Table of Contents

Chapter 0: Introduction and Setup

Lab (Ungraded): Create or Reuse Atlas Cluster

Problem:

Reusing an Atlas Account

If you previously created an Atlas account, you can reuse it for this course.

Go ahead and create a new **Project** and call it **M220**. We will be creating an Atlas free tier cluster (M0) in that project.

Creating an Atlas Account

If you did **not** previously create an Atlas account, we have provided instructions for you to create one, and use it to create a cluster for this course.

To sign up for Atlas, visit the Atlas Registration page and enter your information. Please note that Atlas has a hierarchical structure:

- **Organizations**: The name of the organization will be set using your account *company* name. Feel free to use whatever name you'd like.
- **Projects**: Under *Organizations* you can have several *Projects*. A project is logical group shared by different Atlas users. You can invite friends and colleagues to access a given project. For this course our project will be named **M220**.
- **Clusters**: A cluster is a MongoDB deployment. You can define several different types of clusters (Replica Sets, Shard Clusters) deployed in different Cloud Providers (AWS, Azure, GCP) and with a given Cluster Tier. For this course we will be using an **M0 Free Tier** cluster named **mflix**.

Create Atlas Cluster

1. Creating a New Cluster

Follow the Atlas free tier setup instructions, and make sure of the following:

- When selecting a new region, make sure to choose a region with the "Free Tier Available" flag.
- When selecting a Cluster Tier, choose M0 (free tier) so you don't get charged!
- Please choose the name **"mflix"** for this cluster.

After these selections have been made, click "Create Cluster" so we can get started!

2. Allowing Access to Your Cluster

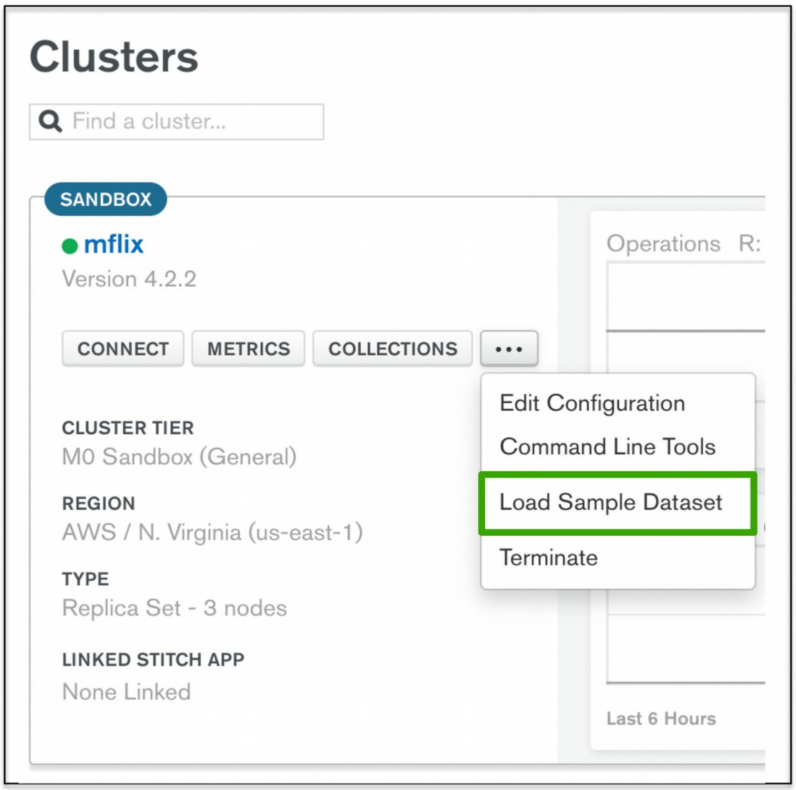
In order to use Atlas with the application, you need to update your IP Whitelist so that your app can talk to the cluster:

1. Navigate to the "**Security**" tab from the **Clusters** page.
2. From the "**IP Whitelist**" tab, there should be an option to "**Add IP Address**" in the top right corner.
3. Choose "**Allow Access from Anywhere**" and then click "**Confirm**".

Note that we do not generally recommend allowing access to your Atlas cluster from anywhere. However, in this class it minimizes network issues and allows us to provide better support.

3. Load Sample Dataset (optional)

Once you've created a cluster, Atlas will notice that there's no data there. It will give you the option to *Load Sample Dataset*. This will load the Atlas sample dataset, containing the MFlix database, into your cluster:



Note: The MFlix database in the Sample Dataset is called "sample_mflix".

If you decide **not** to use the *Load Sample Dataset*, you will need to use **`mongorestore`** in the next step.

Do you have an Atlas cluster ready for M220?

Chapter 1: Driver Setup

OVERVIEW

Overview

We are going to have series of README instructions to be able to setup our MFLIX application successfully. With MFLIX application, you will learn to create and share a database connection, perform the basic Create, Read, Update, and Delete operations through the driver, handle errors, utilize the MongoDB best practices and more.

Mflix is composed of two main components:

- *Frontend*: All the UI functionality is already implemented for you, which includes the built-in React application that you do not need to worry about.
- *Backend*: The project that provides the necessary service to the application. The code flow is already implemented except some functions.

You'll only be implementing the functions which directly call to MongoDB.

Database Layer

We will be using *MongoDB Atlas*, MongoDB's official Database as a Service (DBaaS), so you will not need to manage the database component yourself. However, you will still need to install MongoDB locally to access the command line tools that interact with Atlas, to load data into MongoDB and potentially do some exploration of your database with the shell.

The following README sections are here to get you setup for this course.

Chapter 1: Driver Setup

README: Setting Up MongoDB and Atlas

Table of Contents

MongoDB Installation

It is recommended to connect *Mflix* with *MongoDB Atlas*, so you do not need to have a MongoDB server running on your host machine. The lectures and labs in this course will assume that you are using an *Atlas* cluster instead of a local instance.

That said, you are still required to have the MongoDB server installed, in order to be able to use two server tool dependencies:

- `mongorestore`
  - A utility for importing binary data into MongoDB.
- `mongo`
  - The shell for exploring data in MongoDB.

To download these command line tools, please visit the MongoDB download center and choose the appropriate platform.

MongoDB Atlas Cluster

Mflix uses *MongoDB* to persist all its data.

One of the easiest ways to get up and running with MongoDB is to use *MongoDB Atlas*, a hosted and fully-managed database solution.

If you have taken other MongoDB University courses like M001 or M121, you may already have an account - feel free to reuse that cluster for this course.

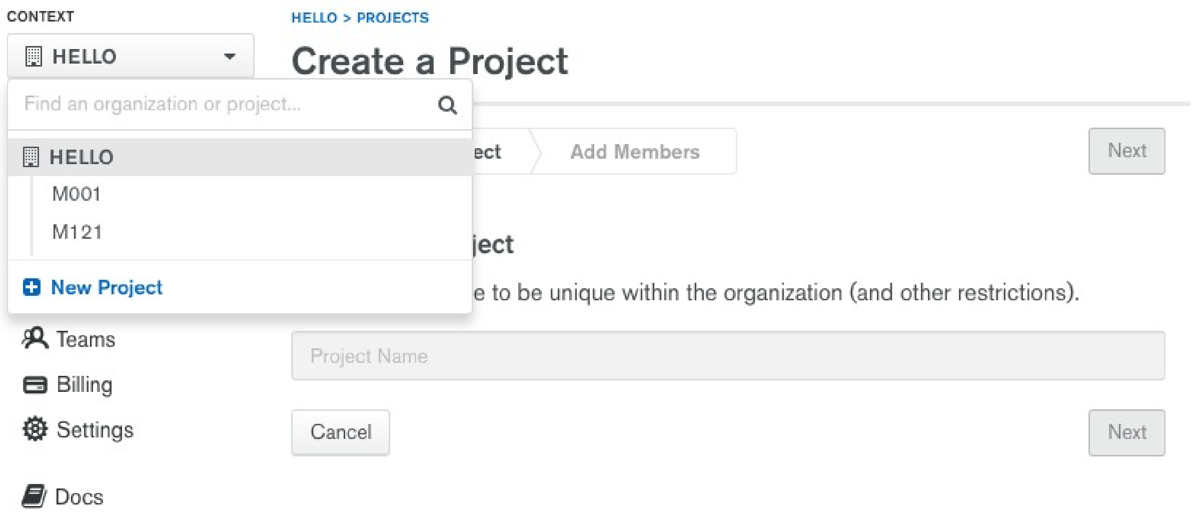
Make sure to use a **free tier cluster** for the application and course.

Note: Be advised that some of the UI aspects of Atlas may have changed since the redaction of this README, therefore some of the screenshots in this file may be different from the actual Atlas UI interface.

Using an existing MongoDB Atlas Account:

If you already have a previous *Atlas* account created, perhaps because you've taken one of our other MongoDB university courses, you can repurpose it for this course.

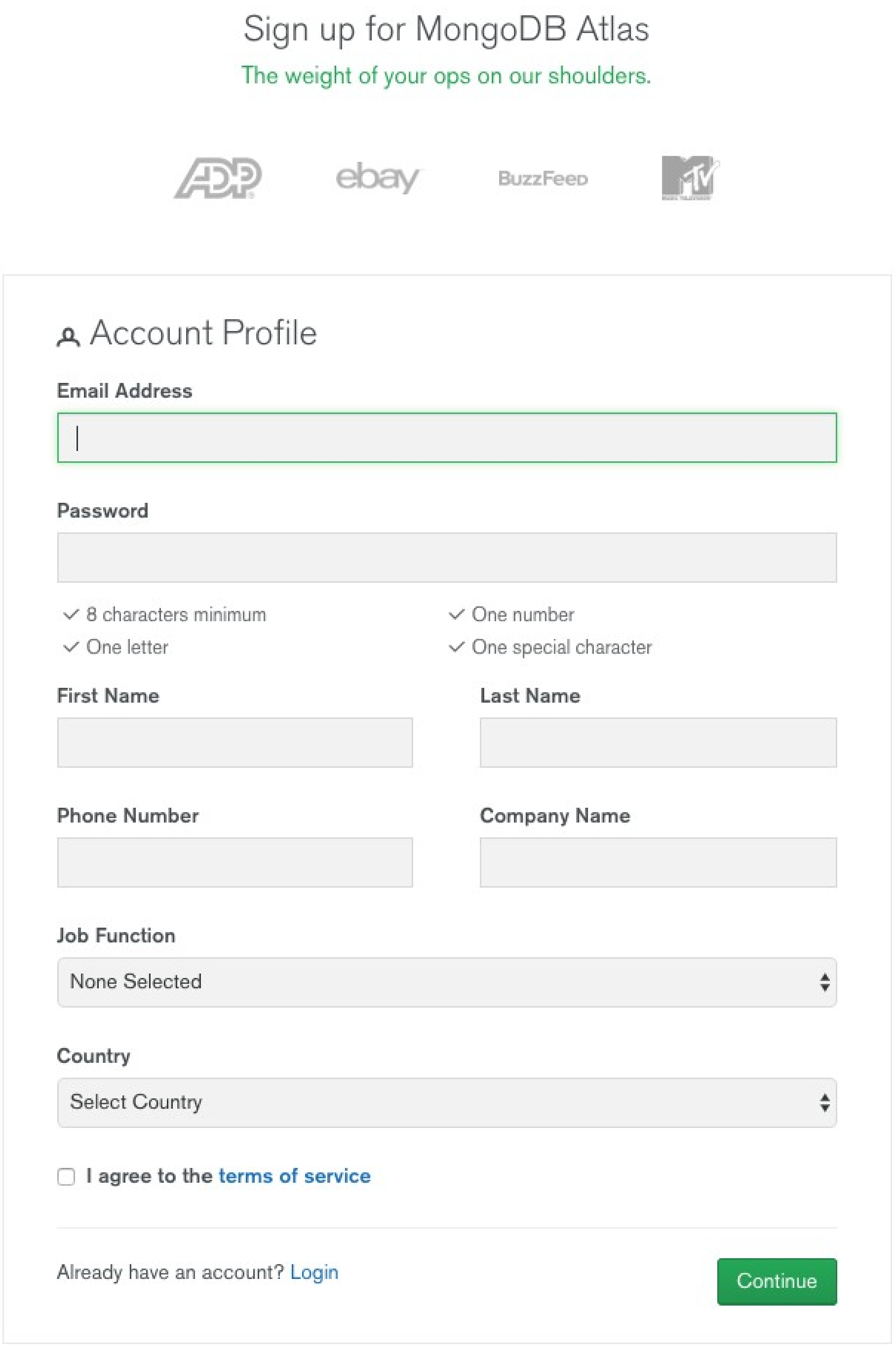
Log into your *Atlas* account and create a new project named **M220** by clicking on the *Context* dropdown menu:



After creating this new project, skip the next section and proceed to the *Creating an M0 free tier cluster Mflix* section.

Creating a new MongoDB Atlas Account:

If you do not have an existing *Atlas* account, go ahead and create an Atlas Account by filling in the required fields:


Sign up for MongoDB Atlas

The weight of your ops on our shoulders.

Account Profile

Email Address

Password

✓ 8 characters minimum
✓ One number
✓ One letter
✓ One special character

First Name

Last Name

Phone Number

Company Name

Job Function

None Selected

Country

Select Country

☐ I agree to the terms of service

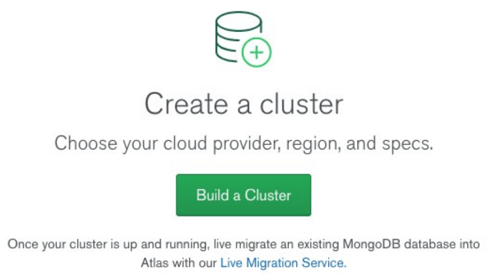
Already have an account? Login

Continue

Creating an M0 free tier cluster mflix:

Note: You will need to do this step even if you are reusing an Atlas account.

