# Serverless Dashboard for Bank Call Report Data Exploration

Drake UNIVERSITY

Blythe Kelly, Jaehyeok Choi, Dr. Eric Manley

## ABSTRACT

The Economics Data Science Research Group's goal is to use machine learning techniques to learn from previous data, so future bank failures can be predicted. But in order for this goal to be achieved, it's crucial for the data to be clean, organized and easily accessible.

Currently, the data has been cleaned and organized in our structured database in AWS. With the bank failure web API that our team has created, the data is easily accessible with simple HTTP requests to designated endpoints to query the desired data.

However, although the data is now clean, organized, and easily accessible, it is hard for us to truly understand the data in bulk JSON format that the API returns via HTTP requests. Since our data also contains the data from different time periods, it is crucial to recognize the changes in data as time progresses.

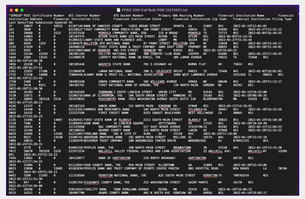
With this dashboard, users now can select the specific bank and the audit and the graph of the audit score over the time periods will be displayed. Now it is easy to spot the change in data over time with visualization.

This dashboard is currently available at

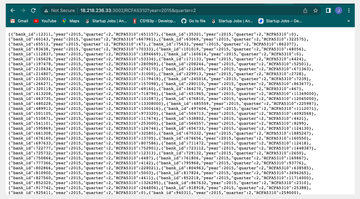
https://7a2sgjfva5.execute-api.us-east-2.amazonaws.com/dev

Or via the QR code =>

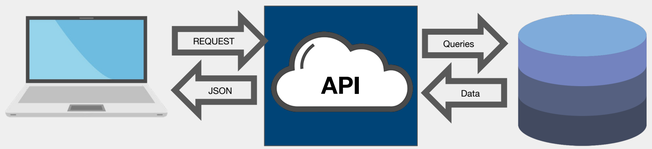
BULK FORMAT



WITH API



Dashboard | API | Database

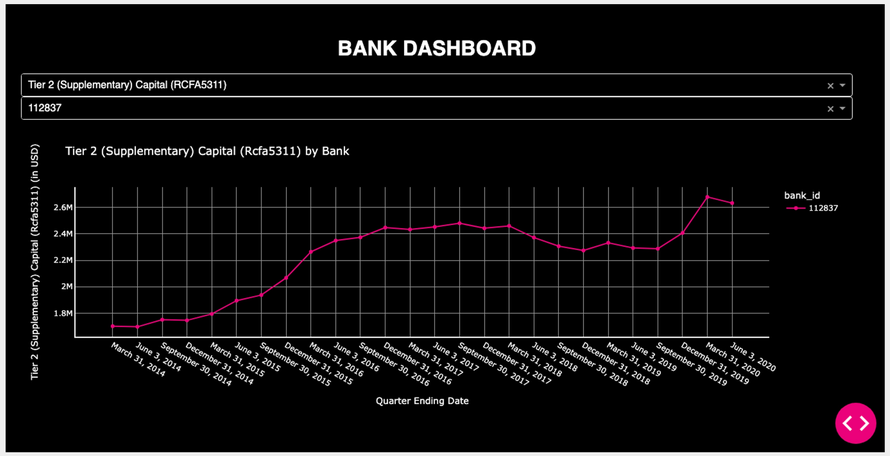



## FUTURE

Solve the issue with cold start related to serverless architecture
Display multiple graphs on same page
Optimization on rendering performance

## ACKNOWLEDGEMENTS

Dr. Eric Manley & Dr. Sean Severe, Drake CS 66 & 67
https://stackoverflow.com/questions/66845303/deploying-a-plotly-dash-app-to-aws-using-serverless-framework




## DASHBOARD LOGIC

- This dashboard is a user interface that helps visualize the financial health and income of each bank through a myriad of indicators. The indicators consist of individual and combinations of measures for risk profiles, capital, and assets.
- The dashboard uses a Dash app embedded in a Flask server to display longitudinal Federal Financial Institutions Examination Council (FFIEC) data from their Call Reports, which include indicators of the financial health of each bank.
- On the dashboard, there are two dropdowns that pass data into the Dash callback function.
  - The first dropdown lists all of the Regulatory Capital Components and Ratios schedule (RCFA) codes that can be individually selected by the user.
  - The RCFA codes are enriched with their meaning through correlating data from the data_dict table in the API.
  - The second dropdown displays the available bank IDs for the selected RCFA code. It is dependent on the first dropdown to avoid errors when a certain bank ID did not report on the specified criteria.

```
_true_get_distribution = pkg_resources.get_distribution
_Dist = namedtuple('_Dist', ['version'])

def _get_distribution(dist):
    if dist == 'flask-compress':
        return _Dist('1.9.0')
    else:
        return _true_get_distribution(dist)

pkg_resources.get_distribution = _get_distribution

server = Flask(__name__)

app = dash.Dash(__name__,
                server=server,
                routes_pathname_prefix='/',
                requests_pathname_prefix='/dev/'
                )
```

```
#First callback to output what bank IDs are included in the second dropdown.
@app.callback(Output(component_id='dropdown_bank_id', component_property= 'options'),
              Output(component_id='dropdown_bank_id', component_property= 'value'),
              [Input(component_id='dropdown_key', component_property= 'value')])

def set_bank_id(value):
    key_item_code=value[-9:-1]

    specific_key_list=requests.get(base_url+"/single/"+key_item_code).json()
    for record in specific_key_list:
        if not(record["bank_id"] in specific_key_bid_list):
            specific_key_bid_list.append(record["bank_id"])

    return specific_key_bid_list, specific_key_bid_list[0]
```

- The data is reported quarterly, and information for the quarter and year is converted into the date used in the graph through the Python function named update_year .
- The Plotly graph illustrates the trends based on the RCFA and bank code chosen.

## DEPLOYMENT

- The most common way to deploy a Python Dash application is by following the traditional pattern. This could mean opening up an EC2 instance, and set up the environment, handling the dependencies, and using gunicorn or other tools to run the application.
- Docker could be used to simplify the process of setting up the environment within the virtual machine and running the application in a containerized environment.
- Both approach that is presented above runs in a traditional server, where the application will be running 24/7, which would increase the cost of the server.
- Goal of this part is to run this application following the serverless architecture
- Serverless framework was used to decrease the deployment process complexity. The specifications are recorded in serverless.yml
- To simplify the process for the students to develop Dash application in serverless manner, the template was created
- https://github.com/Jaethem8y/dash-serverless-template
- For future students, who need to create their own application and want to deploy it on serverless.

- The pros of serverless architecture were obvious as it decreases the total cost of the server, and it is only getting billed as it is used and does not run consistently 24/7.
- The problem was detected as both API and the Dash application are running in serverless architecture; the initial cold start delay significantly decreased the load time of the application when it is first used by the user. Although the requests after had satisfactory performances, the delay in loads of UI components and data load was significant.