Lesson Description - Implementing AWS Interaction

The last unit that we need to implement before we can combine all of our modules into our final tool is the storage strategy for AWS S3.

Documentation For This Video

- The `boto3` package
- The `pytest-mock` package
- The `Mock` class
  </ul>

Installing boto3

To interface with AWS (S3 specifically), we're going to use the wonderful `boto3` package. We can install this to our virtualenv using `pipenv`:

```
(pgbackup-E7nj_BsO) $ pipenv install boto3
```

Configuring AWS Client

The `boto3` package works off of the same configuration file that you can use with the official `aws` CLI. To get our configuration right, let's leave our virtualenv and install the `awscli` package for our user. From there, we'll use its `configure` command to set up our config file:

```
(pgbackup-E7nj_BsO) $ exit
$ mkdir ~/.aws
$ pip3.6 install --user awscli
$ aws configure
$ exec $SHELL
```

The `exec $SHELL` portion reload the shell to ensure that the configuration changes are picked up. Before moving on, make sure to reactivate our development virtualenv:

```
$ pipenv shell
```

Writing S3 test

Following the approach that we've been using, let's write tests for our S3 interaction. To limit the explicit dependencies that we have, we're going to have the following parameters to our `storage.s3` function:

- A client object that has an `upload_fileobj` method.
- A `boto3` client meets this requirement, but in testing, we can pass in a "mock" object that implements this method.
- A file-like object (responds to `read`).
- An S3 bucket name as a string.
- The name of the file to create in S3.

We need an `infile` for all of our tests, so let's extract a fixture for that also.

tests/test_storage.py (partial)

```
import tempfile
import pytest

from pgbackup import storage

@pytest.fixture
def infile():
    infile = tempfile.TemporaryFile('r+b')
    infile.write(b"Testing")
    infile.seek(0)
    return infile

# Local storage tests...

def test_storing_file_on_s3(mocker, infile):
    """
    Writes content from one readable to S3
    """
    client = mocker.Mock()

    storage.s3(client,
            infile,
            "bucket",
            "file-name")

    client.upload_fileobj.assert_called_with(
            infile,
               "bucket",
               "file-name")
```

Implementing S3 Strategy

Our test gives a little too much information about how we’re going to implement our `storage.s3` function, but it should be pretty simple for us to implement now:

src/pgbackup/storage.py (partial)

```
def s3(client, infile, bucket, name):
    client.upload_fileobj(infile, bucket, name)
```

Manually Testing S3 Integration

Like we did with our PostgreSQL interaction, let’s manually test uploading a file to S3 using our `storage.s3` function. First, we’ll create an `example.txt` file, and then we’ll load into a Python REPL with our code loaded:

```
(pgbackup-E7nj_BsO) $ echo "UPLOADED" > example.txt
(pgbackup-E7nj_BsO) $ PYTHONPATH=./src python
>>> import boto3
>>> from pgbackup import storage
>>> client = boto3.client('s3')
>>> infile = open('example.txt', 'rb')
>>> storage.s3(client, infile, 'pyscripting-db-backups',
infile.name)
```

When we check our S3 console, we should see the file there.

Lastly, remove the `example.txt` file and then commit these changes:

```
(pgbackup-E7nj_BsO) $ rm example.txt
(pgbackup-E7nj_BsO) $ git add .
(pgbackup-E7nj_BsO) $ git commit -m 'Implement S3 interactions'
```