

CLOUD COMPUTING PROJECTS

FA18

Gregor von Laszewski
Geoffrey C. Fox

laszewski@gmail.com

CLOUD COMPUTING PROJECTS

Gregor von Laszewski

(c) Gregor von Laszewski, 2018

CLOUD COMPUTING PROJECTS

1 Projects Located Elsewhere

1.1 CM4 project contributors

2 Sample Project Report fa18-523-000 fa18-523-001

2.1 Abstract

2.2 Introduction

2.3 Requirements

2.3.1 Images

2.3.2 Systolic Issues

2.3.3 Copy from this document

2.4 Design

2.5 Architecture

2.6 Dataset

2.7 Implementation

2.8 Benchmark

2.9 Conclusion

2.10 Acknowledgement

2.11 Workbreakdown

3 Raspberry PI Cluster Report fa18-516-03

3.1 Abstract

3.2 Introduction

3.3 Requirements

3.4 Design

3.5 Architecture

3.6 Dataset

3.7 Implementation

3.8 Benchmark

3.9 Conclusion

3.10 Acknowledgement

4 TBD: fa18-516-04

5 Creating a RESTful API Service Using MongoDB and Azure SQL Database fa18-516-06

5.1 Abstract

5.2 Introduction

5.3 Design

[5.4 Architecture](#)

[5.5 Implementation](#)

[5.6 Steps to Reproduce](#)

[5.7 Technologies Used](#)

[5.8 Results](#)

[5.9 Conclusion](#)

[6 Managing AWS Lambda Using REST API](#) fa18-516-08

[6.1 Abstract](#)

[6.2 Introduction](#)

[6.3 Architecture](#)

[6.4 Implementation](#)

[6.5 Testing and Results](#)

[6.6 Recreating Project Environment](#)

[6.7 Technologies Used](#)

[6.8 Conclusion](#)

[6.9 Acknowledgement](#)

[7 Explore OpenFaaS Development and Deployment Aspects](#)

fa18-516-11 fa18-516-23

[7.1 Abstract](#)

[7.2 Introduction](#)

[7.2.1 Micro-Services](#)

[7.2.2 API Gateways](#)

[7.2.3 Serverless](#)

[7.2.4 OpenFaaS](#)

[7.3 Requirements](#)

[7.4 Design](#)

[7.4.1 Neural Network](#)

[7.4.2 Deep Neural Network](#)

[7.4.3 Convolutional Neural Network \(CNN\)](#)

[7.5 Architecture](#)

[7.6 Dataset](#)

[7.7 Implementation](#)

[7.7.1 Install Docker Swarm \(Single-Node Cluster\), Docker and OpenFaaS](#)

[7.7.2 Trouble Shooting](#)

[7.7.3 Build and deploy a serverless OpenFaaS function](#)

[7.7.4 Deploying to AWS](#)

[7.7.5 Project Files](#)

[7.8 Conclusion](#)

[7.9 Team Members and Work Breakdown](#)

[7.10 Acknowledgement](#)

[8 TBD fa18-516-12](#)

[9 HySDS on Kubernetes: ARIA InSAR Processing on XSEDE Jetstream](#)

[fa18-516-14](#)

[9.1 Abstract](#)

[9.2 Background](#)

[9.2.1 Architecture](#)

[9.2.2 Processing Algorithm Library](#)

[9.2.3 Pedigree](#)

[9.3 Requirements](#)

[9.4 Design](#)

[9.5 Architecture](#)

[9.5.1 Mozart](#)

[9.5.2 Metrics](#)

[9.5.3 GRQ](#)

[9.5.4 hysds-k8s](#)

[9.6 Dataset](#)

[9.6.1 Sentinel-1 SLCs](#)

[9.6.2 Sentinel-1 Orbits and Calibrations](#)

[9.6.3 DEMs](#)

[9.7 Implementation](#)

[9.7.1 Create a Kubernetes Cluster on IU Jetstream](#)

[9.7.2 Configuring OpenStack Cloud Provider](#)

[9.7.3 Create HySDS Buckets \(Swift Containers\)](#)

[9.7.4 Create the HySDS cluster](#)

[9.7.5 Register the lightweight-jobs and ariamh repositories in Jenkins](#)

[9.7.6 Submit Sentinel-1 Interferogram Jobs](#)

[9.8 Benchmark](#)

[9.9 Conclusion](#)

[9.10 Acknowledgement](#)

[10 Scalable Data Processing for Retail fa18-516-17](#)

[10.1 Introduction](#)

[10.2 Dataset](#)

[10.2.1 Project dataset.](#)

[10.3 Implementation](#)

[10.3.1 Hadoop](#)

[10.3.2 Linux](#)

[10.3.3 HDFS](#)

[10.3.4 Hive](#)

[10.3.5 API](#)

[10.3.6 Data Storage](#)

[11 Manage Files Across Cloud Providers fa18-516-18](#)

[11.1 Abstract](#)

[11.2 Introduction](#)

[11.3 Requirements](#)

[11.4 Design](#)

[11.5 Implementation](#)

[11.5.1 AWS access from Python:](#)

[11.5.2 Google Cloud Platform:](#)

[11.5.3 MongoEngine GridFS](#)

[11.6 Dataset](#)

[11.7 Benchmark](#)

[11.8 Conclusion](#)

[11.9 Acknowledgement](#)

[11.10 References](#)

[12 LAMP Stacks - Linux Apache, MySQL, PHP](#)

[12.1 Abstract](#)

[12.2 Introduction](#)

[12.3 OpenAPIs](#)

[12.4 Setting up a LAMP Stack](#)

[12.4.1 Setup the environment](#)

[12.4.2 Install Linux \(Ubuntu\)](#)

[12.4.3 Setting Up Vagrant](#)

[12.4.4 Installing Apache](#)

[12.4.5 Installing PHP and MySql](#)

[13 Cloudmesh GraphQL App fa18-516-21 and fa18-516-02](#)

[13.1 Abstract](#)

[13.2 Introduction](#)

[13.3 Requirements](#)

[13.4 Design](#)

- [13.5 Architecture](#)
- [13.6 Dataset](#)
- [13.7 Implementation](#)
- [13.8 Benchmark](#)
- [13.9 Conclusion](#)
- [13.10 Acknowledgement](#)
- [13.11 Workbreakdown](#)
- [14 Open API with AWS EMR and Jupyter](#) fa18-516-22
 - [14.1 Abstract](#)
 - [14.2 Introduction](#)
 - [14.2.1 Open API](#)
 - [14.2.2 AWS EMR](#)
 - [14.2.3 AWS EC2](#)
 - [14.2.4 AWS S3](#)
 - [14.2.5 JupyterHub](#)
 - [14.3 Implementation](#)
 - [14.3.1 Setting up AWS CLI](#)
 - [14.3.2 Setting up AWS Admin Access](#)
 - [14.3.3 Creating and Configuring EC2 Instance to Host API](#)
 - [14.3.4 Create S3 Storage](#)
 - [14.3.5 Codegen Set Up](#)
 - [14.3.6 Building the EMR Rest Service](#)
 - [14.3.7 Deploy S3 Rest Service](#)
 - [14.3.8 Deploy Notebook Rest Service](#)
 - [14.3.9 Running all APIs](#)
 - [14.4 Conclusion](#)
- [15 TBD](#) fa18-516-24
- [16 Hadoop, Hive and Spark multi node Cluster set up on Amazon EC2 instances](#)
 - [16.1 ABSTRACT](#)
 - [16.2 Keywords](#)
 - [16.3 INTRODUCTION](#)
 - [16.4 SOFTWARE VERSIONS](#)
 - [16.5 ARTIFACTS](#)
 - [16.6 REFERENCES](#)
- [17 OpenFaaS & Docker on a Raspberry Pi Multi-Node Cluster with cm-burn Report](#) fa18-516-23

[17.1 Abstract](#)
[17.2 Introduction](#)
[17.3 Requirements](#)
[17.4 Design](#)
[17.5 Architecture](#)
[17.6 Dataset](#)
[17.7 Implementation](#)
[17.8 Benchmark](#)
[17.9 Conclusion](#)
[17.10 Acknowledgement](#)

[18 Morphological Image-based Profiling of Skin Lesions for Scientific Community fa18-523-52](#)

[18.1 Abstract](#)
[18.2 Introduction](#)
[18.3 Requirements](#)
[18.4 Design](#)
[18.5 Architecture](#)
[18.6 Installation](#)
[18.7 Dataset](#)
[18.8 Implementation](#)
[18.9 Benchmark](#)
[18.10 Conclusion](#)
[18.11 Acknowledgement](#)

[19 Orchestrating Microservices for a Credit Scoring Application in Kafka fa18-523-53](#)

[19.1 Abstract](#)
[19.2 Introduction](#)
[19.3 Apache Kafka](#)
[19.4 Requirements](#)
[19.5 Architecture](#)
[19.6 Design](#)
[19.7 Dataset](#)
[19.8 Implementation](#)
[19.9 Conclusion](#)
[19.10 Acknowledgement](#)

[20 Apache NiFi Chapter fa18-523-56](#)

[20.1 Apache NiFi Introduction](#)

[20.2 NiFi History](#)

[20.3 NiFi Features](#)

[20.4 NiFi Architecture](#)

[20.4.1 Web Server](#)

[20.4.2 Flow Controller](#)

[20.4.3 FlowFile Repository](#)

[20.4.4 Content Repository](#)

[20.4.5 Provenance Repository](#)

[20.4.6 Processors](#)

[20.4.7 NiFi Clusters](#)

[20.5 Install NiFi](#)

[20.5.1 Apache NiFi - Windows](#)

[20.6 Building a NiFi Flow](#)

[20.7 Linking NiFi Flow to Apache Kafka](#)

[20.8](#)

[20.9 Alternatives to NIFI](#)

[20.10 Conclusion](#)

[20.11 Acknowledgement](#)

[21 Historical Storm Data Analysis with Cosmos DB](#) fa18-523-57

[fa18-523-58](#)

[21.1 Abstract](#)

[21.2 Introduction](#)

[21.3 Implementation](#)

[21.3.1 Data set](#)

[21.3.2 Related Work](#)

[21.4 Technologies Used](#)

[21.5 Visualization](#)

[21.6 Machine Learning](#)

[21.6.1 Multinomial Naive Bayes](#)

[21.7 Summary](#)

[21.8 Future Work](#)

[21.9 Acknowledgement](#)

[21.10 Work Breakdown](#)

[22 Kickstarter Projects Analysis](#) fa18-523-60, fa18-523-64, fa18-523-72

[22.1 Abstract](#)

[22.2 Introduction](#)

- [22.3 Literature review](#)
 - [22.4 Dataset Description](#)
 - [22.4.1 Kaggle API](#)
 - [22.4.2 Kickstarter Projects Dataset](#)
 - [22.5 Design and Methods](#)
 - [22.6 Technologies](#)
 - [22.6.1 Technologies and Tools Used](#)
 - [22.7 Code Organization](#)
 - [22.7.1 bin](#)
 - [22.8 Architecture](#)
 - [22.9 Observations and Visualizations](#)
 - [22.9.1 Exploratory Analysis](#)
 - [22.9.2 Time Series Analysis](#)
 - [22.9.3 Logistic Regression](#)
 - [22.9.4 MongoDB Queries](#)
 - [22.10 Conclusion](#)
 - [22.11 Acknowledgement](#)
 - [22.12 Workbreakdown](#)
 - [22.12.1 Nishad Tupe](#)
 - [22.12.2 Vishal Bhoyar](#)
 - [22.12.3 Izolda Fetko](#)
 - [22.13 Nishad Tupe, Vishal Bhoyar, Izolda Fetko](#)
- [23 Twitter Text Mining using Python, MongoDB, and Amazon Web Services - fa18-523-61](#)
- [23.1 Abstract](#)
 - [23.2 Introduction](#)
 - [23.3 Twitter API and Python Implementation](#)
 - [23.4 Data](#)
 - [23.5 Twitter Cloud Storage](#)
 - [23.6 Data Science Algorithms for Twitter Data](#)
 - [23.7 Running a Twitter Script in Python](#)
 - [23.7.1 Python Libraries](#)
 - [23.7.2 Visualizations](#)
 - [23.7.3 Machine Learning Algorithms](#)
 - [23.8 Results](#)
 - [23.9 Conclusion](#)
 - [23.10 Future Research](#)

23.11 Acknowledgements

24 Predict EPL results using Tweets fa18-523-62 fa18-523-69

24.1 Abstract

24.2 Introduction

24.3 Related Work

24.4 Implementation

 24.4.1 Data

 24.4.2 MongoDB

 24.4.3 Tweet Extraction

 24.4.4 Machine Learning Approaches

24.5 Results

24.6 Conclusion

24.7 Acknowledgement

24.8 Work Breakdown

25 Analyzing Big Data Sorting Algorithms fa18-523-63

25.1 Abstract

25.2 Introduction

25.3 Requirements

25.4 Design

 25.4.1 Bubble Sort

 25.4.2 Merge Sort

 25.4.3 Insertion Sort

 25.4.4 Shell Sort

 25.4.5 Selection Sort

 25.4.6 Strand Sort

 25.4.7 Python Sort (Timsort)

 25.4.8 Heap Sort

25.5 Architecture

25.6 Dataset

25.7 Implementation

25.8 Benchmark

25.9 Conclusion

25.10 Acknowledgements

26 Data analysis Of Yelp reviews fa18-523-65, fa18-523-67

26.1 Abstract

26.2 Introduction

26.3 Literature review

- [26.4 Dataset](#)
- [26.5 Data processing](#)
- [26.6 Analysis methods](#)
- [26.7 Results](#)
- [26.8 Discussion](#)
- [26.9 Conclusions](#)
- [26.10 Project members and Work Breakdown](#)
- [27 Do I buy or sell? - Using Big Data to predict Stock Prices fa18-523-66 \(...editing\)](#)
 - [27.1 Abstract](#)
 - [27.2 Introduction](#)
 - [27.3 Implementation](#)
 - [27.3.1 Data](#)
 - [27.3.2 Design](#)
 - [27.4 Conclusion](#)
 - [27.5 Acknowledgements](#)
- [28 Image Classification using k-means on TensorFlow fa18-523-68](#)
 - [28.1 Abstract](#)
 - [28.2 Introduction](#)
 - [28.3 Requirements](#)
 - [28.4 Design](#)
 - [28.5 Architecture](#)
 - [28.6 Dataset](#)
 - [28.7 Implementation](#)
 - [28.7.1 TensorFlow](#)
 - [28.8 Benchmark](#)
 - [28.9 Conclusion](#)
 - [28.10 Acknowledgement](#)
 - [28.11 Milestones and Time Plan](#)
 - [28.12 10/12 - 10/18:](#)
 - [28.13 10/19 - 10/25:](#)
 - [28.14 10/26 - 11/08:](#)
 - [28.15 11/09 - 11/26:](#)
- [29 Big Data Application in Recommender Systems fa18-523-70](#)
 - [29.1 Introduction](#)
 - [29.2 What is a recommender system?](#)

29.3 How does the recommender system work?

29.3.1 Collection of Data

29.3.2 Storing the data:

29.3.3 Analyzing the data:

29.3.4 Filtering the data:

29.4 Types of recommender systems

29.5 Algorithms

29.5.1 K-Nearest Neighbors

29.5.2 Association Rules

29.5.3 Matrix Factorization

29.5.4 Deep Neural Networks

29.6 Evaluation of recommender systems

29.6.1 Validation of Recommender System

29.6.2 Root mean squared error

29.6.3 Top N Recommendations

29.7 Acknowledgement

30 Credit Card Defaulter Analysis fa18-523-71 fa18-523-59

30.0.1 Keywords: TBD

30.0.2 Data

30.0.3 Questions

30.0.4 Timeline

30.0.5 References

31 Big Data in Education fa18-523-73, fa18-523-74

31.1 Abstract

31.2 Forms of Big Data in Education institution

31.2.1 Administrative Data

31.3 The Value of Systematic, Real-Time Data

31.4 Learning Analytics

31.5 Requirements

31.5.1 Prediction

31.5.2 Clustering

31.5.3 Relationship Mining

31.6 Architecture

31.6.1 Data Management

31.7 Implementation

31.7.1 Factors to Consider Before Implementing the Big Data

31.7.2 Steps of Successful implementation of the Big Data

31.8 Benchmarking

31.8.1 Planning

31.8.2 Generating Data

31.8.3 Generating Tests

31.8.4 Execution

31.9 Conclusion

32 Big Data in Education fa18-523-73, fa18-523-74

32.1 Abstract

32.2 Introduction

32.3 Forms of Big Data in Education institution

32.3.1 Administrative Data

32.3.2 Learning Process Data

32.4 The Value of Systematic, Real-Time Data

32.5 Learning Analytics

32.6 Requirements

32.6.1 Prediction

32.6.2 Clustering

32.6.3 Relationship Mining

32.7 Architecture

32.7.1 Data Management

32.8 Implementation

32.8.1 Factors to Consider Before Implementing the Big Data

32.8.2 Steps of Successful implementation of the Big Data

32.9 Benchmarking

32.9.1 Planning

32.9.2 Generating Data

32.9.3 Generating Tests

32.9.4 Execution

32.10 Conclusion

33 Utilizing Machine Learning and Visualization Tools to Analyze Breast Tumor Measurements fa18-523-80

33.1 Abstract

33.2 Keywords

33.3 Introduction

33.3.1 Dataset

33.3.2 Xcyt

33.4 Implementation

[33.4.1 Requirements](#)

[33.5 Implementation](#)

[33.6 Results](#)

[33.6.1 Visualizaton](#)

[33.6.2 K-Means Clustering](#)

[33.6.3 Clustering Assessment](#)

[33.6.4 Benchmarks](#)

[33.7 Limitations](#)

[33.8 Conclusion](#)

[34 Analysis of Global Commodity Statistics using PySpark and Watson Analytics fa18-523-81, fa18-523-79, fa18-523-82](#)

[34.1 Abstract](#)

[34.2 Keywords](#)

[34.3 Introduction](#)

[34.4 Data](#)

[34.5 Related work](#)

[34.6 Visualizations](#)

[34.7 Technologies to be used](#)

[34.8 Summary](#)

[34.9 Future Work](#)

[34.10 Acknowledgement](#)

[35 Credit Scoring Algorithm and Its Implementation in Production Environment fa18-523-83](#)

[35.1 Abstract](#)

[35.2 Keywords](#)

[35.3 Introduction](#)

[35.4 Design](#)

[35.4.1 Dataset](#)

[35.4.2 Data Visualization](#)

[35.4.3 Data Preperation](#)

[35.4.4 Model Training](#)

[35.4.5 Result Comparison](#)

[35.5 Implementation](#)

[35.5.1 Technologies Used](#)

[35.5.2 Prerequisites](#)

[35.5.3 Project Code Structure and Components](#)

[35.5.4 Code Running Instruction](#)

35.6 Results

35.6.1 Deployment Benchmarks

35.6.2 Application Benchmarks

35.7 Limitations

35.8 Conclusion

35.9 Acknowledgements

35.10 References

36 An Exploration of the Internet of Things fa18-523-84

37 Analytics in Consumer Behaviors in Black Friday with TensorFlow

fa18-523-85

37.1 Abstract

37.2 Keywords

37.3 Introduction

37.4 Data Description

37.5 Technologies

37.6 Algorithms and Methodology

37.6.1 knn

37.6.2 Naive Bayes Classifier

37.7 Data Analytics

37.7.1 Data Cleaning

37.7.2 Data Exploration and Processing

37.7.3 Data Analysis

37.7.4 Data Visualization

37.8 Conclusion

37.9 Limitations

37.10 Reference

38 Topic: Big data in SAP Ariba fa18-523-86

38.1 Abstract

38.2 Introduction

38.3 Architecture

38.4 Implementation

38.5 Conclusion

39 OCR Extraction Implementation with Tesseract fa18-523-88

39.1 Abstract

39.2 Introduction

39.3 Overview of Optical Character Recognition

39.4 Context Based Extraction Engine

- [39.4.1 Image Thresholding](#)
- [39.4.2 OCR Process](#)
- [39.4.3 Transform HOCR Data](#)
- [39.4.4 Define Candidates](#)
- [39.4.5 Set Context](#)
- [39.4.6 Group Context](#)
- [39.4.7 Score Context](#)
- [39.4.8 Output Result](#)
- [39.5 Example](#)
- [39.6 Tools and Technology](#)
 - [39.6.1 Terresact](#)
 - [39.6.2 Beautiful Soup](#)
 - [39.6.3 FuzzyWuzzy](#)
 - [39.6.4 Python](#)
 - [39.6.5 Numpy](#)
 - [39.6.6 OpenCV](#)
 - [39.6.7 Python Imaging Library](#)
 - [39.6.8 Tkinter](#)
- [39.7 Conclusion](#)
- [39.8 Acknowledgement](#)
- [40 Automation on Drug Interactions Profiling](#) fa18-423-06 fa18-423-03 fa18-423-02 fa18-423-05
 - [40.1 Introduction](#)
 - [40.2 Dataset](#)
 - [40.3 Implementation](#)
 - [40.3.1 parser.py](#)
 - [40.3.2 project.py](#)
 - [40.3.3 query.py](#)
 - [40.3.4 Amazon Web Services \(AWS\) EC2 Usage](#)
 - [40.3.5 Microsoft Azure Cloud Shell Usage](#)
 - [40.4 Results](#)
 - [40.5 Conclusion](#)
 - [40.6 Work Breakdown](#)
- [41 Secchi Disk Visibility Recognition](#)
 - [41.1 Abstract](#)
 - [41.2 Introductions](#)
 - [41.3 Data Collection](#)

- [41.4 Data Flow](#)
- [41.5 Worker Machine](#)
- [41.6 Approach: Use CNN to Determine the Visibility of Secchi Disk](#)
- [41.7 Training of Convolutional Neural Network Model](#)
- [41.8 Tape Measurement Optical Character Recognition\(OCR\)](#)
- [42 Scalable Microservices to Label yelp images using Kuberenets](#)
 - [sp18-616-02](#)
 - [42.1 Introduction](#)
 - [42.2 Docker](#)
 - [42.3 Kubernetes](#)
 - [42.4 Google Cloud Platform](#)
 - [42.5 Google API](#)
 - [42.5.1 Cloud Pub/Sub API](#)
 - [42.5.2 Cloud Vision API](#)
 - [42.6 Redis](#)
 - [42.7 Yelp Dataset](#)
 - [42.8 REST API](#)
 - [42.9 Approach](#)
 - [42.9.1 Initial Setup](#)
 - [42.9.2 Frontend Microservices](#)
 - [42.9.3 Backend Microservice](#)
 - [42.9.4 Mainapp Microservice](#)
 - [42.9.5 Pods, Services, Deployments](#)
 - [42.10 Execution](#)
 - [42.11 Challenges](#)
 - [42.12 Results](#)
 - [42.13 Benchmark](#)
 - [42.14 Conclusion](#)
 - [42.15 Future Scope](#)
 - [42.16 Acknowledgement](#)
- [43 CMD5 Plugin to Create a Docker Swarm Cluster on 3 Raspberry PIs](#)
 - [43.1 Abstract](#)
 - [43.2 Introduction](#)
 - [43.2.1 Docker: Swarm mode, Current Use, Installation and Configuration](#)
 - [43.2.2 Benefits of using Docker](#)

43.2.3 Docker - Services:

43.3 Creating CloudMesh plug-ins

43.4 Raspberry Pi as Platform

43.4.1 Differences between Laptop and a Pi

43.5 Docker and Big Data Platform

43.6 Docker Critique

43.7 Methods: Proposed Solution

43.7.1 Hardware

43.7.2 Raspberry Pi

43.7.3 Micro SD Cards

43.7.4 Micro USB Cables

43.7.5 External monitor

43.7.6 Initial input devices

43.7.7 Software

43.7.8 Docker

43.7.9 Raspbian Installed

43.7.10 Update OS repositories

43.7.11 Remote access setup

43.7.12 Changing hostnames

43.7.13 Steps Followed

43.8 Installing and configuring Docker Swarm

43.8.1 Manager

43.9 Workers

43.10 Additional Research

43.10.1 Other functions considered

43.10.2 Final code

43.11 Other options considered

43.12 Conclusions

43.13 Work Breakdown

44 New Approaches to Managing Metadata at Scale in Research

Libraries hid-sp18-705

44.1 Abstract

44.2 Introduction

44.3 New Approaches to Metadata Management

44.4 Blockchains for Research Libraries

44.5 Design Requirements

44.6 Scope

[44.7 BigchainDB](#)

[44.7.1 Evolution](#)

[44.7.2 Benchmark](#)

[44.7.3 Architecture](#)

[44.8 Dataset](#)

[44.9 Implementation](#)

[44.9.1 Use Case](#)

[44.9.2 Installation](#)

[44.9.3 Data Management](#)

[44.9.4 Role-Based Access Control in BigchainDB](#)

[44.10 Conclusion](#)

[44.11 Acknowledgment](#)

[Refernces](#)

1 PROJECTS LOCATED ELSEWHERE

Please put here your links in the following format

1.1 CM4 PROJECT CONTRIBUTORS

- fa18-516-24, Sachith Withana
- fa18-516-26, Vafa Andalibi
- fa18-516-12, Yu Luo
- fa18-516-25, Chun Sheng Wu
- fa18-516-04, David Demeulenaere
- fa18-516-10, Rui Li

2 SAMPLE PROJECT REPORT

FA18-523-000 FA18-523-

001

Gregor von Laszewski, Albert Zweistein
laszewski@gmail.com, zwei@example.edu
Indiana University, Example University
hid: fa18-523-000 fa18-523-001
github: [cloud](#)
code: [cloud](#)

If you do a project report only without any code,
remove the line with the code url link. Remove this line
also ;-)

Keywords: Cloud, Example

2.1 ABSTRACT

TBD

2.2 INTRODUCTION

One possible way of structuring the document. We may have to tweak this example as we progress.

Make sure paragraphs are 80 chars wide

Place images in an images directory

Use empty lines before and after headings

In [1] we can find a sample report.

Naturally the headings are just suggestions and you may change

them as appropriate for your project.

2.3 REQUIREMENTS

2.3.1 Images

Image locations that start with http are not allowed. All images must be in an images folder within your directory.

All images must be referred to in the text. The words bellow and above must not be used in your paper for images, tables, and code. For code you could use

- | In the previous
- | In the following
- | As explained next
- | We install the softwere with

BUT DO NOT USE THEM FOR IMAGES, ANY IMAGE MUST HAVE A MEANINGFUL CAPTION AS SHOWN IN OUR EXAMPLE

Comment: Above and bellow in a paper means you ask the reader to look at their shoes or the ceiling .

Figure 1 shows a nice figure exported from Powerpoint to png. If you like you can use this as a basis for your drawings.



Figure 1: A simple flow chart

Figures must not be cited with an explicit number, but automated numbering must be used. Here is how we did it for this paper:

```
+@fig:fromonetothether shows a nice  
figure exported from Powerpoint to png.  
If you like you can use this as a basis  
for your drawings.
```

```
![A simple flow chart](images/from-one-to-the-other.png){#fig:fromonetothether}
```

If the paper is copied from another source you MUST use a citation in the caption.



Figure 2: A simple flow chart [1]

This is done as follows

```
![A simple flow chart [@vonLaszewski-fa18-sample-report]](images/from-one-to-the-other.png)  
{#fig:fromonetothethertwo}
```

2.3.2 Stylistic Issues

Your report must not include the phrases

- In this term project we show
- In this project we describe
- In this chapter we have
- In this report we do

or similar

Instead you use

- We show
- We describe
- We have

- We do

You will not use the word `I` but instead you will use `we` or third person.

2.3.3 Copy from this document

In case you want to copy part of this document you need to do this from raw mode

- <https://raw.githubusercontent.com/cloudmesh-community/proceedings-fa18/master/project-report/report.md>

HOWEVER, YOU MUST NOT COPY THIS EXAMPLE IN YOUR REPORTS. IF YOU DO YOU MUST CHANGE THE IMAGE REFERENCES. THIS IS LOGICAL AS ALL IMAGE REFERENCES MUST BE UNIQUE.

2.4 DESIGN

2.5 ARCHITECTURE

2.6 DATASET

2.7 IMPLEMENTATION

2.8 BENCHMARK

2.9 CONCLUSION

2.10 ACKNOWLEDGEMENT

2.11 WORKBREAKDOWN

Only needed if you work in a group.

Jonathan Branam
jobranam@iu.edu
Indiana University
hid: fa18-516-03

github: [cloudpi](#)

code: [cloudpi](#)

code: [cloudpi](#)

Keywords: Cloud, Cluster, Raspberry Pi, Kubernetes

3.1 ABSTRACT

The Raspberry Pi offers a unique platform for developing and exploring cluster computing in a small form factor and for a minimal cost. A single Pi computer can be purchased for around \$35 USD and a large cluster can therefore be built with minimal investment. In this project I improved the existing support for the `cm-burn` tool to setup an SD card for the Pi and also installed and ran experiments on Kubernetes on the Raspberry Pi. I also created a new set of tools to help setup a Raspberry Pi for use in cluster computing.

3.2 INTRODUCTION

3.3 REQUIREMENTS

A network of 4-5 Raspberry Pi computers is recommended to fully experiment with the cluster setup and running different pods across the Kubernetes cluster. We devote one of the Pis to serve as the Kubernetes master node and so a minimum of three other Pis makes the experiment interesting. In addition to the Raspberry Pis themselves, you will need a power supply and networking cables for

our preferred setup. A small 5- or 8-port network hub is also used for connecting the Pis together and joining a local network. You may also wish to purchase or build individual cases or a cluster case for the Pis in order to handle them more conveniently.

In order to use the `cm-burn` tool you must use an operating system that supports the ext file system. Currently only Linux has native support for ext, so if you are using Windows or macOS then you must purchase a 3rd-party ext file system driver. The most commonly used driver is provided by [Paragon Software](#) and is \$20 USD for the Windows version and \$40 USD for the macOS version.

3.4 DESIGN

This project is designed to test the setup and performance of a Kubernetes cluster running on a set of Raspberry Pi tiny computers. The Raspberry Pis run a 64-bit ARM processor at 1.2GHz with 1GB RAM. They have builtin micro SD Card, USB, Ethernet, Bluetooth, WiFi, and HDMI connections. The operating system and storage for the Pi is loaded onto a micro SD Card which limits the overall system capacity. However, using the USB connections a larger external hard drive can be attached to improve their storage capacity.

The design of this cluster is to install the Kubernetes packages onto each of five Raspberry Pi computers and additionally to setup one of them as the cluster master. The other four Pis will then connect to the cluster master to receive their commands. Each computer in a Kubernetes cluster is called a node and each running application is called a pod. Each pod will always be deployed on the same node, that is, on the same computer. A pod may be comprised of one or several containers that are closely related or share a resource such as disk access. An example of a pod might be a database instance or an application instance. A database and application container should usually never be combined into a single pod but rather made as two pods so that they can be hosted on different nodes (computers) to more efficiently utilize resources. Also, the application instances can then be scaled when the database does not need to be.

As part of this project several different pods will be started on the Kubernetes cluster to demonstrate how to start and stop different services. These pods will use the Docker container system which is standard with Kubernetes, although other container solutions are supported.

3.5 ARCHITECTURE

3.6 DATASET

3.7 IMPLEMENTATION

To test the Kubernetes performance on the Raspberry Pis I installed a Python function through OpenFaas. I followed the instructions to [deploy OpenFaas on Kubernetes](#). Since Helm/Tiller don't seem to be well supported on ARM yet, I used the YAML setup option to deploy OpenFaas on Kubernetes.

```
$ git clone https://github.com/openfaas/faas-netes
$ sudo kubectl apply -f faas-netes/namespaces.yml
$ sudo kubectl apply -f faas-netes/yaml_armhf/
$ sudo kubectl get nodes -n openfaas
```

Once the system is running and all nodes are Ready we can deploy some of the functions.

```
$ # ? git clone https://github.com/openfaas/faas-cli
$ curl -sL https://cli.openfaas.com | sudo sh
$ git clone https://github.com/openfaas/faas
$ cd faas
$ faas-cli deploy -f stack_arm.yml -g 10.0.0.101:31112
$ echo -n "Test" | faas-cli invoke echoit -g http://127.0.0.1:31112
```

From my computer:

```
$ ssh -L 8080:127.0.0.1:31112 pi@blue00
```

Then I can open <http://127.0.0.1:8080/ui/> to view the OpenFaas Portal.