1. After creating a new project, you will be prompted to create the first cluster in that project:

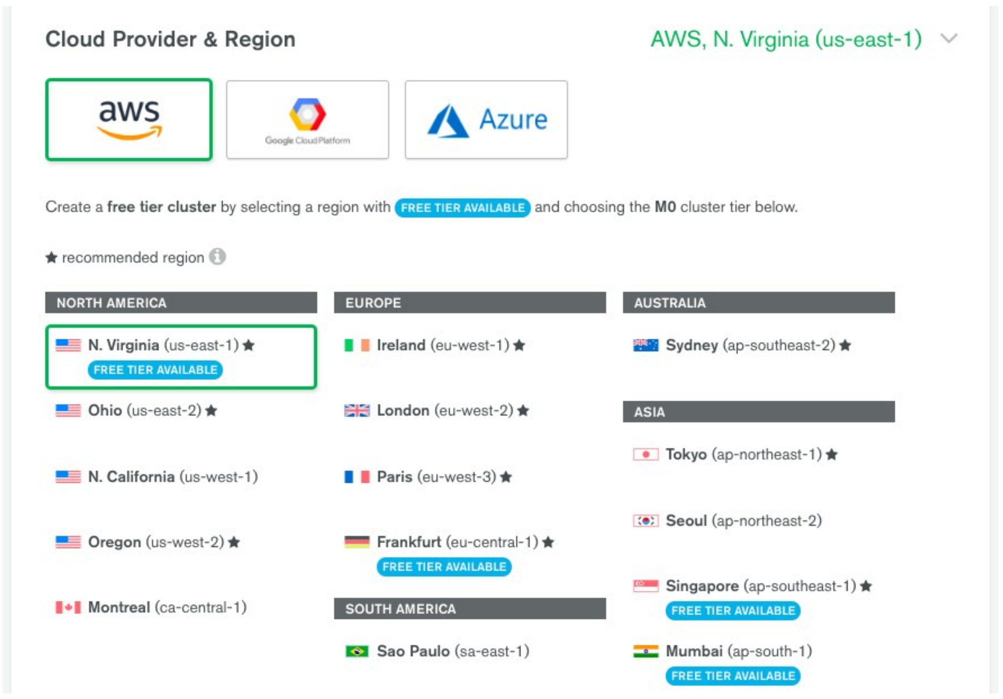


Create a cluster

Choose your cloud provider, region, and specs.

Once your cluster is up and running, live migrate an existing MongoDB database into Atlas with our Live Migration Service.

2. Choose AWS as the cloud provider, in a Region that has the label **Free Tier Available**:



Cloud Provider & Region — AWS, N. Virginia (us-east-1)

Create a **free tier cluster** by selecting a region with FREE TIER AVAILABLE and choosing the **M0** cluster tier below.

★ recommended region

NORTH AMERICA
- **N. Virginia** (us-east-1) ★ FREE TIER AVAILABLE
- **Ohio** (us-east-2) ★
- **N. California** (us-west-1)
- **Oregon** (us-west-2) ★
- **Montreal** (ca-central-1)

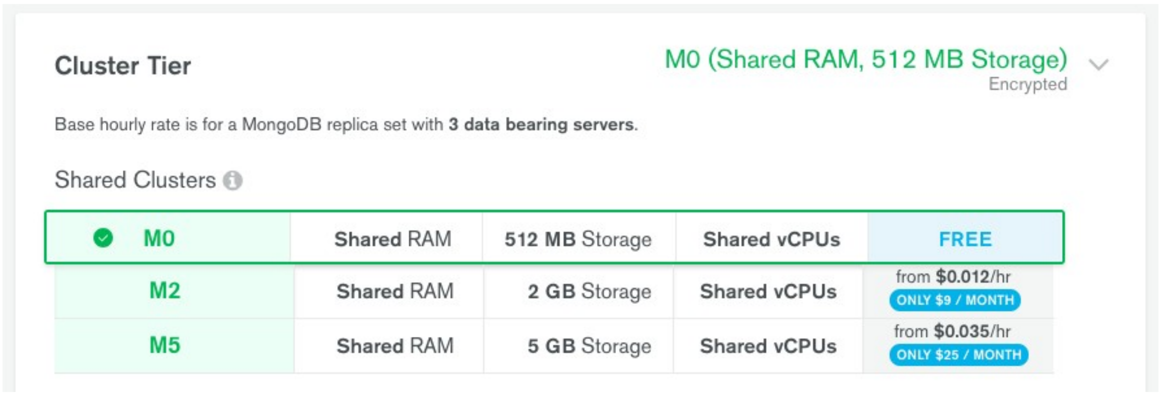
EUROPE
- **Ireland** (eu-west-1) ★
- **London** (eu-west-2) ★
- **Paris** (eu-west-3) ★
- **Frankfurt** (eu-central-1) ★ FREE TIER AVAILABLE

SOUTH AMERICA
- **Sao Paulo** (sa-east-1)

AUSTRALIA
- **Sydney** (ap-southeast-2) ★

ASIA
- **Tokyo** (ap-northeast-1) ★
- **Seoul** (ap-northeast-2)
- **Singapore** (ap-southeast-1) ★ FREE TIER AVAILABLE
- **Mumbai** (ap-south-1) FREE TIER AVAILABLE

3. Select *Cluster Tier* **M0**:



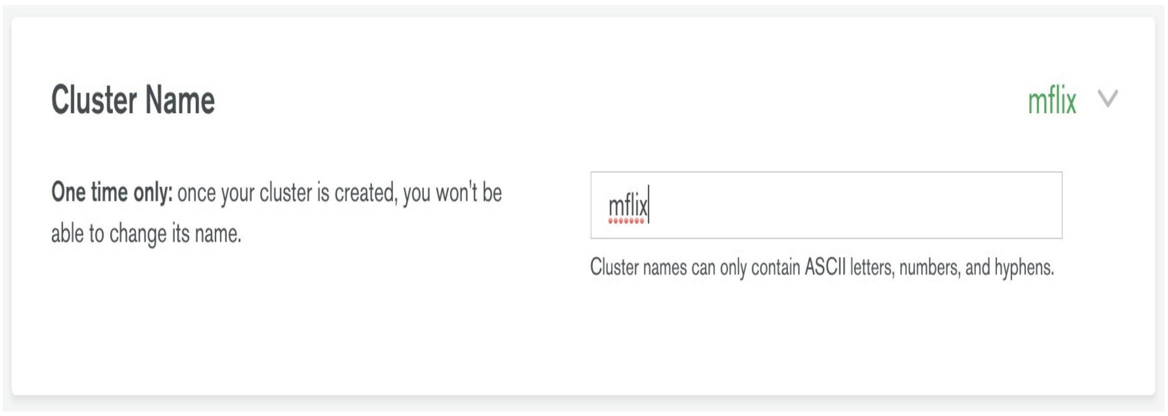
Cluster Tier — M0 (Shared RAM, 512 MB Storage) Encrypted

Base hourly rate is for a MongoDB replica set with **3 data bearing servers**.

Shared Clusters

✓ M0	Shared RAM	512 MB Storage	Shared vCPUs	FREE
M2	Shared RAM	2 GB Storage	Shared vCPUs	from $0.012/hr ONLY $9 / MONTH
M5	Shared RAM	5 GB Storage	Shared vCPUs	from $0.035/hr ONLY $25 / MONTH

4. Set *Cluster Name* to **mflix** by scrolling to the appropriate section and clicking on the default name *Cluster0*, and click *Create Cluster*:



Cluster Name

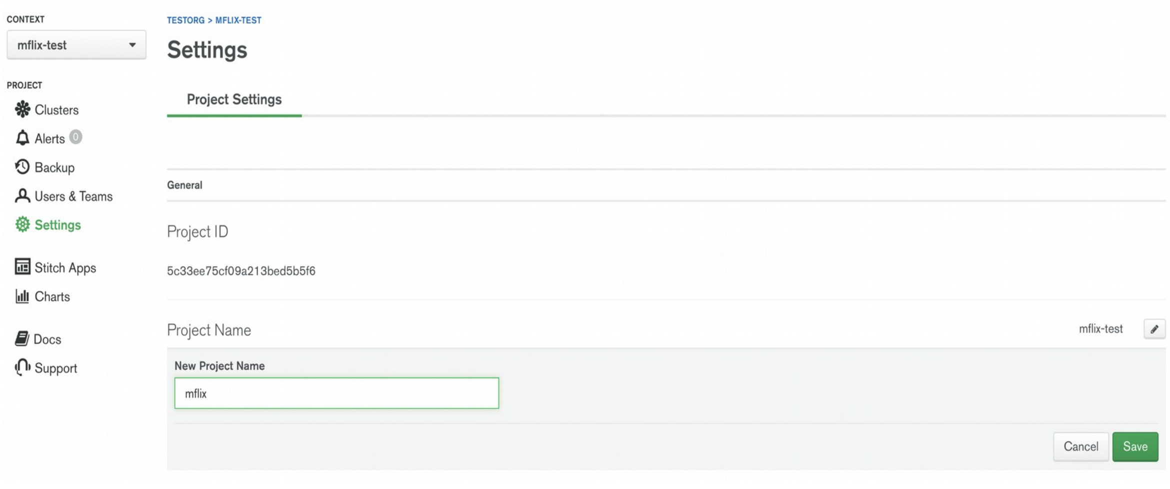
mflix

One time only: once your cluster is created, you won't be able to change its name.

mflix

Cluster names can only contain ASCII letters, numbers, and hyphens.

5. Once you press *Create Cluster*, you will be redirected to the account dashboard. In this dashboard, make sure that the project is named **M220**. If not, go to the *Settings* menu item and change the project name from the default *Project 0* to **M220**:



CONTEXT

mflix-test

PROJECT

Clusters

Alerts 0

Backup

Users & Teams

Settings

Stitch Apps

Charts

Docs

Support

TESTORG > MFLIX-TEST

Settings

Project Settings

General

Project ID

5c33ee75cf09a213bed5b5f6

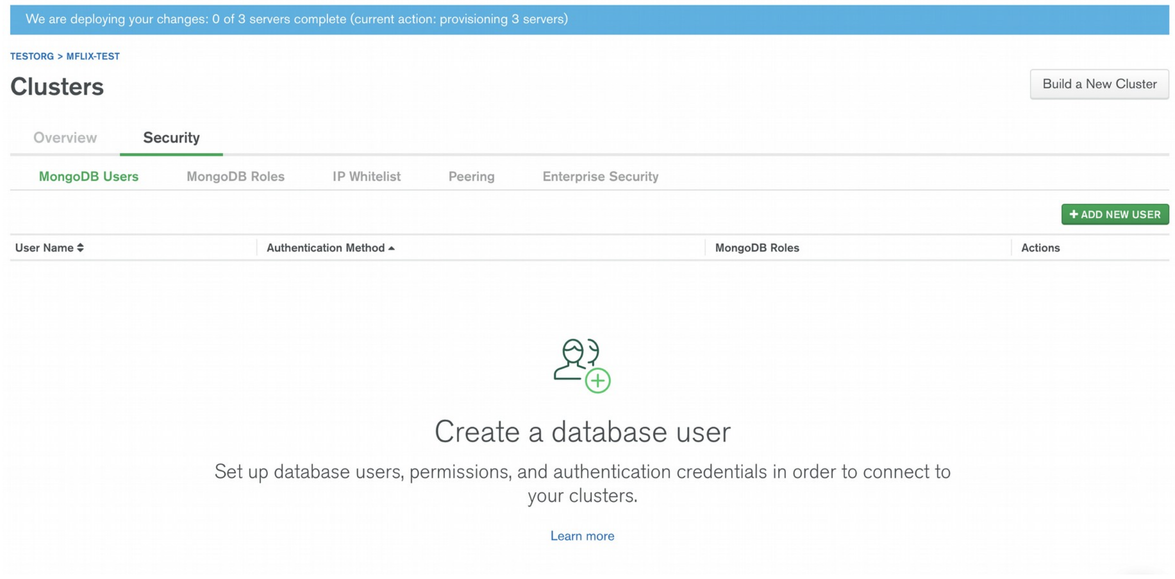
Project Name mflix-test

New Project Name

mflix

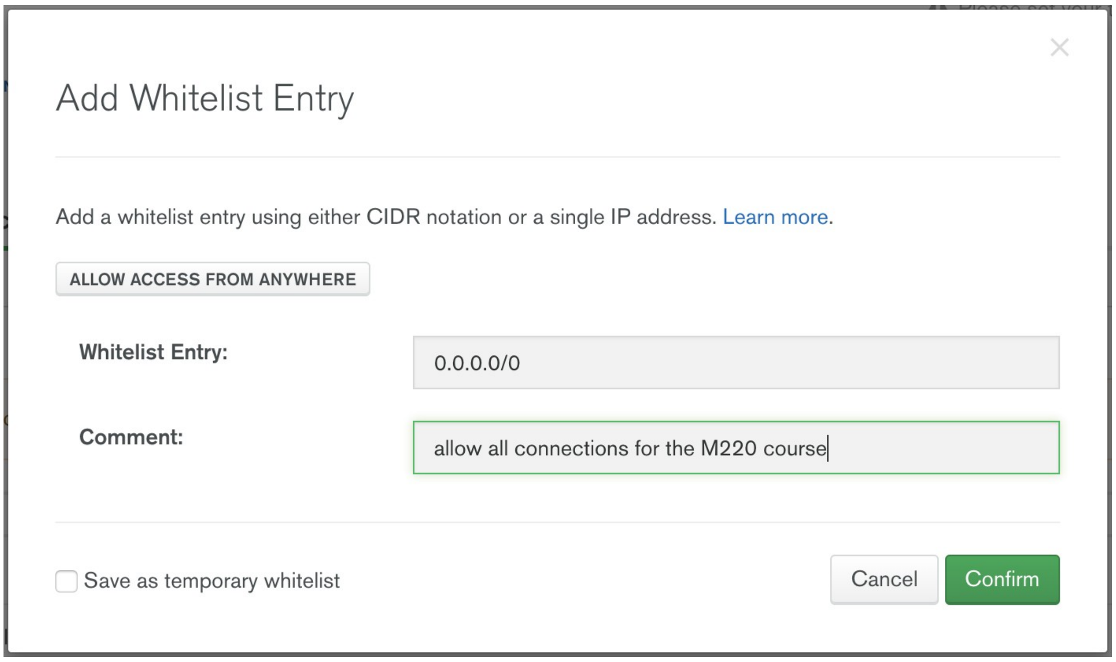
Cancel Save

6. Next, configure the security settings of this cluster, by enabling the *IP Whitelist* and *MongoDB Users*:



Update your IP Whitelist so that your app can talk to the cluster. Click the "Security" tab from the "Clusters" page. Then click "IP Whitelist" followed by "Add IP Address". Finally, click "Allow Access from Anywhere" and click "Confirm".

Note that in a production environment, you would control very tightly the list of IP addresses that can connect to your cluster.


Add Whitelist Entry

Add a whitelist entry using either CIDR notation or a single IP address. Learn more.

