A CLOUD GURU

Lesson Description - Implementing CLI Guided By Tests

We now have some breaking tests to help guide us to the implementation of our `client` module. Let's follow the errors that we see to get our tests passing.

Documentation For This Video

- The `argparse` package
- The `argparse.Action` class
- The `pytest` package
- The `pytest.fixture` function
- Python decorators

Moving Through Failing Tests

Our current test failure is from there not being a `cli.py` file within the `src/pgbackup` directory. Let's do just enough to move onto the next error:

(partial `make` output)

```
(pgbackup-E7nj_Bs0) $ touch src/pgbackup/cli.py
(pgbackup-E7nj_Bs0) $ make
PYTHONPATH=./src pytest
========================================= test session starts
=========================================
platform linux -- Python 3.6.4, pytest-3.3.2, py-1.5.2, pluggy-0.6.0
rootdir: /home/user/code/pgbackup, inifile:
collected 3 items

tests/test_cli.py
FFF
     [100%]

============================================== FAILURES
==============================================
_____________________________________ test_parser_without_driver
_____________________________________

    def test_parser_without_driver():
        """
        Without a specified driver the parser will exit
        """
```

```
        with pytest.raises(SystemExit):
>           parser = cli.create_parser()
E           AttributeError: module 'pgbackup.cli' has no attribute
'create_parser'

tests/test_cli.py:12: AttributeError
...
```

Now we're getting an `AttributeError` because there is not an attribute/function called `create_parser`. Let's implement a version of that function that creates an `ArgumentParser` that hasn't been customized:

~/code/pgbackup/src/pgbackup/cli.py

```
from argparse import ArgumentParser

def create_parser():
    parser = ArgumentParser()
    return parser
```

Once again, let's run our tests:

(partial `make` output)

```
(pgbackup-E7nj_Bs0) $ make
...
self = ArgumentParser(prog='pytest', usage=None, description=None,
formatter_class=, conflict_handler='error', add_help=True)
status = 2
message = 'pytest: error: unrecognized arguments: postgres://
bob:password@example.com:5432/db_one --driver local /some/path\n'

    def exit(self, status=0, message=None):
        if message:
            self._print_message(message, _sys.stderr)
>       _sys.exit(status)
E       SystemExit: 2

/usr/local/lib/python3.6/argparse.py:2376: SystemExit
---------------------------------------- Captured stderr call
-----------------------------------------
usage: pytest [-h]
pytest: error: unrecognized arguments: postgres://
bob:password@example.com:5432/db_one --driver local /some/path
```

```
============================== 1 failed, 2 passed in 0.14 seconds
==============================
```

Interestingly, two of the tests succeeded. Those two tests were the ones that expected there to be a `SystemExit` error. Our tests sent unexpected output to the parser (since it wasn't configured to accept arguments), and that caused the parser to error. This demonstrates why it's important to write tests that cover a wide variety of use cases. If we hadn't implemented the third test to ensure that we get the expected output on success, then our test suite would be green!

Creating Our First Class

For this course, we haven't created any custom classes because it's not something that we'll do all the time, but in the case of our CLI, we need to. Our idea of having a flag of `--driver` that takes two distinct values isn't something that any existing `argparse.Action` can do. Because of this, we're going to follow along with the documentation and implement our own custom `DriverAction` class. We can put our custom class in our `cli.py` file and use it in our `add_argument` call.

src/pgbackup/cli.py

```
from argparse import Action, ArgumentParser

class DriverAction(Action):
    def __call__(self, parser, namespace, values,
option_string=None):
        driver, destination = values
        namespace.driver = driver.lower()
        namespace.destination = destination

def create_parser():
    parser = ArgumentParser(description="""
    Back up PostgreSQL databases locally or to AWS S3.
    """)
    parser.add_argument("url", help="URL of database to backup")
    parser.add_argument("--driver",
            help="how & where to store backup",
            nargs=2,
            action=DriverAction,
            required=True)
    return parser
```

Adding More Tests

Our CLI is coming along, but we probably want to raise an error if the end-user tries to use a driver that we don't understand. Let's add a few more tests that do the following:

1. Ensure that you can't use a driver that is unknown, like `azure`.
2. Ensure that the drivers for `s3` and `local` don't cause errors.

test/test_cli.py (partial)

```
def test_parser_with_unknown_drivers():
    """
    The parser will exit if the driver name is unknown.
    """

    parser = cli.create_parser()

    with pytest.raises(SystemExit):
        parser.parse_args([url, "--driver", "azure", "destination"])
```

```
def test_parser_with_known_drivers():
    """
    The parser will not exit if the driver name is known.
    """

    parser = cli.create_parser()

    for driver in ['local', 's3']:
        assert parser.parse_args([url, "--driver", driver,
"destination"])
```

Adding Driver Type Validation

Since we already have a custom `DriverAction`, we can feel free to customize this to make our CLI a little more intelligent. The only drivers that we are going to support (for now) are `s3` and `local`, so let's add some logic to our action to ensure that the driver given is one that we can work with:

```
known_drivers = ['local', 's3']

class DriverAction(Action):
    def __call__(self, parser, namespace, values,
option_string=None):
```

```
        driver, destination = values
        if driver.lower() not in known_drivers:
            parser.error("Unknown driver. Available drivers are
'local' & 's3'")
        namespace.driver = driver.lower()
        namespace.destination = destination
```

Removing Test Duplication Using `pytest.fixture`

Before we consider this unit of our application complete, we should consider cleaning up some of the duplication in our tests. We create the parser using `create_parser` in every test but using `pytest.fixture` we can extract that into a separate function and inject the `parser` value into each test that needs it.

Here's what our `parser` function will look like:

tests/test_cli.py (partial)

```
import pytest

@pytest.fixture
def parser():
    return cli.create_parser()
```

We haven't run into this yet, but the `@pytest.fixture` on top of our function definition is what's known as a "decorator". A "decorator" is a function that returns a modified version of the function. We've seen that if we don't use parentheses that our functions aren't called, and because of that we're able to pass functions into other functions as arguments. This particular decorator will register our function in the list of fixtures that can be injected into a pytest test. To inject our fixture, we will add an argument to our test function definition that has the same name as our fixture name, in this case, `parser`. Here's the final test file:

tests/test_cli.py

```
import pytest

from pgbackup import cli

url = "postgres://bob@example.com:5432/db_one"

@pytest.fixture
def parser():
    return cli.create_parser()
```

```
def test_parser_without_driver(parser):
    """
    Without a specified driver the parser will exit
    """
    with pytest.raises(SystemExit):
        parser.parse_args([url])

def test_parser_with_driver(parser):
    """
    The parser will exit if it receives a driver without a
destination
    """
    with pytest.raises(SystemExit):
        parser.parse_args([url, "--driver", "local"])

def test_parser_with_unknown_driver(parser):
    """
    The parser will exit if the driver name is unknown.
    """
    with pytest.raises(SystemExit):
        parser.parse_args([url, "--driver", "azure", "destination"])

def test_parser_with_known_drivers(parser):
    """
    The parser will not exit if the driver name is known.
    """
    for driver in ['local', 's3']:
        assert parser.parse_args([url, "--driver", driver,
"destination"])

def test_parser_with_driver_and_destination(parser):
    """
    The parser will not exit if it receives a driver with a
destination
    """
    args = parser.parse_args([url, "--driver", "local", "/some/
path"])

    assert args.url == url
    assert args.driver == "local"
    assert args.destination == "/some/path"
```

Now, all of our tests should pass, and we're in a good spot to make a commit.