3.8 BENCHMARK

Download, load, and process the NYC Green Taxi data from 2017.
Available here

https://s3.amazonaws.com/nyc-tlc/trip+data/green_tripdata_2014-01.csv

...

https://s3.amazonaws.com/nyc-tlc/trip+data/green_tripdata_2014-12.csv

3.9 CONCLUSION

3.10 ACKNOWLEDGEMENT

4 TBD: FA18-516-04

github: [cloud](#)

5 CREATING A RESTFUL API SERVICE USING MONGODB AND AZURE SQL DATABASE FA18-516-06

Paul Filliman
pfillima@iu.edu
Indiana University
hid: fa18-516-06
github: [blue user icon](#)

Learning Objectives

- Create a first REST API
 - Retrieve data from a public API
 - Use MongoDB and Azure SQL as DBaaS
-

Keywords: Azure SQL, MongoDB, Atlas, Swagger, OpenWeatherMap, REST, DBaaS

5.1 ABSTRACT

For the final project, a RESTful API service was created using weather data through the OpenWeatherMap API that retrieves current weather information and stores this information to a persisted data store in either a MongoDB Atlas database or a Microsoft Azure SQL Server database. A user can use this API service in one step to retrieve and store real-time weather data to a personal persisted storage connection.

The goals for this project are to gain experience with creating a RESTful API and use sample data retrieved from a public API to use with MongoDB database as well as with a relational Azure SQL database.

5.2 INTRODUCTION

For this project, we create a REST API service using a swagger yaml file to define our service, server side components to implement the service, creation of a cloud-based MongoDB database using MongoDB Atlas, creation of an Azure SQL Database, and a client side component to call the service through a web URL call.

The data used in this project is current weather data from the public API OpenWeatherMap. Using weather data is simple for the users to understand and provide a real-time data source. The yaml file is structured to retrieve data from the OpenWeatherMap API and the server components use this data to insert into either a MongoDB or an Azure SQL persisted data store. The client user can specify connection strings to their own MongoDB and Azure SQL databases and specify through a URL which data store to which they can persist the data. The user can setup this service to run consistently to insert real-time weather data into their database. Once the user uses this API service to persist data, the user can use this data in analysis.

5.3 DESIGN

The outcome for this project is to build a REST API and have a client user call this API through a URL and have real-time data inserted into either a cloud-based MongoDB database or a cloud-based Azure SQL database. The user will also see the results in a JSON format of the data being inserted into either of the persisted data stores.

As seen in Figure 1, this project can be separated into the client resources and the API server resources. The API server is central to this project. Within the API server components, we have three source files, a Swagger 2.0 yaml file used to outline the sample weather data pull, a server source file in Python to outline the GET/POST routines, and a secondary Python source file to implement the GET/POST routines.

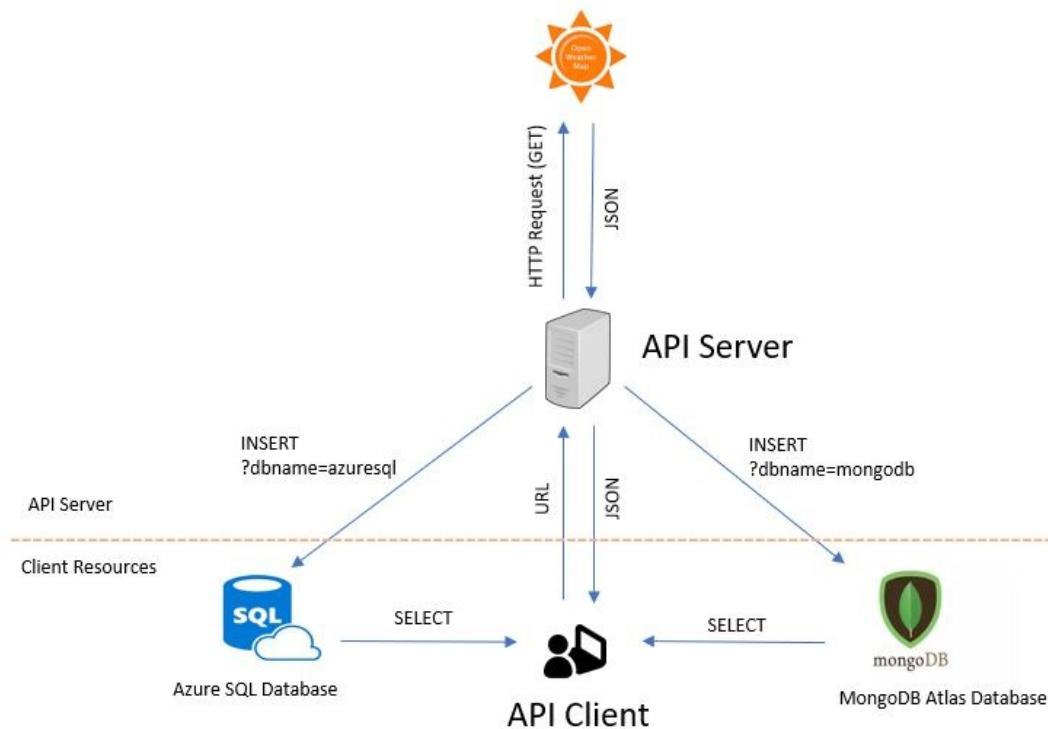
On the API client side, we need to setup a MongoDB database cluster

using the cloud-based MongoDB Atlas. We need to create a MongoDB database within this cluster and a collection within the database. We also need to setup an Azure SQL Database server, a database, and construct a table creation script to insert the sample data. The client will store connection credentials for these database servers into a config file.

The outcome will be for the client to send an HTTP request to the API server and to have the real-time data inserted into either MongoDB or Azure SQL using a URL parameter specifier.

5.4 ARCHITECTURE

The architecture diagram for this project is shown in Figure 1.



{#fig:This figure shows the project architecture diagram.}

The architecture is separated into five components.

5.4.0.1 API Server

The API server is built using swagger 2.0. The swagger.yaml file identifies which attributes of the weather data that will be received from the OpenWeatherMap API. The swagger.yaml file is used by the server.py source file which outlines the HTTP CRUD method requests. Additionally, a secondary source file, weatherapi.py details the methods used when calling the HTTP requests, parsing the parameters out of the client URL request, parsing the connection information to the databases from a config file, transforming raw weather data into decoded information, and determining the proper persisted data store.

5.4.0.2 API Client

The client uses a simple URL request in the format `http://localhost:5000/weather?dbname?mongodb` or `http://localhost:5000/weather?azuresql>` to retrieve information from the REST service. We are using localhost port 5000 in this project. The client will also need to provide connection information in a configuration file to the MongoDB and or Azure SQL databases to store information in their database(s).

5.4.0.3 MongoDB Database

The MongoDB database used in this project is the cloud-based MongoDB Atlas database. First a cluster needs to be setup by the user as well as a database in that cluster and a collection in that database to store the non-relational data from the REST service. The cluster used in this project is a free tier MongoDB Atlas M0 Instance cluster using version 4.0.4 on an AWS server. This cluster can provide up to 512 megabytes free data storage which will be within the limitations for this project fa18-516-06-MongoDB1.

A PFMongoDBCluster0

The screenshot shows the MongoDB interface for a database named 'WeatherDB'. Under the 'Collections' tab, the 'WeatherCollection' is selected. The collection has a size of 6.43KB and contains 14 documents. A query is run with the filter '{ "filter": "example" }', resulting in 14 query results. One result is expanded to show its structure:

```
_id: ObjectId("5bf77ce70f090c16b0340e2a")
coord: Object
  lon: -86.16
  lat: 39.77
weather: Array
  0: Object
    id: 701
    main: "Mist"
    description: "mist"
    icon: "50n"
    base: "stations"
  main: Object
    temp: 274.86
    pressure: 1025
    humidity: 95
    temp_min: 273.45
    temp_max: 275.95
    visibility: 8047
  Wind: Object
    speed: 3.1
    deg: 140
  clouds: Object
    all: 1
    dt: 1542944100
  sys: Object
    type: 1
    id: 1037
    message: 0.0083
```

{#fig:This figure shows the MongoDB DBaaS Interface.}

5.4.0.4 Azure SQL Database

Similar to creating the cloud-based MongoDB database, an Azure SQL Database can be used. For this project, a Basic DTU-based pricing tier Azure SQL Server database server is used. The Basic pricing tier has a max total data size of 2 gigabytes and is under 5 USD per month [fa18-516-06-AzureSQL1]. Once a database server is created, we need to create a database and a table to store the data. Within the table, we need to define type specific columns for character and numeric fields and also provide metadata columns for a primary key and creation dates.

The screenshot shows the Azure SQL Server DBaaS interface. On the left, there's a navigation sidebar with various options like Overview, Activity log, Tags, Diagnose and solve problems, Quick start, and Query editor (preview), which is currently selected. The main area has a search bar and several tabs: Login, Edit Data (Preview), New Query, Open query, Save query, and Feedback. A message box says "Showing limited object explorer here. For full capability please open SSDT." Below this is a "Tables" section listing SalesLT.SalesOrderHeader, dbo.RESTSample, SalesLT.ProductDescription, SalesLT.Address, SalesLT.CustomerAddress, SalesLT.SalesOrderDetail, SalesLT.Customer, SalesLT.ProductCategory, SalesLT.Product, and dbo.OpenWeatherAPI. The "dbo.OpenWeatherAPI" table is expanded, showing columns: ID (PK, int, not null), Time (datetime2, null), Location (varchar, null), Condition (varchar, null), ConditionDescription (varchar, n...), Temperature (numeric, null), Pressure (numeric, null), Humidity (numeric, null), WindSpeed (numeric, null), WindDegrees (numeric, null), and CreatedDate (datetime, not null). There are also rows for SalesLT.ProductModel, SalesLT.ProductModelProductDescri..., dbo.BuildVersion, dbo.ErrorLog, Views, and Stored Procedures. The "Query 1" tab is active, containing the query: "select * from dbo.OpenWeatherAPI". The results pane shows a table with 7 rows of weather data for Indianapolis. The columns are ID, TIME, LOCATION, CONDITION, CONDITIONDESCRIPTION, and TEMPERAT. The data includes entries for Clouds (broken clouds) at 31.49960, Mist (mist) at 38.15600, and Clouds (broken clouds) at 36.73400.

{#fig:This figure shows the Azure SQL Server DBaaS Interface.}

5.4.0.5 OpenWeatherMap API

The OpenWeatherMap public API is where we retrieve our current weather data. The format used to retrieve this is <http://api.openweathermap.org/data/2.5/weather?q=mycityname&appid=myappkey> [fa18-516-06-OpenWeatherMap1]. Again, For this project, there is a free or minimal cost subscription is used. With the free OpenWeatherMap subscription we can retrieve sixty or fewer api calls per minute [fa18-516-06-OpenWeatherMap2].

5.5 IMPLEMENTATION

The implementation to accomplish this project is broken up into source files in the following directories.

1. **/project-code/Makefile** This is the Makefile that is used to install the project Python requirement packages.
2. **/project-code/requirements.txt**
3. **/project-code/packages.txt** The requirements.txt and the

packages.txt files contain the Python packages needed to be installed prior to executing the project on a Linux Ubuntu 16.04 operating system.

4. **/project-code/etc/cloudmesh-weather.yaml** This yaml file contains the connection information attributes for connecting to the MongoDB Atlas cloud database and the Azure SQL cloud Database.
5. **/project-code/services/weather.yaml** This yaml file contains the structure of the data that will be retrieved from the OpenWeatherMap API service and is one of the main source files in this project. This yaml file does many things. First, it uses the swagger version 2.0 and maps the data retrieved to work with our REST API data structure. Secondly, this file contains the parameters for the client URL, for example the database type where the retrieved information will be persisted. Thirdly, this file references the operation identifier method that will be used to interface the retrieved information with the project API.
6. **/project-code/weather/server.py** The server file is the central part of this project. The server.py source file uses the Python package connexion and references weather.yaml in the creation of the connexion app api. This file is the parent file which needs to be run first before all client tasks can run successfully.
7. **/project-code/weather/weatherapi.py** This file is the largest source file and contains all of the detailed tasks that does many things including parsing security information from the connection file cloudmesh-weather.yaml, retrieves and transforms raw weather API data, creates an Azure SQL database table for inserts, parses the raw JSON data into separate columns, creates connection strings for Azure SQL and MongoDB, parses URL parameters for database type, and finally inserts into a MongoDB collection or an Azure SQL Database table.

5.6 STEPS TO REPRODUCE

The following are the steps to reproduce this project running on a Linux Ubuntu 16.04 Operating System using Python version 3.7.1.

1. Using the Makefile under /project-code, install the requirements using the command make install.

```
> cd cloudmesh/fa18-516-06/project-code  
> make install
```

2. If needing to use a separate connection for MongoDB or for Azure SQL Database other than the default connections, edit the cloudmesh-weather.yaml file under /project-code/etc/.
3. Under the directory /project-code/weather, execute the source file server.py with the command python server.py

```
> cd weather  
> python server.py
```

4. Run the client side opening a web browser with the URL <http://127.0.0.1:5000/weather>. From here we can see that the server side piece is running by seeing the current weather data in JSON format.
5. To insert data into the MongoDB specified in step 2, run the URL <http://127.0.0.1:5000/weather?dbname=mongodb>. After running this, we see the JSON data as we did in step 4, but also this has been inserted into our database and collection in MongoDB.
6. To insert data into the specified Azure SQL Database, run the URL <http://127.0.0.1:5000/weather?dbname=azuresql>. We see the weather information in the web browser as well as the subsequent insert into the Azure SQL database.

5.7 TECHNOLOGIES USED

There were many technologies used in this project. The source code was created using Python version 3 with multiple packages including flask, flask_restful, connexion, and pyodbc. There were two database as a service (DBaaS) services used, MongoDB Atlas and Microsoft Azure SQL Server. Finally, a REST service was used from OpenWeatherMap to import data into our REST service.

This project encompassed many different technologies working together. With our REST service at the center as shown in the architecture diagram, Figure 1, we take input from the client URL, retrieve API data from OpenWeatherMap, and import into one of two DBaaS.

5.8 RESULTS

Upon running the steps outlined above in Steps to Reproduce, there are two outcomes to test.

Insert data into MongoDB

With this first test, we want to see a JSON formatted file inserted into our MongoDB collection upon entering the URL <http://127.0.0.1:5000/weather?dbname=mongodb> from a client web browser.

Client

The screenshot shows a JSON viewer interface with the following details:

- URL: 127.0.0.1:5000/weather?dbname=mongodb
- JSON tab is selected.
- Raw Data and Headers tabs are available.
- Save, Copy, Collapse All, and Expand All buttons are present.
- JSON data structure:
 - base: "stations"
 - clouds:
 - all: 75
 - cod: 200
 - coord:
 - lat: 39.77
 - lon: -86.16
 - dt: 1543182900
 - id: 4259418
 - main:
 - humidity: 63
 - pressure: 1002
 - temp: 285.93
 - temp_max: 286.45
 - temp_min: 285.15
 - name: "Indianapolis"
 - sys:
 - country: "US"
 - id: 1035
 - message: 0.004
 - sunrise: 1543149686
 - sunset: 1543184532
 - type: 1
 - visibility: 16093
 - weather:
 - 0:
 - description: "thunderstorm"
 - icon: "11d"
 - id: 211
 - main: "Thunderstorm"
 - wind:
 - deg: 110
 - gust: 12.3
 - speed: 7.7

{#fig:This figure shows the URL using the parameter dbname=mongodb to insert into a collection in our MongoDB instance.}

MongoDB Result

The screenshot shows the MongoDB Atlas interface. On the left, there's a sidebar with a 'CONTEXT' dropdown set to 'Project 0'. Below it is a 'PROJECT' section with links to 'Clusters', 'Alerts', 'Backup', 'Users & Teams', 'Settings', 'Stitch Apps', 'Docs', and 'Support'. The main area displays a JSON document representing a weather forecast for Indianapolis. The JSON structure includes fields like '_id', 'coord', 'weather', 'base', 'main', 'wind', 'clouds', 'sys', and 'cod'. The 'weather' field contains a detailed description of a thunderstorm.

```
_id:ObjectId("5bfbb268d0f090c60232c2f46")
coord:Object
  lon:-86.16
  lat:39.77
weather:Array
  0:Object
    id:211
    main:"Thunderstorm"
    description:"thunderstorm"
    icon:"11d"
base:"stations"
main:Object
  temp:285.93
  pressure:1002
  humidity:63
  temp_min:285.15
  temp_max:286.45
  visibility:10003
wind:Object
  speed:7.7
  deg:110
  gust:12.3
clouds:Object
  all:75
  dt:1543182900
sys:Object
  type:1
  id:1035
  message:0.004
  country:"US"
  sunrise:1543149686
  sunset:1543184532
  id:4259418
  name:"Indianapolis"
  cod:200
```

{#fig:This figure shows results inserted into MongoDB using a URL parameter.}

We can see from figure 3 and figure 4 that the file returned in the result was committed to the MongoDB database.

Insert data into Azure SQL Database

With this second test, we again want to see the JSON formatted file inserted into our Azure SQL Server Database table upon entering the URL <http://127.0.0.1:5000/weather?dbname=azuresql> from a client web browser.

Client

The screenshot shows a JSON viewer interface with the URL `127.0.0.1:5000/weather?dbname=azuresql` in the address bar. The JSON data is displayed in a tree view:

```
base: "stations"
▼ clouds:
  all: 40
  cod: 200
▼ coord:
  lat: 39.77
  lon: -86.16
  dt: 1543185300
  id: 4259418
▼ main:
  humidity: 77
  pressure: 1002
  temp: 285.3
  temp_max: 286.45
  temp_min: 283.15
  name: "Indianapolis"
▼ sys:
  country: "US"
  id: 1037
  message: 0.0047
  sunrise: 1543149686
  sunset: 1543184532
  type: 1
  visibility: 16093
▼ weather:
  ▼ 0:
    description: "scattered clouds"
    icon: "03n"
    id: 802
    main: "Clouds"
▼ wind:
  deg: 110
  speed: 6.7
```

{#fig:This figure shows the URL using the parameter dbname=azuresql to insert into a table in our Azure SQL Database instance.}

Azure SQL Database Result

The screenshot shows the Microsoft Azure portal interface with the 'AdWorksLT - Query editor (preview)' tab selected. On the left, the database schema is visible, showing tables like SalesLT.SalesOrderHeader, SalesLT.CustomerAddress, and the newly created 'dbo.OpenWeatherAPI' table. The 'dbo.OpenWeatherAPI' table has columns: ID (PK, int, not null), Time (datetime2, null), Location (varchar, null), Condition (varchar, null), ConditionDescription (varchar, null), Temperature (numeric, null), Pressure (numeric, null), Humidity (numeric, null), WindSpeed (numeric, null), WindDegrees (numeric, null), and CreatedDate (datetime, not null). A query is run against this table:

```
1 select * from dbo.OpenWeatherAPI
```

The results show 16 rows of weather data for Indianapolis, spanning from November 25, 2018, to November 27, 2018. The data includes various weather conditions like 'Clouds', 'broken clouds', 'Clear', and 'scattered clouds', along with numerical values for temperature, pressure, and humidity.

| ID | Time | Location | Condition | ConditionDescription | Temperature | Pressure | Humidity | WindSpeed | WindDegrees | CreatedDate |
|----|-----------------------------|--------------|-----------|----------------------|-------------|------------|----------|-----------|-------------|-----------------------------|
| 27 | 2018-11-25T14:55:00.0000000 | Indianapolis | Clouds | broken clouds | 56.37200 | 1005.00000 | | | | 2018-11-25T14:55:00.0000000 |
| 28 | 2018-11-25T14:55:00.0000000 | Indianapolis | Clouds | broken clouds | 56.37200 | 1005.00000 | | | | 2018-11-25T14:55:00.0000000 |
| 29 | 2018-11-25T15:35:00.0000000 | Indianapolis | Clear | clear sky | 56.42600 | 1006.00000 | | | | 2018-11-25T15:35:00.0000000 |
| 30 | 2018-11-25T15:35:00.0000000 | Indianapolis | Clear | clear sky | 56.42600 | 1006.00000 | | | | 2018-11-25T15:35:00.0000000 |
| 31 | 2018-11-25T16:35:00.0000000 | Indianapolis | Clouds | broken clouds | 56.48000 | 1004.00000 | | | | 2018-11-25T16:35:00.0000000 |
| 32 | 2018-11-25T16:35:00.0000000 | Indianapolis | Clear | clear sky | 56.66000 | 1004.00000 | | | | 2018-11-25T16:35:00.0000000 |
| 33 | 2018-11-25T17:35:00.0000000 | Indianapolis | Clouds | scattered clouds | 53.87000 | 1002.00000 | | | | 2018-11-25T17:35:00.0000000 |

{#fig:This figure shows results inserted into Azure SQL Database using a URL parameter.}

We can see from figure 5 and figure 6 that the file returned in the result was committed as a row into the prior specified Azure SQL Server database table.

We see from these results that our REST API is working, retrieving real-time weather data through a web API and inserting these rows into either a MongoDB database or a Microsoft Azure SQL Server database.

5.9 CONCLUSION

The goal and outcome for this project is to gain familiarity and experience in creating a RESTful API and use sample data retrieved from a public API to use with MongoDB database as well as with a relational Azure SQL database. To get from input to output, there were several technologies that were used and these separate technologies were needed to communicate with each other.

There are many ways to enhance this REST service created with this project. For example, multiple cities data can be used, or create a city as a new URL parameter. We can also automate this process to

retrieve data every few seconds or every few minutes. Other technologies can be used with larger amounts of streaming data including Spark, Azure Databricks, and Azure Streaming Analytics. These technologies are higher cost options which could be used in corporate environments.

6 MANAGING AWS LAMBDA USING REST API FA18-516-08

Varun Joshi
vajoshi@iu.edu
Indiana University
hid: fa18-516-08
github: [github](#)

Learning Objectives

- Understand RESTful APIs and explore basic CRUD operations
- Learn about OpenAPI documentation using Swagger 2.0
- Apply REST understanding to Amazon's FaaS offering - AWS Lambda
- Learn AWS SDK for Python - boto3
- Use boto3 and Python Flask framework for building RESTful API with AWS Lambda as the resource

Keywords: FaaS, AWS Lambda,serverless,REST,OpenAPI example,Swagger

6.1 ABSTRACT

Amazon's Function as Service offering, AWS Lambda, provides serverless computing and eliminates the overhead of provisioning and managing servers. AWS Lambda can also integrate with other AWS services like S3 and Dynamo DB, extending its capabilities for building highly scalable applications. AWS Lambda can be triggered by other AWS resource events, HTTP endpoints, mobile applications etc. giving the flexibility for serverless computing. This project explores managing AWS Lambda by providing a solution for basic CRUD operations through REST API build using OpenAPI (Swagger 2.0)

specification.

6.2 INTRODUCTION

The goal of this project is to build a solution utilizing REST APIs to manage AWS Lambda service. The solution focuses on providing basic REST CRUD operations with AWS Lambda as resource. Python and Python flask framework is used for constructing REST service and AWS SDK for Python (boto3) is used for managing AWS Lambda.

6.3 ARCHITECTURE

The project is build of three components (see Figure 3).

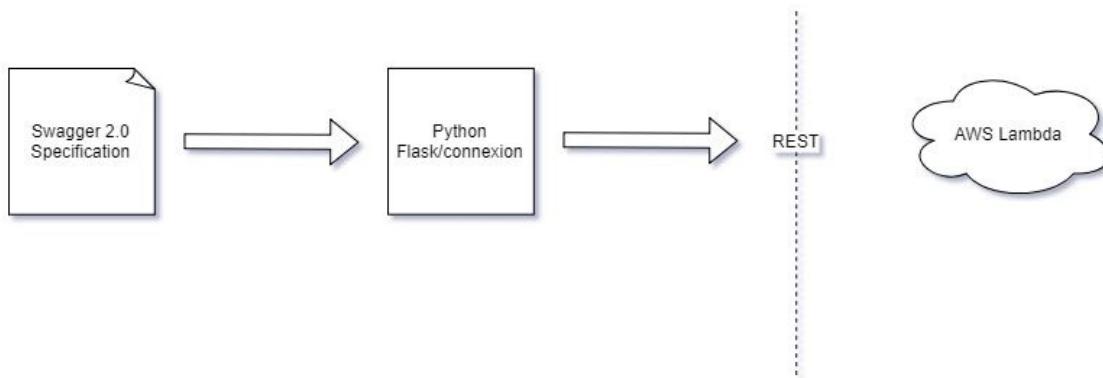


Figure 3: Project Architecture

- Swagger 2.0 is used for writing API specification. The specification describes endpoints for AWS Lambda CRUD operations and defines operation for each endpoint.
- Python flask framework consumes the OpenAPI specification and directs the endpoints to Python functions by building a RESTful app.
- AWS SDK for Python (boto3) is used to define Python functions which operate on endpoints and expose the resource, which is AWS Lambda , over REST API.

6.4 IMPLEMENTATION

Refer to the architecture (see Figure 3). I have enhanced the figure to include implementation details (see Figure 4).

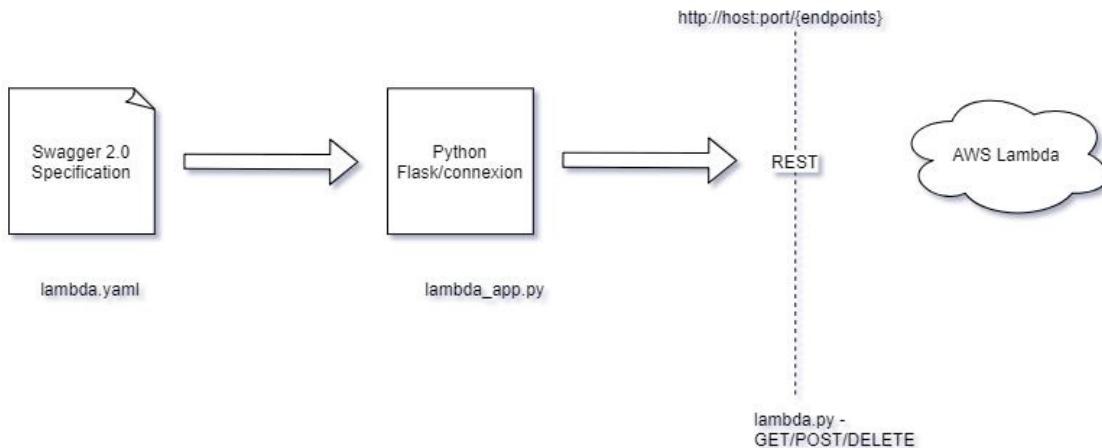


Figure 4: Project Architecture With Implementation

- **OpenAPI specification (Swagger 2.0) :**
 - **/project-code/lambda.yaml** : Defines the endpoints for CRUD operations on AWS lambda
- **Python flask framework :**
 - **/project-code/lambda_app.py** : Creates a connexion app which uses Flask under the hood. Consumes OpenAPI specification (Swagger 2.0) file lambda.yaml and routes the endpoints defined in the specification to the respective Python functions. The app is exposed as REST API running on local host and port 8080.
- **Python program for building endpoints :**
 - **/project-code/lambda.py** : Defines multiple Python functions utilizing boto3 which is AWS SDK for Python. Each function builds the endpoints to support REST operations.
- **Other files :**
 - **/project-code/create_lambda_basic_exec_role.py** : This Python program generates a basic AWS Lambda execution role which is attached to the newly created AWS Lambda function. To create an AWS Lambda function with an enhanced role such as accessing, the policy attribute in the program can be changed to the

desired policy.

- **/project-code/config.yaml** : This is a Python configuration file in YAML format. It holds reusable variables used across the Python programs in this project.
- **/project-code/requirements.txt** : File with Python package and libraries dependency to be installed using pip command.
- **/project-code/test/hello.zip** : Sample AWS Lambda deployment package zip file which holds a Python program for building simple AWS Lambda Function as a Service.
- **/project-code/test/fun.json** : Body parameters for REST POST operation for creating AWS Lambda. The body parameter consumes json content. The format is: { "cfile": "hello.py", "handler": "lambda_handler", "pkg": "hello.zip", "renv": "python3.7" } The key values are required and are explained as below:
 - **"cfile"** : The name of the code file for the application which is used by AWS Lambda for processing the functionality of the application. This project uses simple Hello World Python program as the code file.
 - **"handler"** : Name of the handler or function inside the code file which is invoked by AWS Lambda. In this project lambda_handler is the name of the Python function inside the code file hello.py.
 - **"pkg"** : The name of the zip file which has code file as well as all dependencies required for the code file. In this project this zip file is physically present in the directory /project-code/test.
 - **"renv"** : Runtime environment in AWS Lambda for the code file. This project uses Python 3.7 runtime environment provided by AWS Lambda.
- **/project-code/shell:**

- setup.sh : sets the project environment by cloning this GIT repository
- runAPI.sh : Runs the /project-code/lambda_app.py to start REST service
- testAPI.sh : basic GET/POST/DELETE curl commands to test REST service for managing AWS Lambda
- clean.sh : Deletes the project environment

6.5 TESTING AND RESULTS

In the /project-code directory, run the Python program lambda_app.py: python lambda_app.py

REST service will start on port 8080 (see Figure 5)

```
* Serving Flask app "lambda_app" (lazy loading)
* Environment: production
  WARNING: Do not use the development server in a production environment.
  Use a production WSGI server instead.
* Debug mode: on
* Running on http://0.0.0.0:8080/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 334-271-300
```

Figure 5: Start REST Service

Once the REST service has started , bring up the Swagger UI (see +????: SwaggerUI).curl can also be used in the command line to test the REST service)

Type http://0.0.0.0:8080/lambda/ui/ on a web browser to open Swagger UI:

The screenshot shows a web browser window with the URL `http://0.0.0.0:8080/lambda/ui/`. The page title is "swagger". The main heading is "REST API for AWS Lambda CRUD Operation". Below it, a sub-heading says "Create/Update/Delete AWS Lambda Functions using REST API". It was created by Varun Joshi at `vajoshi@iu.edu`, with Apache as the provider. There are two sections: "Deploy Package (Zip File) For Lambda" and "Lambda". Under "Lambda", there is a note "[BASE URL: /lambda , API VERSION: 0.0.1]". On the right side of the Lambda section, there are "Show/Hide", "List Operations", and "Expand Operations" buttons.

Figure 6: Open Swagger UI

For the LAMBDA tag in the Swagger UI, click “List Operations”. All available REST operations for AWS Lambda resource will be displayed (see Figure 7).

The screenshot shows the expanded REST operations for the AWS Lambda resource. The heading is "REST API for AWS Lambda CRUD Operation". Below it, a sub-heading says "Create/Update/Delete AWS Lambda Functions using REST API". It was created by Varun Joshi at `vajoshi@iu.edu`, with Apache as the provider. The "Lambda" section is expanded, showing the following operations:

| Method | Path | Description |
|--------|-------------------|---|
| DELETE | /function/{fname} | Delete AWS Lambda Function by name |
| GET | /function/{fname} | Get AWS Lambda function configuration by name |
| POST | /function/{fname} | Create AWS Lambda Function |
| PUT | /function/{fname} | Update AWS Lambda Function |
| GET | /functions | Get name of all AWS Lambda Functions |

[BASE URL: /lambda , API VERSION: 0.0.1]

Figure 7: Expand operations for TAG Lambda

Test each operation: * **GET all AWS Lambda functions:** Click on GET /function in the Swagger UI and then click “Try it out!” (see Figure 8).

Implementation Notes
lists all available Lambda Functions (limit 50)

Response Class (Status 200)
Lambda Functions

Model | Example Value

```
{
  "result": "string"
}
```

Response Content Type application/json ⚙️

Try it out!

Figure 8: GET All Lambda Functions

Result for GET functions (see +???).

Curl

```
curl -X GET --header 'Accept: application/json' 'http://0.0.0.0:8080/lambda/functions'
```

Request URL

```
http://0.0.0.0:8080/lambda/functions
```

Response Body

```
[
  {
    "CodeSha256": "0MZhXAbLbLVXxeSu3CCZcNqUtH7f+Fc1i27WFEz22Ho=",
    "CodeSize": 300,
    "Description": "",
    "FunctionArn": "arn:aws:lambda:us-east-2:249033861349:function:testDeploy1",
    "FunctionName": "testDeploy1",
    "Handler": "hello.lambda_handler",
    "LastModified": "2018-11-26T08:20:06.463+0000",
    "MemorySize": 128,
    "RevisionId": "a107d513-c312-4092-999f-193dd82440bb",
    "Role": "arn:aws:iam::249033861349:role/Lambda_Basic_Exec2",
    "Runtime": "python3.7",
    "Timeout": 3,
    "TracingConfig": {
      "Mode": "PassThrough"
    },
    "Version": "$LATEST"
  }
]
```

{#fig: GETFunctionsResult}

- **GET function by name:** Click on GET /function/{fname} in the Swagger UI and then type the function name to get and click

"Try it out!" (see +???: GETFunctionByName).