ALLOW ACCESS FROM ANYWHERE

Whitelist Entry: 0.0.0.0/0

Comment: allow all connections for the M220 course

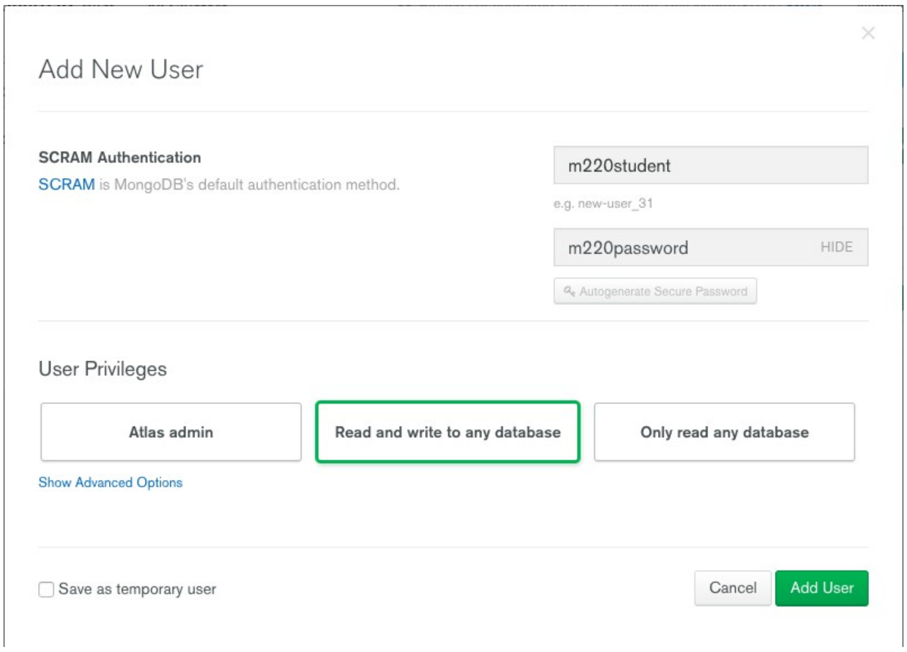
Save as temporary whitelist

Cancel Confirm

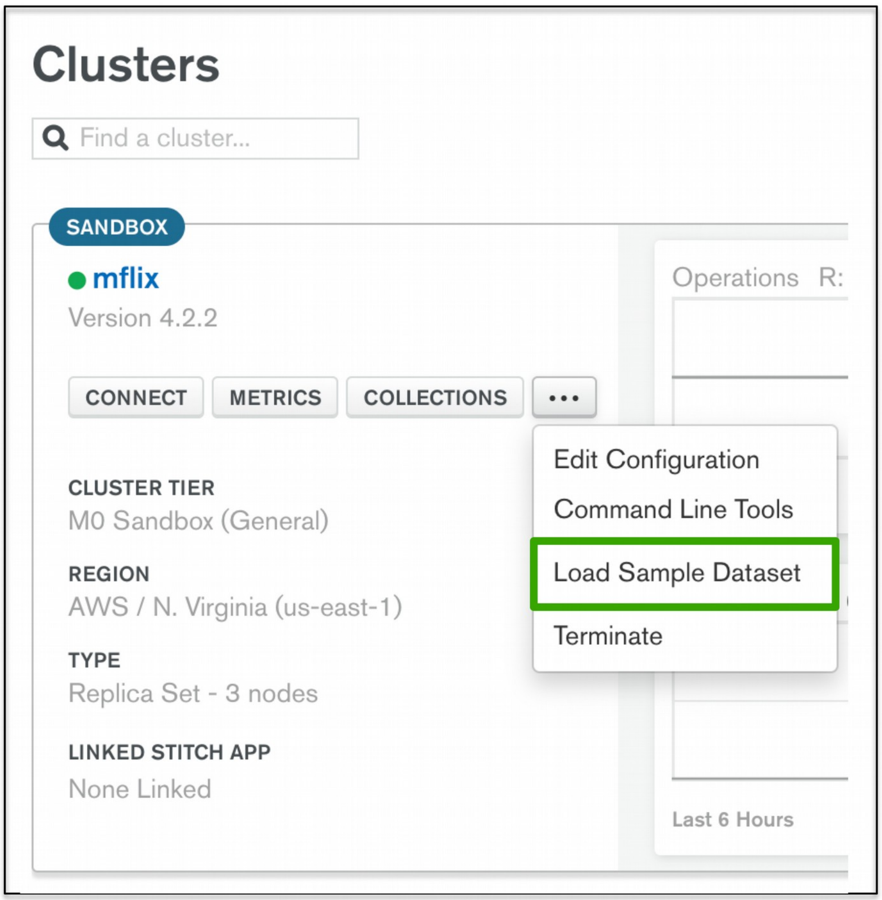
7. Then create the MongoDB database user required for this course:
   - username: **m220student**
   - password: **m220password**

You can create new users through *Security -> MongoDB Users -> Add New User*

Allow this user the privilege to **Read and write to any database**:



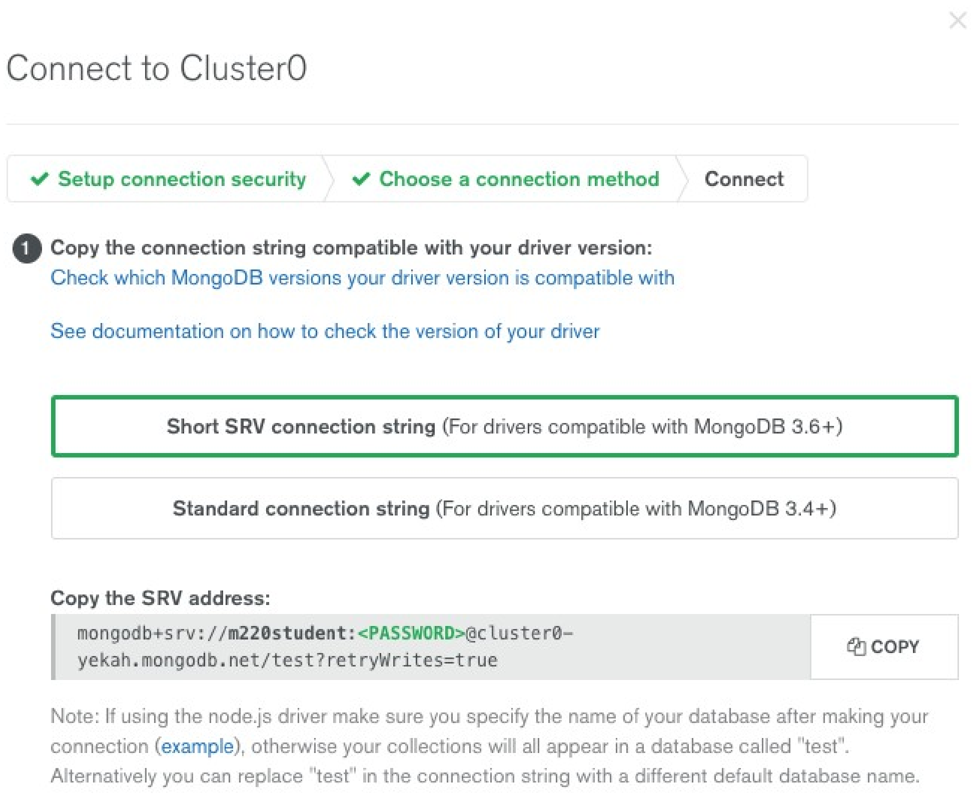
8. When the user is created, and the cluster deployed, you have the option to `Load Sample Dataset`. This will load the Atlas sample dataset, containing the Mflix database, into your cluster:



Note: The Mflix database in the Sample Dataset is called "sample_mflix".

9. Now you can test the setup by connecting via the `mongo` shell. You can find instructions to connect in the *Connect* section of the cluster dashboard. Go to your cluster *Overview* -> *Connect* -> *Connect Your Application*.

   Select the option corresponding to MongoDB version 3.6+ and copy the `mongo` connection URI.



1 Copy the connection string compatible with your driver version:
Check which MongoDB versions your driver version is compatible with

See documentation on how to check the version of your driver

Short SRV connection string (For drivers compatible with MongoDB 3.6+)

Standard connection string (For drivers compatible with MongoDB 3.4+)

Copy the SRV address:

```
mongodb+srv://m220student:<PASSWORD>@cluster0-
yekah.mongodb.net/test?retryWrites=true
```

Note: If using the node.js driver make sure you specify the name of your database after making your connection (example), otherwise your collections will all appear in a database called "test". Alternatively you can replace "test" in the connection string with a different default database name.

The below example connects to *Atlas* as the user you created before, with username **m220student** and password **m220password**. You can run this command from your command line:

```
mongo "mongodb+srv://m220student:m220password@<YOUR_CLUSTER_URI>"
```

By connecting to the server from your host machine, you have validated that the cluster is configured and reachable from your local workstation.

You may see the following message when you connect:

```
Error while trying to show server startup warnings: user is not allowed to do
action [getLog] on [admin.]
```

This is a log message, **not** an error - feel free to ignore it.

Importing Data (Optional)

Note: If you used Load Sample Dataset, you can skip this step.

If you are unable to use atlas cluster because of firewall/connectivity issue, then you can spin up a local mongod and import the data. To continue, please download and unzip the handout zip file given.

The `mongorestore` command necessary to import the data is located below. Copy the command and use the *Atlas SRV* string to import the data (including username and password credentials). If you have earlier versions of MongoDB installed on your machine, please make sure to use MongoDB 3.6 or greater version for this project.

Replace the SRV string below with your own:

```
# navigate to mflix directory
cd mflix-xxxx/mflix/data

# import data into Atlas
mongorestore --drop --gzip --uri
mongodb+srv://m220student:m220password@<YOUR_CLUSTER_URI> data

The entire dataset contains almost 200,000 documents, so importing this data
```

may take 5-10 minutes.

Setting Up mflix

1. Project Structure
2. Local Development Environment Configuration
3. Python Library Dependencies
4. Running the Application
5. Running the Unit Tests

In order to run properly, the MFlix software project has some installation requirements and environmental dependencies.

These requirements and dependencies are defined in this lesson, and they can also be found in the **README.rst** file from the **mflix-python** project, which you will download shortly. This lesson serves as a guide for setting up these necessary tools. After following this README, you should be able to successfully run the MFlix application. First, you will need to download the **mflix-python** project, as described below.

Download the mflix-python.zip file

You can download the **mflix-python.zip** file by clicking the link in the "Handouts" section of this page. Downloading this handout may take a few minutes. When the download is complete, unzip the file and `cd` into the project's root directory, **mflix-python**.

```
cd ~/Downloads
unzip mflix-python.zip
cd mflix-python
```

Project Structure

Everything you will implement is located in the `mflix/db.py` file, which contains all database interfacing methods. The API will make calls to `db.py` to interact with MongoDB.

The unit tests in `tests` will test these database access methods directly, without going through the API. The UI will run these methods in integration tests, and therefore requires the full application to be running.

The API layer is fully implemented, as is the UI. If you need to run on a port other than `5000`, you can edit the **index.html** file in the build directory to modify the value of **window.host**.

Please do not modify the API layer in any way, `movies.py` and `user.py` under the **mflix/api** directory. Doing so will most likely result in the frontend application failing to validate some of the labs.

Local Development Environment Configuration

Anaconda

We're going to use Anaconda to install Python 3 and to manage our Python 3 environment.

Installing Anaconda for Mac

You can download Anaconda from their MacOS download site. The installer will give you the option to "Change Install Location", so you can choose the path where the `anaconda3` folder will be placed. Remember this location, because you will need it to activate the environment.

Once installed, you will have to create and activate a `conda` environment:

```
# navigate to the mflix-python directory
cd mflix-python

# enable the "conda" command in Terminal
echo ". /anaconda3/etc/profile.d/conda.sh" >> ~/.bash_profile
source ~/.bash_profile

# create a new environment for MFlix
conda create --name mflix

# activate the environment
conda activate mflix
```

You can deactivate the environment with the following command:

```
conda deactivate
```

Installing Anaconda for Windows

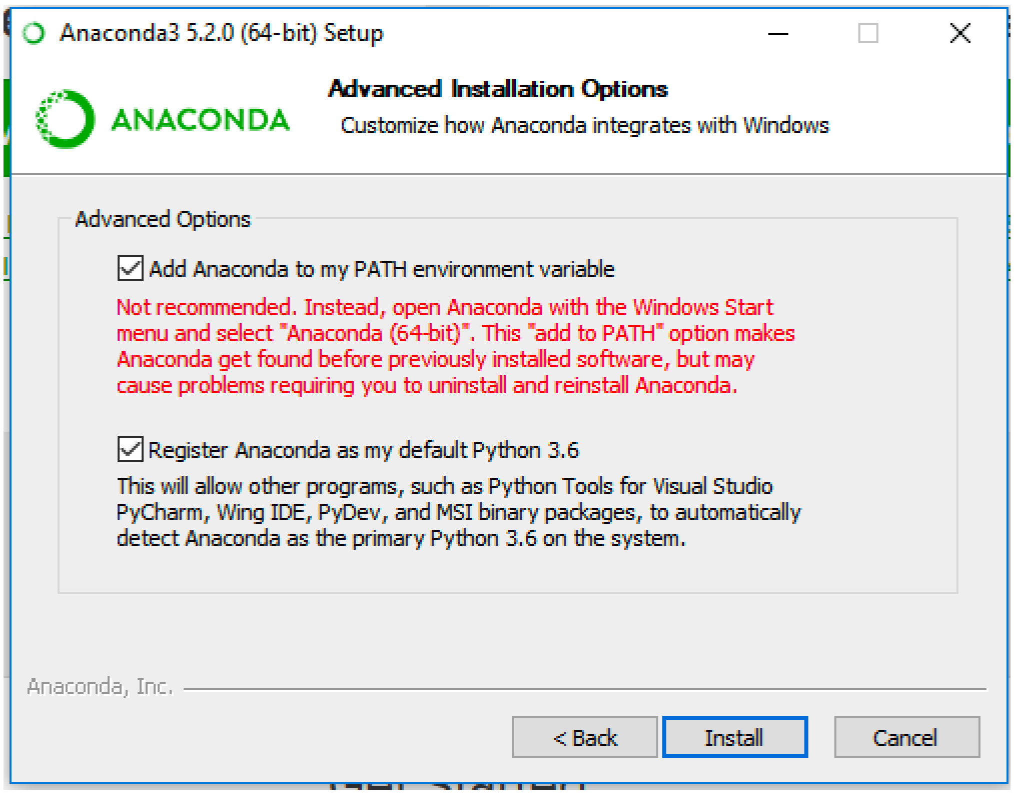
You can download Anaconda from their Download site. Please be careful to select `Windows Tab` before downloading.

The Anaconda installer will prompt you for the following options:

- *Add Anaconda to my PATH environment variable*
- *Register Anaconda as my default Python 3.6*

Please select both of these options. The first option will allow you to use `conda` commands from the Command Prompt, and the second option will allow you to use Anaconda's Python 3.6 as your system's default.

You may see a red error message like the following:


Anaconda3 5.2.0 (64-bit) Setup

Advanced Installation Options

Customize how Anaconda integrates with Windows

Advanced Options

☑ Add Anaconda to my PATH environment variable

Not recommended. Instead, open Anaconda with the Windows Start menu and select "Anaconda (64-bit)". This "add to PATH" option makes Anaconda get found before previously installed software, but may cause problems requiring you to uninstall and reinstall Anaconda.

