A CLOUD GURU

Lesson Description - Introduction to Mocking in Tests

The simplest way that we can get all of the information that we need out of a PostgreSQL is to use the `pg_dump` utility that Postgres itself provides. Since that code exists outside of our codebase, it's not our job to ensure that the `pg_dump` tool itself works, but we do need to write tests that can run without an actual Postgres server running. For this, we will need to "stub" our interaction with `pg_dump`.

Documentation For This Video

- The `pytest-mock` package
- The `subprocess` package
- The `subprocess.Popen` class

Install pytest-mock

Before we can learn how to use mocking in our tests, we need to install the `pytest-mock` package. This will pull in a few packages for us, and mainly provide us with a `mocker` fixture that we can inject into our tests:

```
(pgbackup-E7nj_Bs0) $ pipenv install --dev pytest-mock
```

Writing Tests With Mocking

We're going to put all of the Postgres related logic into its own module called `pgdump`, and we're going to begin by writing our tests. We want this module to do the following:

1. Make a call out to `pg_dump` using `subprocess.Popen`.
2. Returns the subprocess that STDOUT can be read from.

We know how to use the `subprocess` module, but we haven't used `subprocess.Popen` yet. Behind the scenes, the functions that we already know use `Popen`, and wait for it to finish. We're going to use this instead of run, because we want to continue running code instead of waiting, right until we need to write the contents of `proc.stdout` to a file or S3.

To ensure that our code runs the proper third-party utilities, we're going to use `mocker.patch` on the `subprocess.Popen` constructor. This will substitute in a different

implementation that holds onto information like the number of times the function is called and with what arguments. Let's see what this looks like in practice:

tests/test_pgdump.py

```
import pytest
import subprocess

from pgbackup import pgdump

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

def test_dump_calls_pg_dump(mocker):
    """
    Utilize pg_dump with the database URL
    """
    mocker.patch('subprocess.Popen')
    assert pgdump.dump(url)
    subprocess.Popen.assert_called_with(['pg_dump', url],
stdout=subprocess.PIPE)
```

The arguments that we're passing to `assert_called_with` will need to match what is being passed to `subprocess.Popen` when we exercise `pgdump.dump(url)`.