The screenshot shows the AWS Lambda function configuration interface for a GET request. At the top, there's a blue header bar with 'GET' and the URL '/function/{fname}'. To the right of the URL is the text 'Get AWS Lambda function configuration by name'. Below the header, there's a section titled 'Response Class (Status 200)' with a 'Function Search' button. Underneath, there are two tabs: 'Model' (selected) and 'Example Value'. The 'Model' tab contains a JSON snippet: { "result": "string" }. Below this, there's a 'Response Content Type' dropdown set to 'application/json'. A 'Parameters' table follows, with one row for 'fname' set to 'SearchText'. At the bottom left is a 'Try it out!' button.

Figure 9: GET Function by name

Result for GET function by name (see Figure 10).

The screenshot shows the AWS Lambda function configuration interface after a GET request has been made. It includes sections for 'Curl' (containing the command 'curl -X GET --header 'Accept: application/json' 'http://0.0.0.0:8080/lambda/function/SearchText''), 'Request URL' (containing 'http://0.0.0.0:8080/lambda/function/SearchText'), and 'Response Body'. The 'Response Body' section displays a large JSON object representing the Lambda function configuration, including fields like 'CodeSha256', 'CodeSize', 'Description', 'FunctionArn', 'FunctionName', 'Handler', 'LastModified', 'MemorySize', 'RevisionId', 'Role', 'Runtime', 'Timeout', 'TracingConfig', 'Version', and 'VpcConfig'.

Figure 10: GET function by name result

- **POST a new function:** Expand POST /function/{fname} , type in the new AWS Lambda function name to create and in the body parameter type in the json format for the required

values or click the json under “Example Value” to auto pouplate the json in the body parameter. Click “Try it out!” (see Figure 11).

The screenshot shows the AWS Lambda 'Create AWS Lambda Function' interface. At the top, there's a green header bar with 'POST /function/{fname}' and 'Create AWS Lambda Function'. Below this, the 'Response Class (Status 200)' is listed as 'Function Created'. There are two tabs: 'Model' (selected) and 'Example Value'. The 'Model' tab contains a JSON snippet:

```
{
  "result": "string"
}
```

The 'Parameters' section has two entries:

| Parameter | Value | Description | Parameter Type | Data Type |
|--------------|--|---|----------------|---|
| fname | NewFunc | Name for the AWS Lambda Function which will be created | path | string |
| fprop | <pre>{ "cfile": "hello.py", "handler": "lambda_handler", "pkg": "hello.zip", "renv": "python3.7" }</pre> | Required parameters to create AWS Lambda Function -Name of the file containing Lambda code (cfile),Runtime env (renv), AWS Lambda handler name (handler), zip file name of the AWS Lambda Deployment package(pkg) | body | Model Example Value <pre>{ "cfile": "hello.py", "handler": "lambda_handler", "pkg": "hello.zip", "renv": "python3.7" }</pre> |

Below the parameters, there's a note: 'Parameter content type: application/json' with a dropdown arrow.

At the bottom left is a 'Try it out!' button.

Figure 11: POST Function

Result for POST function (see Figure 12).

Curl

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json' -d '{ \
  "cfile": "hello.py", \
  "handler": "lambda_handler", \
  "pkg": "hello.zip", \
  "renv": "python3.7" \
}' 'http://0.0.0.0:8080/lambda/function/NewFunc'
```

Request URL

```
http://0.0.0.0:8080/lambda/function/NewFunc
```

Response Body

```
{
  "CodeSha256": "FjB80CwWt0cLe5LEjeKBPk8xqTs4iTHUIKcw/ASg56I=",
  "CodeSize": 300,
  "Description": "",
  "FunctionArn": "arn:aws:lambda:us-east-2:249033861349:function>NewFunc",
  "FunctionName": "NewFunc",
  "Handler": "hello.lambda_handler",
  "LastModified": "2018-11-28T03:21:09.708+0000",
  "MemorySize": 128,
  "RevisionId": "ab9e6773-580c-44b0-8fed-70db52ce6138",
  "Role": "arn:aws:iam::249033861349:role>Lambda_Basic_Exec5",
  "Runtime": "python3.7",
  "Timeout": 3,
  "TracingConfig": {
    "Mode": "PassThrough"
  },
  "Version": "$LATEST"
}
```

Figure 12: POST function result

- **Delete a function:** Expand DELETE /function/{fname} in the Swagger UI. Type in the function name to delete and then click “Try it out!” (see Figure 13).

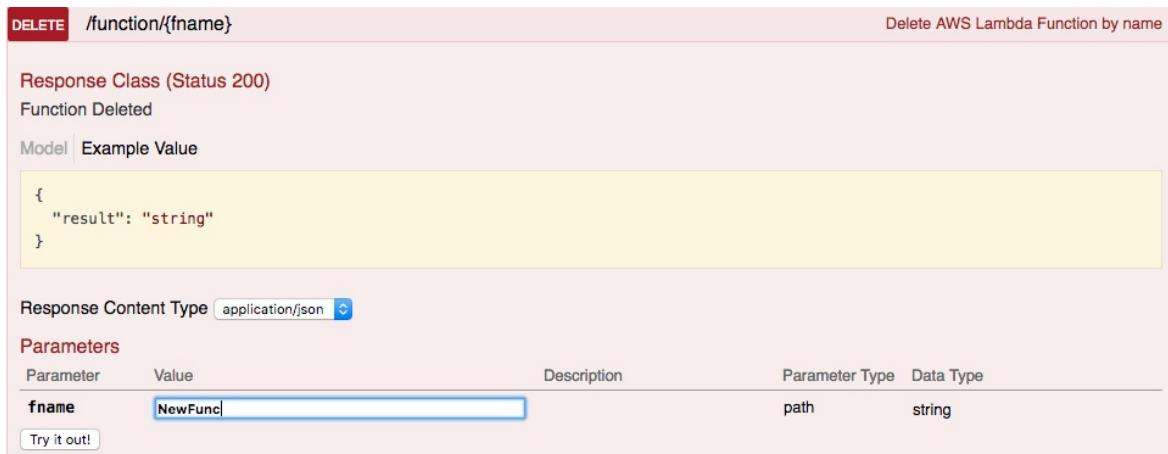


Figure 13: DELETE Function

Result for DELETE function (see Figure 14).

| Parameter | Value | Description | Parameter Type | Data Type | | | |
|--|---------|-------------|----------------|-----------|--|--|--|
| fname | NewFunc | | path | string | | | |
| Try it out! Hide Response | | | | | | | |
| Curl | | | | | | | |
| <pre>curl -X DELETE --header 'Accept: application/json' 'http://0.0.0.0:8080/lambda/function/NewFunc'</pre> | | | | | | | |
| Request URL | | | | | | | |
| <pre>http://0.0.0.0:8080/lambda/function/NewFunc</pre> | | | | | | | |
| Response Body | | | | | | | |
| <pre>"{Message : Requested Function Deleted}"</pre> | | | | | | | |
| Response Code | | | | | | | |
| <pre>200</pre> | | | | | | | |
| Response Headers | | | | | | | |
| <pre>{ "date": "Wed, 28 Nov 2018 03:27:03 GMT", "server": "Werkzeug/0.14.1 Python/3.7.0", "content-length": "41", "content-type": "application/json" }</pre> | | | | | | | |

Figure 14: DELETE Function Result

6.6 RECREATING PROJECT ENVIRONMENT

To recreate the project environment in any Ubuntu machine, use the **/project-code/shell/setup.sh**

The setup.sh has following commands:

- git clone https://github.com/cloudmesh-community/fa18-516-08
- cd fa18-516-08/project-code
- sudo pip install -r requirements.txt

Then, open config.yaml to update your AWS access key id and AWS secret access key for the user id which has permission to create AWS services and resources.

To start REST service use **/project-code/shell/runAPI.sh**

- sudo python lambda_app.py

Commands to test REST are provided in **/project-**

code/shell/testAPI.sh

6.7 TECHNOLOGIES USED

- **OS** : ubuntu 18.04
 - For project runtime environment and to host REST service.
- **Programming Language** : Python 3.7
 - For building REST service using Flask framework with connexion
 - Building functions for REST endpoints
- **Cloud Provider** : Amazon Web services
 - For exposing AWS Lambda, Amazon's FaaS offering , as REST endpoint for CRUD
 - boto3 - AWS SDK for Python
- **API Specification** : OpenAPI
 - Swagger 2.0 specification for building REST API
 - Swagger UI editor for testing REST operations

6.8 CONCLUSION

The goal of this project was to learn about OpenAPI documentation for writing REST API using Swagger 2.0 and applying this learning to explore REST CRUD operations for AWS Lambda Function as a Service. The end result of the project produced a nice OpenAPI (Swagger 2.0) documentation with AWS Lambda as endpoints. The endpoint operations were written using Python 3.7. The API documentation generated as part of this project can be reused to operate on Lambda endpoints utilizing any supported programming language and not just Python 3.7. The reusability of this project and demonstration of AWS Lambda exposed as REST endpoints is the biggest gain of this project learning exercise. The OpenAPI documentation can also be reused to expose more endpoints over REST which are similar to AWS Lambda such as AWS API Gateway, AWS S3 or AWS Dynamo DB.

6.9 ACKNOWLEDGEMENT

I would like to thank professor Gregor von Laszewski and associate instructors for the course ENGR 516 for providing their guidance and help in achieving the goal of this project learning.

7 EXPLORE OPENFAAS DEVELOPMENT AND DEPLOYMENT ASPECTS FA18-516-11 FA18-516-23

Murali Cheruvu, Anand Sriramulu
mcheruvu@iu.edu, asriram@iu.edu

Indiana University
hid: fa18-516-11 fa18-516-23

github: [cloudnative](#)
code: [cloudnative](#)

Keywords: OpenFaaS, Serverless, Micro-Services, Function-as-a-service (FaaS)

7.1 ABSTRACT

Explore creating micro-services using OpenFaaS that makes serverless functions simple for containers like Docker and Kubernetes. Integrate OpenFaaS serverless functions into public cloud offerings such as, AWS and Azure, to make them even better.

7.2 INTRODUCTION

Goals of this project is to learn how to create cloud-native micro-services using server-less concepts for better scalability and cheaper to maintain. Function-as-a-service (FaaS) methodology allows to decouple each functionality from the rest of the application, for better support, isolated deployment and scalability at each function level. We will use OpenFaaS, an open source alternative framework to develop and deploy micro-services as FaaS in cloud-agnostic way. OpenFaaS has an opiniated way of developing FaaS and deploying them to achieve the required scalability without much depending on the infrastructure of the public cloud offerings, using container concepts wrapped around our FaaS functions. In the context of

developing loosely couple components, we can interchangeably use micro-service for function-as-a-service (FaaS).

Let us introduce some of the key concepts that are related to the function-as-a-service.

7.2.1 Micro-Services

Micro-Services is a new paradigm in the software architecture to break down complex monolithic applications into more manageable and decoupled components that can be created and supported in silos. Loosely coupled components offer scalability and also we can use programming-of-choice based on the nature and complexity of the component, anywhere from advanced Object-Oriented Programming (OOP) languages like Java and C#.net, to modern functional-programming (FP) languages like Scala or Python. Deploying micro-services can be as flexible as deploying each individual functionality as a separate micro-service to grouping of related micro-services into one deployment package [2]. Micro-Services, targeting for web-based interfaces, can be implemented using simple REST-based API methodology.

7.2.2 API Gateways

Micro-Services offer flexibility and scalability but they bring complexity into the deployment and support. Too many micro-services can create confusion in discovering them and also, client applications may have to make multiple micro-service calls, hence create more network traffic even to populate a single webpage. As an example, Amazon uses lots of micro-services to display a typical product search result webpage. To reduce the network round-trips, it is advised, to create a gateway - API gateway, so that clients make unified calls to the gateway and all the related micro-services to fulfill a request will be made within the server and the results of these micro-services are bundled into one result, hence reduce the number of calls to make [3]. Cloud Offerings such as Microsoft Azure and AWS offer API Gateways with automatic API discovery and quota-based

usage along with lots of DevOp tools for continuous monitoring the scalability, performance and health-check of the micro-services. OpenFaaS has built-in API Gateway and integrates well with continuous monitoring tools such as **Grafana** and **Prometheus** [4].

7.2.3 Serverless

Serverless is the new methodology that cloud providers such as AWS, Azure or Google Cloud brought to simplify the deployment, execution and support of micro-services. Cloud providers take care the responsibility of the deployment, execution of the code and supporting - monitoring, tracking and notifying errors, auto-scaling (up or down) and performance metrics. Application providers are responsible to develop in a cloud-native fashion and leave the rest of the responsibilities to the cloud providers. AWS Lambda, Azure Functions, Google Functions are serverless offerings that are available today with high scalability turned on and cheaper to host such serverless functions [5].

7.2.4 OpenFaaS

OpenFaaS (Functions as a Service) is a framework for creating micro-services that can be hosted in containers like Docker or Kubernetes and make these services ready to be served in a serverless fashion [6]. OpenFaaS can easily deployed into all the popular public cloud providers such as AWS, Azure and Google Cloud.

![7](images/open-faas.png){#fig:OpenFaaS}

7.3 REQUIREMENTS

This project has two goals - (1) How to deploy machine learning algorithm to production and make it to work as serverless function to get best scalability and (2) Explore OpenFaaS and related tools to develop, test and deploy onto public cloud providers such as Azure, AWS and Google Cloud.

High level requirements include: setting up OpenFaaS locally within the development environment in Windows and also create deployable aspects: containerized OpenFaaS to be able to deploy to public cloud offerings including AWS, Azure and Google Cloud. Create python based project exploring high level concepts of serverless/micro-services for web. Explore container features in the process.

- Development Environment: Windows 10 Enterprise
- Install Docker Community Edition and Docker Swarm with single-node cluster
- Setup developer account with Docker Hub for publishing Docker Images on the internet
- Install OpenFaaS CLI - command line interface for OpenFaaS
- Deploy OpenFaaS into the development environment to make it ready to use
- Install Python and related libraries to make it ready for the actual project for machine learning algorithms to run and also write OpenFaaS functions in python.
- Write Python based code for object (image) detection using Convolved Deep Neural Network (CNN) algorithms
- Train the model using around 20,000 images of dogs and cats provided by one of Kaggle competition [8].
- create OpenFaaS function to predict the uploaded image - whether it is a dog or cat
- create another OpenFaaS function to detect the uploaded image as what animal using pre-trained model by Keras library.

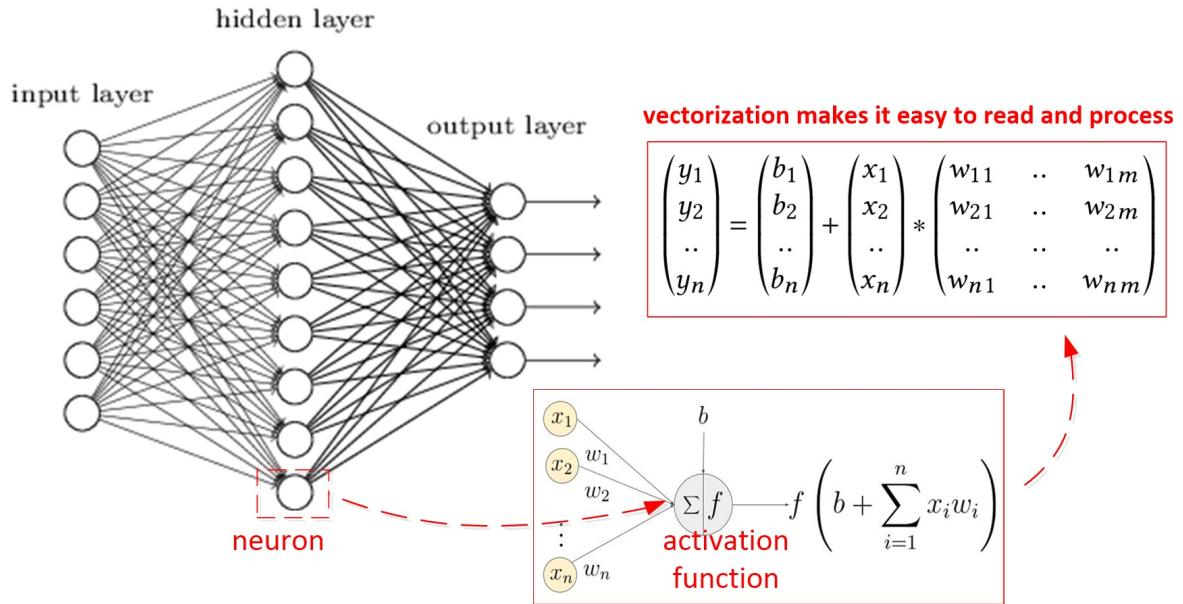
7.4 DESIGN

Create image object detection and classification model using Convolutional Neural Network by exploring very small number of training examples, from the dataset of 20,000 images of dogs and cats, using Python libraries such as Keras and Tensorflow. We will train the small neural network as a baseline and apply fine-tuning of an existing pre-trained network provided by popular VGGNet image

network [9]. We will provide some background on the concepts of neural network to understand the project domain.

7.4.1 Neural Network

Neural Network is modeled after the human brain, specifically the way it solves complex problems. Perceptron, the first generation neural network, created a simple mathematical model or a function, mimicking neuron - the basic unit of the brain, by taking several binary inputs and produced single binary output. Sigmoid Neuron improved learning by giving some weightage to the input based on importance of the corresponding input to the output so that tiny changes in the output due to the minor adjustments in the input weights (or biases) can be measured effectively. Neural Network is, a directed graph, organized by layers and layers are created by number of interconnected neurons (or nodes). Every neuron in a layer is connected with all the neurons from the previous layer; there will be no interaction of neurons within a layer. As shown in Figure 15, a typical Neural Network contains three layers: input (left), hidden (middle) and output (right) [10]. The middle layer is called hidden only because the neurons of this layer are neither the input nor the output. However, the actual processing happens in the hidden layer as the data passes through layer by layer, each neuron acts as an activation function to process the input. The performance of a Neural Network is measured using cost or error function and the dependent input weight variables. Forward-propagation and backpropagation are two techniques, neural network uses repeatedly until all the input variables are adjusted or calibrated to predict accurate output. During, forward-propagation, information moves in forward direction and passes through all the layers by applying certain weights to the input parameters. Back-propagation method minimizes the error in the weights by applying an algorithm called gradient descent at each iteration step.



An example of a neuron showing the input ($x_1 - x_n$), their corresponding weights ($w_1 - w_n$), a bias (b) and the activation function f applied to the weighted sum of the inputs.

Figure 15: Neural Network [10]

7.4.2 Deep Neural Network

Deep Learning is an advanced neural network, with multiple hidden layers (thousands or even more deep), that can work well with supervised (labeled) and unsupervised (unlabeled) datasets. Applications, such as speech, image and behavior patterns, having complex relationships in large-set of attributes, are best suited for Deep Learning Neural Networks. Deep Learning vectorizes the input and converts it into output vector space by decomposing complex geometric and polynomial equations into a series of simple transformations. These transformations go through neuron activation functions at each layer parameterized by input weights. For it to be effective, the cost function of the neural network must guarantee two mathematical properties: continuity and differentiability.

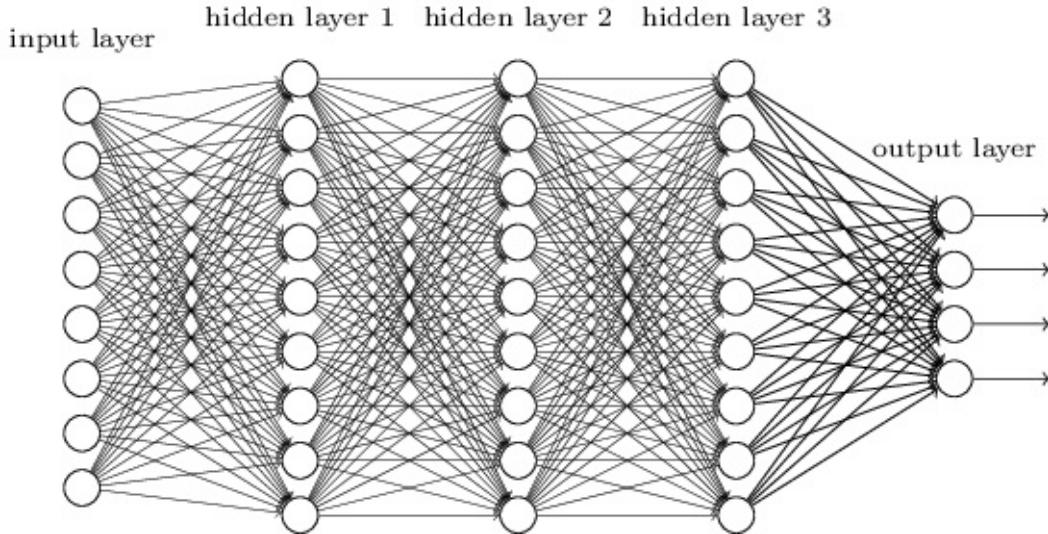


Figure 16: Deep Neural Network [10]

7.4.3 Convolutional Neural Network (CNN)

Convolutional Neural Network (CNN), also called multilayer perceptron (MLP), is a deep feedforward network, consists of (1) convolutional layers - to identify the features using weights and biases, followed by (2) fully connected layers - where each neuron is connected from all the neurons of previous layers - to provide nonlinearity, sub-sampling or max-pooling, performance and control data overfitting [11]. CNN is used in image and voice recognition applications by effectively using multiples copies of same neuron and reusing group of neurons in several places to make them modular. CNNs are constrained by fixed-size vectorized inputs and outputs.

Convolution Neural Network has two key components: (a) feature extraction - in this component, the network performs a series of convolutions (mathematical operation) and pooling operations to create the feature-maps, the list of features from the images. (b) classification - fully connected layers will serve as a classifier on top of these extracted features. They will assign probability for the object on the image being what the algorithm predicts it is.

- Softmax Layer: Classifier

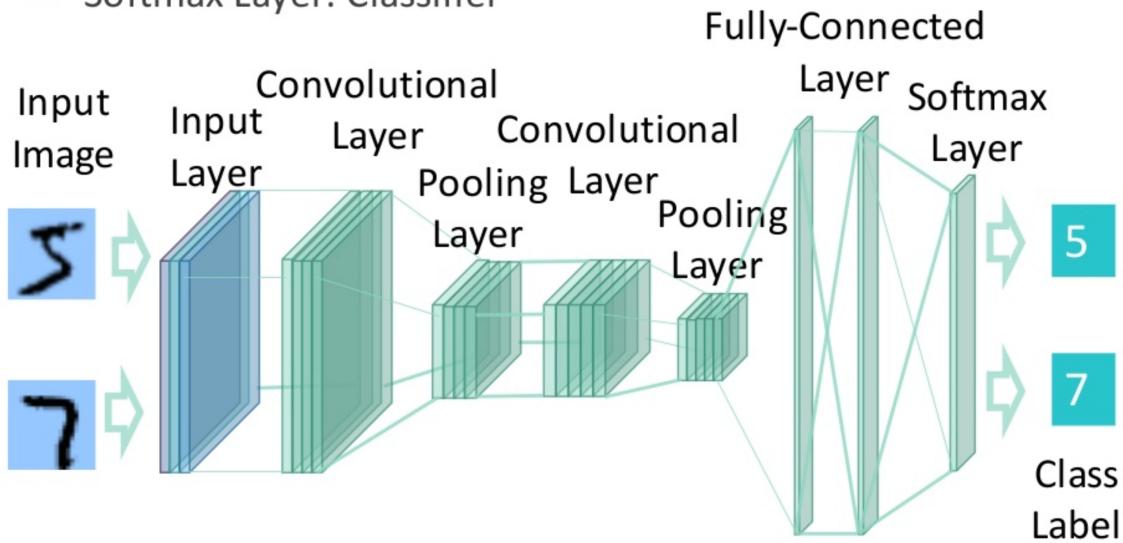


Figure 17: Convolutional Neural Network (CNN) [12]

7.5 ARCHITECTURE

Computer vision is an interdisciplinary field that deals with how computers can be made for gaining high-level understanding from digital images or videos. Images are treated as a matrix of pixel values. By applying convolutional, mathematical operation, features such as edges, brightness or blur can be extracted as feature maps from the images. This process goes through series of convolutions followed by pooling to reduce the size of the images for further processing. In the end, various features of the image are flattened into vector and create deep neural network to classify the image, in our case, whether it is a dog or cat. VGGNet is very popular image network that reduced the errors in the image classification and improved the processing performance from the predecessor image network models. We will create our based neural network model using a small sample dataset of 2000 images from the given 20,000 images and apply this on top of the pre-trained model that is optimized based on VGGNet algorithm. So that our effort is incremental and minimal.

7.6 DATASET

About 20,000 images of dogs and cats are provided part of the Kaggle competition. Dataset can be downloaded from [HERE](#).

7.7 IMPLEMENTATION

7.7.1 Install Docker Swarm (Single-Node Cluster), Docker and OpenFaaS

- **Prerequisites:** Windows 10 Professional or Enterprise Edition, open the command prompt in Administrator mode
- **Step 1:** Install Docker [Community Edition](#)
- **Step 2:** Install Git Bash for pulling the latest OpenFaaS artifacts and all the other software from GitHub
- **Step 3:** Run **docker swarm init** to set up the single-node docker swarm cluster
- **Step 4:** Create an account with Docker Hub, if the created docker images to be shared with others through internet
- **Step 5:** Run **docker login** to make sure docker is linked to your account
- **Step 6:** Download latest **faas-cli.exe** from [HERE](#)
- **Step 7:** Copy the **faas-cli.exe** to *C:folder to make it available for the command prompt. Or you will need to add the path of the faas-cli.exe into the system environment variables
- **Step 8:** Test the faas-cli using the command - **faas-cli version**
- **Step 9:** Clone the OpenFaaS artifacts from GitHub using : git clone https://github.com/openfaas/faas
- **Step 10:** Go into the **faas** folder that to checkout the git master repository - cd faas and git checkout master
- **Step 11:** Run **deploy_stack.sh -no-auth** to deploy the latest OpenFaaS into our environment
- **Step 12:** Run **docker service ls** to verify whether openfaas has been deployed to our environment

7.7.2 Trouble Shooting

If there are any issues with the docker and/or OpenFaaS functions,

we can reset the environment using the following commands

- **restart docker** - to restart the docker
- **docker stack rm func** - to remove all the functions
- **docker swarm leave -force** - to shutdown the docker cluster
- **docker swarm init** - to initialize docker swarm cluster
- **{open_faas.github.folder}/deploy_stack.sh** - to pull the latest code from openfaas GitHub

7.7.3 Build and deploy a serverless OpenFaaS function

7.7.3.1 Get FaaS-CLI

```
curl -sSL https://cli.openfaas.com | sudo sh
```

7.7.3.2 Build, deploy and push to Docker Hub

```
$ cd fa18-516-11/project-code
$ docker build -t faas-ressnet .
$ faas-cli deploy --image faas-ressnet --name faas-ressnet
$ docker tag faas-ressnet $anandid/faas-ressnet
$ docker push $anandid/faas-ressnet
```

7.7.3.3 Testing OpenFaaS function

7.7.3.3.1 Test Request : 1



faas - OpenFaas - tiger

```
Input:
curl -X POST -H \
--data-binary @data/tiger.png \
"http://127.0.0.1:8080/function/faas-resnet"
```

Output:

```
Predicted: [('n02129604', 'tiger', 0.92411584), ('n02123159', 'tiger_cat', 0.04635064),
('n02391049', 'zebra', 0.017654872)]
```

7.7.3.3.2 Test Request : 2



faas - OpenFaas - tiger

```
Input:  
curl -X POST -H \  
--data-binary @data/tiger.png \  
"http://127.0.0.1:8080/function/faas-resnet"
```

Output:

```
Predicted: [('n02403003', 'ox', 0.55445725), ('n03868242', 'oxcart', 0.36393312),  
('n02109047', 'Great_Dane', 0.035532992)]
```

7.7.4 Deploying to AWS

7.7.4.1 Setup AWS Instance

1. Purchased Spot Instance for Ubuntu 16.04, and with instance type we'll use m4.xlarge
2. Enabled Security group allowing ports 22, 31112, and 6443 for ingress
3. Created a key-pair file, so that we can SSH in to the instance
4. Test the Instance

```
ssh -i "faas.pem" ubuntu@ec2-18-191-176-209.us-east-2.compute.amazonaws.com
```

7.7.4.2 Setting up Kubernetes on AWS

1. Prep the machine by installing some necessary components. Run the following commands to enter superuser mode, install some necessary components from this gist, then exit back into the ubuntu user.