☑ Register Anaconda as my default Python 3.6

This will allow other programs, such as Python Tools for Visual Studio PyCharm, Wing IDE, PyDev, and MSI binary packages, to automatically detect Anaconda as the primary Python 3.6 on the system.

This is expected. Please select both of the options above.

If you forget to select the *PATH* option before installing, no worries. The installer will let you choose an "Install Location" for Anaconda, which is the directory where the `Anaconda3` folder will be placed.

Using your machine's location of `Anaconda3` as `<path-to-Anaconda3>`, run the following commands to activate `conda` commands from the Command Prompt:

```
set PATH=%PATH%;<path-to-Anaconda3>;<path-to-Anaconda3>\Scripts\
```

Once Anaconda is installed, you will have to create and enable a `conda` environment.

```
# enter mflix-python folder
cd mflix-python

# create a new environment for MFlix
conda create --name mflix

# activate the environment
activate mflix
```

You can deactivate the environment with the following command:

```
deactivate
```

Virtualenv

Note: If you installed Anaconda instead, skip this step.

As an alternative to Anaconda, you can also use `virtualenv`, to define your Python 3 environment. You are required to have a Python 3 installed in your workstation.

You can find the virtualenv installation procedure on the PyPA website.

Once you've installed Python 3 and `virtualenv`, you will have to setup a `virtualenv` environment:

```
# navigate to the mflix-python directory
cd mflix-python

# create the virtual environment for MFlix
virtualenv -p YOUR_LOCAL_PYTHON3_PATH mflix_venv

# activate the virtual environment
source mflix_venv/bin/activate
```

You can deactivate the virtual environment with the following command:

```
deactivate
```

Python Library Dependencies

Once the Python 3 environment is activated, we need to install our python dependencies. These dependencies are defined in the `requirements.txt` file, and can be installed with the following command:

```
pip install -r requirements.txt
```

Running the Application

In the `mflix-python` directory you can find a file called `dotini`.

Open this file and enter your Atlas SRV connection string as directed in the comment. This is the information the driver will use to connect. Make sure **not** to wrap your Atlas SRV connection between quotes:

```
MFLIX_DB_URI = mongodb+srv://...
```

Rename this file to `.ini` with the following command:

```
mv dotini_unix .ini  # on Unix
ren dotini_win .ini # on Windows
```

Note: Once you rename this file to `.ini`, it will no longer be visible in Finder or File Explorer. However, it will be visible from Command Prompt or Terminal, so if you need to edit it again, you can open it from there:

```
vi .ini       # on Unix
notepad .ini  # on Windows
```

To start MFlix, run the following command:

```
python run.py
```

This will start the application. You can then access the MFlix application at http://localhost:5000/.

Running the Unit Tests

To run the unit tests for this course, you will use `pytest` and needs to be run from `mflix-python` directory. Each course lab contains a module of unit tests that you can call individually with a command like the following:

```
pytest -m LAB_UNIT_TEST_NAME
```

Each ticket will contain the command to run that ticket's specific unit tests. For example to run the Connection Ticket test your shell command will be:

```
pytest -m connection
```

Chapter 1: Driver Setup - MongoClient

Running the Jupyter Notebooks

This lesson, as many lessons in this course, was created using a Jupyter Notebook. These notebooks facilitate a more modular approach to writing Python.

Jupyter was included in `requirements.txt`, so it should already be installed in your environment.

To open and run the notebooks in this course, navigate to the `notebooks` directory and launch the Jupyter notebook server:

```
cd mflix-python/notebooks
jupyter notebook
```

This should give you a URL link to the notebook server (this server is running on your machine), and you can paste it into your browser to view the notebooks. Navigate between cells in the notebook with the arrow keys. You can run each cell in the notebook with `Shift-Enter`. Feel free to edit the individual cells and re-run the notebook.

The SRV string in the notebooks will not work, because the demo Atlas cluster for this course has been shut off. Please replace this with your own connection string.

Chapter 1: Driver Setup - Lab (Ungraded): Import Dataset

Problem:

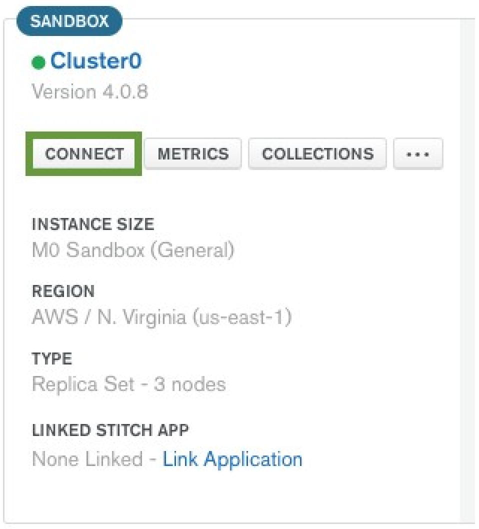
Note: There should be a section of the `README.rst` titled "Importing Data". If you imported data into Atlas during the **README** lesson, you can skip this lesson. **If you used Load Sample Dataset, you can also skip this step.**

Now that your Atlas cluster is configured correctly, let's populate it with some data. All the data required for MFlix is contained in the `data` directory in the handout.

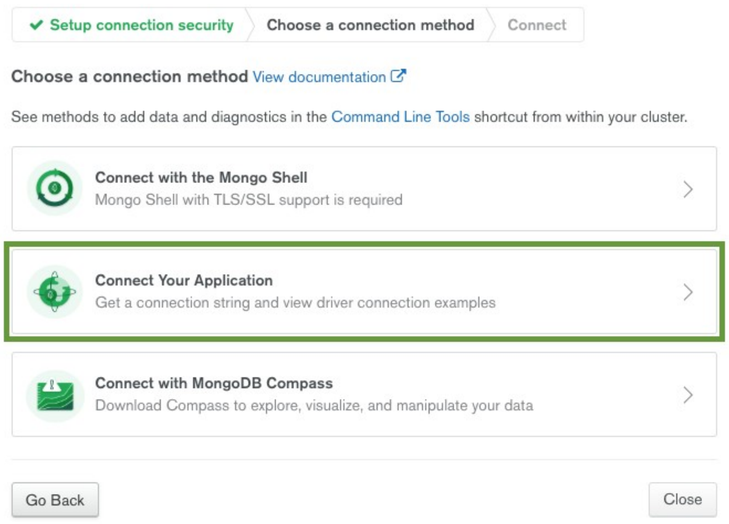
The `mongorestore` command necessary to import the data is located below. Copy the command and use the Atlas SRV string to import the data (including username and password credentials).

You can retrieve this SRV string from the Atlas website:

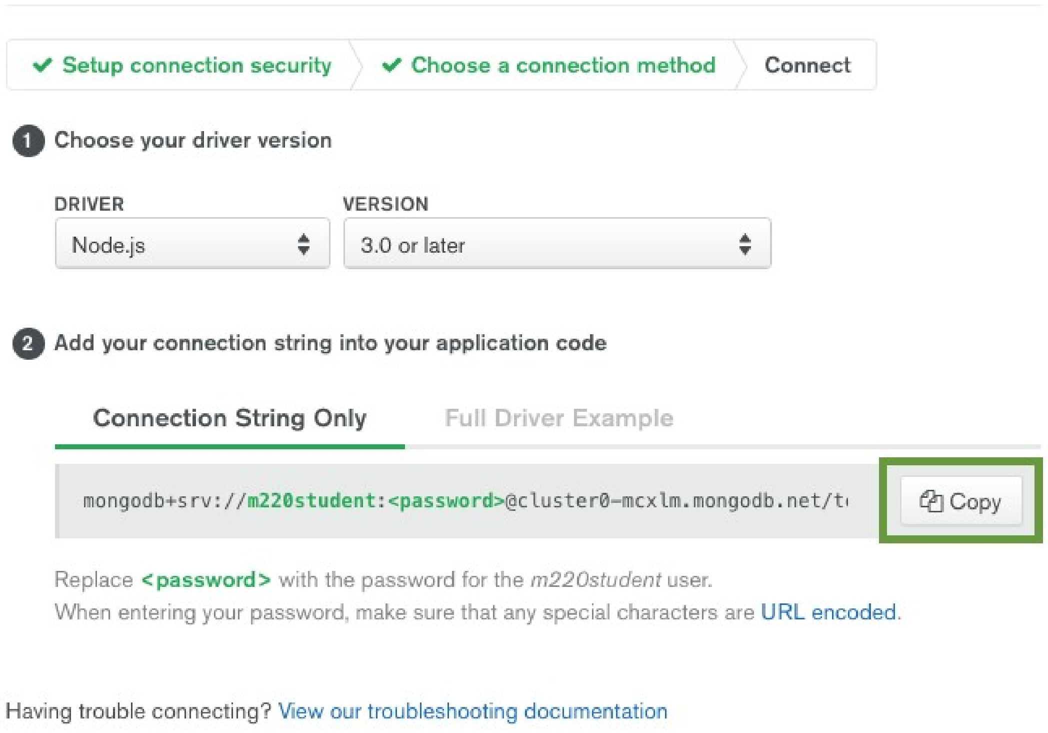
1. Click *Connect*:



2. Then click *Connect Your Application*:


Setup connection security
Choose a connection method
Connect
Choose a connection method View documentation
See methods to add data and diagnostics in the Command Line Tools shortcut from within your cluster.
Connect with the Mongo Shell
Mongo Shell with TLS/SSL support is required
Connect Your Application
Get a connection string and view driver connection examples
Connect with MongoDB Compass
Download Compass to explore, visualize, and manipulate your data
Go Back
Close

3. Then copy the SRV string to your clipboard:


✔ Setup connection security
✔ Choose a connection method
Connect
1 Choose your driver version
DRIVER
Node.js
VERSION
3.0 or later
2 Add your connection string into your application code
Connection String Only
Full Driver Example
Copy

```
mongodb+srv://m220student:<password>@cluster0-mcxlm.mongodb.net/t
```

Replace <password> with the password for the *m220student* user.
When entering your password, make sure that any special characters are URL encoded.

Having trouble connecting? View our troubleshooting documentation

Remember to replace `<password>` with your own password!

Once you have the SRV string, pass it to the `--uri` parameter:

```
mongorestore --drop --gzip --uri
mongodb+srv://m220student:m220password@<YOUR_CLUSTER_URI> data
```

A few tips when running `mongorestore`:

- The `--username` and `--password` flags cannot be used with and SRV string. Instead, the username and password should be **included** in the SRV string (i.e. `mongodb+srv://m220student:m220password@<your-cluster-address>`)
- In order to work properly, this command must be run from the top-level of the `mflix-<language>/` directory, where `<language>` is your chosen programming language.

You can verify that the data was imported by connecting to Atlas from the `mongo` shell:

```
mongo mongodb+srv://m220student:m220password@<YOUR_CLUSTER_URI>
```

Did you successfully import all the MFlix data into your Atlas cluster?

Chapter 1: Driver Setup - Ticket: Connection

Problem:

Task

MFlix will use MongoDB as a storage layer, so for this ticket you'll be required to perform some application setup.

1. First, make sure you've created a user on your Atlas cluster with read and write access to the **mflix** database.
    - The user name should be `m220student` and the password should be `m220password`
    - Don't forget to whitelist your IP address!
2. Copy the connection string. Select that you'd like to connect with the `mongo` shell, version *3.6 or later* - this will give you the **srv** connection string. Make sure this URI string contains your username and password!
3. Locate the file called **dotini_win** or **dotini_unix** (depending on your operating system) and replace the information within with your own **srv** connection string. The `[TEST]` URI will be used by the unit tests, while the integration tests will use the `[PROD]` URI:

    ```
    [PROD]
    SECRET_KEY = super_secret_key_you_should_change
    MFLIX_DB_URI = mongodb+srv://m220student:m220password@<your-atlas-cluster-
    address>
    MFLIX_NS = sample_mflix

    [TEST]
    SECRET_KEY = super_secret_testing_key
    MFLIX_DB_URI = your_testing_db_uri (can be the same as Atlas, or a local
    MongoDB database)
    MFLIX_NS = sample_mflix
    ```

    - It's highly suggested you also change the `SECRET_KEY` to some very long, very random string. While this application is only meant for local use during this course, software has a strange habit of living a long time.
4. Rename **dotini_win** or **dotini_unix** to **.ini**. You can do this by running the following command from the `mflix-python` directory:

    ```
    mv dotini_unix .ini   # on Unix
    ren dotini_win .ini  # on Windows
    ```

Note: Once you rename this file to `.ini`, it will no longer be visible in Finder or File Explorer. However, it will be visible from Command Prompt or Terminal, so if you need to edit it again, you can open it from there:

```
vi .ini       # on Unix
notepad .ini  # on Windows
```

Testing and Running the Application

In order to reinforce good development practices, everything asked of you in this course is backed up by unit tests. Reading through the tests for a specific exercise will tell you exactly what is expected.

You can run the unit tests for this ticket by running:

```
pytest -m connection
```

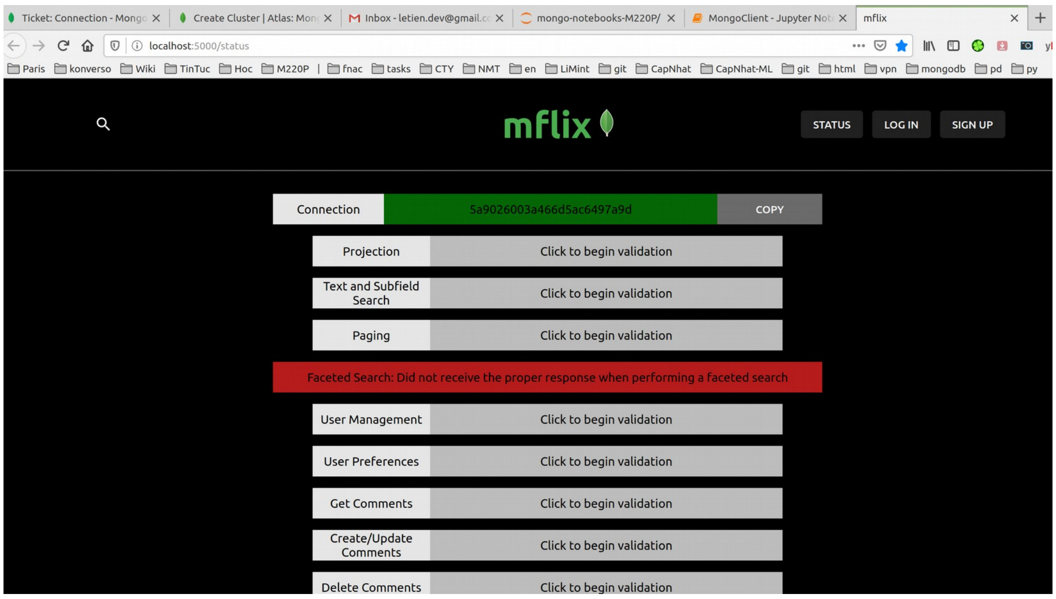
Once the unit tests are passing, run the application with:

```
python run.py
```

Now proceed to the status page to run the full suite of integration tests and get your validation code.

