucket names are global to a S3 region or location
    so you need to choose a unique name.
    bucket_name - The name of the bucket to be created.
```

```
    location - The location in which the bucket should be
               created.  The Location class is a simple
               enum-like static class that has the following attributes:
               DEFAULT|EU|USWest|APNortheast|APSoutheast
    """
    s3 = boto.connect_s3()
    # First let's see if we already have a bucket of this name.
    # The lookup method will return a Bucket object if the
    # bucket exists and we have access to it or None.
    bucket = s3.lookup(bucket_name)
    if bucket:
        print 'Bucket (%s) already exists' % bucket_name
    else:
        # Let's try to create the bucket.  This will fail if
        # the bucket has already been created by someone else.
        try:
            bucket = s3.create_bucket(bucket_name,
location=location)
        except s3.provider.storage_create_error, e:
            print 'Bucket (%s) is owned by another user' %
bucket_name
    return bucket
```

Store Private Data

```
import boto
def store_private_data(bucket_name, key_name, path_to_file):
    """
    Write the contents of a local file to S3 and also store custom
    metadata with the object.
    bucket_name   The name of the S3 Bucket.
    key_name      The name of the object containing the data in
S3.
    path_to_file  Fully qualified path to local file.
    """
    s3 = boto.connect_s3()
    bucket = s3.lookup(bucket_name)
```

```
 # Get a new, blank Key object from the bucket.  This Key object
only
    # exists locally until we actually store data in it.
    key = bucket.new_key(key_name)
    # First let's demonstrate how to write string data to the Key
    data = 'This is the content of my key'
    key.set_contents_from_string(data)
    # Now fetch the data from S3 and compare
    stored_key = bucket.lookup(key_name)
    stored_data = stored_key.get_contents_as_string()
    assert stored_data == data
    # Now, overwrite the data with the contents of the file
    key.set_contents_from_filename(path_to_file)
    return key
```

Store Metadata with an Object

```
import boto
def store_metadata_with_key(bucket_name,
                            key_name,
                            path_to_file,
                            metadata):
    """
    Write the contents of a local file to S3 and also store custom
    metadata with the object.
    bucket_name   The name of the S3 Bucket.
    key_name      The name of the object containing the data in
S3.
    path_to_file  Fully qualified path to local file.
    metadata      A Python dict object containing key/value
                  data you would like associated with the object.
                  For example: {'key1':'value1', 'key2':'value2'}
    """
    s3 = boto.connect_s3()
    bucket = s3.lookup(bucket_name)
```

```
 # Get a new, blank Key object from the bucket.  This Key object
only
    # exists locally until we actually store data in it.
    key = bucket.new_key(key_name)
    # Add the metadata to the Key object
    key.metadata.update(metadata)

    # Now, write the data and metadata to S3
    key.set_contents_from_filename(path_to_file)
    return key
```

```
def print_key_metadata(bucket_name, key_name):
    """
    Print the metadata associated with an S3 Key object.

    bucket_name   The name of the S3 Bucket.
    key_name      The name of the object containing the data in
S3.
    """
    s3 = boto.connect_s3()
    bucket = s3.lookup(bucket_name)
```

```
key = bucket.lookup(key_name)
  print key.metadata
```

Computing Total Storage Used by a Bucket

```
import boto
def bucket_du(bucket_name):
    """
    Compute the total bytes used by a bucket.
    NOTE: This iterates over every key in the bucket.  If you have
millions of
          keys this could take a while.
    """
    s3 = boto.connect_s3()
    total_bytes = 0
    bucket = s3.lookup(bucket_name)
    if bucket:
       for key in bucket:
          total_bytes += key.size
```

```
else:
        print 'Warning: bucket %s was not found!' % bucket_name
    return total_bytes
```

```
Copy an Existing Object to Another Bucket
import boto
def copy_object(src_bucket_name,
                src_key_name,
                dst_bucket_name,
                dst_key_name,
                preserve_metadata=True):
    """
    Copy an existing object to another location.
    src_bucket_name   Bucket containing the existing object.
    src_key_name      Name of the existing object.
    dst_bucket_name   Bucket to which the object is being
copied.
    dst_key_name      The name of the new object.
    preserve_metadata If True, all metadata on the original
object
                      will be preserved on the new object.  If False
                      the new object will have the default metadata.
    """
```

```
 s3 = boto.connect_s3()
    bucket = s3.lookup(src_bucket_name)
    # Lookup the existing object in S3
    key = bucket.lookup(src_key_name)
    # Copy the key back on to itself, with new metadata
    return key.copy(dst_bucket_name, dst_key_name,
preserve_acl=preserve_acl)
```

Modify the Metadata of an Existing Object