```
$ sudo su
$ curl -sSL
https://gist.githubusercontent.com/ericstoeckl/1d4372e9398d9cec7ec028629b2c36e2/raw/6f03cf3...
| sh
exit
```

2. Deploy Kubernetes

```
$ sudo kubeadm init --kubernetes-version stable-1.8
```

3. Networking layer for the cluster, to allow inter-pod communication

```
$ kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=$(kubectl version | base64 | tr -d '\n')"
```

4. To Allow container placement on the master node and confirm the cluster is running

```
$ kubectl taint nodes --all node-role.kubernetes.io/master-
$ kubectl get all -n kube-system
```

7.7.4.3 Deploying OpenFaas on Kuberetes using faas-netes

1. Clone the node, Deploy the Whole Stack and deploy OpenFaas

```
$ git clone https://github.com/openfaas/faas-netes
$ kubectl apply -f https://raw.githubusercontent.com/openfaas/faas-
netes/master/namespaces.yml
$ cd faas-netes && \
kubectl apply -f ./yaml
```

2. Install the CLI, deploy samples

```
$ curl -sL https://cli.openfaas.com | sudo sh
$ git clone https://github.com/openfaas/faas-cli
```

3. Pull docker image (OpenFaas functions)

```
docker pull anandid:faas-resnet
```

4. Deploy OpenFaas Functions

```
faas-cli deploy --image anandid/faas-resnet --name faas-resnet --gateway
http://18.191.176.209:31112
```

5. Test OpenFaas function

```
curl http://18.191.176.209:31112/function/faas-resnet --data-binary @data/tiger.png
```

7.7.4.4 Install Python Libraries

Following are the steps used to install Python libraries:

- Install latest Anaconda for Windows 64-bit for Python 3
- Once Anaconda installed, use the Anaconda command prompt, to install Tensorflow and Keras
- We can add Python exe folder into windows system environment path variable, so that we can use python from the regular command prompt
- We can install Jupyter Notebook as well, given we are using Jupyter notebook to write the CNN algorithm for the image classification
- Detailed steps are provided in the installation/deployment instructions

7.7.5 Project Files

| File | Description |
|------------|--|
| readme.md | Instructions on how to deploy and use OpenFaaS Serverless Functions. |
| Dockerfile | Docker Image file with our OpenFaaS function and all the python dependencies that can be deployed onto typical public cloud providers: Azure, AWS, Google Cloud. |
| | OpenFaaS serverless function folder with related files to classify the uploaded animal |

| | |
|------------------------------------|---|
| resnet_pretrained_classify | image using ResNet image network pre-trained model using Keras and Tensorflow libraries. The classification happens very quick, hence qualifies to be a serverless function. |
| index.py | Entry python file to call in the docker image. |
| image_classifier_dogsandcats.ipynb | Jupyter notebook with detailed analysis and exploration of Convolutional Neural Network to train the model using about 20,000 images of dogs and cats. The saved model will be used in the OpenFaaS function to test the classification of the uploaded image. Python uses Keras and Tensorflow Neural Network to perform the modeling. |
| classify_pre_trained_model.ipynb | Jupyter notebook to classify image of an animal using ResNet50 pre-trained model through Keras and Tensorflow. |

7.8 CONCLUSION

OpenFaaS facilitates clean design, development, deployment and

support of the function-as-a-service (micro-services) implementations. OpenFaaS creates co-opetitive (co-operation and competition) environment with public cloud providers. With all the needed built-in methodologies - API Gateway, FaaS, etc. and tools - security, logging, integrations with DevOp tools, etc., OpenFaaS is already a very good open source alternative for building micro-services and maturity of this framework is drastically increasing with growing usage community and adoption. OpenFaaS can be helpful to host variety of FaaS functions all the way from http-based functions to complex functions such as machine learning based predictions and classifications.

7.9 TEAM MEMBERS AND WORK BREAKDOWN

- Murali Cheruvu worked on the CNN, OpenFaaS function, Docker Image and Deploying Azure (Single-Node Cluster)
- Anand Sriramulu worked on OpenFaaS function, Docker Image and Deploying to Raspberry Pi (Multi-Node Cluster)

7.10 ACKNOWLEDGEMENT

The author would like to thank Dr. Gregor von Laszewski and the Teaching Assistants for their support and valuable suggestions. Author would also like to thank authors listed in the bibliography along with OpenFaas and the community of OpenFaaS for great collaboration and providing invaluable documentation and sample projects.

8 TBD

FA18-516-12

github: [cloud911](#)

9 HySDS ON KUBERNETES: ARIA InSAR PROCESSING ON XSEDE JETSTREAM FA18-516-14

Gerald Manipon, Gregor von Laszewski, Hook Hua

gmanipon@iu.edu, laszewski@gmail.com, hook.hua@jpl.nasa.gov

Indiana University, Indiana University, NASA Jet Propulsion Laboratory

hid: fa18-516-14

github: [cloud](#)

code: [cloud](#)

Keywords: E516, NASA, JPL, HySDS, Hybrid Cloud Science Data System, InSAR, Interferogram, Radar Interferometry, Sentinel-1, ISCE, InSAR Scientific Computing Environment, Kubernetes, OpenStack, Jetstream, Python, Terraform

9.1 ABSTRACT

Developed at NASA's Jet Propulsion Laboratory, the core of HySDS (Hybrid Cloud Science Data System) [13] has been running on AWS (Amazon Web Services) and OpenStack since 2011 generating science data products for projects such as ARIA (Advanced Rapid Imaging and Analysis) [14], GRFN (Getting Ready for NISAR) [15], and WVCC (Water Vapor Cloud Climatology) [16]. The upcoming NASA missions, SWOT (Surface Water and Ocean Topography) [17] and NISAR (NASA-ISRO SAR Mission) [18], are slated to launch in September 2021 and January 2022, respectively, and will use HySDS in AWS as the baseline science data system. To mitigate the risk of vendor lock-in and increase the capabilities of the system, HySDS now needs to fully function on the IaaS (Infrastructure as a Service) services provided by other public and private cloud vendors such as Google Cloud Platform (GCP), Microsoft Azure, FutureSystems, Jetstream, ChameleonCloud, High Performance Computing/High End Computing and other XSEDE compute resources. Multiple funded efforts are currently in progress at JPL and EOS (Earth Observatory of Singapore)

to add support for Microsoft Azure, Google Cloud Platform and the Pleiades Supercomputer however there is currently no plan to add support for NSF-funded research-based cloud resources such as those provided by XSEDE: Jetstream, ChameleonCloud and FutureSystems. In addition, HySDS was originally developed to operate at the IaaS layer however by adapting HySDS to run on Kubernetes, a container orchestration framework open-sourced by Google, portability to other cloud vendors can be vastly improved and simplified. HySDS can leverage Kubernetes as a PaaS (Platform as a Service) service that provides an abstraction layer to the vendor-specific IaaS services. In this project for the Fall 2018 E516 course at Indiana University at Bloomington, we pathfind a potential avenue for utilizing the NSF-funded XSEDE cloud resources by prototyping a real-life science use case: ARIA InSAR processing of Sentinel-1 SLC data running on HySDS on a Kubernetes cluster on IU's Jetstream Cloud. We found that by generalizing the way HySDS handles product generation executables (PGEs) and their inputs, HySDS can successfully leverage the IaaS abstraction provided by Kubernetes and can thus be provisioned onto any private or public cloud vendor capable of running Kubernetes. Further work however is needed to assess the additional overhead and complexity that Kubernetes adds to the end-to-end science data system as well as how well such a system performs and scales in a production environment.

9.2 BACKGROUND

HySDS was originally developed as the framework used to create the science data system (SDS) that "scalably powers the ingestion, metadata extraction, cataloging, high-volume data processing, and publication of the geodetic data products for the Advanced Rapid Imaging & Analysis (ARIA), Getting Ready for NISAR (GRFN), and Water Vapor Cloud Climatology (WVCC) projects at JPL" [13]. The role of the SDS is to process the raw level0 (L0) data downloaded by the ground data system (GDS) from the satellites to higher-order level1 (L1), level2 (L2) and level3 (L3) products which are more usable to end-users. The SDS then delivers these products to a distributed active

archive center (DAAC) for the purpose of providing public distribution, access, and discovery of these products. Figure 18 depicts the role of the SDS in the end-to-end system for the upcoming NISAR mission.

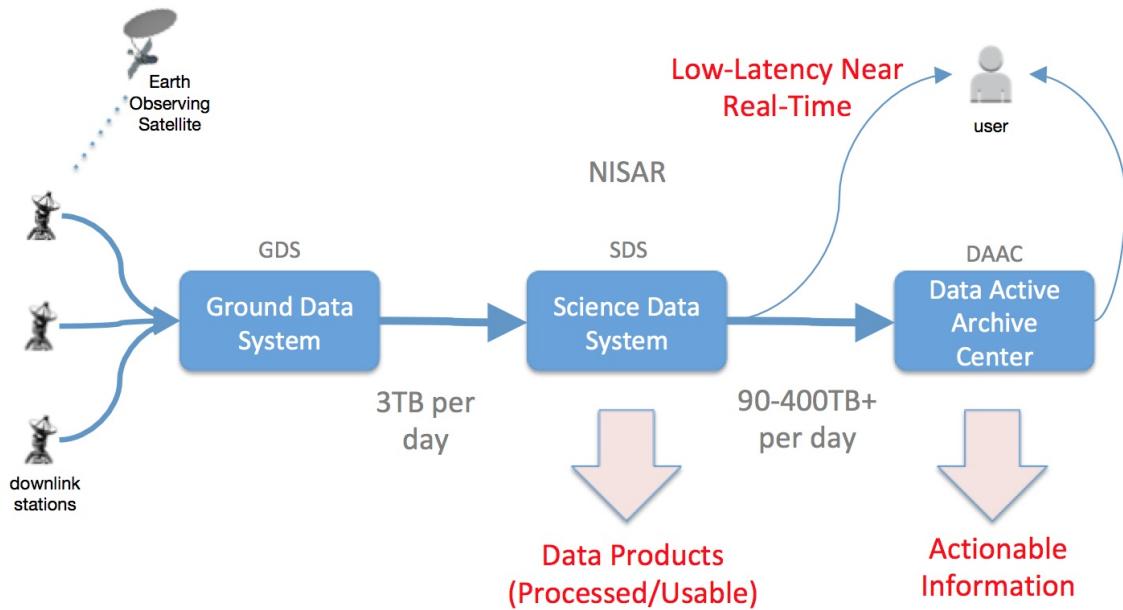


Figure 18: Large Data Flow

9.2.1 Architecture

The SDS itself is comprised of various components to orchestrate and facilitate the processing of the raw L0 satellite data. Figure 19 depicts the functional architecture of the SDS in the context of running in AWS.

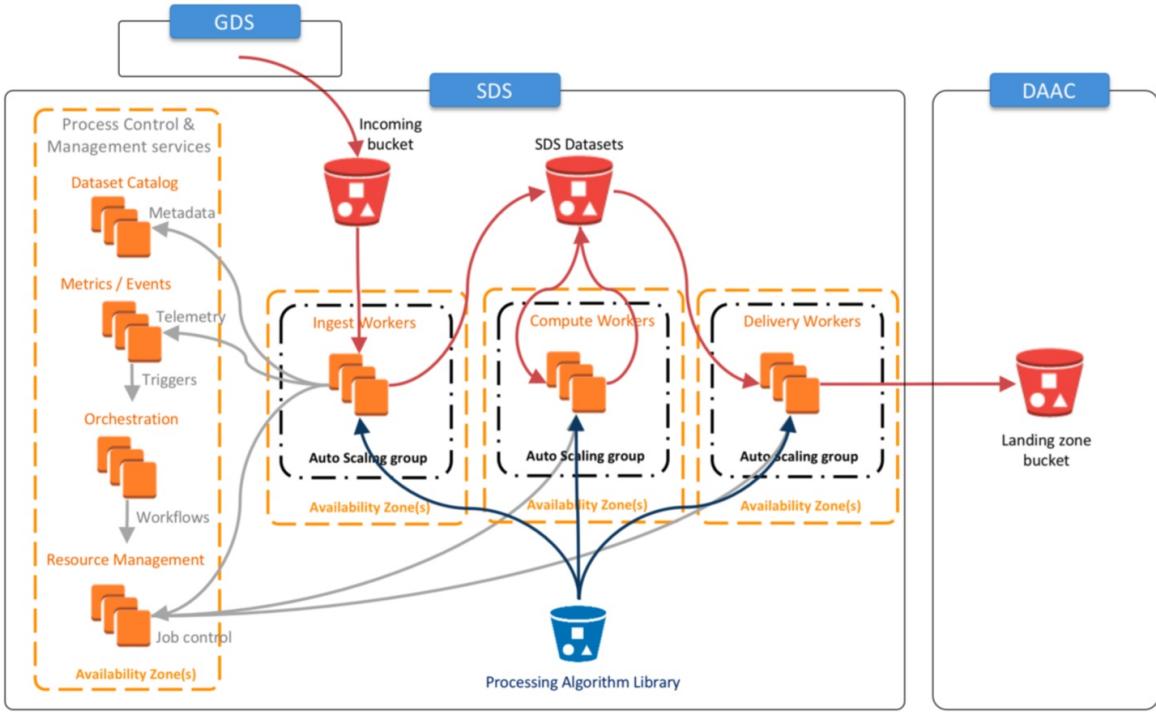


Figure 19: Functional Architecture

The GDS system will stage L0 data to the incoming bucket of the HySDS cluster. When an L0 dataset file is deposited, an event trigger submits an ingestion job to the resource manager which is then pulled by an autoscaling group of workers specifically provisioned for the purpose of dataset/product ingestion. During the ingestion job, the L0 data file is verified, metadata is extracted and browse images are generated. They are then copied to the SDS dataset bucket and the metadata/location of the L0 dataset is indexed into the dataset catalog. Upon indexing of the dataset into the catalog, trigger rules are evaluated and if their conditions are met, further downstream processing jobs are submitted to the resource manager. For example, the use-case we will be prototyping is the ARIA InSAR processing of Sentinel-1 SLCs into L2 interferograms. In the ARIA SDS, ingestion of an SLC triggers the Sentinel-1 interferogram workflow. This workflow, which runs on a different autoscaling group, determines if all the conditions have been met (e.g. pertinent ancillary orbit files, calibration files, digital elevation models exist) for processing an interferogram job. If so, it proceeds to generate the interferogram dataset which in turn gets ingested into the dataset bucket and

indexed into the dataset catalog. Finally, the ingestion of these higher-order products trigger an evaluation of a delivery rule which runs on yet a different autoscaling group to deliver the interferogram to the DAAC's bucket or to notify them that it can be picked up from the SDS's dataset bucket.

9.2.2 Processing Algorithm Library

Note also in Figure 19 that HySDS provides a processing algorithm library which serves as the catalog of all PGEs (product generation executors) that can be used by the worker fleet. Processing algorithms are usually developed by subject matter experts (e.g. radar scientists) and integrated into HySDS by installing them in container images (Docker, Singularity, rkt) along with interface documents needed by the rest of the system to submit and run jobs. The Processing Algorithm Library serves as the "App Store" for all scientific data processing that would be run on a particular HySDS cluster and provides a mechanism to import or export these PGEs for exchange.

9.2.3 Pedigree

The SDS architecture detailed in Figure 19 is NOT novel and is very much aligned with the architectural design patterns of most other cloud data processing systems out there (e.g. AWS Batch, GCP Dataflow, Azure Batch). In fact, other data processing systems take a more cloud-native, all-in approach to their implementation. The goal of HySDS from day-one was to be as vendor-neutral and cross-cloud interoperable as possible all the while taking advantage of any performance gains by utilizing vendor-specific cloud-native features. The value-added that HySDS provides is the experience, pedigree and lessons-learned from operating science data systems at high scale since 2011. One such example is the suite of HySDS watchdogs built into the system. These watchdog subcomponents of the resource manager are dispatched to monitor and mitigate any potential issues. For example, take an SDS that has 100 jobs submitted to the resource manager. There are 5 compute workers running and executing jobs. If

one of those workers had an issue, for example the root disk volume was mounted as read-only on boot-up, the job immediately fails. The compute worker then pulls the next job but that too immediately fails. Within seconds, the rogue compute worker has drained the job queue resulting in what is known as “job drain”. A highly-resilient data processing system would employ a job drain watchdog to detect this situation, determine that the rogue compute worker was not healthy, and shut down that worker immediately before more damage can be done.

Many of these lessons-learned have come from employing HySDS on forward processing and bulk reprocessing campaigns for various NASA missions and projects. For example, during a Pleiades maintenance downtime in 2015, the OCO-2 mission [19] needed to run some major bulk reprocessing but did not have enough resources on-premise to provide the results in time. They utilized HySDS on AWS to process 6 million soundings (1.6 TB output) in under a day by scaling out to 1000 compute workers. Figure 20 depicts the number EC2 instances running OCO-2 sounding PGEs over time.



Figure 20: OCO-2 Autoscaling

9.3 REQUIREMENTS

In this project, we propose to extend the capabilities of the open-source HySDS core components to support running on a Kubernetes

cluster provisioned on Indiana University's Jetstream Cloud, a compute resource provided through the XSEDE (Extreme Science and Engineering Discovery Environment) program. This requires that the HySDS system must be adapted to run process and control management nodes (PCM) and compute workers on a Kubernetes cluster. We will also adapt the ingestion, processing, and cataloguing of products generated through a scientific data pipeline running on the HySDS cluster to utilize cloud-specific but highly scalable features of the XSEDE resources. Once the HySDS Core system is completely functional on Kubernetes, research and flight mission adaptations can then be integrated and applied to provide real-world production use-cases. One such adaption is the ARIA SAR interferogram generation pipeline. The adapted HySDS/Kubernetes/Jetstream system would utilize JPL's ingested synthetic-aperture radar data, orbits, calibrations, and DEMs in order to run this pipeline to generate L2 interferogram products. Use cases for running this pipeline on XSEDE resources such as Jetstream include forward processing of level 2 SAR interferometry products as well as generating real-time urgent response products to aid in disaster response to earthquakes, hurricanes, volcanic eruptions and flooding. A sample forward-processing scenario will be conducted to assess the stability, efficiency, and capabilities of the HySDS/Kubernetes/Jetstream system. We will also compare and benchmark the execution of the ARIA interferogram pipeline on this system to the baseline performance of the ARIA HySDS system on AWS. A large number of jobs will be submitted to the system for a given amount of time. Afterwards, the results and observations will be documented to understand the system's performance.

9.4 DESIGN

Figure [21](#) provides a high level overview of the various infrastructure fabric that can be used by HySDS.

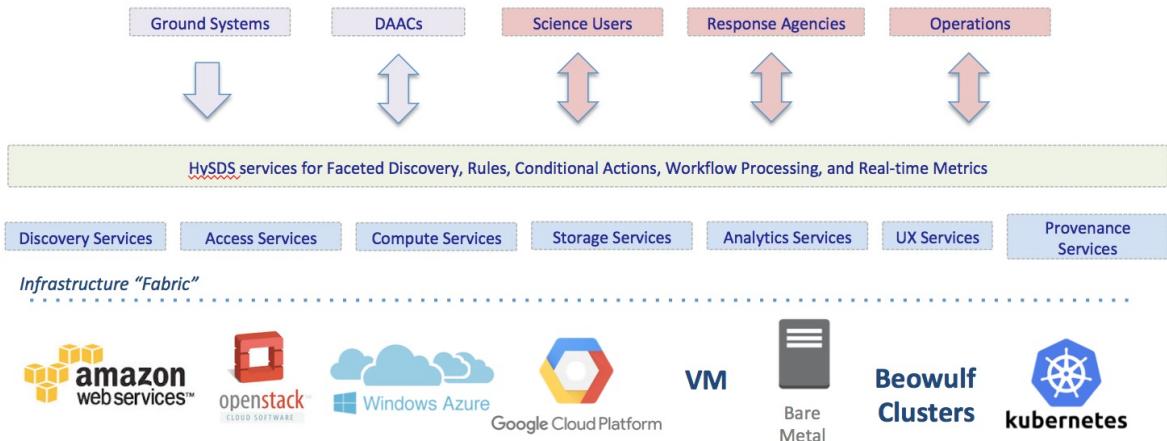


Figure 21: HySDS Infrastructure Fabric

In this project, we are adding support for HySDS to run on Kubernetes. To do this, we need to re-architect the HySDS components to conform to the way Kubernetes does things. More specifically, we need to decompose mozart (resource management), metrics (job and worker metrics/analysis), and grq (dataset catalog) components into Kubernetes pods and services.

9.5 ARCHITECTURE

9.5.1 Mozart

Figure [22](#) shows the architecture of the resource management component of HySDS, mozart.

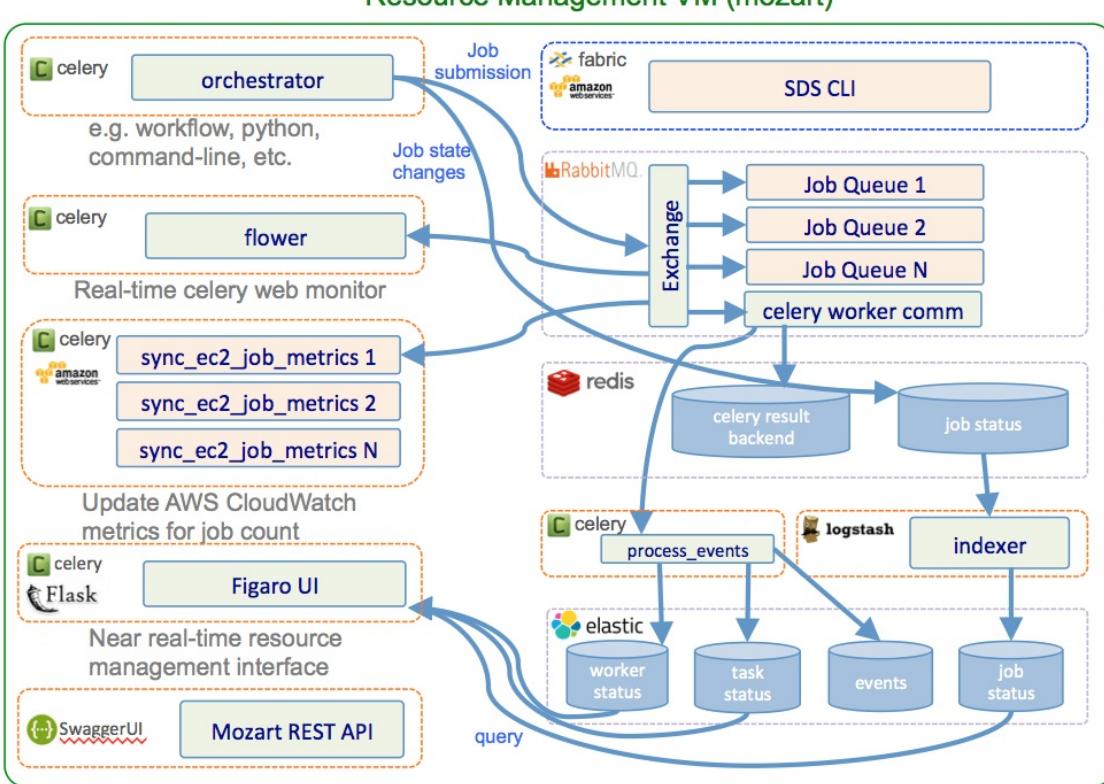


Figure 22: Mozart VM

To conform to Kubernetes best practices regarding microservices adaptation, we decided to decompose the components to the following Kubernetes pods:

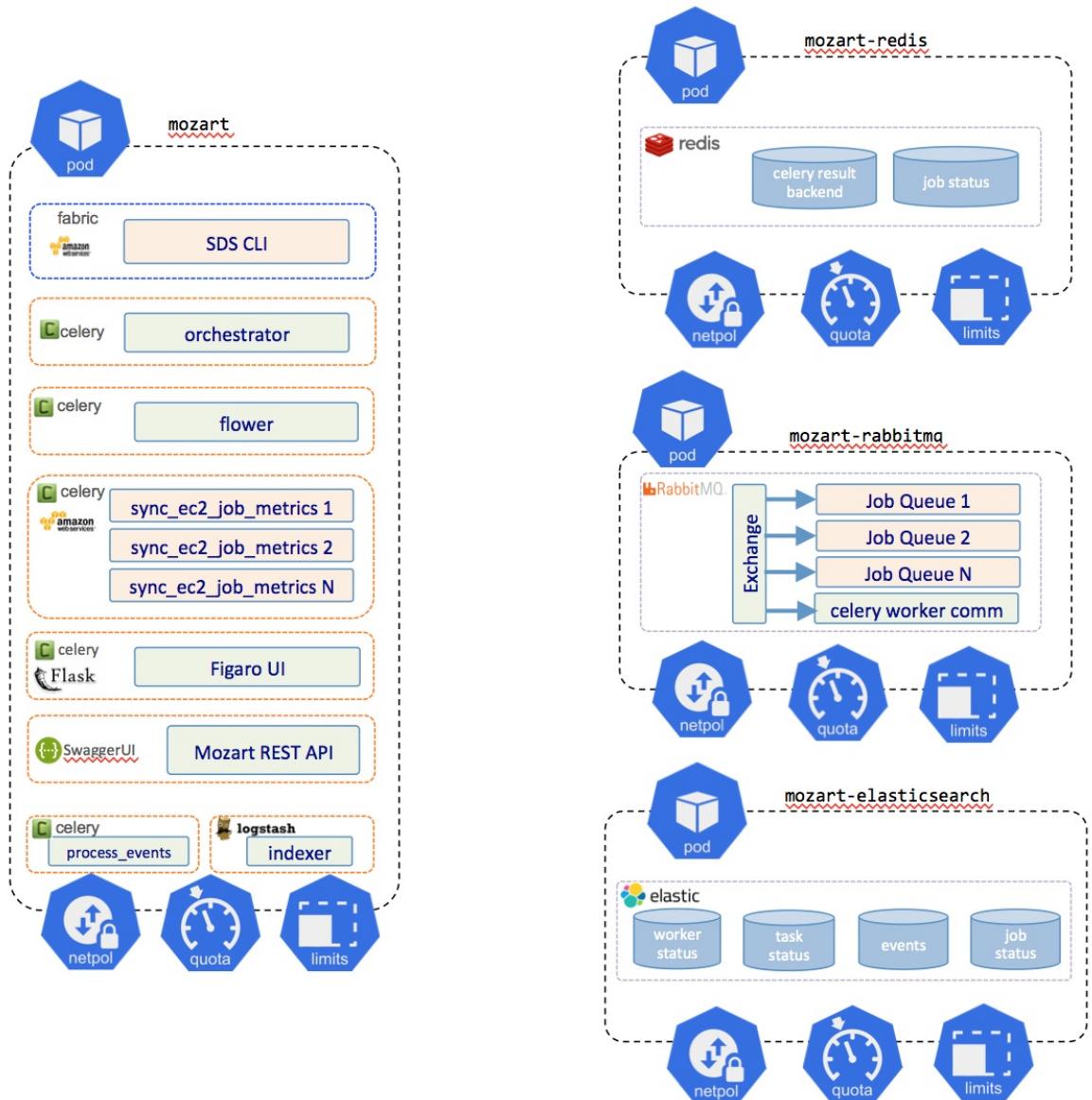


Figure 23: Mozart Pods/Services

9.5.2 Metrics

Similarly, Figure 24 shows the architecture of the metrics component of HySDS and its subsequent decomposition to Kubernetes pods:

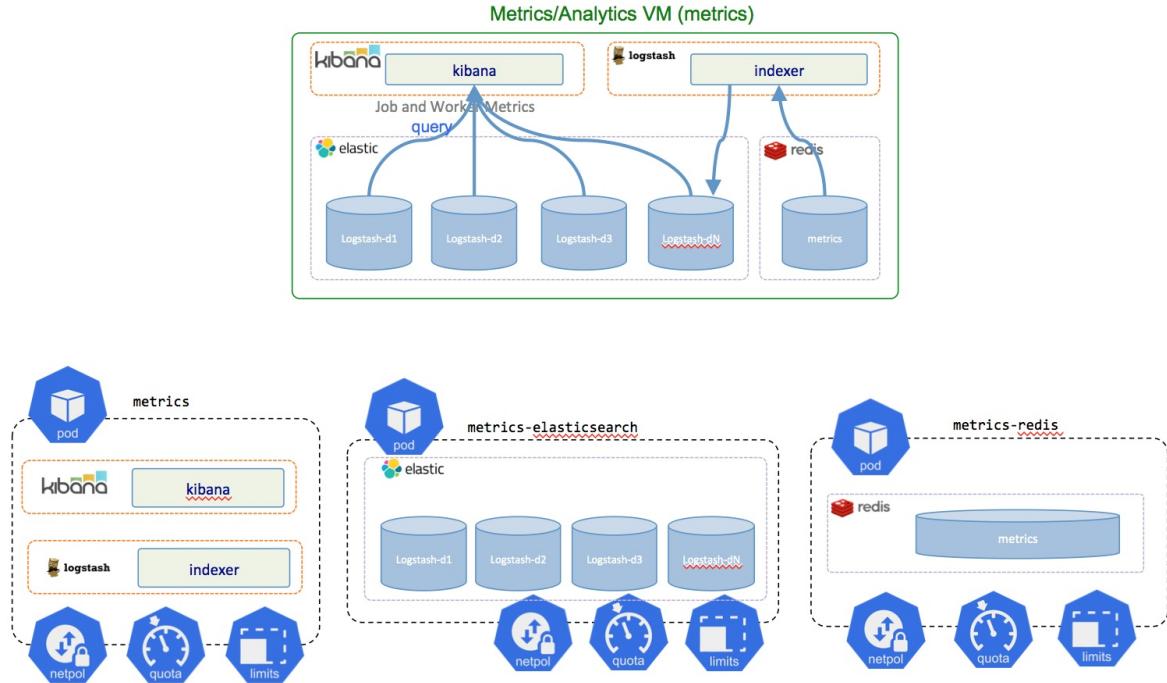


Figure 24: Metrics VM

9.5.3 GRQ

Finally, Figure 25 shows the architecture of the dataset catalog component of HySDS and its subsequent decomposition to Kubernetes pods:

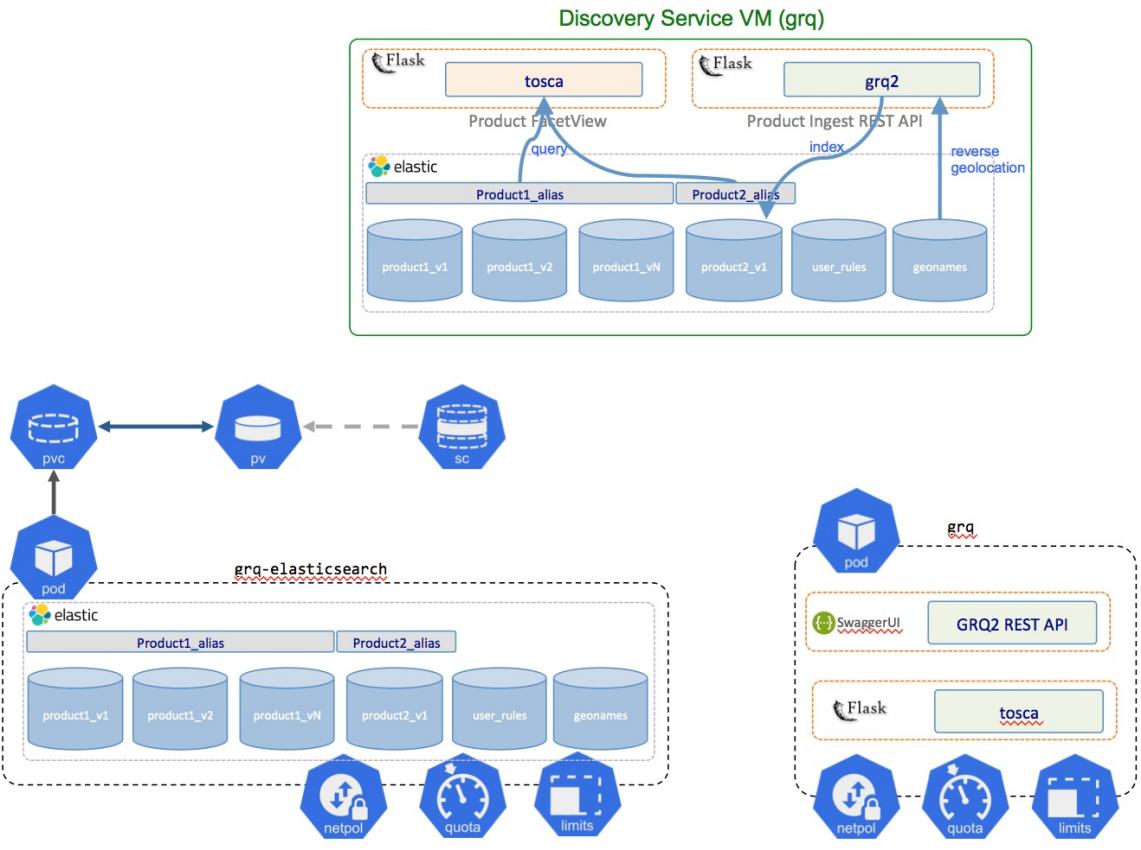


Figure 25: GRQ VM

9.5.4 hysds-k8s

The decomposition of the HySDS components to microservice pods was implemented in a new github repository located at <https://github.com/pymonger/hysds-k8s> on the “grfn-jetstream-iu” branch: <https://github.com/pymonger/hysds-k8s/tree/grfn-jetstream-iu>. To verify and validate the operation of the HySDS cluster running on Kubernetes, we exercised the “Hello World” and “Hello Dataset” tutorials located at <https://github.com/hysds/hysds-framework/wiki>Hello-World> and <https://github.com/hysds/hysds-framework/wiki>Hello-Dataset>, respectively.

9.6 DATASET

As stated before, once we've adapted HySDS to run on Kubernetes, we will run a real-world use case on the HySDS cluster: the ARIA

Sentinel-1 interferogram pipeline. Figure [26](#) gives an overview of the various pipelines that ingest input datasets needed by the Sentinel-1 interferogram pipeline.

S1-IFG Overview

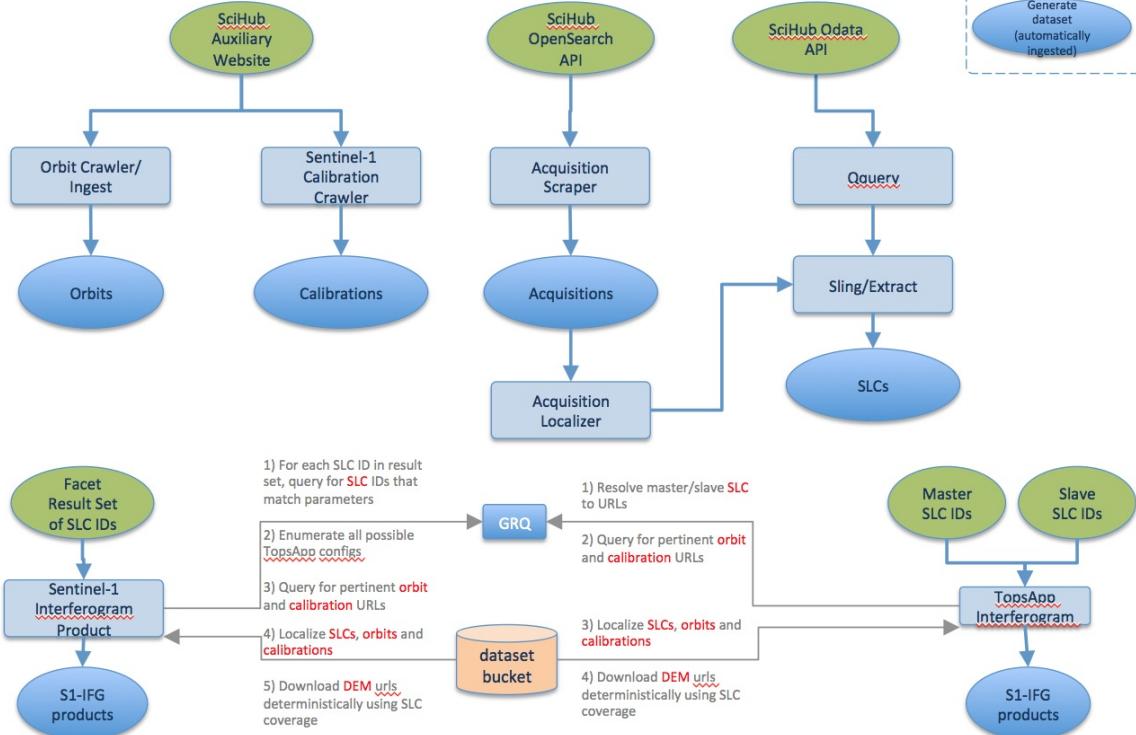


Figure 26: Sentinel-1 Interferogram

The Sentinel-1 interferogram pipeline wraps the actual PGE (product generation executor): ISCE (InSAR Scientific Computing Environment). ISCE is open source and available for download at <https://winsar.unavco.org/software/isce>. To process an interferogram, the ISCE PGE needs the following inputs:

- Sentinel-1 SLCs (single look complex)
- Sentinel-1 precise orbits
- Sentinel-1 calibrations
- USGS DEMs (digital elevation models)

9.6.1 Sentinel-1 SLCs

The ESA data product description describes the SLC dataset as follows:

"Level-1 Single Look Complex (SLC) products consist of focused SAR

data geo-referenced using orbit and attitude data from the satellite and provided in zero-Doppler slant-range geometry. The products include a single look in each dimension using the full transmit signal bandwidth and consist of complex samples preserving the phase information.” [20]

The main source for this dataset is the SciHub portal at <https://scihub.copernicus.eu/>. ESA provides a rest API to query and download the SLCs as described at <https://scihub.copernicus.eu/twiki/do/view/SciHubWebPortal/APIHub>

9.6.2 Sentinel-1 Orbits and Calibrations

In order to process the Sentinel-1 SLC dataset into an interferogram, the ISCE PGE needs multiple ancillary files pertaining to the satellite acquisition date of each SLC. It needs the precise orbit files (https://qc.sentinel1.eo.esa.int/aux_poeorb/) and it needs the auxiliary calibration files (https://qc.sentinel1.eo.esa.int/aux_cal/).

9.6.3 DEMs

Finally, the ISCE PGE requires the DEM (digital elevation model) for the area covered by the SLC scene to process the interferogram. According to the USGS:

“Digital Elevation Models (DEMs) consist of a raster grid of regularly spaced elevation values that have been primarily derived from the USGS topographic map series.” [21]

The DEMs are made available at the following website: <https://e4ftl01.cr.usgs.gov/MEASURES/SRTMGL1.003/2000.02.11/>.

9.7 IMPLEMENTATION

9.7.1 Create a Kubernetes Cluster on IU Jetstream

The first order of business after we have access to the IU Jetstream

OpenStack cluster is to stand up a Kubernetes cluster.

1. Provision Base CentOS7 VMs for a master and 5 worker nodes according to <https://iujetstream.atlassian.net/wiki/spaces/JWT/pages/44826>

The screenshot shows the OpenStack Instances page with the following details:

| Instance Name | Image Name | IP Address | Flavor | Key Pair | Status | Availability Zone | Task | Power State | Time since created | Actions |
|----------------------------|----------------------|--|-----------|----------|--------|-------------------|------|-------------|----------------------|-----------------|
| hyds-kube-node-6.novalocal | hyds-kub-e-node-v1.0 | 10.0.0.8 Floating IPs: 149.165.169.221 | m1.medium | hyds | Active | zone-r3 | None | Running | 10 hours, 57 minutes | Create Snapshot |
| hyds-kube-node-5.novalocal | hyds-kub-e-node-v1.0 | 10.0.0.16 Floating IPs: 149.165.157.245 | m1.xlarge | hyds | Active | zone-r1 | None | Running | 13 hours, 32 minutes | Create Snapshot |
| hyds-kube-node-4.novalocal | hyds-kub-e-node-v1.0 | 10.0.0.7 Floating IPs: 149.165.169.136 | m1.medium | hyds | Active | zone-r1 | None | Running | 1 week, 1 day | Create Snapshot |
| hyds-kube-node-3.novalocal | hyds-kub-e-node-v1.0 | 10.0.0.4 Floating IPs: 149.165.170.108 | m1.medium | hyds | Active | zone-r1 | None | Running | 1 week, 2 days | Create Snapshot |
| hyds-kube-node-2.novalocal | hyds-kub-e-node-v1.0 | 10.0.0.12 Floating IPs: 149.165.170.183 | m1.medium | hyds | Active | zone-r6 | None | Running | 1 week, 2 days | Create Snapshot |
| hyds-kube-node-1.novalocal | hyds-kub-e-node-v1.0 | 10.0.0.5 Floating IPs: 149.165.168.35 | m1.medium | hyds | Active | zone-r2 | None | Running | 1 week, 2 days | Create Snapshot |

Jetstream Instances

2. Log into kube-master via ssh and sudo to root.
3. Elasticsearch requires the following kernel tuning parameter in `/etc/sysctl.conf`:

```
vm.max_map_count=262144
```

Ensure this is configured on each node you bring up (master + worker nodes). To set this on a live system without having to reboot:

```
sudo sysctl -w vm.max_map_count=262144
```

4. Configure kubernetes yum repo:

```
cat <<EOF > /etc/yum.repos.d/kubernetes.repo
```

```
[kubernetes]
name=Kubernetes
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86_64
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg
https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg
exclude=kube*
EOF
```

5. Disable SELinux:

```
setenforce 0
```

6. Update and install kubernetes:

```
yum update -y
yum install -y kubelet kubeadm kubectl --disableexcludes=kubernetes
```

7. Start up kubelet service via systemd:

```
systemctl enable kubelet && systemctl start kubelet
```

8. Initialize your master:

```
kubeadm init

kubeadm init
[init] using Kubernetes version: v1.12.1
[preflight] running pre-flight checks
[preflight/images] Pulling images required for setting up a Kubernetes cluster
[preflight/images] This might take a minute or two, depending on the speed of your
internet connection
[preflight/images] You can also perform this action in beforehand using 'kubeadm config
images pull'
[kubelet] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-
flags.env"
[kubelet] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[preflight] Activating the kubelet service
[certificates] Generated ca certificate and key.
[certificates] Generated apiserver certificate and key.
[certificates] apiserver serving cert is signed for DNS names [js-156-171.jetstream-
cloud.org kubernetes kubernetes.default kubernetes.default.svc
kubernetes.default.svc.cluster.local] and IPs [10.96.0.1 172.28.26.7]
[certificates] Generated apiserver-kubelet-client certificate and key.
[certificates] Generated front-proxy-ca certificate and key.
[certificates] Generated front-proxy-client certificate and key.
[certificates] Generated etcd/ca certificate and key.
[certificates] Generated etcd/server certificate and key.
[certificates] etcd/server serving cert is signed for DNS names [js-156-171.jetstream-
cloud.org localhost] and IPs [127.0.0.1 ::1]
[certificates] Generated etcd/peer certificate and key.
[certificates] etcd/peer serving cert is signed for DNS names [js-156-171.jetstream-
cloud.org localhost] and IPs [172.28.26.7 127.0.0.1 ::1]
[certificates] Generated etcd/healthcheck-client certificate and key.
[certificates] Generated apiserver-etcd-client certificate and key.
[certificates] valid certificates and keys now exist in "/etc/kubernetes/pki"
```

```
[certificates] Generated sa key and public key.
[kubeconfig] Wrote KubeConfig file to disk: "/etc/kubernetes/admin.conf"
[kubeconfig] Wrote KubeConfig file to disk: "/etc/kubernetes/kubelet.conf"
[kubeconfig] Wrote KubeConfig file to disk: "/etc/kubernetes/controller-manager.conf"
[kubeconfig] Wrote KubeConfig file to disk: "/etc/kubernetes/scheduler.conf"
[controlplane] wrote Static Pod manifest for component kube-apiserver to
"/etc/kubernetes/manifests/kube-apiserver.yaml"
[controlplane] wrote Static Pod manifest for component kube-controller-manager to
"/etc/kubernetes/manifests/kube-controller-manager.yaml"
[controlplane] wrote Static Pod manifest for component kube-scheduler to
"/etc/kubernetes/manifests/kube-scheduler.yaml"
[etcd] Wrote Static Pod manifest for a local etcd instance to
"/etc/kubernetes/manifests/etcd.yaml"
[init] waiting for the kubelet to boot up the control plane as Static Pods from directory
"/etc/kubernetes/manifests"
[init] this might take a minute or longer if the control plane images have to be pulled
[apiclient] All control plane components are healthy after 31.502384 seconds
[uploadconfig] storing the configuration used in ConfigMap "kubeadm-config" in the "kube-
system" Namespace
[kubelet] Creating a ConfigMap "kubelet-config-1.12" in namespace kube-system with the
configuration for the kubelets in the cluster
[markmaster] Marking the node js-156-171.jetstream-cloud.org as master by adding the label
"node-role.kubernetes.io/master=''"
[markmaster] Marking the node js-156-171.jetstream-cloud.org as master by adding the
taints [node-role.kubernetes.io/master:NoSchedule]
[patchnode] Uploading the CRI Socket information "/var/run/dockershim.sock" to the Node
API object "js-156-171.jetstream-cloud.org" as an annotation
[bootstraptoken] using token: r7zbx5.c5p8wc2yrwx0iikj
[bootstraptoken] configured RBAC rules to allow Node Bootstrap tokens to post CSRs in
order for nodes to get long term certificate credentials
[bootstraptoken] configured RBAC rules to allow the csapprover controller automatically
approve CSRs from a Node Bootstrap Token
[bootstraptoken] configured RBAC rules to allow certificate rotation for all node client
certificates in the cluster
[bootstraptoken] creating the "cluster-info" ConfigMap in the "kube-public" namespace
[addons] Applied essential addon: CoreDNS
[addons] Applied essential addon: kube-proxy
```

Your Kubernetes master has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

```
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

You should now deploy a pod network to the cluster.

Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
<https://kubernetes.io/docs/concepts/cluster-administration/addons/>

You can now join any number of machines by running the following on each node
as root:

```
kubeadm join 172.28.26.7:6443 --token r7zbx5.c5p8wc2yrwx0iikj --discovery-token-ca-cert-
hash sha256:e887df59ebb48a3483e55a101fe9e7ed0475182420608734725e46941817f35c
```

Make a record of the kubeadm join command that kubeadm init outputs. You need this command to join nodes to your cluster.

If you ever lose the join command, run this on master:

```
kubeadm token generate  
kubeadm token create <generated-token> --print-join-command --ttl=0
```

8. Exit out of root back to a normal user and configure for administration:

```
exit  
mkdir -p $HOME/.kube  
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config  
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

9. Deploy a pod network (Weave Net):

```
sudo sysctl net.bridge.bridge-nf-call-iptables=1  
kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=$(kubectl version | base64 | tr -d '\n')"
```

Confirm that it is working by checking that the CoreDNS pod is running in the output of `kubectl get pods --all-namespaces`. Once the CoreDNS pod is up and running, you can continue by joining your nodes:

| kubectl get pods --all-namespaces | | | READY | STATUS |
|-----------------------------------|--|----------|-------|---------------|
| NAMESPACE | NAME | RESTARTS | AGE | |
| kube-system | coredns-576cbf47c7-dd68p | 0 | 9m35s | 1/1 Running 0 |
| kube-system | coredns-576cbf47c7-m7r6t | 0 | 9m35s | 1/1 Running 0 |
| kube-system | etcd-js-156-171.jetstream-cloud.org | 0 | 8m52s | 1/1 Running 0 |
| kube-system | kube-apiserver-js-156-171.jetstream-cloud.org | 0 | 8m33s | 1/1 Running 0 |
| kube-system | kube-controller-manager-js-156-171.jetstream-cloud.org | 0 | 8m52s | 1/1 Running 0 |
| kube-system | kube-proxy-vt9cn | 0 | 9m35s | 1/1 Running 0 |
| kube-system | kube-scheduler-js-156-171.jetstream-cloud.org | 0 | 8m36s | 1/1 Running 0 |
| kube-system | weave-net-wrtn6 | 0 | 2m3s | 2/2 Running 0 |

9. Repeat step 2-6 above for each worker node you want to have join your cluster.
10. For each worker, ssh into it, sudo to root and run the join command output by `kubeadm init`:

```
kubeadm join --token <token> <master-ip>:<master-port> --discovery-token-ca-cert-hash sha256:<hash>
```

For example:

```

kubeadm join 172.28.26.7:6443 --token r7zbx5.c5p8wc2yrwx0iikj --discovery-token-ca-cert-
hash sha256:e887df59ebb48a3483e55a101fe9e7ed0475182420608734725e46941817f35c
[preflight] running pre-flight checks
    [WARNING RequiredIPVSKernelModulesAvailable]: the IPVS proxier will not be used,
because the following required kernel modules are not loaded: [ip_vs_wrr ip_vs_sh ip_vs
ip_vs_rr] or no builtin kernel ipvs support: map[nf_conntrack_ipv4:{} ip_vs:{} ip_vs_rr:{}]
ip_vs_wrr:{} ip_vs_sh:{}]
you can solve this problem with following methods:
  1. Run 'modprobe -r' to load missing kernel modules;
  2. Provide the missing builtin kernel ipvs support

[discovery] Trying to connect to API Server "172.28.26.7:6443"
[discovery] Created cluster-info discovery client, requesting info from
"https://172.28.26.7:6443"
[discovery] Requesting info from "https://172.28.26.7:6443" again to validate TLS against
the pinned public key
[discovery] Cluster info signature and contents are valid and TLS certificate validates
against pinned roots, will use API Server "172.28.26.7:6443"
[discovery] Successfully established connection with API Server "172.28.26.7:6443"
[kubelet] Downloading configuration for the kubelet from the "kubelet-config-1.12"
ConfigMap in the kube-system namespace
[kubelet] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-
flags.env"
[preflight] Activating the kubelet service
[tlsbootstrap] Waiting for the kubelet to perform the TLS Bootstrap...
[patchnode] Uploading the CRI Socket information "/var/run/dockershim.sock" to the Node
API object "js-156-120.jetstream-cloud.org" as an annotation

```

This node has joined the cluster:

- * Certificate signing request was sent to apiserver and a response was received.
- * The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the master to see this node join the cluster.

Verify that the worker node was registered by running `kubectl get nodes` on the master node:

| kubectl get nodes | | | | |
|--------------------------------|--------|--------|-----|---------|
| NAME | STATUS | ROLES | AGE | VERSION |
| js-156-120.jetstream-cloud.org | Ready | <none> | 41s | v1.12.1 |
| js-156-171.jetstream-cloud.org | Ready | master | 18m | v1.12.1 |

9.7.2 Configuring OpenStack Cloud Provider

In order to enable the OpenStack Cloud Provider for Kubernetes, the following steps must be performed:

1. Log onto the master node and run `hostname -f` to determine it's FQDN
2. Use the OpenStack dashboard or CLI to ensure that the instance name matches the FQDN

```
(env) centos@hysds-kube-node-1:~$ source ~/TG-CDA180009-openrc-v3.sh
Please enter your OpenStack Password for project TG-CDA180009 as user gmanipon:
(env) centos@hysds-kube-node-1:~$ openstack server list
+-----+-----+-----+
| ID | Image | Flavor | Name | Status | Networks |
+-----+-----+-----+
| 53455d81-0644-4a8a-949d-e1cb1fa09622 | hysds-kube-node-3 | m1.medium | ACTIVE | hysds_net=10.0.0.4,149.165.170.108 | hysds-kube-node-v1.0 |
| 56c94413-f1b6-4416-9ce3-688e765cb159 | hysds-kube-node-2 | m1.medium | ACTIVE | hysds_net=10.0.0.12,149.165.170.183 | hysds-kube-node-v1.0 |
| 5f2e73e3-26fa-418d-b3d4-169f90e2d15a | hysds-kube-node-1 | m1.medium | ACTIVE | hysds_net=10.0.0.5,149.165.168.35 | hysds-kube-node-v1.0 |
+-----+-----+-----+
(env) centos@hysds-kube-node-1:~$ hostname -f
hysds-kube-node-1.novalocal
```

3. In this example, the master node's FQDN is `hysds-kube-node-1.novalocal` while the instance name in openstack is `hysds-kube-node-1`.
4. They need to match for the in-tree OpenStack cloud provider to work so use the dashboard or the CLI to change the instance name to match the FQDN.
5. Do this for the master and all kube nodes. ##### On the master node
6. Create the cloud config file which contains the pertinent OpenStack configuration as `/etc/kubernetes/cloud.conf`. This info can be pulled from the OpenStack rc file. Note that if the `availability zone` for Nova instances and Cinder volumes are different, set the `ignore-volume-az` parameter to `true`. You can determine this by creating an openstack instance and cinder volume and comparing the `availability zone` they exist in [Cloud Providers - Kubernetes](#):

```
[Global]
auth-url=https://iu.jetstream-cloud.org:5000/v3
username=gmanipon
password=*****
tenant-id=b16ff2e9abbb41f0b3dcb6b5ad0bd423
domain-id=decf397762654fa2945ae7d4cc49d8c2
tenant-name=TG-CDA180009
domain-name=tacc
region = RegionOne

[BlockStorage]
ignore-volume-az=true
```

7. Under `/etc/kubernetes/manifests` modify the `kube-controller-manager.yaml` and `kube-apiserver.yaml` to add the `cloud-config` and `cloud-provider` options as well as volume mounts to the `cloud.conf` you just created:

```
[...]
spec:
  containers:
    - command:
        - kube-controller-manager (and kube-apiserver)
        - --cloud-provider=openstack
        - --cloud-config=/etc/kubernetes/cloud.conf

[...]
  volumeMounts:
    - mountPath: /etc/kubernetes/cloud.conf
      name: cloud-config
      readOnly: true

[...]
  volumes:
    - hostPath:
        path: /etc/kubernetes/cloud.conf
        type: FileOrCreate
        name: cloud-config
[...]
```

8. Once edited, the KCM and Kube API server will automatically restart. If you receive the message “The connection to the server x was refused—did you specify the right host or port?”, don’t worry, the Kube API server is restarting. It’s normal to lose the connection with it. To confirm that these components have been configured, you can run:

```
kubectl describe pod kube-controller-manager -n kube-system | grep
'/etc/kubernetes/cloud.conf'

  - cloud-config=/etc/kubernetes/cloud.conf
  /etc/kubernetes/cloud.conf from cloud-config (ro)
  Path: /etc/kubernetes/cloud.conf
```

9.7.2.1 On all nodes (including the master)

9. Add the `/etc/kubernetes/cloud.conf` created on the master node to file to the same path on all nodes.
10. Edit the `/etc/systemd/system/kubelet.service.d/10-kubeadm.conf` file, adding the `--cloud-provider=openstack` and `--cloud-config=/etc/kubernetes/cloud.conf` parameters in the `KUBELET_CONFIG_ARGS` environment variable:

```

# Note: This dropin only works with kubeadm and kubelet v1.11+
[Service]
Environment="KUBELET_KUBECONFIG_ARGS=--bootstrap-kubeconfig=/etc/kubernetes/bootstrap-
kubelet.conf --kubeconfig=/etc/kubernetes/kubelet.conf"
#Environment="KUBELET_CONFIG_ARGS=--config=/var/lib/kubelet/config.yaml"
Environment="KUBELET_CONFIG_ARGS=--config=/var/lib/kubelet/config.yaml --cloud-
provider=openstack --cloud-config=/etc/kubernetes/cloud.conf"
# This is a file that "kubeadm init" and "kubeadm join" generates at runtime, populating
the KUBELET_KUBEADM_ARGS variable dynamically
EnvironmentFile=-/var/lib/kubelet/kubeadm-flags.env
# This is a file that the user can use for overrides of the kubelet args as a last resort.
Preferably, the user should use
# the .NodeRegistration.KubeletExtraArgs object in the configuration files instead.
KUBELET_EXTRA_ARGS should be sourced from this file.
EnvironmentFile=-/etc/sysconfig/kubelet
ExecStart=
ExecStart=/usr/bin/kubelet $KUBELET_KUBECONFIG_ARGS $KUBELET_CONFIG_ARGS
$KUBELET_KUBEADM_ARGS $KUBELET_EXTRA_ARGS

```

11. Restart kubelet service using:

```

sudo systemctl daemon-reload
sudo systemctl restart kubelet

```

12. Ensure that the node was successfully registered:

```

sudo tail -f /var/log/messages

Oct 27 15:51:32 hysds-kube-node-3 kubelet: I1027 15:51:32.470352      3320
reconciler.go:207] operationExecutor.VerifyControllerAttachedVolume started for volume
"weavedb" (UniqueName: "kubernetes.io/host-path/b61b79a2-d9c2-11e8-b5d5-fa163e02ac5c-
weavedb") pod "weave-net-mfd9" (UID: "b61b79a2-d9c2-11e8-b5d5-fa163e02ac5c")
Oct 27 15:51:32 hysds-kube-node-3 kubelet: I1027 15:51:32.470366      3320
reconciler.go:154] Reconciler: start to sync state
Oct 27 15:51:32 hysds-kube-node-3 kubelet: I1027 15:51:32.898253      3320
kubelet_node_status.go:324] Adding node label from cloud provider:
beta.kubernetes.io/instance-type=3
Oct 27 15:51:32 hysds-kube-node-3 kubelet: I1027 15:51:32.898300      3320
kubelet_node_status.go:335] Adding node label from cloud provider: failure-
domain.beta.kubernetes.io/zone=zone-r1
Oct 27 15:51:32 hysds-kube-node-3 kubelet: I1027 15:51:32.898313      3320
kubelet_node_status.go:339] Adding node label from cloud provider: failure-
domain.beta.kubernetes.io/region=RegionOne
Oct 27 15:51:32 hysds-kube-node-3 kubelet: I1027 15:51:32.901236      3320
kubelet_node_status.go:70] Attempting to register node hysds-kube-node-3.novalocal
Oct 27 15:51:33 hysds-kube-node-3 kubelet: I1027 15:51:33.227417      3320
kubelet_node_status.go:112] Node hysds-kube-node-3.novalocal was previously registered
Oct 27 15:51:33 hysds-kube-node-3 kubelet: I1027 15:51:33.458241      3320
kubelet_node_status.go:73] Successfully registered node hysds-kube-node-3.novalocal

```

You should now have a running Kubernetes cluster configured with the in-tree OpenStack Cloud Provider:

```
(env) centos@hyds-kube-node-1:~$ kubectl get svc,deploy,pod,no --all-namespaces
NAME          NAME        TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)           AGE
default       service/ci   NodePort   10.103.13.33   <none>        32226/TCP,8080:31514/TCP,8085:31520/TCP,9001:31440/TCP   11h
default       service/factotum   NodePort   10.103.13.240  <none>        32236/TCP,80:30617/TCP,443:30895/TCP,8085:32236/TCP,9001:32441/TCP   11h
default       service/gr-elasticsearch   NodePort   10.103.243.29  <none>        32237/TCP,80:32666/TCP,443:32633/TCP,8878:32205/TCP,8879:30165/TCP,9001:31047/TCP   11h
default       service/grq-redis   ClusterIP  10.96.1.97    <none>        6379/TCP,30362/TCP,9300:31836/TCP   11h
default       service/kubernetes   ClusterIP  10.96.0.1     <none>        443/TCP   44h
default       service/metrics-elasticsearch   NodePort   10.101.36.156  <none>        32238/TCP,80:30311/TCP,443:30311/TCP,5601:30713/TCP,9001:30962/TCP   11h
default       service/metrics-redis   ClusterIP  10.104.1.97    <none>        6379/TCP,30362/TCP,9300:31517/TCP   11h
default       service/mozart   ClusterIP  10.104.1.97    <none>        22:30781/TCP,80:32763/TCP,443:32708/TCP,5555:32623/TCP,8888:31085/TCP,8898:31677/TCP,9001:31336/TCP   11h
default       service/mozart-elasticsearch   NodePort   10.106.14.114  <none>        9200:32630/TCP,9300:30831/TCP   11h
default       service/mozart-rabbitmq   NodePort   10.106.14.239  <none>        5671:32631/TCP,15672:31045/TCP   11h
default       service/mozart-redis   ClusterIP  10.111.72.186  <none>        6379/TCP   11h
default       service/verdi   NodePort   10.104.121.191  <none>        22:32417/TCP,80:32574/TCP,443:32359/TCP,8085:31773/TCP,9001:32008/TCP   43m
kube-system   kube-dns   ClusterIP  10.96.0.10   <none>        53/UDP,53/TCP   44h

NAME          NAME        DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
default       deployment.extensions/ci   1         1         1         1         11h
default       deployment.extensions/factotum   1         1         1         1         11h
default       deployment.extensions/gr-elasticsearch   1         1         1         1         11h
default       deployment.extensions/grq-redis   1         1         1         1         11h
default       deployment.extensions/gru-redis   1         1         1         1         11h
default       deployment.extensions/metrics-elasticsearch   1         1         1         1         11h
default       deployment.extensions/metrics-redis   1         1         1         1         11h
default       deployment.extensions/mozart-elasticsearch   1         1         1         1         11h
default       deployment.extensions/mozart-rabbitmq   1         1         1         1         11h
default       deployment.extensions/mozart-redis   1         1         1         1         11h
default       deployment.extensions/verdi   1         1         1         1         11h
kube-system   deployment.extensions/coredns   2         2         2         2         44h

NAME          NAME        READY     STATUS    RESTARTS   AGE
default       pod/c1-6b994988c7-jb7hw   1/1     running   0          11h
default       pod/factotum-756485399t-k7vkp   1/1     running   0          11h
default       pod/gr-6tdkrfrbs-tjtjg   1/1     running   0          11h
default       pod/grq-redis-5658d69f-8dr1l   1/1     running   0          11h
default       pod/grq-redis-587c5b6d45-8c92s   1/1     running   0          11h
default       pod/metrics-redis-76599a7c4-x2cbh   1/1     running   0          11h
default       pod/mozart-redis-7bbcc4c94-nje1b   1/1     running   0          11h
default       pod/mozart-redis-647f9b5f6-f6sfv   1/1     running   0          11h
default       pod/mozart-elasticsearch-737fc7b6-9mdv6   1/1     running   0          11h
default       pod/mozart-metrics-57645374x9   1/1     running   0          11h
default       pod/mozart-redis-548774ff6-b71dh   1/1     running   0          11h
default       pod/verdi-659d8d467-vs2w   1/1     running   0          43m
kube-system   pod/coredns-5c87447c-9cp   1/1     running   0          44h
kube-system   pod/etcd-hyds-kube-node-1.novalocal   1/1     running   0          44h
kube-system   pod/kube-apiserver-hyds-kube-node-1.novalocal   1/1     running   0          44h
kube-system   pod/kube-controller-manager-hyds-kube-node-1.novalocal   1/1     running   0          44h
kube-system   pod/kube-proxy-881fr   1/1     running   0          11h
kube-system   pod/kube-proxy-czgbm   1/1     running   0          12h
kube-system   pod/kube-proxy-kfrdf   1/1     running   0          44h
kube-system   pod/kube-proxy-kwlpz   1/1     running   0          44h
kube-system   pod/kube-proxy-xn5d   1/1     running   0          44h
kube-system   pod/weave-net-1r5gf   2/2     running   1          14h
kube-system   pod/weave-net-47ewk   2/2     running   0          44h
kube-system   pod/weave-net-1gtzz   2/2     running   0          12h
kube-system   pod/weave-net-1tpp5   2/2     running   0          44h
kube-system   pod/weave-net-nmcgg   2/2     running   1          11h
kube-system   pod/weave-net-thwmnt   2/2     running   0          44h

NAME          NAME        STATUS   ROLES   AGE   VERSION
node/hyds-kube-node-1.novalocal   Ready   master   44h   v1.12.2
node/hyds-kube-node-1.novalocal   Ready   <none>   44h   v1.12.2
node/hyds-kube-node-2.novalocal   Ready   <none>   44h   v1.12.2
node/hyds-kube-node-3.novalocal   Ready   <none>   44h   v1.12.2
node/hyds-kube-node-4.novalocal   Ready   <none>   44h   v1.12.2
node/hyds-kube-node-5.novalocal   Ready   <none>   14h   v1.12.2
node/hyds-kube-node-6.novalocal   Ready   <none>   11h   v1.12.2

(env) centos@hyds-kube-node-1:~$
```

Figure 27: Kubernetes Info

9.7.3 Create HySDS Buckets (Swift Containers)

Using the OpenStack dashboard, we now need to create 2 buckets: one for the PGE docker images and one for the datasets.

Figure 28: Code Bucket

Figure 29: Dataset Bucket

9.7.4 Create the HySDS cluster

Next we provision our HySDS cluster:

1. Log into kube-master via ssh.
2. Clone the `hysds-k8s` repository and checkout the `grfn-jetstream-iu` branch:

```
git clone https://github.com/pymonger/hysds-k8s
cd hysds-k8s
git checkout grfn-jetstream-iu
```

3. Run the `create_hysds_cluster.sh` script to provision all HySDS-related Kubernetes resources:

```
./create_hysds_cluster.sh
```

Note that the script requires 4 arguments: - AWS Access Key for the ARIA datasets bucket hosted by JPL - AWS Secret Key for the ARIA datasets bucket hosted by JPL - Swift Access Key for the HySDS code and dataset buckets hosted on IU Jetstream - Swift Secret Key for the HySDS code and dataset buckets hosted on IU Jetstream 4. Run the `get_urls.sh` script to list the urls for all HySDS web interfaces and REST APIs:

```
./get_urls.sh
```

9.7.5 Register the lightweight-jobs and ariamh repositories in Jenkins

1. Log into kube-master via ssh.
2. Log into the mozart pod using kubectl:

```
kubectl exec -ti mozart bash
```

3. Register the lightweight-jobs repo to Jenkins:

```
sds ci add_job -k -b master https://github.com/hysds/lightweight-jobs.git s3
```

4. Register the ariamh repo to Jenkins:

```
sds ci add_job -k -b jetstream-iu-k8s https://github.com/hysds/ariamh.git s3
```

You should now see both jobs registered in Jenkins:

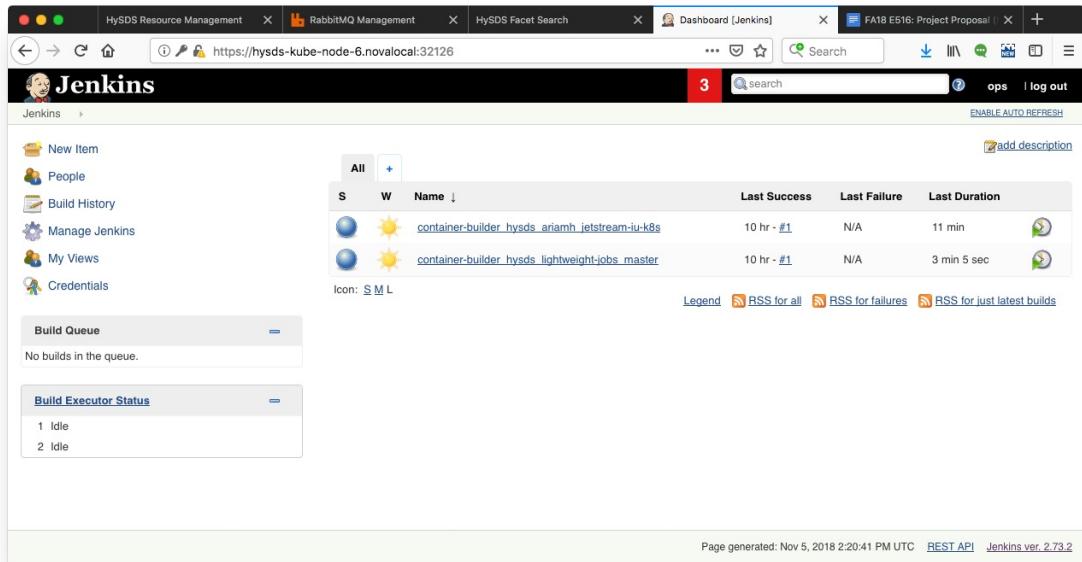


Figure 30: Jenkins Jobs

5. Log into the Jenkins web interface and push the builds for lightweight-jobs and ariamh. You should see the jobs build to completion via the console log output:

```

Started by user ops
Building in workspace /home/ops/jenkins/workspace/container-builder_hyds_ariamh_jetstream-iu-k8s
Cloning the remote Git repository
  > git init /home/ops/jenkins/workspace/container-builder_hyds_ariamh_jetstream-iu-k8s # timeout=10
Fetching upstream changes from https://github.com/hyds/ariamh.git
  > git --version # timeout=10
  > git fetch --tags --progress https://github.com/hyds/ariamh.git +refs/heads/*:refs/remotes/origin/*
  > git config remote.origin.url https://github.com/hyds/ariamh.git # timeout=10
  > git config remote.origin.url https://github.com/hyds/ariamh.git # timeout=10
Checking up Revision 9e28821cc24bf4d229ecbd99af49d37cbfc4bbc (refs/remotes/origin/jetstream-iu-k8s)
  > git config core.sparsecheckout # timeout=10
  > git checkout -f 9e28821cc24bf4d229ecbd99af49d37cbfc4bbc
Commit message: 'remove .netrc binding'
First time build. Skipping changelog.
[container-builder_hyds_ariamh_jetstream-iu-k8s] $ /bin/bash /tmp/jenkins4912048385419354008.sh
fatal: log exactly matches '9e28821cc24bf4d229ecbd99af49d37cbfc4bbc'
[CI] Is checkout a tag: 0
[CI] Last log: commit 9e28821cc24bf4d229ecbd99af49d37cbfc4bbc
Author: Gerald Manipon <pymonger@gmail.com>
Date:   Sun Nov 4 18:15:42 2018 +0000

remove .netrc binding
[CI] Skip image build flag: 0
[WARNING] ml-trainer does not define a hyds-io.json
[WARNING] ml-trainer does not define a job-spec.json
[WARNING] sciflo-ifg-stitcher defines hyds-io parameter query without a type
[WARNING] sciflo-ifg-stitcher defines hyds-io parameter name without a type
[WARNING] sciflo-ifg-stitcher defines hyds-io parameter username without a type
[WARNING] sciflo-dense_offset defines hyds-io without 'submission_type'
[WARNING] sciflo-dense_offset defines hyds-io parameter query without a type
[WARNING] sciflo-dense_offset defines hyds-io parameter name without a type
[WARNING] sciflo-dense_offset defines hyds-io parameter username without a type
[WARNING] sciflo-cor defines hyds-io without 'submission_type'
[WARNING] sciflo-cor defines hyds-io parameter localize_url without a type
[WARNING] sciflo-defines hyds-io parameter path without a type
[WARNING] sciflo-topsapp-slcp defines hyds-io parameter query without a type
[WARNING] sciflo-topsapp-slcp defines hyds-io parameter name without a type
[WARNING] sciflo-topsapp-slcp defines hyds-io parameter username without a type
[WARNING] sciflo-validate-ifg defines hyds-io parameter query without a type
[WARNING] sciflo-validate-ifg defines hyds-io parameter name without a type
[WARNING] sciflo-validate-ts defines hyds-io parameter query without a type
[WARNING] sciflo-validate-ts defines hyds-io parameter name without a type
[WARNING] sciflo-validate-ts defines hyds-io parameter username without a type
[WARNING] sciflo-sl-ifg defines hyds-io without 'submission_type'
[WARNING] sciflo-sl-ifg defines hyds-io parameter query without a type
[WARNING] sciflo-sl-ifg defines hyds-io parameter name without a type
[WARNING] sciflo-sl-ifg defines hyds-io parameter username without a type
[WARNING] sciflo-sl-slcp defines hyds-io parameter query without a type
[WARNING] sciflo-sl-slcp defines hyds-io parameter name without a type
[WARNING] sciflo-sl-slcp defines hyds-io parameter username without a type
[WARNING] sciflo-topsapp-ifg defines hyds-io parameter query without a type
[WARNING] sciflo-topsapp-ifg defines hyds-io parameter name without a type
[WARNING] sciflo-topsapp-ifg defines hyds-io parameter username without a type
[WARNING] sciflo-sl-slcp-mrpe defines hyds-io without 'submission_type'
[WARNING] sciflo-sl-slcp-mrpe defines hyds-io parameter preReferencePairDirection without a type
[WARNING] sciflo-sl-slcp-mrpe defines hyds-io parameter postReferencePairDirection without a type

```

Figure 31: Build ariamh

9.7.6 Submit Sentinel-1 Interferogram Jobs

To facilitate running the Sentinel-1 Interferogram pipeline in Kubernetes without having to ingest 1TB of input datasets, we copy the dataset indexes from the ARIA HySDS cluster at JPL. Access to these indexes requires special permissions so please contact me at gmanipon@jpl.nasa.gov for more information. Assuming the indexes

have been set up on the grq component of the HySDS/Kubernetes/Jetstream cluster, you should see SLCs, precise orbits and calibration files available in the tosca interface:

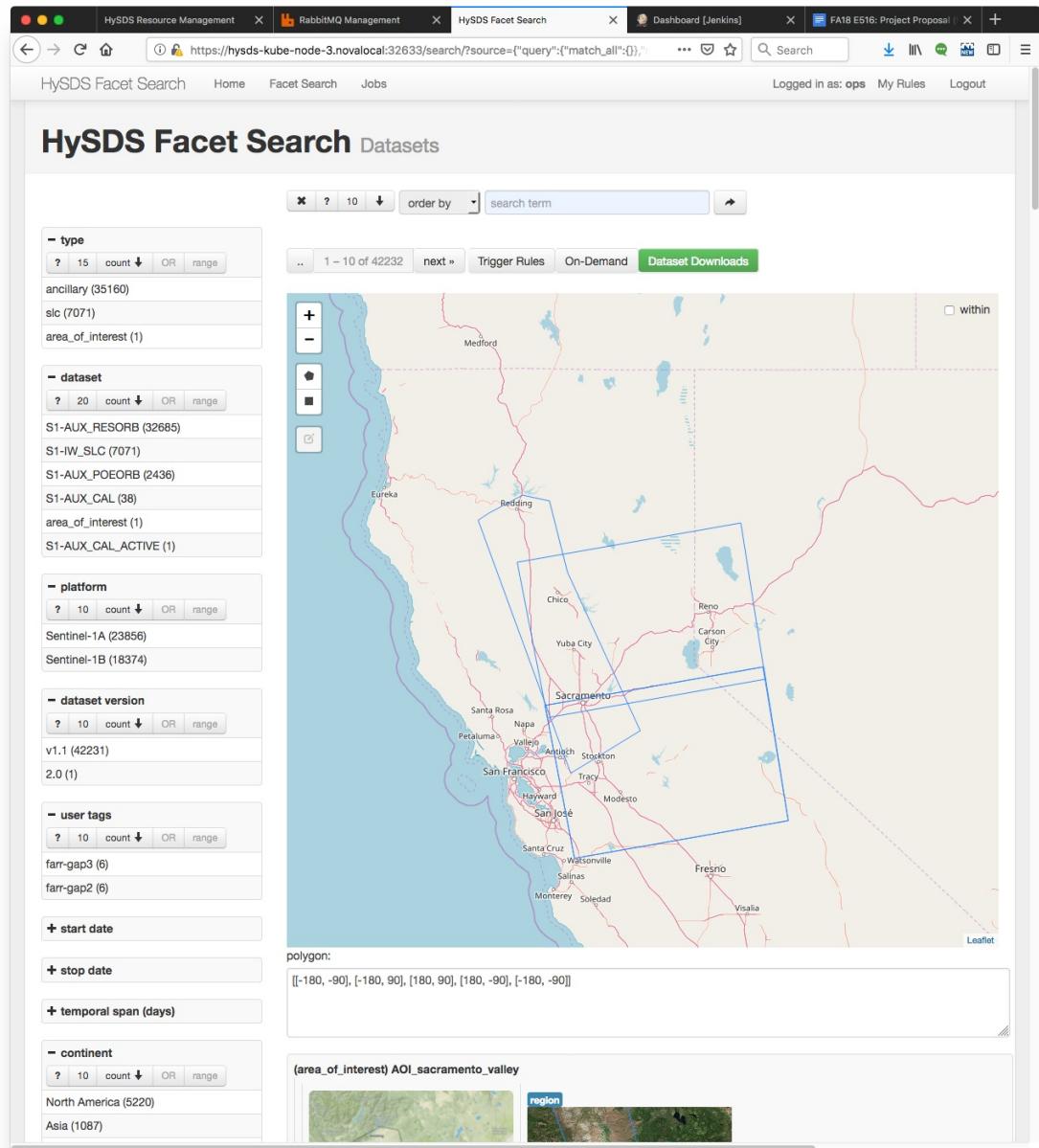


Figure 32: Input Datasets

We can now use the tosca interface to submit Sentinel-1 interferogram jobs.

1. In the tosca interface, draw a bounding box over an area of interest. In our case, we draw a box over the area north of Los Angeles. We then facet down to the S1-IW_SLC dataset type and track 71:

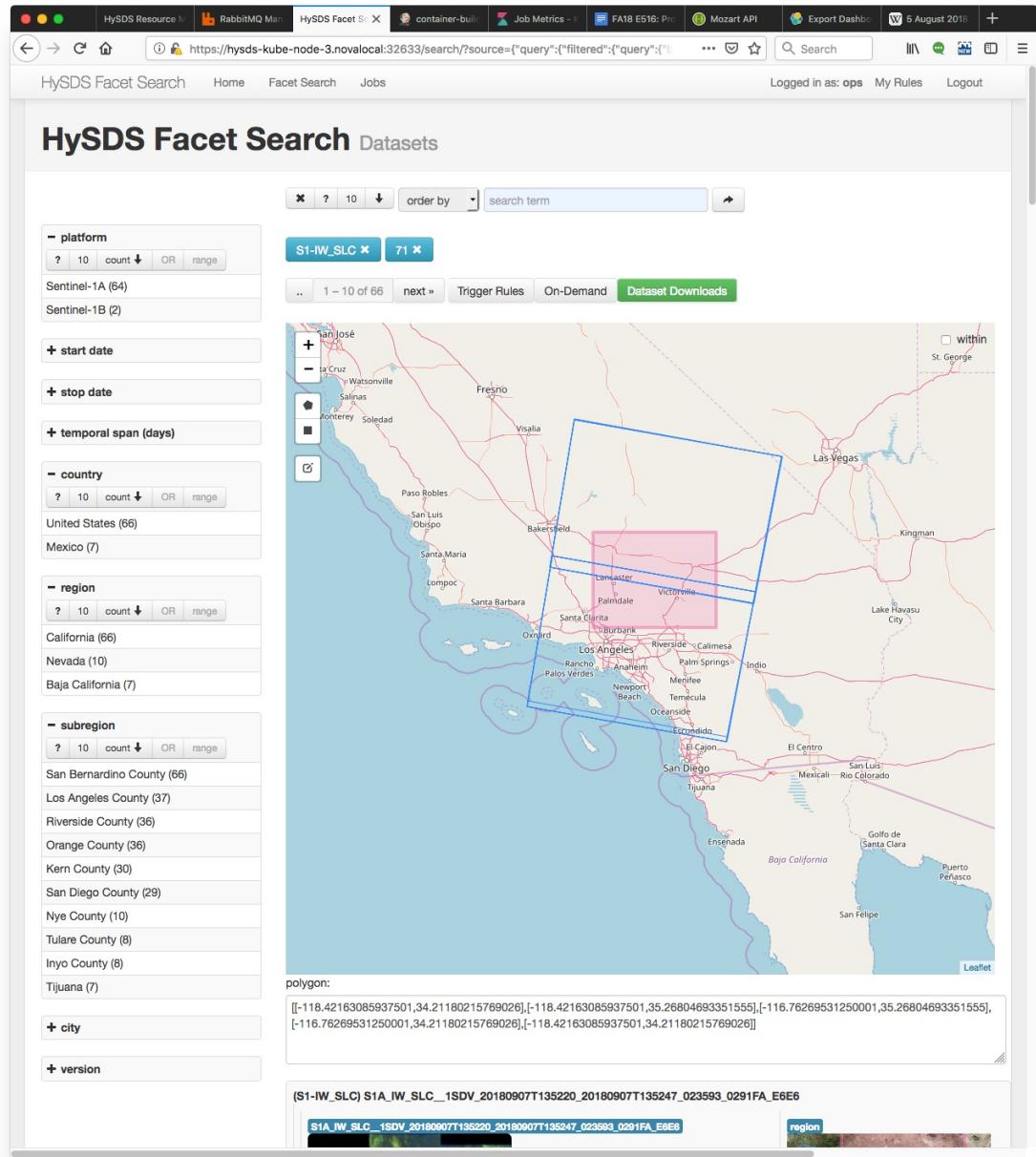


Figure 33: AOI - Los Angeles track 71

2. Click on the “On-Demand” button. This will bring up a modal window that will allow you to submit jobs based on the result set that was faceted down to. Select the “Sentinel-1 Interferogram Product [jetstream-iu-k8s]” action and select the “grfn-job_worker-small” queue:

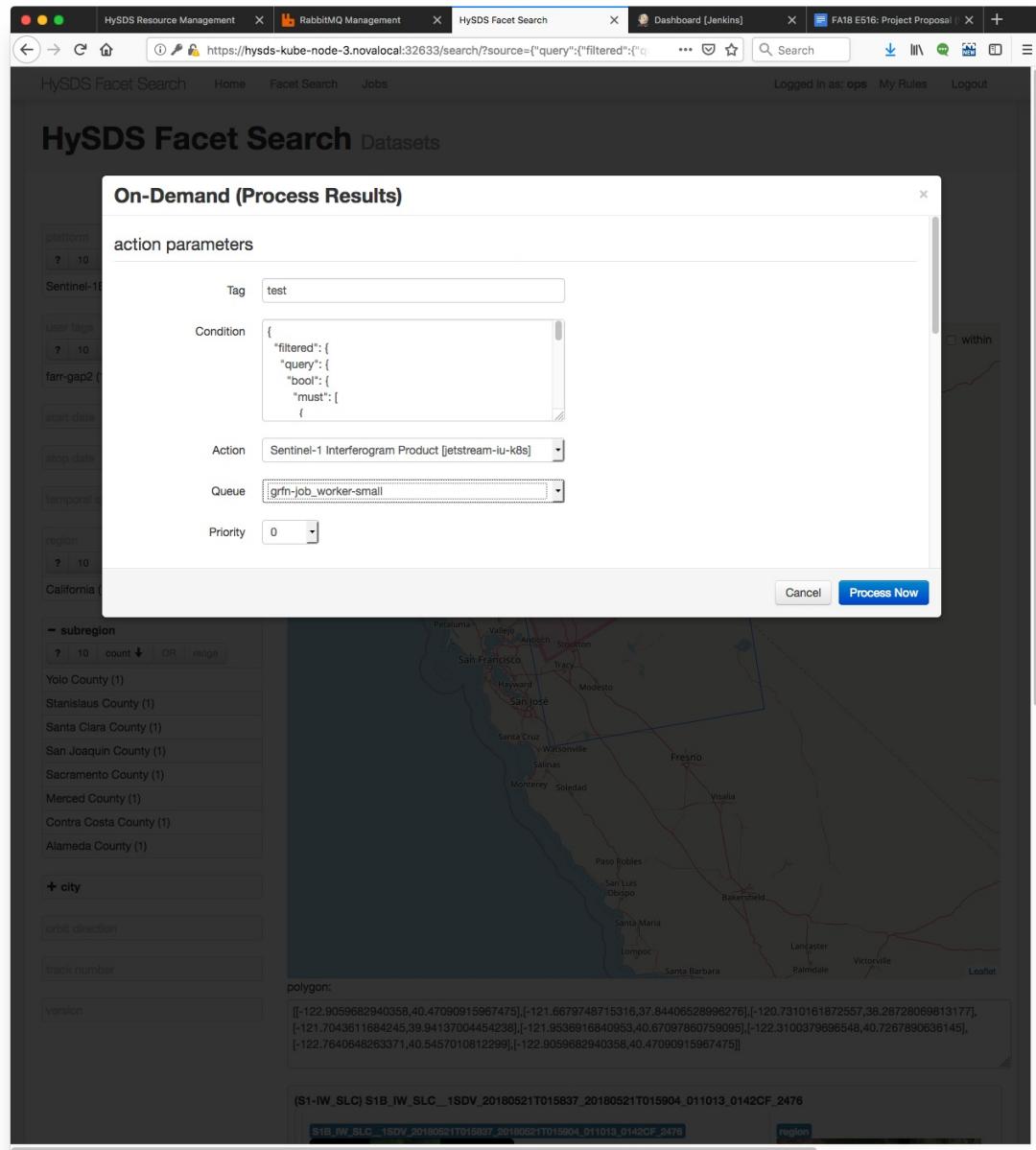


Figure 34: Configure Job

The PGE-specific parameters below can be used unchanged. Click on “Process Now” to submit the job:

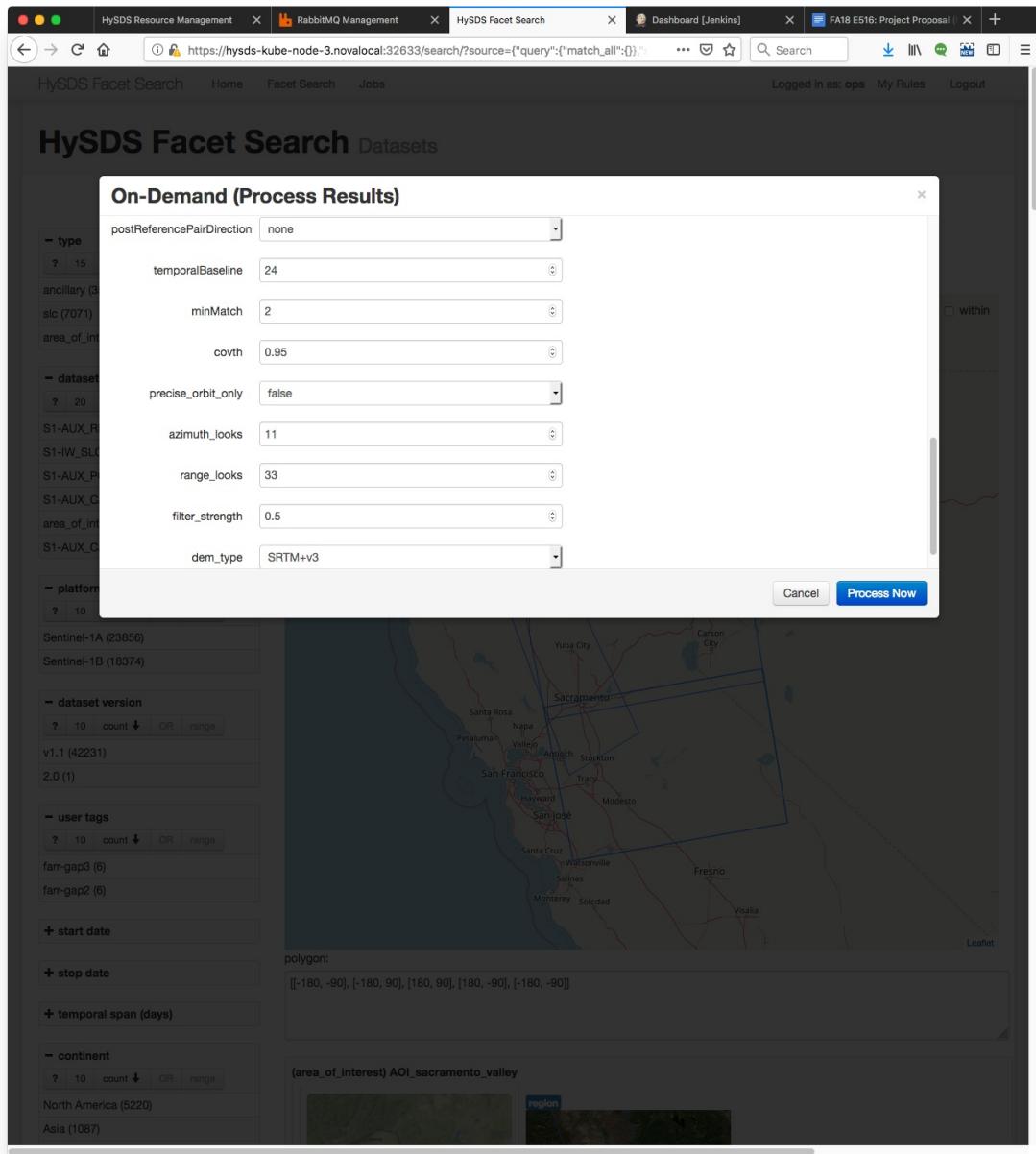


Figure 35: Submit Job

Alternately, the Mozart REST API is a Swagger/OpenAPI-compliant REST interface that can be used to submit jobs:

Mozart API 0.1

[Base URL: /mozart/api/v0.1]
<https://hyds-kube-node-3.novalocal:32708/mozart/api/v0.1/swagger.json>

API for HySDS job submission and query.

queue Mozart queue operations >

job_spec Mozart job-specification operations >

job Mozart job operations >▼

- GET** /job/info Get complete info on submitted job based on id
- GET** /job/list Paginated list submitted jobs
- GET** /job/status Gets the status of a submitted job based on job id
- POST** /job/submit Submits a job to run inside HySDS

container Mozart container operations >

hysds_io HySDS IO operations >

event HySDS event stream operations >

Models >▼

```

Job Status Response(JSON) ✓ {
    status*           string
                      example: job-queued
                      job status
                      Enum
                      > Array [ 6 ]
    message*         string
                      message describing success or failure
    success*         boolean
                      if 'false', encountered exception; otherwise no errors occurred
}

```