After passing the relevant tests, what is the validation code for **Connection**?

Answer: 5a9026003a466d5ac6497a9d



Connection	5a9026003a466d5ac6497a9d
Projection	Click to begin validation
Text and Subfield Search	Click to begin validation
Paging	Click to begin validation

Faceted Search: Did not receive the proper response when performing a faceted search

User Management	Click to begin validation
User Preferences	Click to begin validation
Get Comments	Click to begin validation
Create/Update Comments	Click to begin validation
Delete Comments	Click to begin validation

Chapter 1: Driver Setup - Ticket: Projection

Problem:

User Story

"As a user, I'd like to be able to search movies by country and see a list of movie titles. I should be able to specify a comma-separated list of countries to search multiple countries."

Task

Implement the **get_movies_by_country** method in `db.py` to search movies by country and use projection to return the `title` field.

You can find examples in `notebooks/your_first_read.ipynb`.

MFlix Functionality

Once you complete this ticket, the UI will allow movie searches by one or more countries.

Testing and Running the Application

Make sure to look at the tests in `test_projection.py` to understand what is expected.

You can run the unit tests for this ticket by running:

```
pytest -m projection
```

Once the unit tests are passing, run the application with:

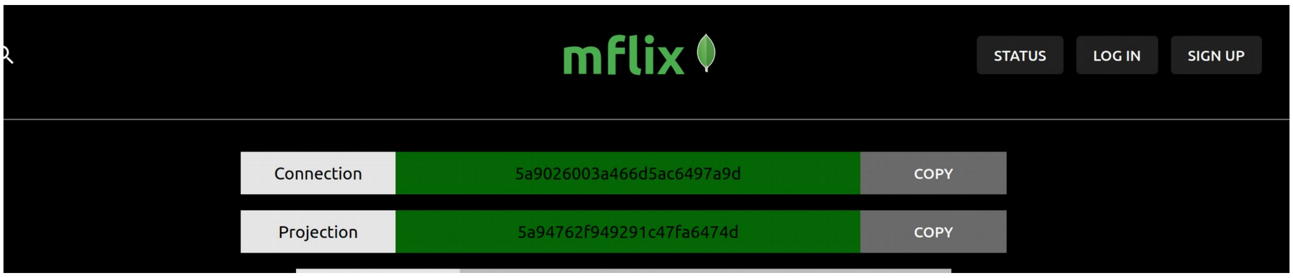
```
python run.py
```

Now proceed to the status page to run the full suite of integration tests and get your validation code.

After passing the relevant unit tests, what is the validation code for **Projection**?

Answer: **5a94762f949291c47fa6474d**

```
# TODO: Projection
# Find movies matching the "countries" list, but only return the title
# and _id. Do not include a limit in your own implementation, it is
# included here to avoid sending 46000 documents down the wire.
return list(db.movies.find({"countries": {"$in":countries}},
                           {"title": 1, "_id": 1}).limit(46000))
```



Connection	5a9026003a466d5ac6497a9d	COPY
Projection	5a94762f949291c47fa6474d	COPY

Chapter 1: Driver Setup - Ticket: Text and Subfield Search

Problem:

User Story

"As a user, I'd like to be able to search movies by cast members, genre, or perform a text search of the plot summary, full plot, and title."

Task

For this ticket, you will need to modify the method **build_query_sort_project** in `db.py` to allow the following movie search criteria:

- `genres`: finds movies that include any of the wanted genres.

Already, the **build_query_sort_project** method is able to return results for **two different types** of movie search criteria:

- `text`: performs a text search in the movies collection
- `cast`: finds movies that include any of the wanted cast

You just need to construct the query that queries the `movies` collection by the `genres` field.

Hint

> Check the implementation of similar formats of search criteria - the `genres` query should be similar.

MFlix Functionality

Once you complete this ticket, the UI will allow movie searches by members of the `cast`, movie `genres`, movie `title`, and `plot` summary.

A text index was created for you when you restored the collections with **mongorestore**, so these queries will be performant once they are implemented.

Testing and Running the Application

Make sure to look at the tests in `test_text_and_subfield_search.py` to understand what is expected.

You can run the unit tests for this ticket with:

```
pytest -m text_and_subfield_search
```

Once the unit tests are passing, run the application with:

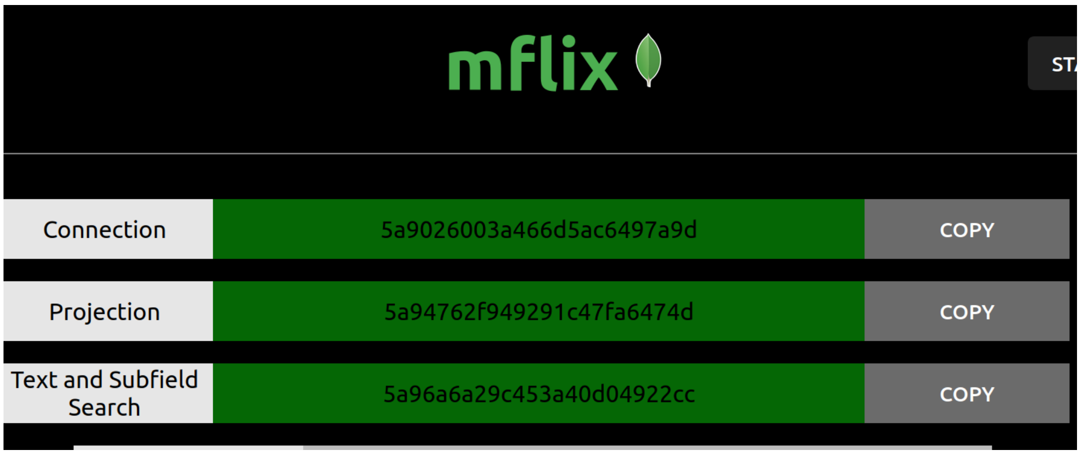
```
python run.py
```

Now proceed to the status page to run the full suite of integration tests and get your validation code.

After passing the relevant tests, what is the validation code for **Text and Subfield Search**?

Answer: 5a96a6a29c453a40d04922cc

```
# TODO: Text and Subfield Search
# Construct a query that will search for the chosen genre.
query = {"genres": {"$in": filters["genres"]}}
```



Connection	5a9026003a466d5ac6497a9d	COPY
Projection	5a94762f949291c47fa6474d	COPY
Text and Subfield Search	5a96a6a29c453a40d04922cc	COPY

Chapter 2: User-Facing Backend - Ticket: Paging

Problem:

User Story

"As a user, I'd like to get the next page of results for my query by scrolling down in the main window of the application."

Task

Modify the method **get_movies** in `db.py` to allow for paging. You can see how the page is parsed and sent in the **api_search_movies** method from `movies.py`.

You can find examples of using cursor methods in `notebooks/cursor_methods_agg_equivalents.ipynb`

MFlix Functionality

The UI is already asking for infinite scroll! You may have noticed a message stating "paging not implemented" when scrolling to the bottom of the page.

Once this ticket is completed, this message will go away, and scrolling to the bottom of the page will result in a new page of movies.

Testing and Running the Application

Make sure you look at the tests in `test_paging.py` to look at what is expected.

You can run the unit tests for this ticket by running:

```
pytest -m paging
```

Once the unit tests are passing, run the application with:

```
python run.py
```

Now proceed to the status page to run the full suite of integration tests and get your validation code.

After passing the relevant tests, what is the validation code for **Paging**?

Answer: 5a9824d057adff467fb1f526

```
# TODO: Paging
# Use the cursor to only return the movies that belong on the current page.
movies = cursor.skip(movies_per_page * page).limit(movies_per_page)
```

Chapter 2: User-Facing Backend - Basic Aggregation

o learn more about the Aggregation Framework, check out M121: The MongoDB Aggregation Framework.

The Aggregation Pipeline in this lesson was produced using MongoDB Compass.

Using Compass' Aggregation Pipeline Builder feature, we can easily create, delete and rearrange the stages in a pipeline, and evaluate the output documents in real time. Then we can produce a version of that pipeline in Python, Java, C# or Node.js, using Compass' Export-to-Language feature.

You can download Compass from the MongoDB Download Center if you'd like to give it a try.

Export-to-Language is new in Compass version 1.15.1.

Chapter 2: User-Facing Backend - Ticket: Faceted Search

Problem:

User Story

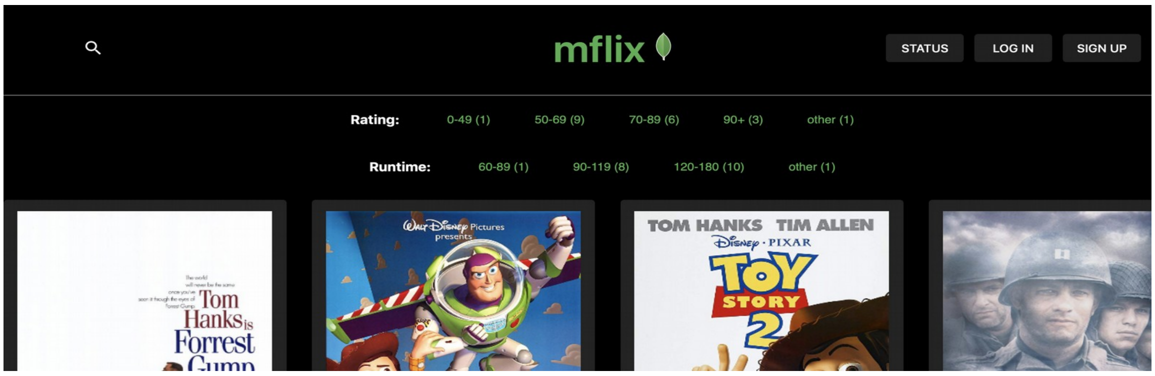
"As a user, I want to be able to filter cast search results by one facet, **metacritic** rating."

Task

For this Ticket, you'll be required to implement one method in `db.py`, **get_movies_faceted**, so the MFlix application can perform faceted searches.

MFlix Functionality

Once the change is implemented for this ticket, the user interface will reflect this change when you search for cast (e.g. "Tom Hanks"), then additional search parameters will be added as shown below:


mflix
STATUS
LOG IN
SIGN UP
Rating: 0-49 (1) 50-69 (9) 70-89 (6) 90+ (3) other (1)
Runtime: 60-89 (1) 90-119 (8) 120-180 (10) other (1)
Tom Hanks is Forrest Gump
TOM HANKS TIM ALLEN
TOY STORY 2

What is a Faceted Search?

Faceted search is a way of narrowing down search results as search parameters are added. For example, let's say MFlix allows users to filter movies by a rating from 1 to 10, but Kate Winslet has only acted in movies that have a rating of 6 or higher.

If we didn't specify any other search parameters, MFlix would allow us to choose a rating between 1 and 10. But if we first search for Kate Winslet, the application would only let us choose a rating between 6 and 10, because none of the movie documents in the result set have a rating below 6.

If you're curious, you can read more about Faceted Search here.

Faceted Search in MFlix

Faceted searches on the MFlix site cannot be supported with the basic search method **get_movies**, because that method uses the Mongo query language. For faceted searches, the application must use the Aggregation Framework.

The method **get_movies_faceted** uses the Aggregation Framework, and the individual *stages* in the pipeline have already been completed. Follow instructions in the `db.py` file to append the required stages to the pipeline object.

Testing and Running the Application

Look in the `test_facets.py` file in your **tests** directory to view the unit tests for this ticket.

You can run the unit tests for this ticket by running:

```
pytest -m facets
```

Once the unit tests are passing, run the application with:

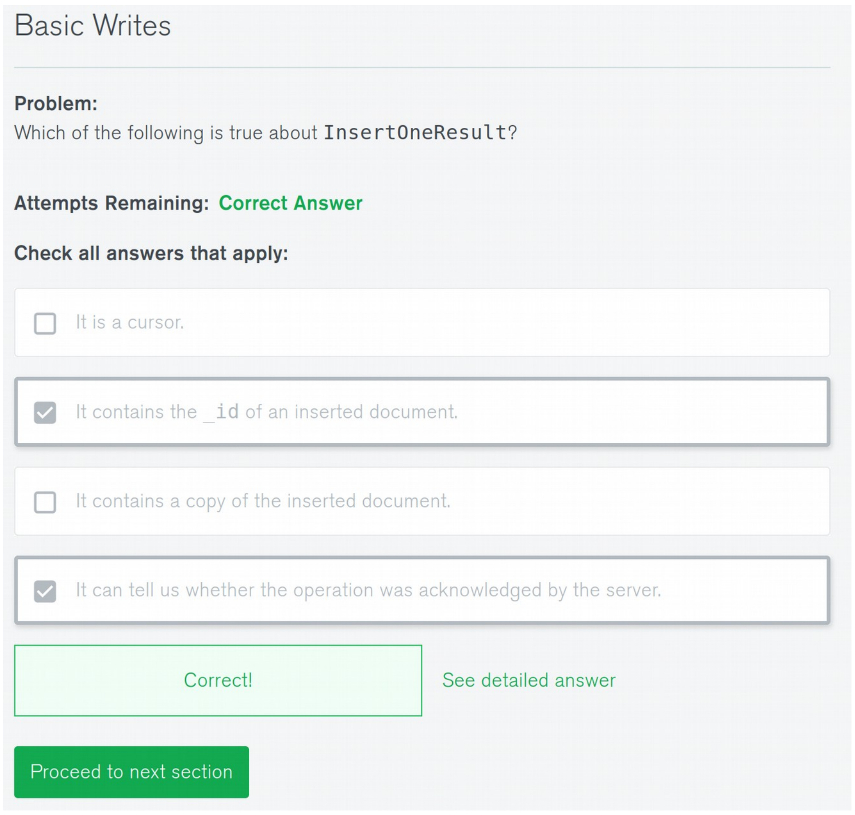
```
python run.py
```

Now proceed to the status page to run the full suite of integration tests and get your validation code.

After passing the relevant tests, what is the validation code for **Faceted Search**?

Answer:

```
# TODO: Faceted Search
# Add the necessary stages to the pipeline variable in the correct order.
# pipeline.extend(...)
pipeline.extend([skip_stage])
pipeline.extend([limit_stage])
pipeline.extend([facet_stage])
```



Basic Writes

Problem:
Which of the following is true about `InsertOneResult`?

