Lesson Description - Wiring the Units Together

We've successfully written the following:

- CLI parsing
- Postgres Interaction
- Local storage driver
- AWS S3 storage driver

Now we need to wire up an executable that can integrate these parts. Up to this point we've used TDD to write our code. These have been "unit tests" because we're only ever testing a single unit of code. If we wanted to write tests that ensure our application worked from start to finish, we could do that and they would be "integration" tests. Given that our code does a lot with the network, and we would have to do a lot of mocking to write integration tests, we're not going to write them. Sometimes the tests aren't worth the work that goes into them.

Documentation For This Video

- The `boto3` package
- The `setuptools` script creation
- The `time.strftime` function

Add "console_script" to project

We can make our project create a console script for us when a user runs `pip install`. This is similar to the way that we made executables before, except we don't need to manually do the work. To do this, we need to add an entry point in our `setup.py`:

setup.py (partial)

```
    install_requires=['boto3'],
    entry_points={
        'console_scripts': [
            'pgbackup=pgbackup.cli:main',
        ],
    }
```

Notices that we're referencing our `cli` module with a `:` and a `main`. That `main` is the function that we need to create now.

Wiring The Units Together

Our `main` function is going to go in the `cli` module, and it needs to do the following:

1. Import the `boto3` package.
2. Import our `pgdump` and `storage` modules.
3. Create a parser and parse the arguments.
4. Fetch the database dump.
5. Depending on the driver type do one of the following:
    - create a `boto3` S3 client and use `storage.s3` or
    - open a local file and use `storage.local`

src/pgbackup/cli.py

```
def main():
    import boto3
    from pgbackup import pgdump, storage

    args = create_parser().parse_args()
    dump = pgdump.dump(args.url)
    if args.driver == 's3':
        client = boto3.client('s3')
        # TODO: create a better name based on the database name and
the date
        storage.s3(client, dump.stdout, args.destination,
'example.sql')
    else:
        outfile = open(args.destination, 'wb')
        storage.local(dump.stdout, outfile)
```

Let's test it out:

```
$ pipenv shell
(pgbackup-E7nj_BsO) $ pip install -e .
(pgbackup-E7nj_BsO) $ pgbackup --driver local ./local-dump.sql
postgres://demo:password@54.245.63.9:80/sample
(pgbackup-E7nj_BsO) $ pgbackup --driver s3 pyscripting-db-backups
postgres://demo:password@54.245.63.9:80/sample
```

Reviewing the Experience

It worked! That doesn't mean there aren't things to improve though. Here are some things we should fix:

- Generate a good file name for S3
- Create some output while the writing is happening
- Create a shorthand switch for `--driver` (`-d`)

Generating a Dump File Name

For generating our filename, let's put all database URL interactions in the `pgdump` module with a function name of `dump_file_name`. This is a pure function that takes an input and produces an output, so it's a prime function for us to unit test. Let's write our tests now:

tests/test_pgdump.py (partial)

```
def test_dump_file_name_without_timestamp():
    """
    pgdump.db_file_name returns the name of the database
    """
    assert pgdump.dump_file_name(url) == "db_one.sql"

def test_dump_file_name_with_timestamp():
    """
    pgdump.dump_file_name returns the name of the database
    """
    timestamp = "2017-12-03T13:14:10"
    assert pgdump.dump_file_name(url, timestamp) ==
"db_one-2017-12-03T13:14:10.sql"
```

We want the file name returned to be based on the database name, and it should also accept an optional timestamp. Let's work on the implementation now:

src/pgbackup/pgdump.py (partial)

```
def dump_file_name(url, timestamp=None):
    db_name = url.split("/")[-1]
    db_name = db_name.split("?")[0]
    if timestamp:
        return f"{db_name}-{timestamp}.sql"
    else:
        return f"{db_name}.sql"
```

Improving the CLI and Main Function

We want to add a shorthand `-d` flag to the `driver` argument, let’s add that to the `create_parser` function:

src/pgbackup/cli.py (partial)

```
def create_parser():
    parser = ArgumentParser()
    parser.add_argument("url", help="URL of the PostgreSQL database
to backup")
    parser.add_argument("--driver", "-d",
            help="how & where to store the backup",
            nargs=2,
            action=DriverAction,
            metavar=("DRIVER", "DESTINATION"),
            required=True)
    return parser
```

Lastly, let’s print a timestamp with `time.strftime`, generate a database file name, and print what we’re doing as we upload/write files.

src/pgbackup/cli.py (partial)

```
def main():
    import time
    import boto3
    from pgbackup import pgdump, storage

    args = create_parser().parse_args()
    dump = pgdump.dump(args.url)

    if args.driver == 's3':
        client = boto3.client('s3')
        timestamp = time.strftime("%Y-%m-%dT%H:%M", time.localtime())
        file_name = pgdump.dump_file_name(args.url, timestamp)
        print(f"Backing database up to {args.destination} in S3 as
{file_name}")
        storage.s3(client,
                dump.stdout,
                args.destination,
                file_name)
    else:
        outfile = open(args.destination, 'wb')
        print(f"Backing database up locally to {outfile.name}")
        storage.local(dump.stdout, outfile)
```

Feel free to test the CLI’s modifications and commit these changes.