```

HySDS IO Addition Response(JSON) ✓ {
    message*         string
                      message describing success or failure
    result*          string
                      HySDS IO ID
    success*         boolean
                      if 'false', encountered exception; otherwise no errors occurred
}

```

Job Type List Response(JSON) >

Figure 36: Mozart REST API

3. Monitor the jobs that were submitted by navigating to the figaro interface:

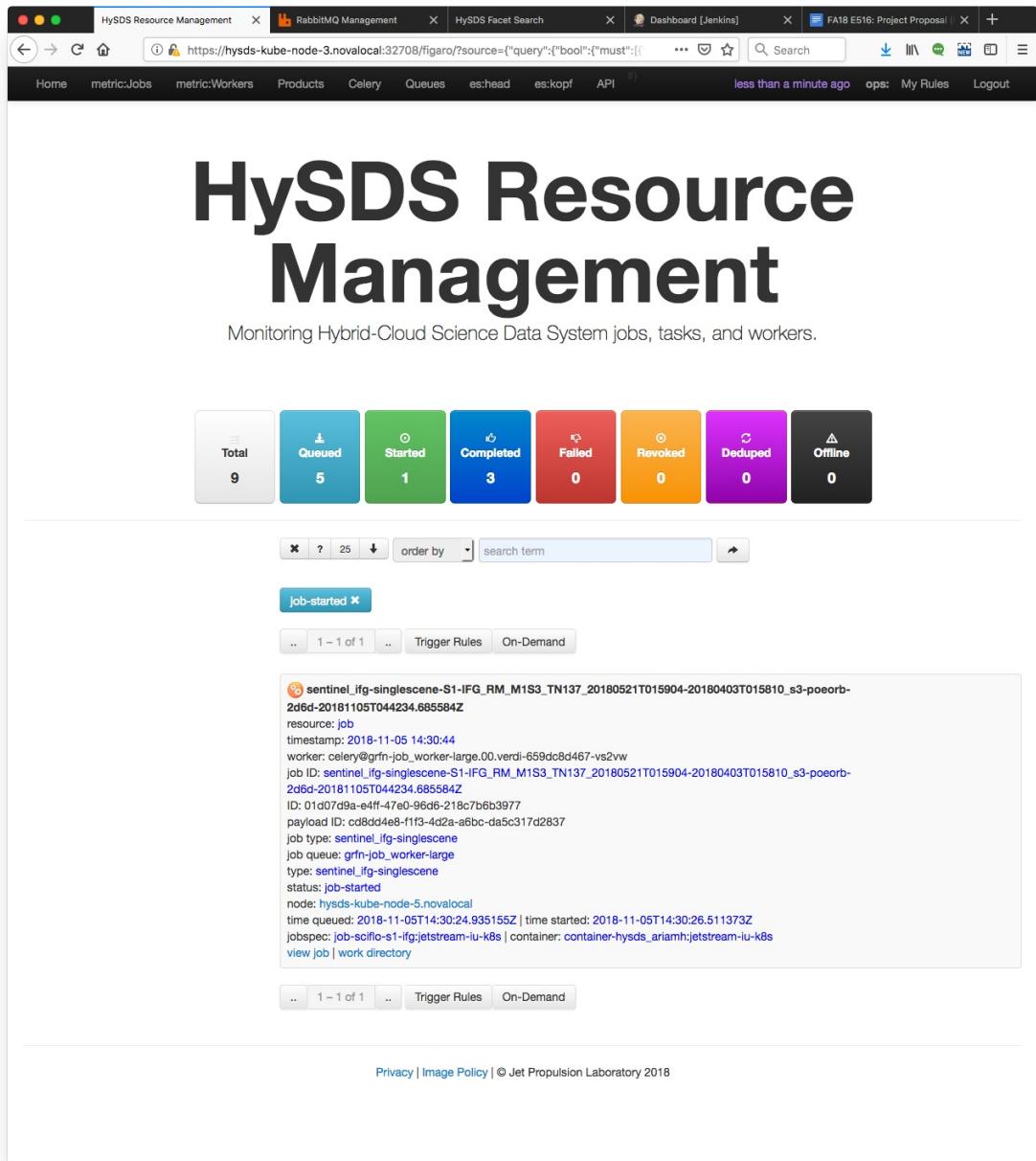


Figure 37: Figaro

The figaro interface provides a faceted view of the HySDS cluster's resource management. This includes job state information as well as information on tasks, workers, and events. Alternately, the RabbitMQ admin interface provides a real-time view of the job queues:

| Name | Features | State | Messages | | | Message rates | | |
|---------------------------|----------|-------|----------|---------|-------|---------------|---------------|--------|
| | | | Ready | Unacked | Total | incoming | deliver / get | ack |
| aria-job_worker-large | D Pri | idle | 0 | 0 | 0 | | | |
| aria-job_worker-small | D Pri | idle | 0 | 0 | 0 | | | |
| dataset_processed | D Pri | idle | 0 | 0 | 0 | 0.00/s | 0.00/s | 0.00/s |
| factotum-job_worker-large | D Pri | idle | 0 | 0 | 0 | | | |
| factotum-job_worker-small | D Pri | idle | 0 | 0 | 0 | | | |
| grfn-job_worker-large | D Pri | idle | 5 | 1 | 6 | 0.00/s | 0.00/s | 0.00/s |
| grfn-job_worker-small | D Pri | idle | 0 | 0 | 0 | 0.00/s | 0.00/s | 0.00/s |
| jobs_processed | D Pri | idle | 0 | 0 | 0 | 0.00/s | 0.00/s | 0.00/s |
| system-jobs-queue | D Pri | idle | 0 | 0 | 0 | 0.00/s | 0.00/s | 0.00/s |
| user_rules_dataset | D Pri | idle | 0 | 0 | 0 | 0.00/s | 0.00/s | 0.00/s |
| user_rules_job | D Pri | idle | 0 | 0 | 0 | 0.00/s | 0.00/s | 0.00/s |
| user_rules_trigger | D Pri | idle | 0 | 0 | 0 | | | |

Add a new queue

HTTP API Server Docs Tutorials Community Support Community Slack Commercial Support Plugins GitHub Changelog

Figure 38: RabbitMQ Admin

4. When the Sentinel-1 interferogram jobs complete, they can be viewed via the tosca interface by faceting on the S1-IFG dataset type:

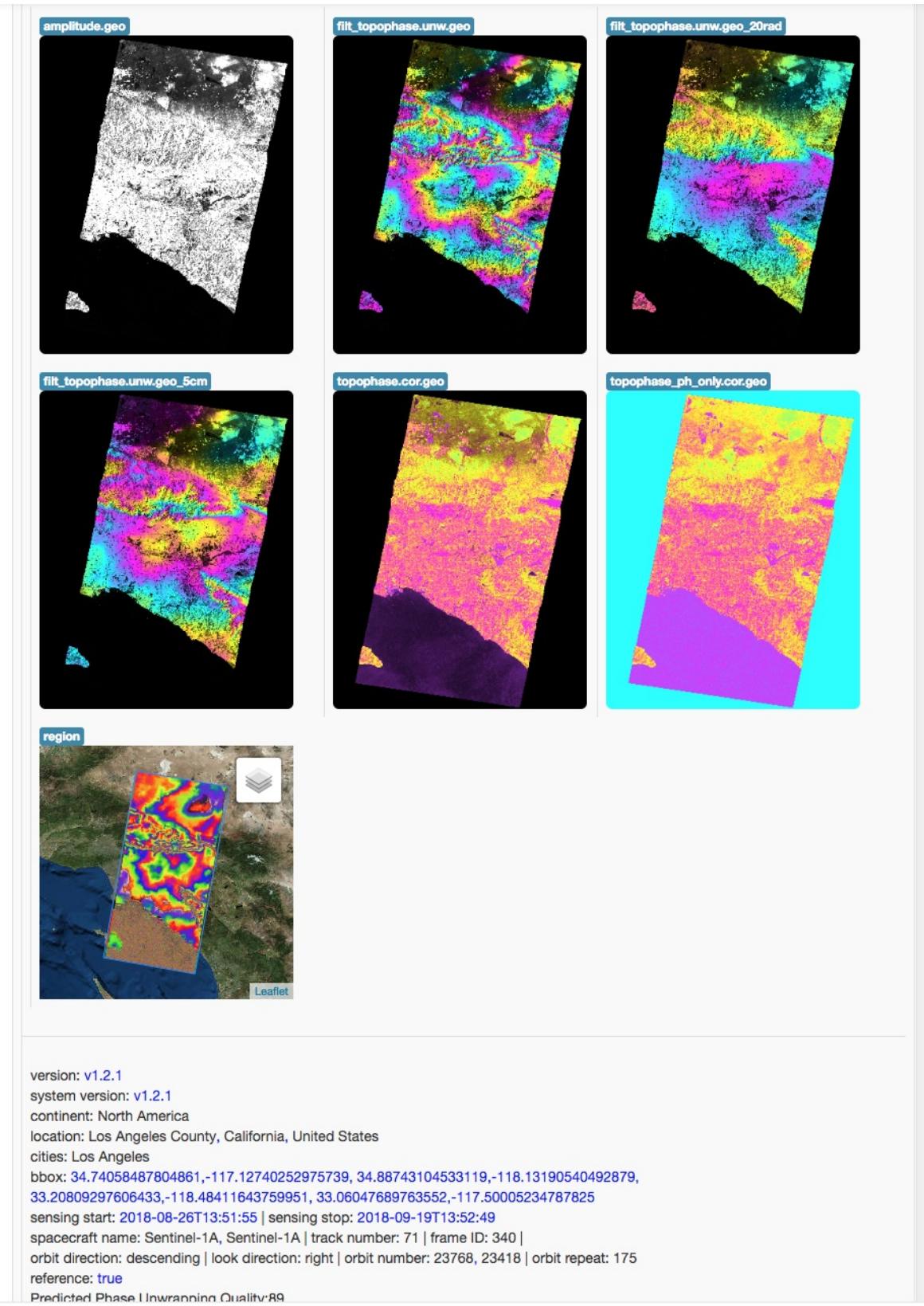


Figure 39: Sentinel-1 Interferogram

9.8 BENCHMARK

After running a few Sentinel-1 interferogram jobs, we can log into the Kibana job metrics interface on the metrics node to view the average execution time of these jobs:

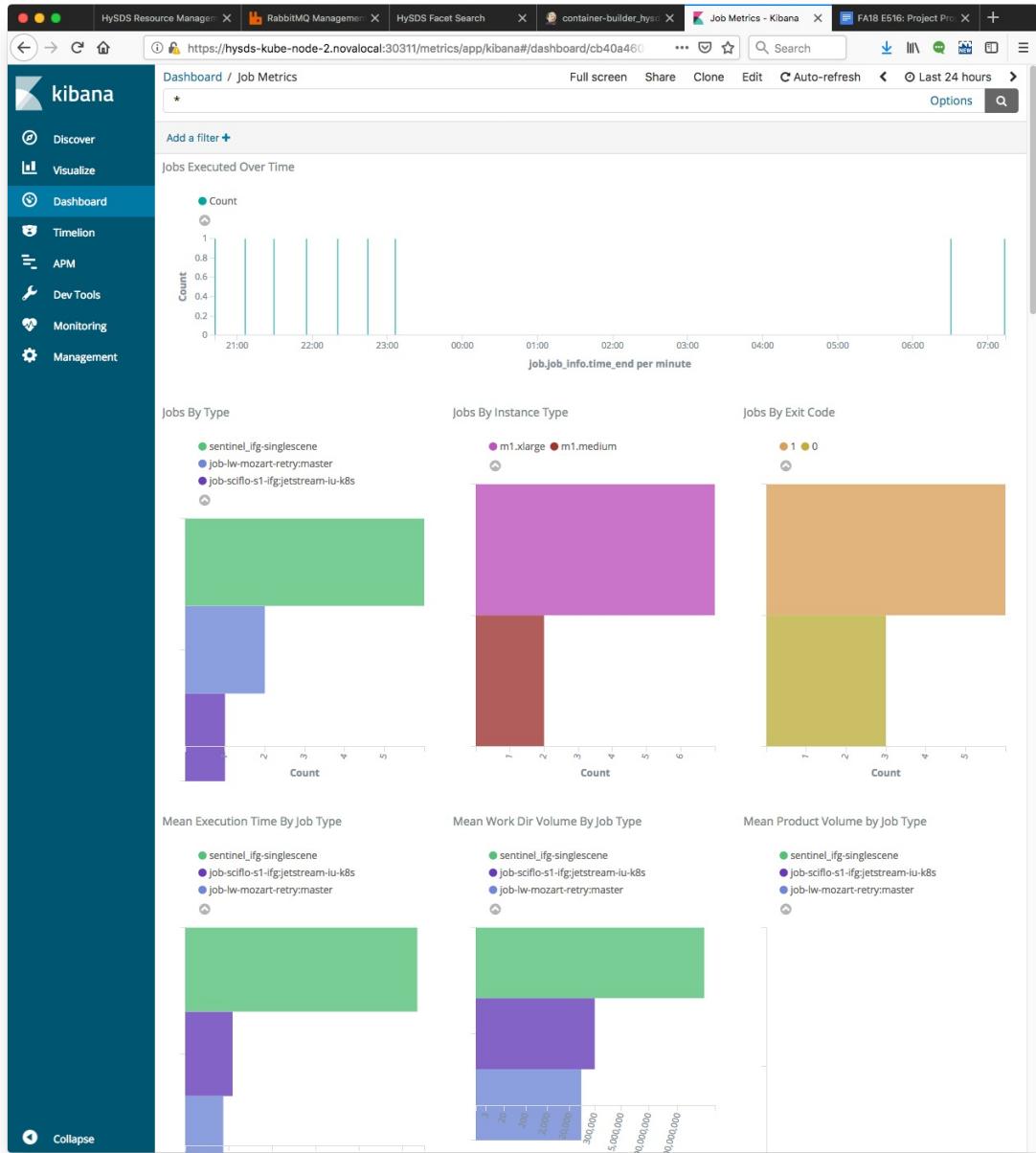


Figure 40: Sentinel-1 Interferogram Job Metrics

We note that it takes an average of 45 minutes to run each job on the HySDS/Kubernetes/Jetstream cluster. In comparison, the ARIA HySDS cluster on AWS takes an average of 25 minutes to run each job. Worker nodes on the ARIA HySDS cluster utilize the c5.9xlarge EC2 instance type (36 vCPUs and 72GB RAM) vs. the worker nodes on the HySDS/Kubernetes/Jetstream cluster which utilize the m1.xlarge flavor (24 vCPUs and 60GB RAM). This discrepancy shows that the

ISCE PGE performs better when it is given more vCPUs and RAM to work with.

9.9 CONCLUSION

The Hybrid Cloud Science Data System was developed in 2011 for the ARIA project to process SAR data to higher-level data products for the purpose of hazard monitoring and urgent response to earthquakes, floods, volcano eruptions, and other natural disasters. It was originally developed to run on OpenStack and later on AWS. As technologies develop and new cloud vendors put forth competitive pricing, there is a constant need to evolve HySDS to utilize these new technologies and cloud vendors. With the advent of container orchestration frameworks such as Kubernetes, HySDS can leverage their built-in support for interfacing with the underlying IaaS services of the various public and private cloud vendors. In this project, we re-architected the HySDS components to run on Kubernetes by decomposing them into Kubernetes pods and services. To test out running HySDS on a Kubernetes cluster on the IU Jetstream OpenStack cloud, we adapted the cluster to run the ARIA Sentinel-1 interferogram pipeline and successfully generated interferograms. In comparing the benchmark execution times of these jobs running on Jetstream infrastructure vs. those that run on AWS, we found that the ARIA worker nodes used a beefier instance type than those used on Jetstream and thus the jobs on the Jetstream cluster took almost twice as long to generate the products: 45 minutes vs. 25 minutes. Future work needs to be done to assess an apples-to-apples comparison of end-to-end HySDS cluster performance and ISCE PGE performance on various OpenStack instance flavors.

9.10 ACKNOWLEDGEMENT

"We thank the XSEDE help desk for assisting with providing access to the IU Jetstream OpenStack API and dashboard, which was made possible through the XSEDE Extended Collaborative Support Service (ECSS) program."

10 SCALABLE DATA PROCESSING FOR RETAIL

FA18- 516-17

Brad Pope
popebradleyt@gmail.com
Indiana University
hid: fa18-516-17
github: [cloud](#)
code: [cloud](#)

Learning Objectives

- Describe the unique data storage and compute needs of the Retail Industry
- Implement portions of a cloud computing environment that would meet those needs
- Describe the performance of the prescribed cloud computing environment on a sample retail dataset
- Highlight some additional architecture opportunities outside the scope of this project that would further help the Retail sector

Keywords: Hadoop, Hive, WebHCat, Java, Retail

10.1 INTRODUCTION

As with other industries, the retail industry today is undergoing a major shift. Traditional brick and mortar stores used to rely on practiced merchandising techniques such as printed flyers, television advertising and low prices to drive traffic into stores. Today, shoppers are electing for convenience more than ever before. This is driving growth in a variety of time-saving Omni channel purchasing behaviors. Shoppers are voting with their wallets to have more

product shipped to their door and for more convenient services such as grocery pick-up.

In tandem with a more demanding shopper, there is also a fundamental shift in the sheer amount of data collected on the path taken to purchase products. For example:

- > * Shoppers now research and initiate more purchases from their smart devices.
- > * There are more shopper interaction datapoints captured in digital advertisements
- > * Social media impact on brands and products
- > * Multiple ecommerce outlets reselling product through new and different channels

In short, there is more data than ever available for retailers to leverage.

To survive, Retailers need to be open to change and use every advantage they have. A fundamental element of developing their strategy is having a saleable cloud architecture to leverage the masses of data that they should be collecting about their shoppers. The challenge is that as technology has evolved each retailer responded differently and implemented a piecemeal data strategy to accommodate data as it becomes available. The challenge is to offer a cloud based highly scalable distributed architecture that will allow retailers to process the data a variety and integrate new data sets as they become available.

10.2 DATASET

In order to appreciate the storage and computational requirements it is important to understand the idiosyncrasies of a retail data set.

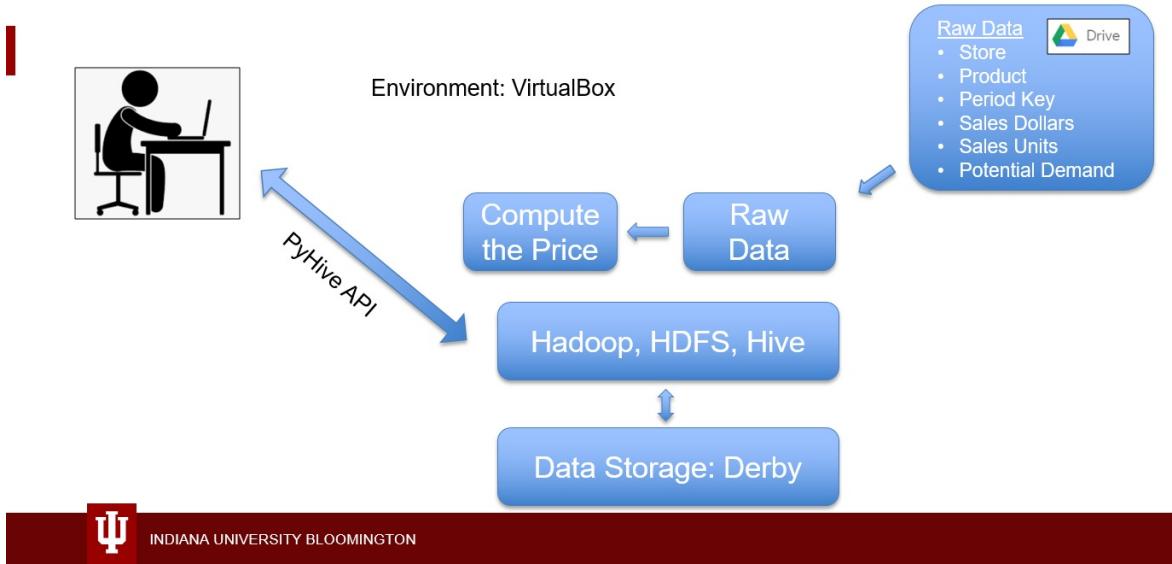


Figure 41: Retail Project Diagram

Data available to retailers is heavily nuanced as each retailer collects and houses data in a different way. There are a host of internal metrics required in the ordinary course of business. In addition, there are a mounting number of external datasets now required to effectively compete. All of this makes for a challenge when it comes to acquiring, blending and putting data to use.

Retailer generated data (Internal data): This is data that the retailer creates during the normal course of business. This includes transactional data such as what product was sold in each store at any given time including what other products it sold with in a specific transaction. At an operational level it includes the purchase orders they use to get more product to sell from suppliers, inventory levels in warehouses and stores. Operational data is also future looking with anticipated through merchandise forecasts. There are other data sets that track what products should be on shelves, how much product should be there and where the product should go. In short, there is complexity for Retailers on their internal data sets range greatly in terms their uses and the metrics gathered.

- Granularity describes the level of depth of a

dataset. On one side of the scale, transactional data sets are at a shopper, store, item, date/timestamp level of granularity and include important features such as what products are purchased together. Commonly retailers make operational available at the store, product, day or week levels of granularity. For example, units per store per day is a normalized measure of how quickly a product sells in a given store set.

- Frequency describes how often a dataset is refreshed. Some measures are important operationally and refreshed continually. Other datasets are more static. For example forecasts could be refreshed weekly, reference information like competitive stores could be updated monthly and exchange rates for planning purposes could be refreshed annually.
- Latency describes how much of a lag exists with a data set. For example, while a dataset may be updated daily it may have two or three-day latency to get it from the stores where the product is selling. Often times, it takes that much time to flow it to the central repository for further processing and reporting.
- Restatement or trickle data occurs when updated data becomes available for past time periods. When this occurs processes need to be in place to remove previous data with the updates. Controls or versions of data should be in place for critical numbers such as actual sales values that get reported to Walstreet for their shareholders.

Externally generated data: Not all data that a retailer uses comes from their internally systems as it only gives partial insight to

shoppers and competitors. To supplement internal data, retailers are continually assessing shopper preferences. In order to do this many subscribe to services offered by IRI, Nielsen and InfoScout. This type of data allows them to understand who their shoppers are and if they are getting their fair share of wallet or percent of purchases of each category were made in their locations. It also allows them to see what other products those shoppers buy and which of their competitors those shoppers are purchasing them.

There are several external sources of data that have recently entered the market.

- > * Social Media: Insights from social media have become highly sought in recent years making that a highly coveted dataset.
- > * Omnichannel: Omnichannel and path to purchase data varies greatly and can holds insights as to how consumers shop for certain products and brands.
- > * Ecommerce: Purchases made on-line is also a desirable subset of the data. Ecommerce continues to pull purchases away from traditional retailers and understanding which products have better potential to sell on-line is considered essential. The complexity with this data is that it can be hard to combine and mine with a retailer's internal transactional and operational data.

10.2.1 Project dataset.

To begin this journey, a sample dataset with very common metrics was masked so the proprietary retailer and supplier are unrecognizable:

- Store – This column represents a single store. The number of stores varies greatly by retailer. The dataset included 300 stores. Larger retailer chains can have more than 10,000 stores internationally.
- Product – This is a proxy for a product UPC or Item Number. Retailers will sell thousands of products in any given retail location. For our purposes here 23 products were included.
- Period_Key – Date information in a YYYYMMDD format. Many

retailer datasets have two years of history available. This dataset contains daily data from 10/23/2015 through 10/15/2017 which equates to 723 unique dates.

- Sales Dollars – Dollar value associated for each product, store and period.
- Sales Units – Number of units sold for each product, store and period.
- Potential Demand – potential revenue associated with having the product available for sale (no out of stocks). This is illustrative of a calculated metric calculated from existing metrics.

10.3 IMPLEMENTATION

Hadoop was chosen as a foundation with the overall solution shown in the diagram below.

While a learning exercise there were reasons behind the selection of the components:

10.3.1 Hadoop

Apache Hadoop for retailers was chosen for several reasons. First, the benefits Hadoop is capable of are well aligned with the needs of retailers. According to the home page of the Hadoop project at Apache Software Foundation, Apache Hadoop offers highly reliable, scalable, distributed processing of large data sets using simple programming models. With the ability to be built on clusters of commodity computers, Hadoop provides a cost-effective solution for storing and processing structured, semi- and unstructured data with no format requirements [22].

First and foremost, the cost advantages associated with open source software should not be overlooked. Retailers working with razor thin margins are not only battling their on-line counterparts, there are a host of aggressive discount retailers such as Aldi, Family Dollar and Lidl that have been aggressively attacking with low cost private label product. Saving money on operations for retailers is critical to

maintain a margin.

From a retailer standpoint this is a good choice since it can reliably process for a range of data needs, and distribute them in a scalable way. Operation data is the lifeblood of a retailer. If the computer assisted order system does not know the inventory and sales from a store as well as the future forecast it will not be able to issue purchase orders to suppliers and keep product on stores for ongoing business. Rather than relying on the hardware to rely on redundancy and high-availability, Hadoop detects and handles failures at the application layer. This delivers a high-availability service on top of a cluster of computers which could individually fail [23].

The application layer also allows for adding more datasets as they become available which is a key retailer need. The Hadoop framework scales from processing on a single server to thousands of machines and uses the computation and storage available on each. As new formats of semi and unstructured data become available.

New data formats in semi- and unstructured formats such as social media, shopper sentiment can be handled by Hadoop as well. Traditional Enterprise Data Warehouse (EDW) would struggle with that integration. This data when coupled with the other operational data and help provide more accurate analytic decisions retailers need [22].

10.3.2 Linux

Also, as a foundation Hadoop was installed on a Linux VirtualBox. While Hadoop can be installed on Windows, Unix and Mac OS X there is an informal consensus on the internet that if you want to run production Hadoop with the fewest number of issues it is best to use Linux.

10.3.3 HDFS

Next, the native distributed file system for Hadoop was set and

integrate it with other computational platforms. The Hadoop Distributed File system (HDFS) is Hadoop's way of proven ability to store very large files on a cluster of commodity hardware [24].

For a retailer with thousands of stores and thousands of products capturing all of the data necessary results in petabytes of data. In addition, data is constantly being created as products move in the supply chain, sell in the stores and forecasting and planning are continually being done in the background. HDFS accommodates these demands with the way it is designed.

First, it is possible to store files of any size on HDFS. The distributed file system breaks files down to files that potentially petabytes in size into smaller pieces or blocks and each piece could be stored on a different machine. In this system a HDFS master node is known as a NameNode and a slave node is called a DataNode. NameNodes maintain and manage all information about all files and directories stored on HDFS including the file system tree and metadata. DataNodes are the actual storage for the blocks. NameNode sends blocks to the DataNodes to store and DataNodes continuously report their status back including the list of blocks they are storing. This makes it possible to store files of any size on a Hadoop Cluster.

For the purposes of the project I'm keeping the NameNode and the DataNode on the same VirtualBox with a small set of sample data, in order to do this at scale leveraging the HDFS structure using NameNodes and DataNodes distributed across many servers would be required. In addition, to fully leverage HDFS the goal would be to store the retailer data in larger files instead of millions of smaller files to be able to have the NameNode use less memory.

Commodity Hardware means using standard commonly used hardware without the need to specialized high-end systems. Being able to use more common server configuration systems to build a reliable cluster with HDFS results in saving money which is extremely important to this demographic.

10.3.4 Hive

Since its incubation in 2008, Apache Hive is considered the defacto standard for interactive SQL queries over petabytes of data in Hadoop. Data analysts use Hive to query, summarize, explore and analyze that data, then turn it into actionable business insight [25].

Hive works well with HDFS because it organizes the data into databases, tables, buckets and clusters. Partitioning these tables and bucketing options allows for efficient storage and data retrieval. This abstract structure allows the system to only load the relevant parts of the table during query processing. Querying less data results in faster query execution times [26].

Just like in other industries, Retail IT professionals often have SQL skills so using the HiveQL which is very similar to SQL would be beneficial for adoption. In addition, the hive structure will do an effective job pruning the large datasets that a retailer has to just the relevant measures and attributes needed in a report or analysis.

Since this is the case, SQL against Hive to compute the average price was leveraged here. Since the dataset was smaller for this project, all of the Hive organization features weren't leveraged. Against a larger, more complex retail dataset more features would be used.

10.3.5 API

There needs to be a way to access the data through an API interface. As a simpler implementation with python, PyHive is the logical API for illustrative purposes with Hive. Per the Python Software foundation, Pyhive is a collection of Python DB-API and SQLAlchemy interfaces for Presto and Hive [27].

Since Retail data can be large and complex it may make sense to scale to other API technologies. WebHCat is a REST API for HCatalog which is the storage management layer for Hadoop. WebHCat also works well with HDInsight making it ideal for an Azure blob Microsoft

implementation. It is a logical alternative as implementation requirements grow with the data size.

10.3.6 Data Storage

Hadoop and HDFS require a database and the dataset in the example is small Derby was a natural choice. Derby has a small footprint and is easy to install, deploy, and use. It also supports a client server model and is based on Java which is already a requirement of Hadoop [28].

With Retail data the data size and concurrent processing needs quickly increase. Traditional relational databases have been optimized for years and are work for conventional use cases where data is well defined and structured. As we add in unstructured data Hadoop and HDFS Azure blob storage scales well. Azure blobs are capable of storing a variety of types of files from documents to database backups. Similar blobs are stored in containers. Both containers and blobs do not have hard size limitations.

Azure Blob storage can be accessed from Hadoop (available through HDInsight). HDInsight can use a blob container in Azure Storage as the default file system for the cluster. Through a Hadoop distributed file system (HDFS) interface provided by a WASB driver, the full set of components in HDInsight can operate directly on structured or unstructured data stored as blobs. Azure Blob storage can also be accessed via Azure SQL Data Warehouse using its PolyBase feature [29]

11 MANAGE FILES ACROSS CLOUD PROVIDERS

516-18

FA18-

Richa Rastogi
rirastog@iu.edu
Indiana University
hid: fa18-516-18
github: [github](#)

Keywords: Multi-cloud data service, Cloud Computing, Python, Open API, Cloud Providers, MongoDB, Swagger

11.1 ABSTRACT

The goal of this project is to manage files across different cloud providers. There are many cloud providers where we can store data in form of files like Amazon AWS, Microsoft Azure, Google cloud, etc. Here we are going to build an OpenAPI to manage these files, operations like copy, upload, download or delete from any provider. This system is self sufficient to work as a file manager.

11.2 INTRODUCTION

The objective of this project is to manage data across different cloud providers. We are going to build an RESTFUL OpenAPI for managing the data between all the cloud storages. We will analyse how these different clouds work and then build python methods to handle data across them. Final step will be to expose these functionalities as a RESTFUL API. This way we can also take advantage of cloud providers for cheaper solutions for storage by dividing the data across them. Since this project has its own MongoDB and User profiling so it can be used a file manager in itself.

11.3 REQUIREMENTS

This project requires knowledge about Cloud Providers like AWS, Azure, Google Cloud, etc. * This project is using Amazon AWS and Google cloud as two cloud providers and their storage functionality. We can expand this * This also needs a database so we are using MongoDB through MongoEngine and store the files and User data in it. * Overall functionality can be accessed through console or RESTFUL OpenAPI which is built using Flask and Swagger.

11.4 DESIGN

This project involves developing a RESTFUL API to manage files. We can perform following operations like upload, download, list, copy, rsync and delete. We can also use this project to store files in MongoDB and assign specific User Role permissions to access the files.

- The very first thing required for that is to create accounts in AWS and Google(Since those are the two providers we are using).
- Then use their API Keys to connect to these providers through Python code.
- After we have a connection, we use their APIs to access the bucket by providing the name through Yaml file/API input.
- Now since we have the bucket, we can list, download or upload the files.
- There is Command console from where we can execute these functions directly using console or script file.
- On top of that there is an open API built to perform these functions using REST.
- We also have MongoDB storing the downloaded files.

11.5 IMPLEMENTATION

This project is using following technologies for implementation: * Python 3.7.0 for python code development * Swagger 2.0 for writing

API specification. This specification describes REST endpoints for managing files across providers.

- * Python flask framework which consumes the OpenAPI specification and directs the endpoints to Python functions by building a RESTful app.
- * MongoEngine as a Document-Object Mapper for working with MongoDB from Python.

Enable a virtual environment so that all installations can be done specifically in that env.

```
python3 -m virtualenv /home/richa/venv/      //to install venv  
source /home/richa/venv/bin/activate        //to activate venv  
  
Now my console looks like:  
  
(venv) (3.7.0) richa@richa-VirtualBox:~$
```

11.5.1 AWS access from Python:

- Install apache-libcloud by “pip install apache-libcloud”
- Follow instructions to create an AWS account - <https://github.com/cloudmesh-community/book/blob/master/chapters/iaas/aws/aws.md>
- Select S3 from Services and create a bucket
- To access this bucket, go to IAM and create a user and then create a new Access Key in “Security Credentials”
- While creating this key, system will prompt to download pem file. Save that pem file onto your working machine.

11.5.2 Google Cloud Platform:

- Install “pip install google-cloud-storage”
- pip install google-auth google-auth-httplib2
- pip install -upgrade google-api-python-client
- Create an account on Google Cloud by going to <https://cloud.google.com/>
- Create a new Project from the top of the page.
- Create a new storage bucket in google cloud, select Storage -> Storage -> Browser
- To access this bucket now, follow <https://cloud.google.com/storage/docs/reference/libraries>

- This will download a JSON file in your working VM and use that file for authentication to access Google Cloud Storage.

All the dependencies can be installed easily by running requirements.txt inside project-code so no need to do any pip install.

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

AWS and Google Cloud specific functionality python files are under directory structure project-code/cloudmesh/data. cloudmesh-data.yaml is the yaml file holding all the information about these cloud setup. Aws_setup.py and google_cloud_setup.py uses this yaml file to authenticate the cloud providers and setup the connection to the cloud services.

It also has command.py under here to run the functionality from console passing in relevant input. To execute commands from console using cmddata commands, we need to setup cmddata by running in project-code dir:

```
pip install .
```

Now we can run all cmddata commands as given below. We can also test if cmddata is working by running a test command:

```
cmddata test

Usage:
  cmddata test
  cmddata set provider=PROVIDER
  cmddata set dir=BUCKET
  cmddata data add PROVIDER BUCKETNAME FILENAME
  cmddata data get PROVIDER BUCKETNAME FILENAME USER_UUID
  cmddata data ls PROVIDER BUCKETNAME
  cmddata data copy FILENAME PROVIDER PROVIDER_BUCKET DEST DEST_BUCKET
  cmddata data rsync FILENAME SOURCE DEST
  cmddata data del PROVIDER BUCKETNAME FILENAME
  cmddata update user USER file FILENAME
  cmddata (-h | --help)
  cmddata --version

Options:
  -h --help      Show this screen.
  --version      Show version.
  --config       Location of a cmddata.yaml file

Description:
  put a description here

Example:
  cmddata test
  cmddata data add AWS richa-516 MapReduce.docx
```

```
cmddata data get AWS richa-516 MapReduce.docx 1234
cmddata data ls AWS richa-516
cmddata data copy xyz.txt AWS richa-516 GOOGLE richa-google-516
```

We also have MongoDB installed to save the downloaded files into the database. We are using MongoEngine as Document-Object Mapper to add records and save the file as a FileField into DB. File is stored into MongoDB using GridFS.

11.5.3 MongoEngine GridFS

GridFS is a specification for storing and retrieving files into MongoDB.

Instead of storing a file in a single document, GridFS divides the file into parts, or chunks, and stores each chunk as a separate document. By default, GridFS uses a default chunk size of 255 kB; that is, GridFS divides a file into chunks of 255 kB with the exception of the last chunk. The last chunk is only as large as necessary. Similarly, files that are no larger than the chunk size only have a final chunk, using only as much space as needed plus some additional metadata.

GridFS uses two collections to store files. One collection stores the file chunks, and the other stores file metadata. The section GridFS Collections describes each collection in detail.

When you query GridFS for a file, the driver will reassemble the chunks as needed. You can perform range queries on files stored through GridFS. You can also access information from arbitrary sections of files, such as to “skip” to the middle of a video or audio file.

GridFS is useful not only for storing files that exceed 16 MB but also for storing any files for which you want access without having to load the entire file into memory. See also When to Use GridFS.

This File database table structure is read from project-code/file.yml definitions and it has a primary key as the name of the file so that we can search based on this field. This also has user_uuid field to provide specific user access to the files.

```
class File(Document):
    name = StringField(primary_key=True)
    endpoint = StringField()
    checksum = StringField()
    size = StringField()
    timestamp = DateTimeField(default=datetime.datetime.now)
    last_modified = DateTimeField(default=datetime.datetime.now)
    user_uuid = StringField()
    file_content = FileField()
```

Similarly we have a User table to store Users with their roles and group.

This project also has RESTFUL APIs to perform all the above operations and their Swagger UI looks like below. For File APIs, please refer to screenshot below for Swagger UI for File APIs (refer to FileSwaggerAPI.png).

file 1.0.0

[Base URL: /cloudmesh]

A file is a computer resource allowing storage of data that is being processed. The interface to a file provides the mechanism to appropriately locate a file in a distributed system. File identification includes the name, endpoint, checksum, and size. Additional parameters, such as the last access time, could also be stored. The interface only describes the location of the file. The file object has name, endpoint (location), size in GB, MB, Byte, checksum for integrity check, and last accessed timestamp.

default

▼

GET /files/{provider}

GET /file/{provider}

POST /file/{provider}

POST /file/copy

POST /file/rsync

DELETE /file/delete

Figure 42: FileSwaggerAPI

For User APIs, please refer to screenshot below for Swagger UI for User APIs (refer to UserSwaggerAPI.png).

user 1.0.0

[Base URL: /cloudmesh]

User profiles are used to store information about users. User information can be reused in other services. This is useful to access files and upload or download them to cloud machines. Profiles can be added, removed and listed. A group in the profile can be used to augment users to be part of one or more groups. A number of roles can specify a specific role of a user.

The screenshot shows the 'profile' section of the UserSwaggerAPI. It contains three API endpoints:

- GET /user/profile**
- PUT /user/profile** Create a new profile
- GET /user/profile/{uuid}**

Figure 43: UserSwaggerAPI

11.6 DATASET

Database records for File table. This shows that file_content is stored in another table as per GridFS described above in fs.chunks and fs.files:

```
{'_id': 'MapReduce.docx', 'endpoint': 'AWS', 'checksum': '2c716d77f0916df41147f16c05c91e10', 'size': '149.5 KB', 'timestamp': datetime.datetime(2018, 12, 2, 14, 45, 32, 564000), 'last_modified': datetime.datetime(2018, 12, 2, 14, 45, 32, 564000), 'user_uuid': 'richa', 'file_content': ObjectId('5c04608cf8724304b52eac8a')}, {'_id': 'aws_lambda.png', 'endpoint': 'AWS', 'checksum': 'a73e3d8449ab6e7366a7b1e7f24dab35', 'size': '99.6 KB', 'timestamp': datetime.datetime(2018, 12, 2, 14, 46, 9, 241000), 'last_modified': datetime.datetime(2018, 12, 2, 14, 46, 9, 241000), 'user_uuid': 'richa', 'file_content': ObjectId('5c0460b1f87243053dd214de')}}
```

Database records for User table.

```
{'_id': '11111', 'username': 'richa.rastogi', 'group': 'test', 'role': 'test', 'resource': 'test', 'context': 'test', 'description': 'test', 'firstname': 'richa', 'lastname': 'rastogi', 'publickey': 'test', 'email': 'rrastogi@iu.edu'}
```

11.7 BENCHMARK

11.8 CONCLUSION

The main objective of this project was to gain knowledge and understanding of different cloud providers and create RESTFUL APIs using OpenAPI architecture using either flask or Eve and then to use MongoDB database to manage this data coming in from cloud providers. There were several other technologies been used to bring this whole project together.

This project can be enhanced even further by including many other cloud providers like Openstack, Azure etc. and all these clouds can perform operations within themselves which can reduce costs for certain people since they will not be exceedingly dependent on only just one provider. Since this project has its own access and database system enabled so this can be used as a file manager in itself.

11.9 ACKNOWLEDGEMENT

I am very thankful to Professor Gregor von Laszewski for helping me throughout this project development as I am new to all the technologies used in this project. I also took help from nist and cm projects to understand the OpenAPI development using flask.

11.10 REFERENCES

- [30] <https://github.com/cloudmesh-community/nist/tree/master/services>
- [31] <https://github.com/cloudmesh-community/cm/tree/master/cm4>

12 LAMP STACKS - LINUX APACHE, MySQL, PHP

De'Angelo Rutledge
derutled@iu.edu
Indiana University
hid: fa18-516-19
github: [cloud](#)
code: [cloud](#)

Keywords: Linux, Apache, MySQL, PHP, Application Programming Interface (API's), REST, IOT, OAS, LAMP

- LAMP is a set of open source software packages used in a collaborative model to create web applications and web services.
- LAMP stands for Linux, Apache, MySQL, PHP.

12.1 ABSTRACT

The use of open API's to transfer data over the web is increasing as Web applications and the Internet of things (IOT) proliferate. Open APIs make it easier to share and pass data over the web.

API's are built to expose resources for access to public data. APIs are used to fetch data from a database, fulfill a request and process data, using a web server to send information to a client or another service.

12.2 INTRODUCTION

LAMP is a set of open source software packages used in a collaborative model to create web applications and web services. LAMP stands for Linux, Apache, MySQL , PHP. The LAMP stack contains open source software packages when integrated creates a SaaS model. The four packages are:

1. Linux - The operating system
2. Apache - The Web Server
3. MySQL - The backend database
4. PHP - The Web Programming Language

The use of open source networks to transfer data over the web is increasing as Web applications and the Internet of things (IOT) proliferate. Open source software and APIs, make it easier to share and pass data over the web. APIs known as Application Programming Interfaces are used to fulfill request from clients or other services. They specify how to create, read, update and delete the state of shared objects. API's can fetch data, process data, or pass data to another web service. Much attention is given to the LAMP stack because it is a common approach for creating web applications and is used with open API's.

There are other alternatives to the LAMP stack as each component can be exchanged for another application. Some list the LAMP stack as Linux, Apache, MySQL & PHP/Python/Perl because each one of the last P components can be a substitute for another.

There[32] are several variants of the four stack model as well. ...

- WAMP: Windows, Apache, MySQL & PHP
- WISA: Windows, IIS, SQL & ASP.net
- MAMP: MacOS, Apache, MySQL & PHP

illegal use of http in images. explained before

![figure 2]([https://user-images.githubusercontent.com/42589474/47969473-24879d00-e046-11e8-9a47-f818323f88aa.png\)](https://user-images.githubusercontent.com/42589474/47969473-24879d00-e046-11e8-9a47-f818323f88aa.png)

12.3 OPENAPIs

Web applications and the Internet of things (IOT) are reasons Open API's are increasing in use. Sharing information between applications is easier when the requester and the sender use a standard protocol to communicate and exchange information. This is why the LAMP

stack is so popular because each component of the stack integrates well.

The OpenAPI Specification Initiative has worked to standardize API design to facilitate standardization of design. The OpenAPI committee states [fa18-516-19-www-oai-b].

When properly defined via OpenAPI, a consumer can understand and interact with the remote service with a minimal amount of implementation logic.[fa18-516-19-OAI]. OAS states, The OpenAPI Specification (OAS) defines a standard, programming language-agnostic interface description for REST APIs, ... [fa18-516-19-OAI]

RESTful stands for Representational state transfer. When referring to RESTful here; I refer to the W3C working groups definition of RESTful which is [fa18-516-19-www-restful], arbitrary Web Service, in which the service may expose an arbitrary set of operations. An example is HTTP request to GET or POST information from a URI.

- <https://www.w3.org/TR/ws-arch/>

Rigor.com likens the job of an API to a waiter, taking the request, then fetching and returning the data.

Your waiter writes down your order, delivers to the kitchen, picks up your food when it's ready, and serves it to you at your table. [fa18-516-19-www-rigor-com]

12.4 SETTING UP A LAMP STACK

12.4.1 Setup the environment

First Load the operating system for your environment in the case of a LAMP stack Linux is the operating system. You may select one of the Linux distributions such a ubuntu.

Click the link, locate your distro and select download.

- <https://www.ubuntu.com/download/desktop>

Download Virtual Box to setup a virtual environment in which you will run the LAMP stack

- <https://www.virtualbox.org/wiki/Downloads>

Install Virtualbox

- <https://www.virtualbox.org/manual/UserManual.html>

12.4.2 Install Linux (Ubuntu)

Install ubuntu on the virtual machine

Install Vagrant

- <https://www.vagrantup.com/downloads.html>
- <https://www.vagrantup.com/intro/index.html>

12.4.3 Setting Up Vagrant

Use vagrant to initialize ubuntu on the virtual machine(vm) by typing the following command:

```
vagrant init "your linux distro" (ubuntu-bionic64)
vagrant -v  (check the version of vagrant that installed)
vagrant up  (starts the VM)
```

12.4.4 Installing Apache

Apache is the web server used in a LAMP stack. Apache is an open source software created and managed by the Apache Software Foundation and the Apache Server Project. The Apache web server establishes connections between the backend database and a client and host services available to the client. According to Apache [fa18-516-19-www-apache-httpd] The Apache HTTP Server Project is an effort to develop and maintain an open-source HTTP server for modern operating systems including UNIX and Windows.

To install Apache from the command line type:

```
sudo apt-get install -y apache2
sudo a2enmod rewrite
```

a2enmod rewrite - is an Apache module that generates readable url's. [fa18-516-19-www-digitalocean] One of the most useful modules for Apache is mod_rewrite. This module allows you to generate unique and easily readable urls for content requested on the server.

<https://www.digitalocean.com/community/tutorials/how-to-install-configure-and-use-modules-in-the-apache-web-server>

12.4.4.1 Performance Tuning and Server Settings

After you start the Apache server there are a few settings you might want to change in order to maintain server performance. Two of these settings are ServerLimit and MaxClients.

It is best to configure the server to provide the shortest response time possible especially during peak times. Two variables that impact response to request are MaxClients and ServerLimits. Both variables are set to a default values of 256 initially.

According to APACHE HTTP SERVER PROJECT [fa18-516-19-www-apache-httpd], The biggest single hardware issue affecting web server performance is RAM. APACHE recommends adjusting the MaxRequestWorkers setting to minimize swapping.

As the name indicates this setting controls the maximum number of connections that can be processed simultaneously. As stated by APACHE [fa18-516-19-www-apache-httpd] The MaxRequestWorkers directive sets the limit on the number of simultaneous request that will be served. The ServerLimit setting can be changed by updating the httpd.conf file.

- https://httpd.apache.org/docs/current/mod/mpm_common.html

12.4.4.2 Server status

12.4.4.3 To list the server status type:

```
sudo systemctl status apache2
```

12.4.4.4 Adding Apache mods

One Apache mode that is recommended to install is the spamhaus mod. The purpose of the mod is to block black listed URL's. This list was created and is maintained by Spamhaus. According to Spamhouse they are

an international nonprofit organization that tracks spam and related cyber threats such as phishing, malware and botnets, provides realtime actionable and highly accurate threat intelligence to the Internet's major networks, corporations and security vendors, ...
[fa18-516-19-www-spamhause]

- <https://www.spamhaus.org/organization/>

To install the mod type:

```
sudo apt-get install libapache2-mod-spamhaus
```

12.4.5 Installing PHP and MySql

To install PHP type the following bash command at the terminal command prompt:

```
$ sudo apt-get install -y php7.2
```

Now install the apache PHP mod by typing the following bash command at the terminal prompt:

```
$ apt-get install -y libapache2-mod-php7.2
```

Restart the apache server by typing the following bash command

```
$ service apache2 restart
```

Next install the following PHP mods, the first package loads common modules for PHP. The php7.2-mcrypt packages provides a number of functions to handle encryption and decryption.