Attempts Remaining: Correct Answer

Check all answers that apply:

- [ ] It is a cursor.
- [✓] It contains the `_id` of an inserted document.
- [ ] It contains a copy of the inserted document.
- [✓] It can tell us whether the operation was acknowledged by the server.

Chapter 2: User-Facing Backend - Ticket: User Management

Problem:

User Story

"As a user, I should be able to register for an account, log in, and logout."

Task

For this Ticket, you'll be required to implement all the methods in `db.py` that are called by the API endpoints in `user.py`. Specifically, you'll implement:

- **get_user**
- **add_user**
- **login_user**
- **logout_user**
- **get_user_session**
- **delete_user**

For this ticket, you will need to use the `find_one()`, `update_one()` and `delete_one()` methods. You can find examples in the following notebooks:

- `notebooks/deletes.ipynb`
- `notebooks/your_first_write.ipynb`

MFlix Functionality

Once this ticket is completed, users will be able to register for a new account, log in, logout, and delete their account.

Registering should create an account and log the user in, ensuring an entry is made in the **sessions** collection. There is a unique index on the `user_id` field in **sessions**, so we can efficiently query on this field.

Testing and Running the Application

Look within the `test_user_management.py` file in your **tests** directory to view the unit tests for this ticket.

You can run the unit tests for this ticket by running:

```
pytest -m user_management
```

Once the unit tests are passing, run the application with:

```
python run.py
```

Now proceed to the status page to run the full suite of integration tests and get your validation code.

After passing the relevant tests, what is the validation code for **User Management**?

Answer: 5a8d8ee2f9588ca2701894be

Chapter 2: User-Facing Backend - Ticket: Durable Writes

Problem:

Task

For this ticket, you'll be required to increase the durability of the **add_user** method from the default write concern of `w: 1`.

When a new user registers for MFlix, their information must be added to the database before they can do anything else on the site. For this reason, we want to make sure that the data written by the **add_user** method will not be rolled back.

We can completely eliminate the chances of a rollback by increasing the write durability of the **add_user** method. To use a non-default write concern with a database operation, use Pymongo's with_options flag when issuing the query.

You can find examples of write concerns in `notebooks/write_concerns.ipynb`.

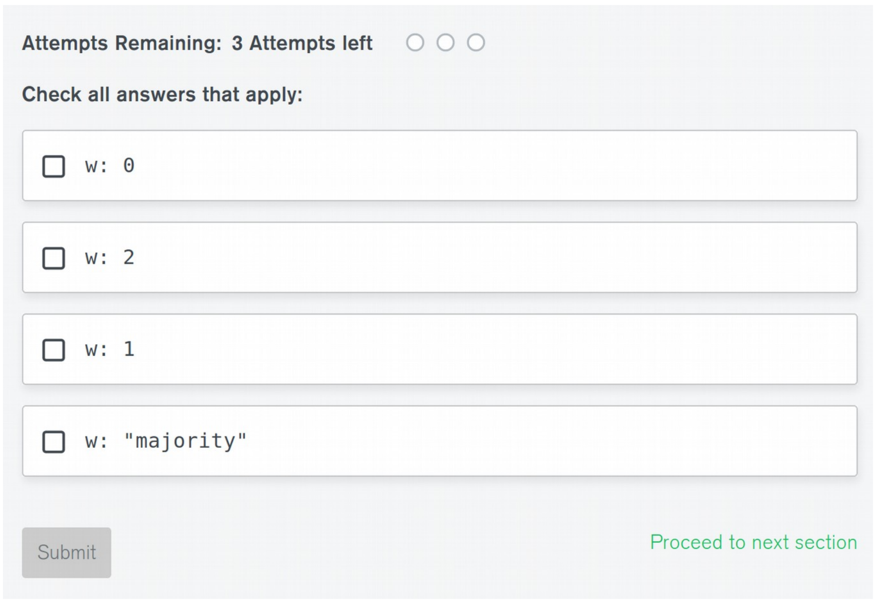
Testing and Running the Application

There are no unit or integration tests for this lab.

Please complete the multiple choice question below, and then implement the correct write concern in the **add_user** method.

The implementation of this task will not be tested, but using the default of `w: 1` might result in a rollback of your users' account data!

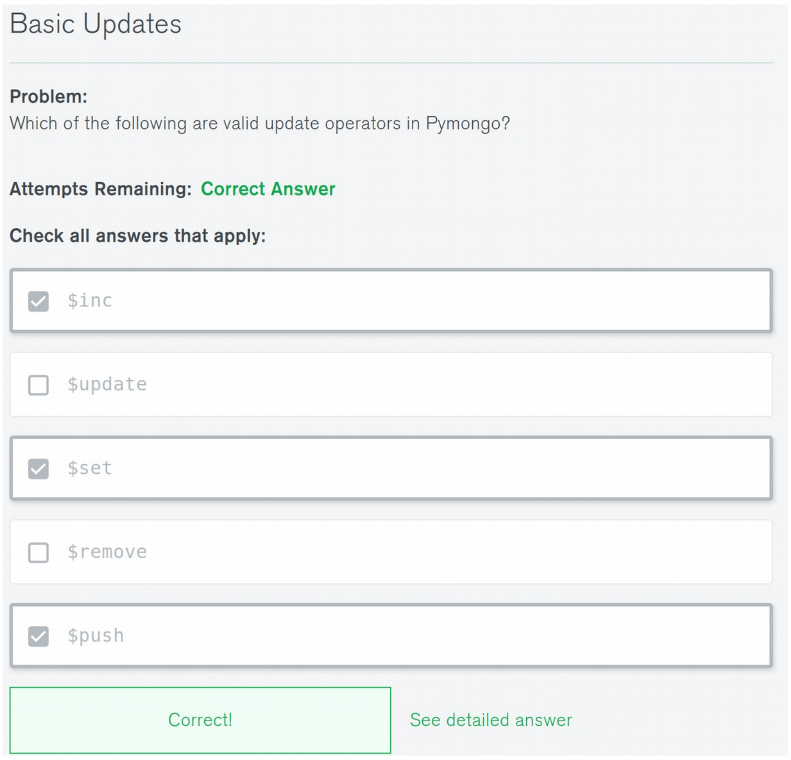
Which of the following write concerns are more durable than the default?



Attempts Remaining: 3 Attempts left ○ ○ ○

Check all answers that apply:

- ☐ `w: 0`
- ☐ `w: 2`
- ☐ `w: 1`
- ☐ `w: "majority"`



Basic Updates

Problem:
Which of the following are valid update operators in Pymongo?

Attempts Remaining: Correct Answer

Check all answers that apply:

- [✓ $inc]
- $update
- [✓ $set]
- $remove
- [✓ $push]

Chapter 2: User-Facing Backend - Ticket: User Preferences

Problem:

User Story

"As a user, I want to be able to store preferences such as my favorite cast member and preferred language."

Task

For this Ticket, you'll be required to implement one method in `db.py`, **update_prefs**. This method allows updates to be made to the `"preferences"` field in the `users` collection.

MFlix Functionality

Once this ticket is completed, users will be able to save preferences in their account information.

Testing and Running the Application

You can run the unit tests for this ticket by running:

```
pytest -m user_preferences
```

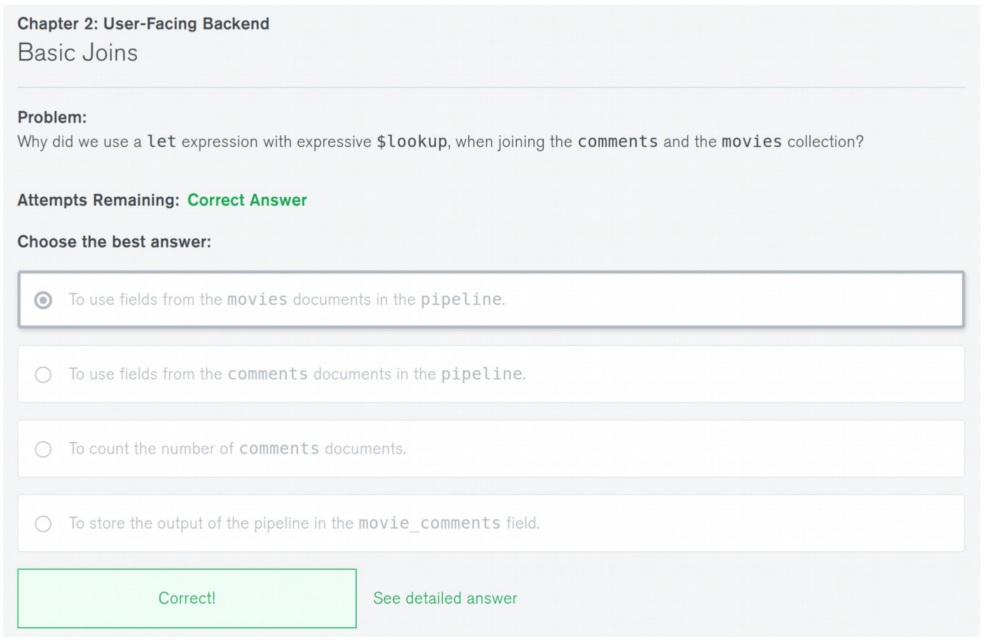
Once the unit tests are passing, run the application with:

```
python run.py
```

Now proceed to the status page to run the full suite of integration tests and get your validation code.

After passing the relevant tests, what is the validation code for **User Preferences**?

Answer: 5aabe31503ac76bc4f73e267


Chapter 2: User-Facing Backend

Basic Joins

Problem:
Why did we use a `let` expression with expressive `$lookup`, when joining the `comments` and the `movies` collection?

Attempts Remaining: Correct Answer

Choose the best answer:

- [✓ To use fields from the `movies` documents in the `pipeline`.]
- To use fields from the `comments` documents in the `pipeline`.
- To count the number of `comments` documents.
- To store the output of the pipeline in the `movie_comments` field.

Chapter 2: User-Facing Backend - Ticket: Get Comments

Problem:

User Story

"As a user, I want to be able to view comments for a movie when I look at the movie detail page."

Task

For this ticket, you'll be required to extend the **get_movie** method in `db.py` so that it also fetches the comments for a given movie.

The comments should be returned in order from most recent to least recent using the `date` key.

Movie comments are stored in the `comments` collection, so this task can be accomplished by performing a `$lookup`. Refer to the Aggregation Quick Reference for the specific syntax.

You can find examples of Aggregation with the Python driver in `notebooks/basic_aggregation.ipynb`.

MFlix Functionality

Once this ticket is completed, each movie's comments will be displayed on that movie's detail page.

Testing and Running the Application

You can run the unit tests for this ticket by running:

```
pytest -m get_comments
```

Once the unit tests are passing, run the application with:

```
python run.py
```

Now proceed to the status page to run the full suite of integration tests and get your validation code.

After passing the relevant tests, what is the validation code for **Get Comments**?

Hint: We need to sort the comments in the `$lookup` stage.

Answer: 5ab5094fb526e43b570e4633

Chapter 2: User-Facing Backend - Ticket: Create/Update Comments

Problem:

User Story

"As a user, I want to be able to post comments to a movie page as well as edit my own comments."

Task

For this ticket, you'll be required to implement two methods in `db.py`, **add_comment** and **update_comment**.

Ensure that **update_comment** only allows users to update their own comments, and no one else's comments.

MFlix Functionality

Once this ticket is completed, users will be able to post comments on their favorite (and least favorite) movies, and edit comments they've posted.

Testing and Running the Application

You can run the unit tests for this ticket by running:

```
pytest -m create_update_comments
```

Once the unit tests are passing, run the application with:

```
python run.py
```

Now proceed to the status page to run the full suite of integration tests and get your validation code.

After passing the relevant unit tests, what is the validation code for **Create/Update Comments**?

Answer: 5aba8d5113910c25d7058f8f

Chapter 2: User-Facing Backend - Ticket: Delete Comments

Problem:

User Story

"As a user, I want to be able to delete my own comments."

Task

For this ticket, you'll be required to modify one method in `db.py`, **delete_comment**. Ensure the delete operation is limited so only the user can delete their own comments, but not anyone else's comments.

You can find examples of `delete_one()` in `notebooks/deletes.ipynb`.

MFlix Functionality

Once this ticket is completed, users will be able to delete their own comments, but they won't be able to delete anyone else's comments.

Testing and Running the Application

You can run all the tests for this ticket by running:

```
pytest -m delete_comments
```

Once the unit tests are passing, run the application with:

```
python run.py
```

Now proceed to the status page to run the full suite of integration tests and get your validation code.

After passing the relevant tests, what is the validation code for **Delete Comments**?

Answer: 5ac25c280a80ed6e67e1cecb

Chapter 3: Admin Backend - Ticket: User Report

Problem:

User Story

"As an administrator, I want to be able to view the top 20 users by their number of comments."

Task

For this ticket, you'll be required to modify one method in `db.py`, **most_active_commenters**. This method produces a report of the 20 most frequent commenters on the MFlix site.

Hint

> This report is meant to be run from the backend by a manager that is very particular about the accuracy of data. Ensure that the read concern used in this read, avoids any potential document rollback.

Remember to add the necessary changes in the pipeline to meet the requirements. More information can be found in the comments of the method.

You can find examples of Aggregation with the Python driver in `notebooks/basic_aggregation.ipynb`.

MFlix Functionality

Once this ticket is completed, users with database access can make users administrators. Administrators will be able to generate a user report listing top commenters.

Testing and Running the Application

You can run the unit tests for this ticket by running:

```
pytest -m user_report
```

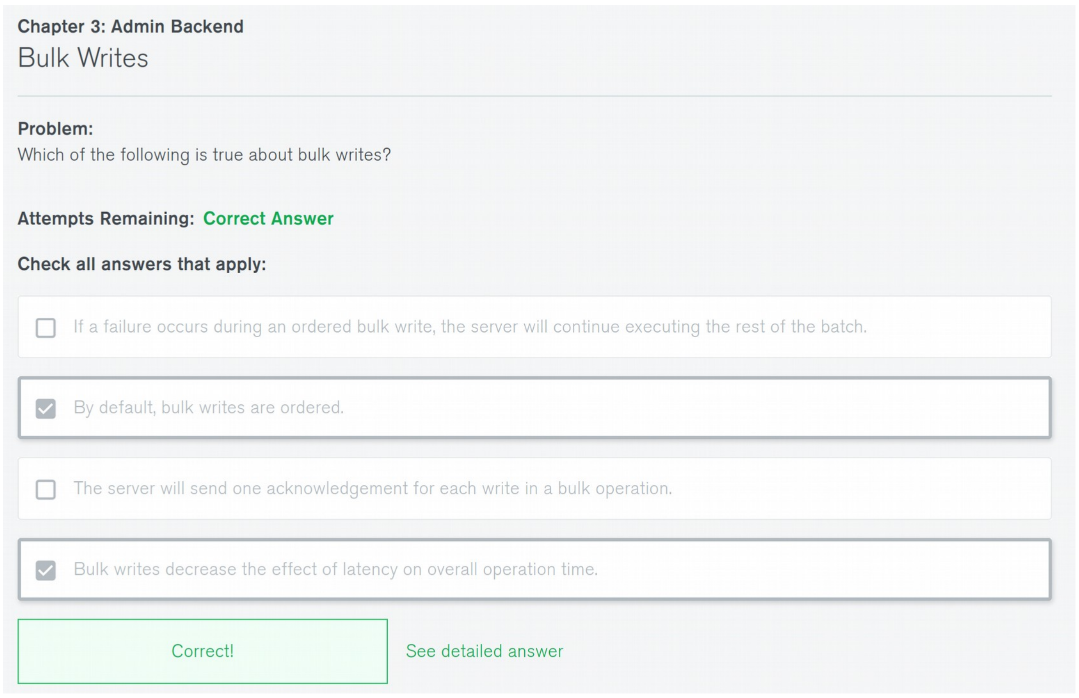
Once the unit tests are passing, run the application with:

```
python run.py
```

Now proceed to the status page to run the full suite of integration tests and get your validation code.