```
import boto
def modify_metadata(bucket_name,
                    key_name,
                    metadata):
    """
    Update the metadata with an existing object.
    bucket_name   The name of the S3 Bucket.
    key_name      The name of the object containing the data in
S3.
    metadata      A Python dict object containing the new
metadata.
                  For example: {'key1':'value1', 'key2':'value2'}
    """
    s3 = boto.connect_s3()
    bucket = s3.lookup(bucket_name)
```

```
 # Lookup the existing object in S3
    key = bucket.lookup(key_name)
    # Copy the key back on to itself, with new metadata
    key.copy(bucket.name, key.name, metadata,
preserve_acl=True)
    return key
```

Enable Logging on an Existing Bucket

```
import boto
def enable_logging(bucket_name,
                   log_bucket_name,
                   log_prefix=None):
    """
    Enable logging on a bucket.
    bucket_name     Bucket to be logged.
    log_bucket_name Bucket where logs will be written.
    log_prefix      A string which will be prepended to all log file
names.
    """
    s3 = boto.connect_s3()
    bucket = s3.lookup(bucket_name)
    log_bucket = s3.lookup(log_bucket_name)
    # First configure log bucket as a log target.
    # This sets permissions on the bucket to allow S3 to write
logs.
    log_bucket.set_as_logging_target()
```

```
 # Now enable logging on the bucket and tell S3
    # where to deliver the logs.
    bucket.enable_logging(log_bucket, target_prefix=log_prefix)
def disable_logging(bucket_name):
    """
    Disable logging on a bucket.
    bucket_name     Bucket that will no longer be logged.
"""
```

```
s3 = boto.connect_s3()
  bucket = s3.lookup(bucket_name)
  bucket.disable_logging()
```

Reduce the Cost of Storing Noncritical Data

```
import boto
import os
def upload_file_rrs(local_file,
                    bucket_name,
                    key_name=None):
    """
    Upload a local file to S3 and store is using Reduced
Redundancy Storage.
    local_file  Path to local file.
    bucket_name Bucket to which the file will be uploaded.
    key_name    Name of the new object in S3.  If not provided,
the basename
                of the local file will be used.
    """
    s3 = boto.connect_s3()
    bucket = s3.lookup(bucket_name)
    # Expand common shell vars in filename.
    local_file = os.path.expanduser(local_file)
    local_file = os.path.expandvars(local_file)
```

```
# If key_name was not provided, use basename of file.
if not key_name:
      key_name = os.path.basename(local_file)
   # Create a new local key object.
   key = bucket.new_key(key_name)
   # Now upload file to S3
   key.set_contents_from_filename(local_file,
reduced_redundancy=True)
```

```
def copy_object_to_rrs(bucket_name,
                       key_name):
    """
    Will change an existing standard storage class object to a
    Reduced Redundancy storage class object.

    bucket_name Bucket in which the existing key is located.
    key_name    Name of the existing, standard storage key.
    """
    s3 = boto.connect_s3()
    bucket = s3.lookup(bucket_name)
    key = bucket.lookup(key_name)

    return key.copy(bucket_name, key_name,
reduced_redundancy=True,
                    preserve_acl=True)
```

Hosting Static Websites on S3

```
import boto
import os
import time
def upload_website(bucket_name,
                   website_dir,
                   index_file,
                   error_file=None):
    """
    Upload a static website contained in a local directory to
    a bucket in S3.
    bucket_name The name of the bucket to upload website to.
    website_dir Fully-qualified path to directory containing
                website.
    index_file  The name of the index file (e.g. index.html)
    error_file  The name of the error file.  If not provided
                the default S3 error page will be used.
    """
```

```
 s3 = boto.connect_s3()
    bucket = s3.lookup(bucket_name)
    # Make sure bucket is publicly readable
    bucket.set_canned_acl('public-read')
    for root, dirs, files in os.walk(website_dir):
        for file in files:
            full_path = os.path.join(root, file)
            rel_path = os.path.relpath(full_path, website_dir)
            print 'Uploading %s as %s' % (full_path, rel_path)
            key = bucket.new_key(rel_path)
            key.content_type = 'text/html'
            key.set_contents_from_filename(full_path,
policy='public-read')
    # Now configure the website
    bucket.configure_website(index_file, error_file)
    # A short delay, just to let things become consistent.
    time.sleep(5)
    print 'You can access your website at:'
    print bucket.get_website_endpoint()
```