After passing the relevant tests, what is the validation code for **User Report**?

Answer: 5accad3272455e5db79e4dad



Chapter 3: Admin Backend
Bulk Writes

Problem:
Which of the following is true about bulk writes?

Attempts Remaining: Correct Answer

Check all answers that apply:

- [ ] If a failure occurs during an ordered bulk write, the server will continue executing the rest of the batch.
- [✓] By default, bulk writes are ordered.
- [ ] The server will send one acknowledgement for each write in a bulk operation.
- [✓] Bulk writes decrease the effect of latency on overall operation time.

Chapter 3: Admin Backend - Ticket: Migration

Problem:

Task

For this ticket, you'll be required to complete the command-line script located in the `migrations` directory of `mflix` called **movie_last_updated_migration.py**.

Things always change, and a requirement has come down that the `lastupdated` value in each document of the `movies` collection needs to be stored as an **ISODate** rather than a **String**.

Complete the script so it updates the values using the bulk API.

To perform the migration, run the script:

```
python movie_last_updated_migration.py
```

You can find examples of Bulk Operations in `notebooks/bulk_writes.ipynb`.

Testing and Running the Application

You can run the unit tests for this ticket by running:

```
pytest -m migration
```

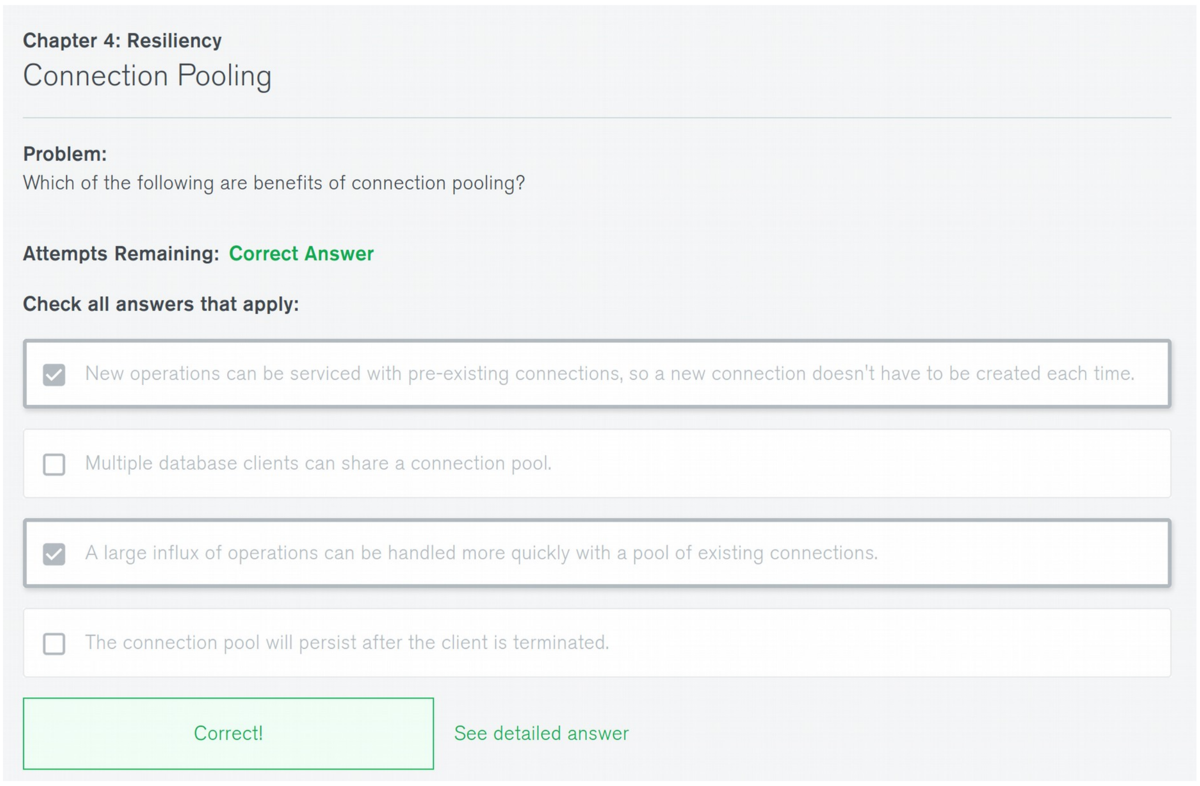
Once the unit tests are passing, run the application with:

```
python run.py
```

Now proceed to the status page to run the full suite of integration tests and get your validation code.

After passing the relevant tests, what is the validation code for **Migration**?

Answer: 5ad9f6a64fec134d116fb06f



Chapter 4: Resiliency

Connection Pooling

Problem:
Which of the following are benefits of connection pooling?

Attempts Remaining: Correct Answer

Check all answers that apply:

- [✓] New operations can be serviced with pre-existing connections, so a new connection doesn't have to be created each time.
- [ ] Multiple database clients can share a connection pool.
- [✓] A large influx of operations can be handled more quickly with a pool of existing connections.
- [ ] The connection pool will persist after the client is terminated.

Chapter 4: Resiliency - Ticket: Connection Pooling

Problem:

Task

For this ticket, you'll be required to modify the configuration of `MongoClient` to set the maximum size of the connection pool to 50 connections.

The `MongoClient` in `db.py` is initialized in the **get_db** method. A link to the `MongoClient` documentation is included here for your reference.

You can learn more about Connection Pooling in `notebooks/connection_pooling.ipynb`.

Testing and Running the Application

You can run the unit tests for this ticket by running:

```
pytest -m connection_pooling
```

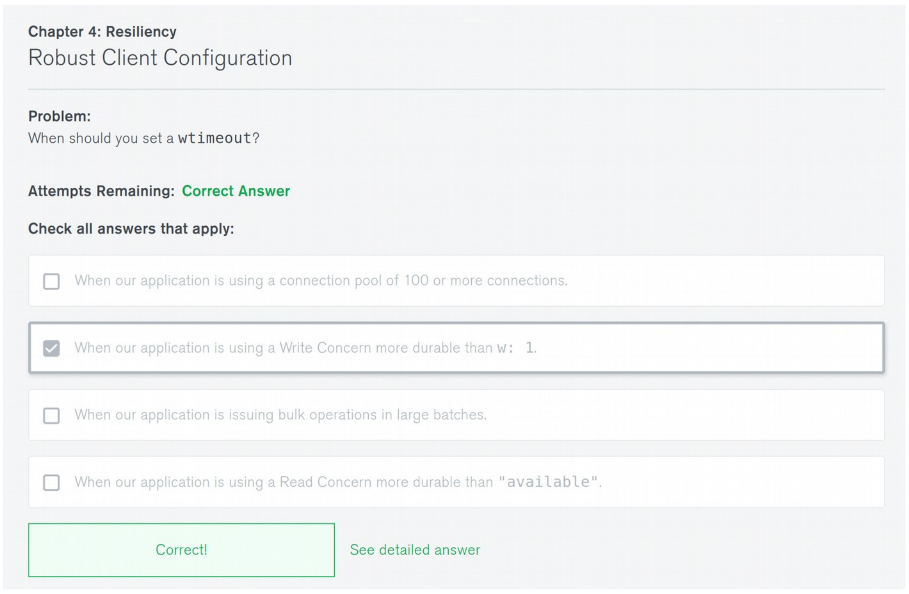
Once the unit tests are passing, run the application with:

```
python run.py
```

Now proceed to the status page to run the full suite of integration tests and get your validation code.

After passing the relevant tests, what is the validation code for **Connection Pooling**?

Answer: 5ad4f4f58d4b377bcf55d742



Chapter 4: Resiliency

Robust Client Configuration

Problem:

When should you set a `wtimeout`?

Attempts Remaining: Correct Answer

Check all answers that apply:

- [ ] When our application is using a connection pool of 100 or more connections.
- [x] When our application is using a Write Concern more durable than `w: 1`.
- [ ] When our application is issuing bulk operations in large batches.
- [ ] When our application is using a Read Concern more durable than `"available"`.

Chapter 4: Resiliency - Ticket: Timeouts

Problem:

Task

For this ticket, you'll be required to modify the connection information for `MongoClient` to set a write timeout of 2500 milliseconds.

The `MongoClient` in `db.py` is initialized in the **get_db** method. A link to the relevant documentation is included here for your reference.

You can learn more about timeouts in `notebooks/robust_applications.ipynb`.

Testing and Running the Application

You can run the unit tests for this ticket by running:

```
pytest -m timeouts
```

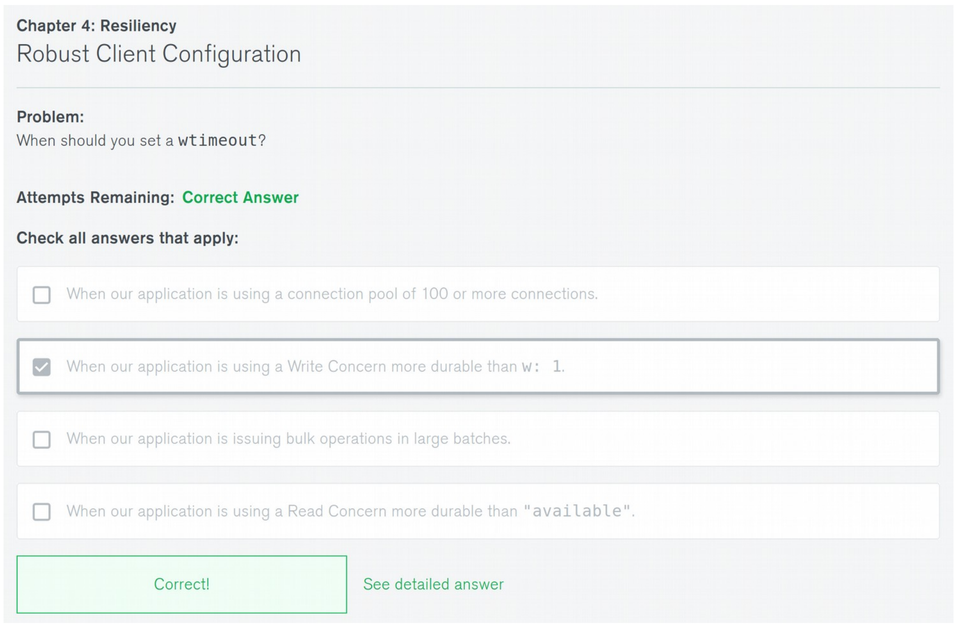
Once the unit tests are passing, run the application with:

```
python run.py
```

Now proceed to the status page to run the full suite of integration tests and get your validation code.

After passing the relevant tests, what is the validation code for **Timeouts**?

Answer: 5addf035498efdeb55e90b01



Chapter 4: Resiliency
Robust Client Configuration

Problem:
When should you set a `wtimeout`?

Attempts Remaining: Correct Answer

Check all answers that apply:

- [ ] When our application is using a connection pool of 100 or more connections.
- [x] When our application is using a Write Concern more durable than `w: 1`.
- [ ] When our application is issuing bulk operations in large batches.
- [ ] When our application is using a Read Concern more durable than `"available"`.

Correct! See detailed answer

Chapter 4: Resiliency - Ticket: Handling Errors

Problem:

Task

For this ticket, you'll be required to make the API more robust by handling exceptions. Specifically, what would happen should an incorrectly formatted **_id** be passed to **get_movie** in `db.py`?

To determine the exact exception that will be thrown in this case, please consult the documentation to see which exceptions `pymongo` and `bson` can raise:

- pymongo exceptions
- bson exceptions

Once you've determined the exception you need to handle, make sure to catch it in the **get_movie** method.

Although this ticket will only test that you've handled exceptions for the `get_movie` method, it's also a good idea to look over the entirety of **db.py** to look for other potential exceptions and handle them!

You can find examples of Error Handling in `notebooks/error_handling.ipynb`.

Testing and Running the Application

You can run the unit tests for this ticket by running:

```
pytest -m error_handling
```

Once the unit tests are passing, run the application with:

```
python run.py
```

Now proceed to the status page to run the full suite of integration tests and get your validation code.

After passing the relevant unit tests, what is the validation code for **Error Handling**?

Answer: 5ae9b76a703c7c603202ef22

Chapter 4: Resiliency - Ticket: Principle of Least Privilege

Problem:

Task

For this ticket, you'll be required to add a new user on your Atlas cluster for the MFlix application to connect with.

The user should follow credentials:

- username: **mflixAppUser**
- password: **mflixAppPwd**

This user should have the **readWrite** role on the **sample_mflix** database. Use **Add Default Privileges** to assign the user this specific role.

After you have created this user, modify the SRV connection string in your configuration file so the application connects with the new username and password.

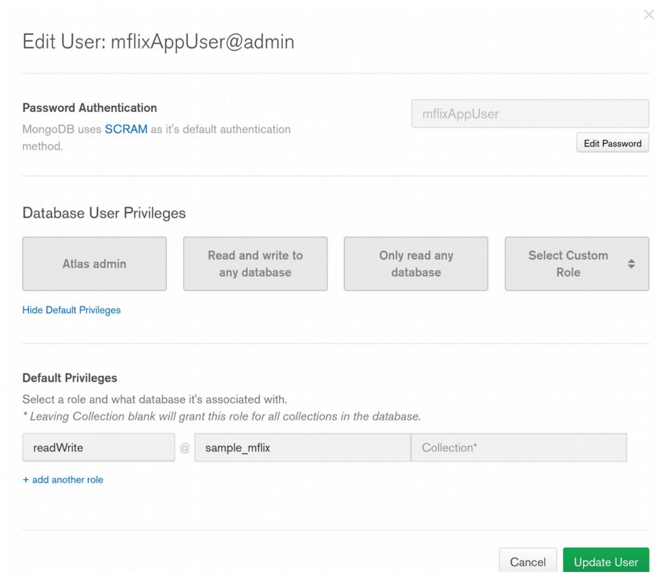
Testing and Running the Application

There are no unit tests associated with this ticket.

Once you have modified the connection string, stop and restart the application.

Now proceed to the status page to run the full suite of integration tests and get your validation code.

What is the validation code for **Principle of Least Privilege**?



Edit User: mflixAppUser@admin

Password Authentication
MongoDB uses SCRAM as it's default authentication method.

mflixAppUser

Edit Password

Database User Privileges

Atlas admin | Read and write to any database | Only read any database | Select Custom Role

Hide Default Privileges

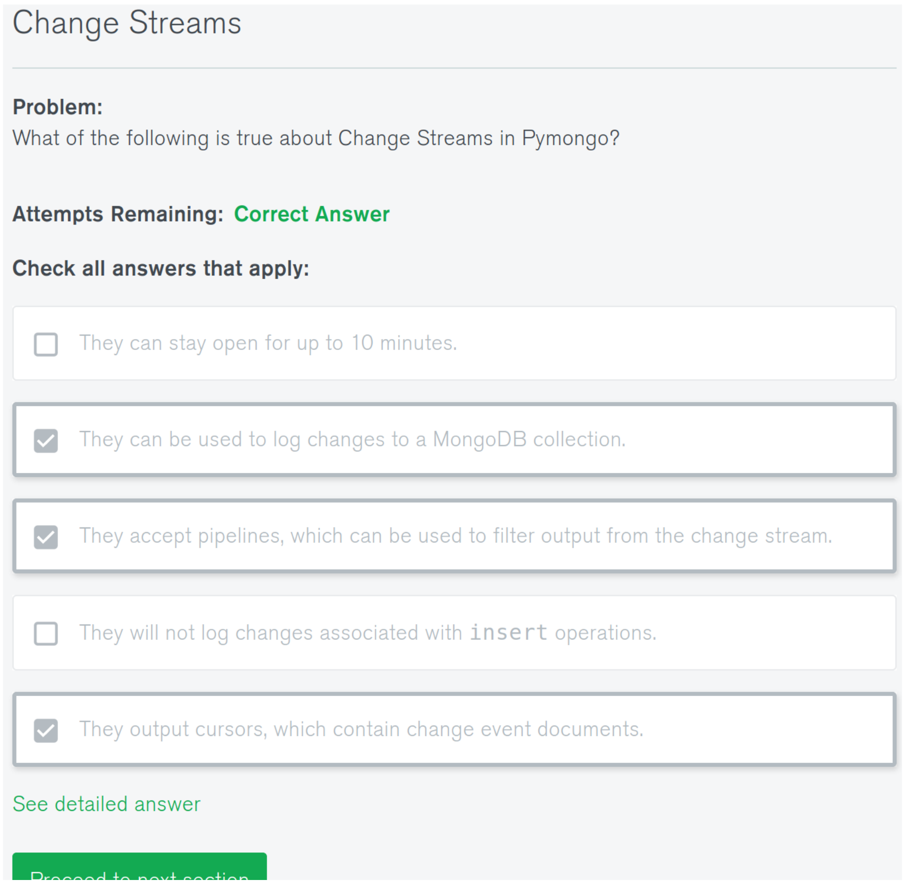
Default Privileges
Select a role and what database it's associated with.
* Leaving Collection blank will grant this role for all collections in the database.

readWrite @ sample_mflix | Collection*

+ add another role

Cancel | Update User

mongodb+srv://**mflixAppUser**:**mflixAppPwd**@mflix-lujgj.azure.mongodb.net/test
Answer: 5b61be29094dbae03bf30616



Change Streams

Problem:
What of the following is true about Change Streams in Pymongo?

Attempts Remaining: Correct Answer

Check all answers that apply:

- [ ] They can stay open for up to 10 minutes.
- [✓ They can be used to log changes to a MongoDB collection.]
- [✓ They accept pipelines, which can be used to filter output from the change stream.]
- [ ] They will not log changes associated with `insert` operations.
- [✓ They output cursors, which contain change event documents.]

Final Exam

Final: Question 1

Problem:

Assume a collection called `elections` that holds data about all United States Presidential Elections since 1789. All the documents in the `elections` collection look like this:

```
{
  "year" : 1828,
  "winner" : "Andrew Jackson",
  "winner_running_mate" : "John C. Calhoun",
  "winner_party" : "Democratic",
  "winner_electoral_votes" : 178,
  "total_electoral_votes" : 261
}
```

`total_electoral_votes` represents the total number of electoral votes that year, and `winner_electoral_votes` represents the number of electoral votes received by the winning candidates.

Which of the following queries will retrieve *all* the **Republican** winners with at least **160** electoral votes?

```
db.elections.find( { "winner_party": "Republican",
                     "winner_electoral_votes": { "$gte": 160 } } )
```

Final: Question 2

Problem:

Consider a collection of phones called `phones`, used by a phone manufacturer to keep track of the phones currently in production.

Each document in `phones` looks like this:

```
{
  "model": 5,
  "date_issued" : ISODate("2016-07-27T20:27:52.834Z"),
  "software_version": 4.8,
  "needs_to_update": false
}
```

There is an update required for phones with `software_version` earlier than 4.0. Anyone still using a version older than 4.0 will be asked to update.

The phone manufacturer wants to set the flag `needs_to_update` to `true` when the value of `software_version` is lower than 4.0. For example, a document like this one:

```
{
  "model": 5,
  "date_issued" : ISODate("2014-03-04T14:23:43.123Z"),
  "software_version": 3.7,
  "needs_to_update": false
}
```

Should be updated to look like this:

```
{
  "model": 5,
  "date_issued" : ISODate("2014-03-04T14:23:43.123Z"),
  "software_version": 3.7,
  "needs_to_update": true
}
```

Which of the following update statements will correctly perform this update?

```
db.phones.update_many( { "software_version": { "$lt": 4.0 } },
                       { "$set": { "needs_to_update": True } } )
```

Final: Question 3

Problem:

Suppose an instance of `MongoClient` is created with the *default settings*:

```
from pymongo import MongoClient

uri = "mongodb+srv://m220-user:m220-pass@m220-lessons-mcxlm.mongodb.net/test"
mc = MongoClient(uri)

mc.stats
```

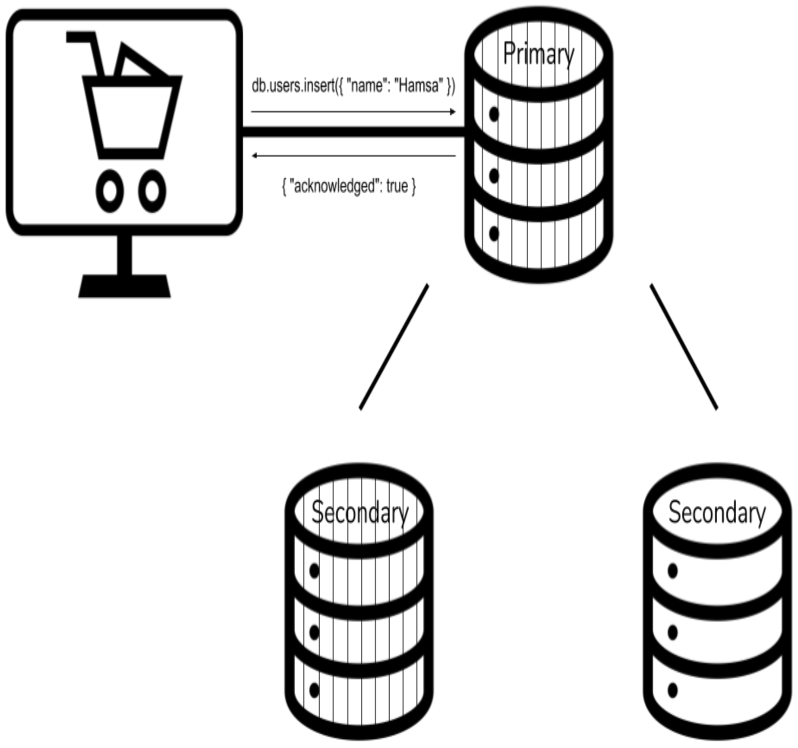
What will be the output of `mc.stats`?

```
Database(MongoClient(host=['m220-lessons-shard-00-02-
mcxlm.mongodb.net:27017', 'm220-lessons-shard-00-00-
mcxlm.mongodb.net:27017', 'm220-lessons-shard-00-01-
mcxlm.mongodb.net:27017'], document_class=dict, tz_aware=False,
connect=True, authsource='admin', replicaset='m220-lessons-shard-
0', ssl=True), 'stats')
```

Final: Question 4

Problem:

Suppose a client application is sending writes to a replica set with 3 nodes:



Before returning an acknowledgement back to the client, the replica set **waits**.

When the write has been applied by the nodes marked in **stripes**, it returns an acknowledgement back to the client.

What Write Concern was used in this operation?

`w: majority`

Final: Question 5

Problem:

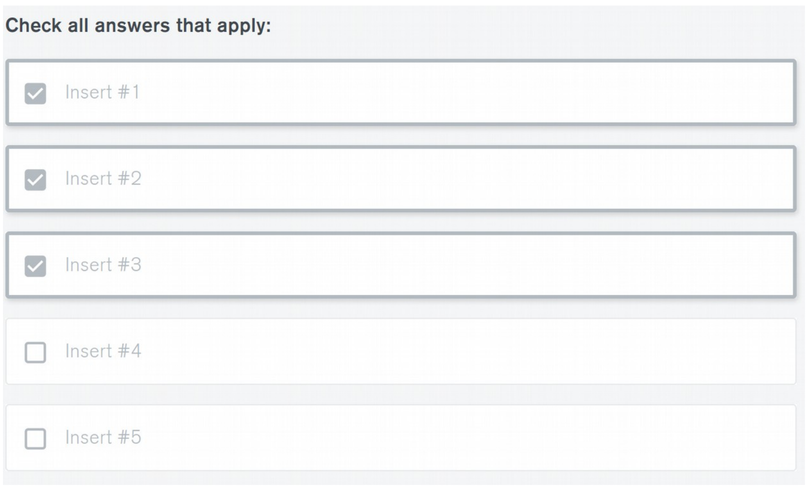
Given the following bulk write statement, to a collection called `employees`:

```
requests = [
  InsertOne({ '_id': 11, 'name': 'Edgar Martinez', 'salary': "8.5M" }),    # Insert #1
  InsertOne({ '_id': 3, 'name': 'Alex Rodriguez', 'salary': "18.3M" }),    # Insert #2
  InsertOne({ '_id': 24, 'name': 'Ken Griffey Jr.', 'salary': "12.4M" }),  # Insert #3
  InsertOne({ '_id': 11, 'name': 'David Bell', 'salary': "2.5M" }),        # Insert #4
  InsertOne({ '_id': 19, 'name': 'Jay Buhner', 'salary': "5.1M" })         # Insert #5
]

response = employees.bulk_write(requests)
```

Assume the `employees` collection is empty, and that there were no network errors in the execution of the bulk write.

Which of the insert operations in `requests` will succeed?



Check all answers that apply:

- [✓ Insert #1]
- [✓ Insert #2]
- [✓ Insert #3]
- Insert #4
- Insert #5

Correct Answers:

Inserts #1, #2, and #3 will succeed.

These writes do not conflict with each other, and they should all succeed if there are no network errors.

Incorrect Answers:

Inserts #4 and #5 will not succeed.

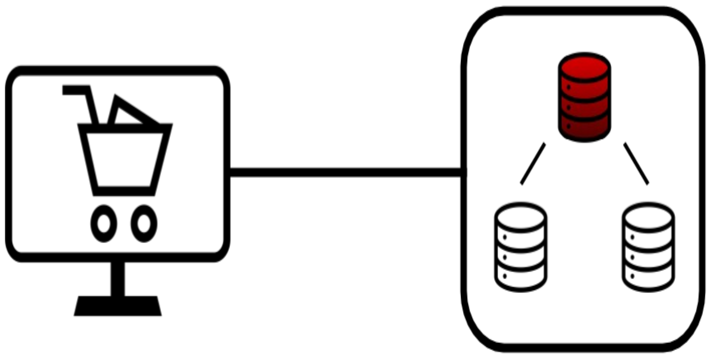
Insert #4 has the same `_id` as Insert #1, and the driver will throw a `DuplicateKeyError` on this insert, after receiving a similar exception from the server.

The default behavior for bulk writes is an *ordered* execution of the batch. So when Insert #4 fails, Insert #5 will not be executed.

Final: Question 6

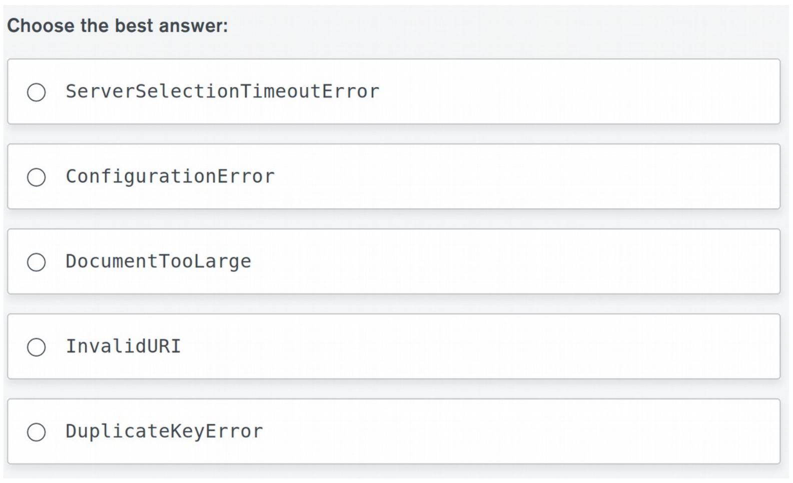
Problem:

Suppose a client application is sending writes to a replica set with three nodes, but the primary node stops responding:



Assume that none of the connection settings have been changed, and that the client is only sending insert statements with write concern `w: 1` to the server.

After 30 seconds, the client still cannot connect to a new primary. Which of the following exceptions will be raised by Pymongo?



Choose the best answer:

- ○ `ServerSelectionTimeoutError`
- ○ `ConfigurationError`
- ○ `DocumentTooLarge`
- ○ `InvalidURI`
- ○ `DuplicateKeyError`

`ServerSelectionTimeoutError`

Final: Question 7

Problem:

Assume a collection called `people_heights` with documents that look like this:

```
{
  "name": "Ada",
  "height": 1.7
}
```

Which of the following queries will find **only** the 4th- and 5th-tallest people in the `people_heights` collection?