```
$ sudo apt-get install -y php7.2-common  
$ sudo apt-get install -y php7.2-mcrypt  
$ sudo apt-get install -y php7.2-zip
```

- <http://php.net/manual/en/book.mcrypt.php>

12.4.5.1 Set MySql User & Pass

```
debconf-set-selections <<< 'mysql-server mysql-server/root_password  
password root' debconf-set-selections <<< 'mysql-server mysql-  
server/root_password_again password root'
```

12.4.5.2 Install MySQL

```
$ apt-get install -y mysql-server
```

12.4.5.3 PHP-MYSQL lib

```
$ apt-get install -y php7.2-mysql
```

After installing the database restart the apache server before testing the stack and writing data to the database.

12.4.5.4 Restart Apache

```
$ sudo service apache2 restart
```

13 CLODMESH GRAPHQL APP FA18-516-21 AND FA18-516-02

Mihir Shanishchara, Vineet Barshikar
mshanish@iu.edu, vbarshik@indiana.edu
Indiana University
hid: fa18-516-21 fa18-516-02
github: [cloudmesh-client](#)
code: [cloudmesh-client](#)

Keywords: GraphQL, Cloudmesh client, mongoengine, Flask, Electron

13.1 ABSTRACT

13.2 INTRODUCTION

Cloudmesh cm4 is an ongoing project worked upon by entire class to create a network of computers that run parallel jobs. Currently it accepts commands via command line.

Our project provides an user interface to Cloudmesh cm4. In our project, we have implemented a client-server application which will accept commands from user interface and pass it to server which will perform corresponding appropriate actions. Our second aim with this project is to demonstrate client server communication through GraphQL Apis.

More info for GraphQL is available as a chapter in cloud computing handbook.

13.3 REQUIREMENTS

- A cross platform desktop application which can be

- redistributed to users
- Client App should show data from MongoDB using GraphQL APIs
 - Client App should send user action to server and mutate data
 - Client and server should be able to handle more than 10000 VMs

13.4 DESIGN

Cloudmesh App is divided in two parts

- Client App: Client app can be distributed to users. This app will provide a simple interface to user using which user can execute all commands provided by cm4. Client app will communicate with GraphQL server and based on input from user server will execute commands.
- GraphQL server: This GraphQL server will be running on one cloud instance to which all client apps can connect.

13.5 ARCHITECTURE

Client App is designed using following technologies

- ElectronJS ??? : Using ElectronJS we can build cross platform desktop apps with JavaScript, HTML and CSS. ElectronJS combines power of native apps with beautiful web interface.
- BackboneJS ??? : BackboneJS provides an MVC structure with models, collections and views. For code reusability views have been divided in to two categories
 - Smart View: Knows how to communicate with server but doesn't know about representation
 - Dumb View: Doesn't know how to communicate with server but knows how to render data
- HandlebarsJS ???: Handlebars provides set of functions which

lets us build generic HTML templates easily. It also provides a way to extend helper function and create custom helpers to use in templates.

- All custom helper functions for tempaltes are defined at utils/helpers space
- jQuery ???: jQuery provides set of functions which are very useful for DOM manipulation
- Material UI ???: Material UI is open source design spec which is mainly developed by Google. We are using web component implementation of Material UI.
- Webpack ???: Webpack is a module bundler and also it provides a way to specify loaders for different file types. For example handlebars loader is used to load and compile handlebar template before creating bundle.

13.6 DATASET

Used faker to generate fake data for testing.

13.7 IMPLEMENTATION

- Checkout project-code and execute following commands

```
cd app  
npm install
```

After all UI dependencies are installed execute following commands

```
cd ..  
python3 -m venv cloudmesh-graphql-server  
cd cloudmesh-graphql-server  
source bin/activate  
pip install -r requirements.txt
```

Now to start graphql server execute following command

```
python app.py
```

Once the server is started open another terminal and go to app directory. To start client app execute

```
npm start
```

13.8 BENCHMARK

13.9 CONCLUSION

13.10 ACKNOWLEDGEMENT

13.11 WORKBREAKDOWN

| Task | Author |
|--|--------|
| Initial code setup with client server integration | Mihir |
| Integration of mongoengine | Vineet |
| Implementation of login page | Mihir |
| Add routing between pages | Mihir |
| Use Flask in server code | Vineet |
| GraphQL query to fetch VMs list | Vineet |
| Use Faker to generate mock VMs | Mihir |
| GraphQL mutation to change VM state | Vineet |
| Add Tabs for various VM types (AWS, Azure, etc.) | Mihir |
| Add variables for GraphQL APIs | Mihir |
| Make mutations generic | Vineet |
| Make custom helper classes for Handlebar templates | Mihir |
| Lazy loading (infinite scrolling on UI) of VMs | Vineet |
| Mutation for set/unset Favorite VM | Vineet |
| Add code to show VM details | Mihir |

| | |
|---|--------|
| Add code to mimic DB update from cloudmesh data | Mihir |
| Add code to show notifications | Mihir |
| Add helper to prettify JSON | Mihir |
| Implement sort by dropdown and integrate with API | Mihir |
| Added table view to the list of VMs | Vineet |
| Added sorting by IsFavorite and host | Vineet |
| Implement image page to list image | Mihir |
| Implement list all image APIs | Mihir |

14 OPEN API WITH AWS EMR AND JUPYTER

FA18- 516-22

Ian Sims
isims@iu.edu
Indiana University
hid: fa18-516-22
github: [cloud](#)
code: [cloud](#)

Keywords: AWS, Open API, EC2, EMR, Jupyter, S3

Learning Objectives

- Use Open API to interact with various AWS products
- Learn to deploy an AWS EMR Cluster
- Interact with Jupyter notebooks stored in S3 buckets

14.1 ABSTRACT

The goal of this project is to create an AWS cloud environment to facilitate analytical work at scale. This includes creating an API to facilitate the creation of an analytical environment as well as APIs to aid a user in determining the types of analytical processes that already exist.

Specifically, this project involves building an API for the creation of an AWS EMR cluster and APIs to give users the ability to determine available analytical datasets on AWS S3. Finally, we will create an API that interacts with the S3 API and gives the ability to parse through Jupyter notebook files to determine which analytical processes utilize a given dataset.

The project architecture can be visualized as follows:

Figure 44 shows the proposed architecture for this project

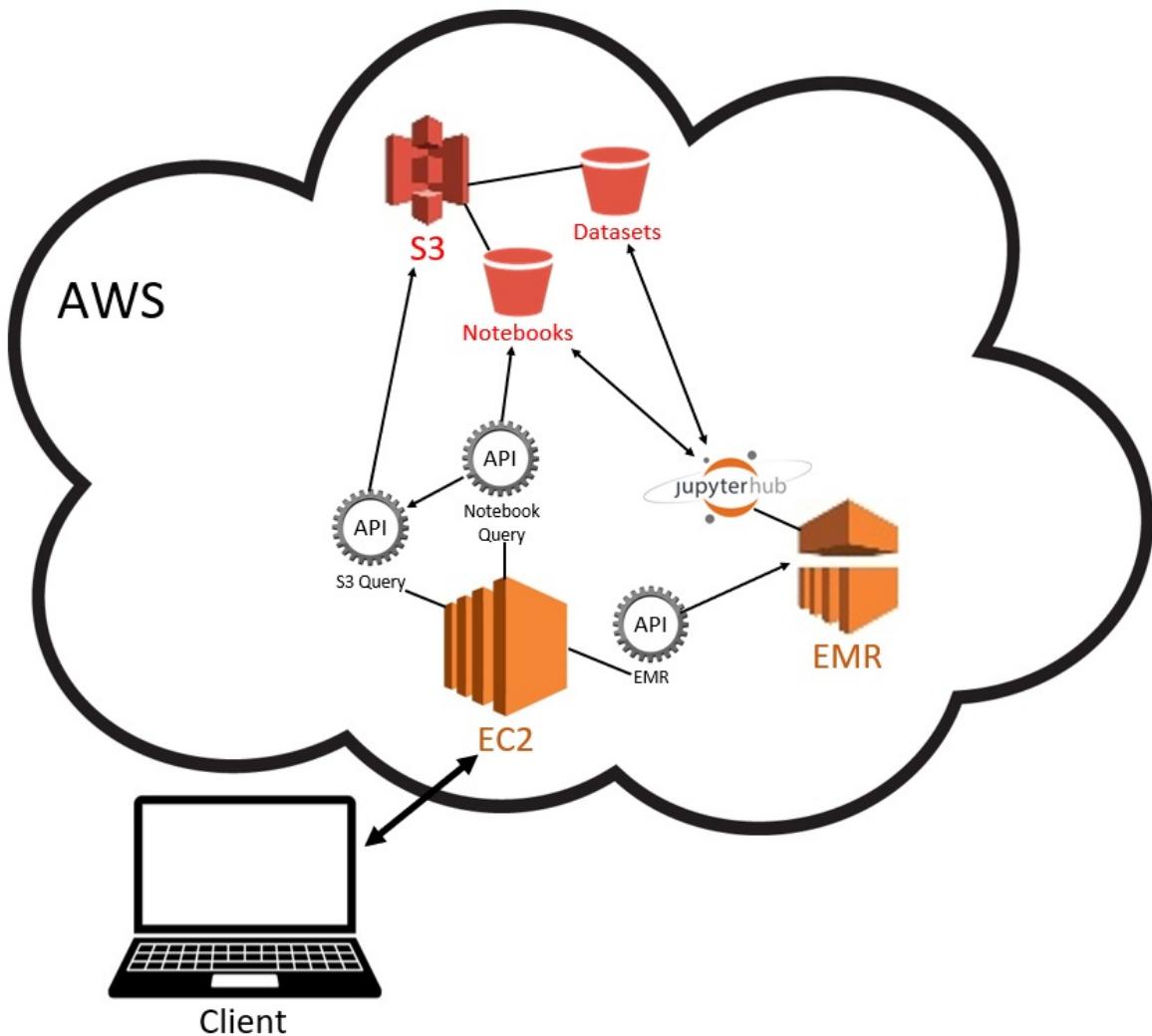


Figure 44: Project Architecture

14.2 INTRODUCTION

The following provides a description of the different technologies and products that were used in the project.

14.2.1 Open API

Rest API's allow for the creation of services that can interact with multiple applications. This project will seek to develop an API for interacting with various AWS products. These products include AWS

EMR and S3. In addition, these APIs will be hosted on an AWS EC2 instance.

Open API is an open source project intended to create a consistent format for creating REST services. Open API describes this project as:

“The OpenAPI Initiative (OAI) was created by a consortium of forward-looking industry experts who recognize the immense value of standardizing on how REST APIs are described.” [33]

14.2.2 AWS EMR

EMR is an Amazon product that allows for the creation of clusters of Elastic Compute Cloud (EC2) instances. EMR allows users to take advantage of distributed computing capabilities. As the name suggests this product is designed to allow users to easily scale their cluster to meet their computing needs.

EMR clusters can be created through relatively simple web interfaces or can be created through code using CLI. EMR Clusters can be configured for size and can be provisioned with open-source distributed frameworks such as SPARK and HBase.

For this project we will interact with EMR using an API. This API will allow for the creation and termination of an EMR cluster. It will also allow a user to retrieve the status of an EMR cluster. This EMR cluster will also include the installation of Jupyter Hub to enable the development of notebooks for analytical purposes.

14.2.3 AWS EC2

EC2 is an Amazon product that enables cloud computing. Amazon describes this product as:

“Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides secure, resizable compute capacity in the cloud. It is designed to make web-scale

cloud computing easier for developers." [34]

For this project APIs will be hosted on an EC2 instance.

14.2.4 AWS S3

S3 is one of Amazon's data storage solutions. For this project we will configure an EMR cluster to read/write to an S3 bucket. This bucket would potentially be used for accessing data for analytical purposes and to store log files associated with the cluster. In addition, the Jupyter Hub instance installed on the EMR cluster will store created notebooks on S3. This will allow for terminating the cluster when it is not in use with the ability to retrieve saved notebooks for future use.

14.2.5 JupyterHub

JupyterHub is an open-source project intended to allow a wide range of users to interact with and organize notebooks for analysis. The open-source project describes JupyterHub as follows:

"JupyterHub brings the power of notebooks to groups of users. It gives users access to computational environments and resources without burdening the users with installation and maintenance tasks. Users - including students, researchers, and data scientists - can get their work done in their own workspaces on shared resources which can be managed efficiently by system administrators." [35]

For this project we will build an API that creates an Amazon EMR cluster that includes an installation of JupyterHub.

14.3 IMPLEMENTATION

14.3.1 Setting up AWS CLI

After setting up an AWS account account: [AWS Account](#) and an [AWS](#)

[Key Pair](#), we needed to be able to work with AWS products from the command line. To do this we utilized the AWS Command-Line Interface (CLI) from a Linux environment.

First we set up a [Linux](#) environment using VirtualBox. We then installed [Python](#) and [PIP](#) on that environment. Finally we installed CLI using the following Bash command:

```
$ pip install awscli
```

The following item had to be configured for CLI:

- AWS Access Key ID
- AWS Secret Access Key
- Default region name (this is the default region that will be used when you create EC2 instances)
- Default output format (the default format is json)

14.3.2 Setting up AWS Admin Access

In order to work from the command line with various AWS products we had to set up admin access. Using CLI we ran the following commands:

```
$ aws iam create-group --group-name Admins
```

```
$ aws iam attach-group-policy --group-name Admins --policy-arn arn:aws:iam::aws:policy/AdminsPolicy
```

Then through the [AWS Console](#) we assigned users to the admin group. Under 'Group Actions', we selected 'Add Users to Group'.

Figure 45 shows the AWS Console screen for adding users to a admin security group

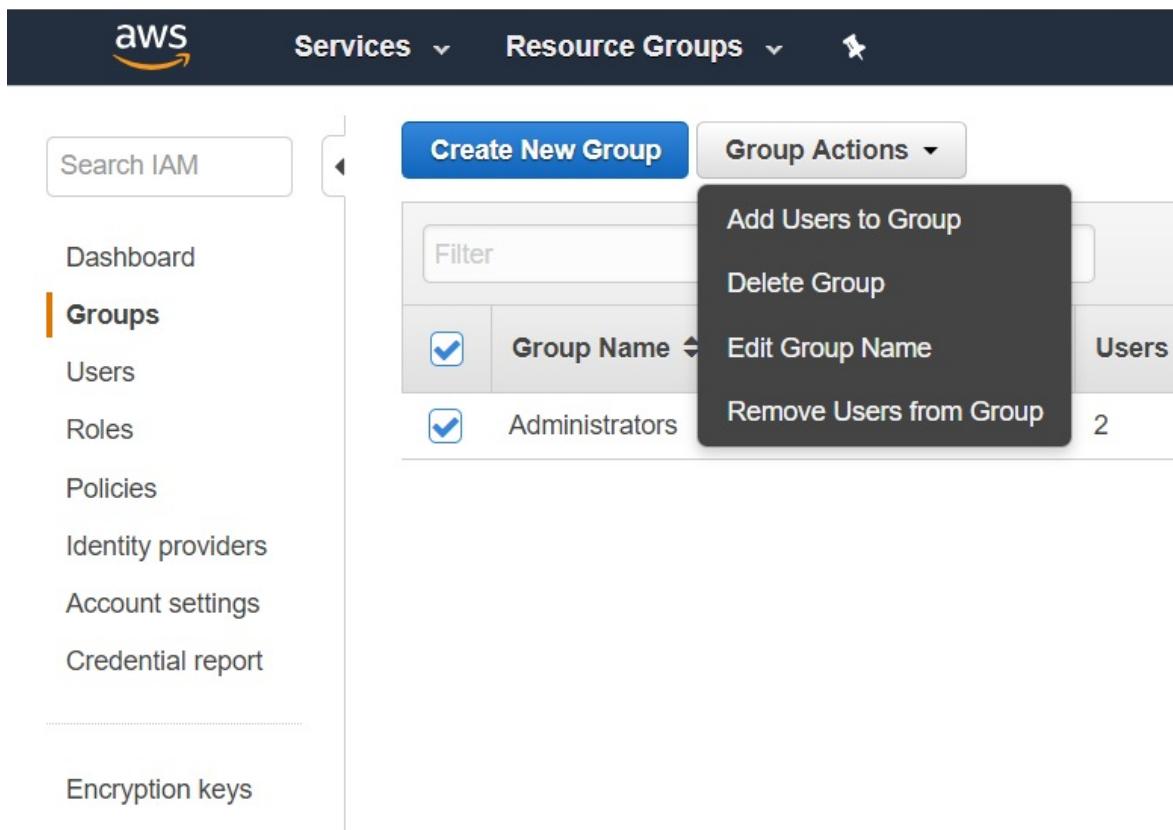


Figure 45: AWS Security [36]

14.3.3 Creating and Configuring EC2 Instance to Host API

14.3.3.1 EC2 Security Group

To set up the EC2 instance for hosting our APIs we first used the Amazon Console to set up a security group.

Navigating to: [EC2 Security Group](#) we selected 'Create Security Group'

Figure 46 shows the screen to create an AWS security group

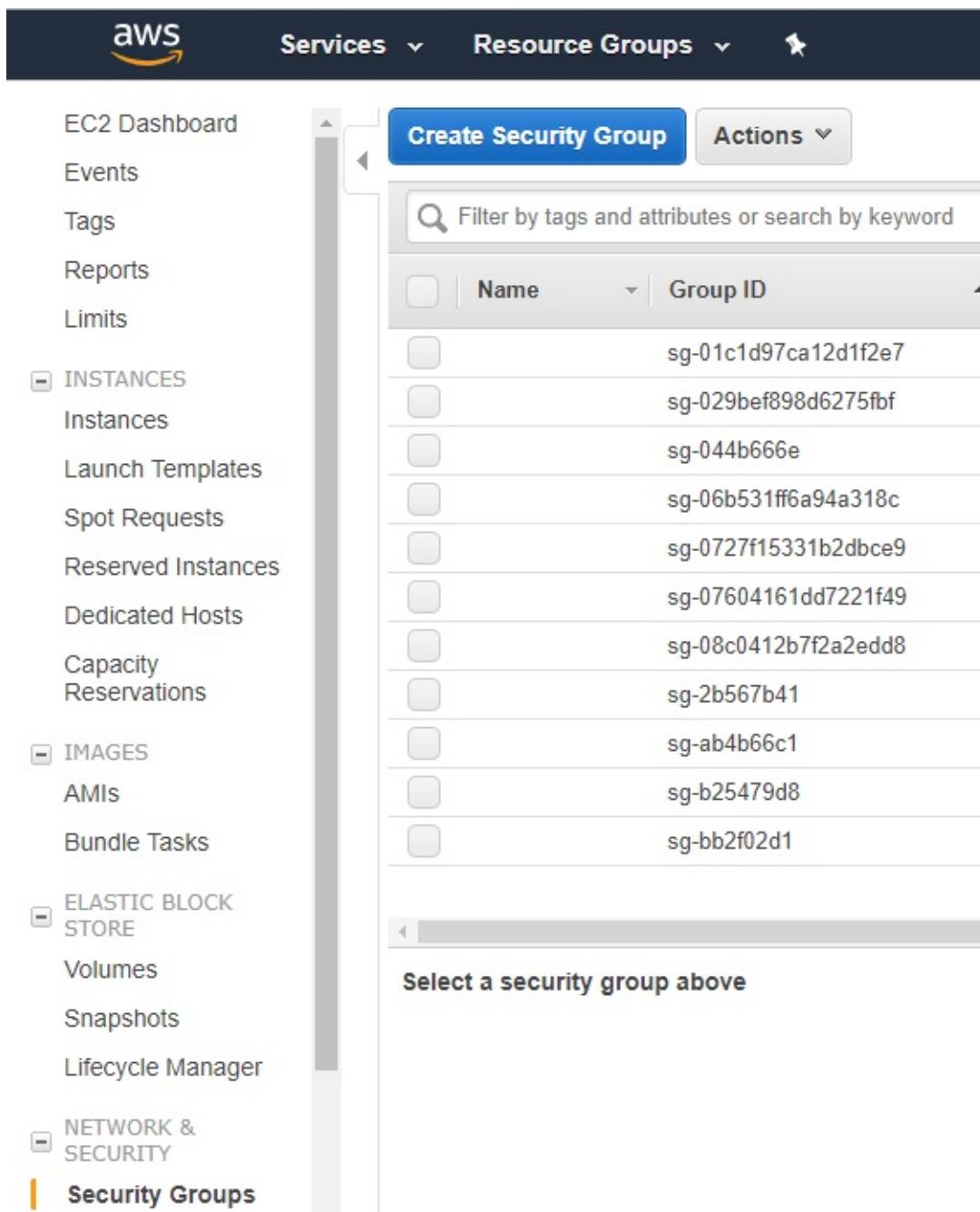


Figure 46: AWS Security [37]

We then gave the security group a name, selected the default VPC and added two rules. One that opens ports 8080, 8081, and 8082 for http traffic and one to allow ssh access from a single ip. Ports 8080, 8081, and 8082 will be used for accessing the APIs.

Figure [47](#) shows the AWS screen for defining a security group

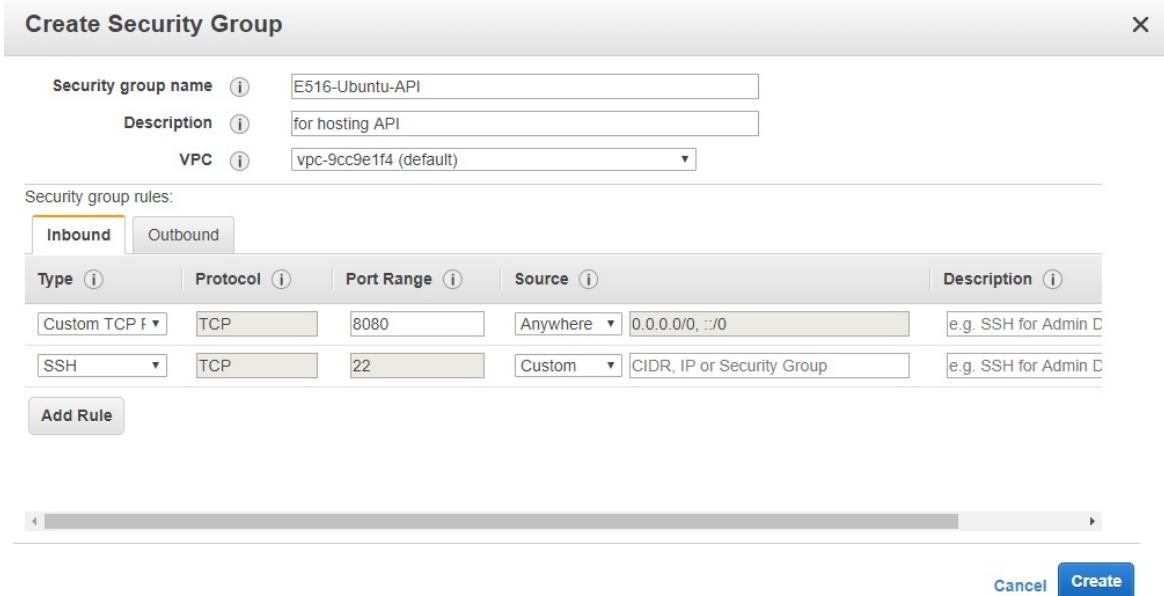


Figure 47: AWS Security ???

14.3.3.2 EC2 Create Instance

Now it was time to create an EC2 instance using the AWS Console: [Launch EC2](#). We clicked the 'Launch Instance' button.

Figure 48 shows the AWS Console screen for launching an EC2 instance

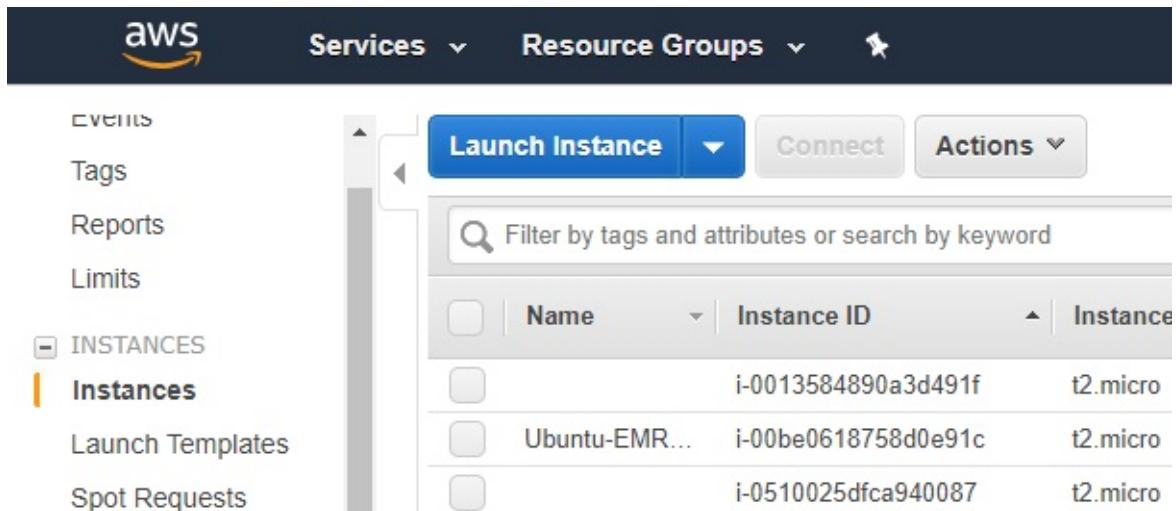


Figure 48: AWS EC2 [38]

We selected the Ubuntu version, using version 18.04:

Figure 49 shows the AWS screen used for selecting an EC2 operating system

Step 1: Choose an Amazon Machine Image (AMI)

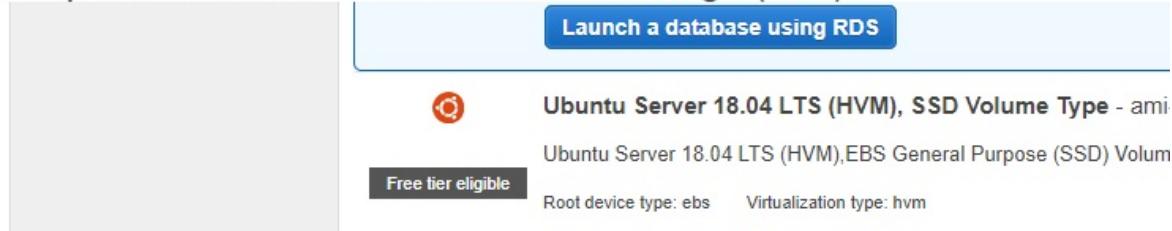


Figure 49: AWS EC2 OS [38]

We selected a small instance type and went to "Next: Configure Instance Details.

Figure 50 shows the AWS Console screen for selecting the type of instance

The screenshot shows the 'Step 2: Choose an Instance Type' section of the AWS EC2 console. The top navigation bar includes 'Services' and 'Resource Groups'. Below the navigation, a progress bar shows steps 1 through 5: '1. Choose AMI', '2. Choose Instance Type' (which is underlined in yellow), '3. Configure Instance', '4. Add Storage', and '5. Add...'. The main area has a heading 'Step 2: Choose an Instance Type' and a sub-instruction: 'Amazon EC2 provides a wide selection of instance types optimized to fit different use cases. It's a mix of resources for your applications. Learn more about instance types and how they can me...'. Below this is a table with columns: 'Filter by:' (set to 'All instance types'), 'Current generation' (set to 'Current generation'), and 'Show/Hide Columns'. A note says 'Currently selected: t2.micro (Variable ECUs, 1 vCPUs, 2.5 GHz, Intel Xeon Family, 1 GiB rr...'. The table lists instance types:

| | Family | Type | vCPUs |
|-------------------------------------|-----------------|--------------------------------|-------|
| <input type="checkbox"/> | General purpose | t2.nano | 1 |
| <input checked="" type="checkbox"/> | General purpose | t2.micro Free tier eligible | 1 |

Figure 50: AWS EC2 Type [38]

We made sure the default VPC is selected and then went to 'Configure

Security Group'.

Figure 51 shows the AWS Console screen for configfirguring security on an EC2 instance

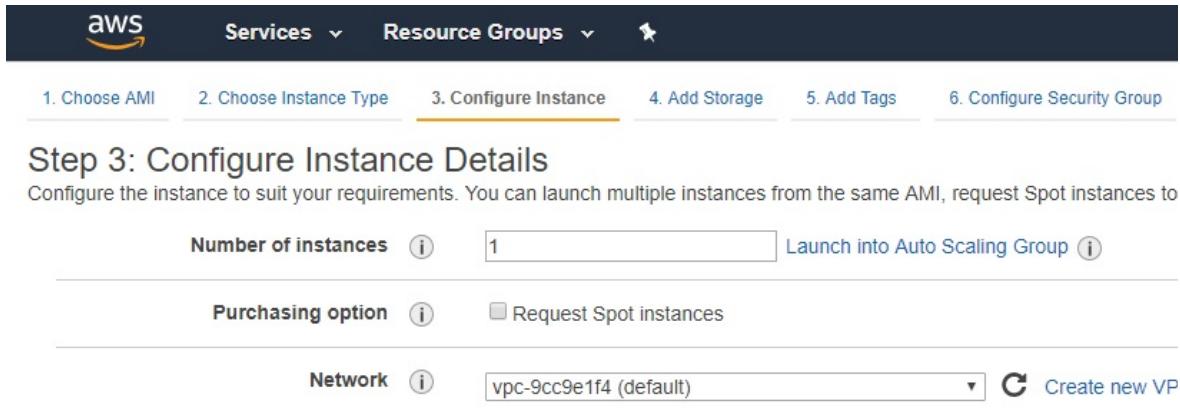


Figure 51: AWS EC2 Security Config [38]

We clicked 'Select and existing security group' and selected the group created earlier.

Figure 52 shows the AWS Console screen for selecting a security group for EC2

The screenshot shows the AWS EC2 instance creation wizard at Step 6: Configure Instance. The top navigation bar includes 'Services' and 'Resource Groups'. Below the steps, the current step is 'Step 6: Configure Security Group'. A descriptive text explains that a security group is a set of firewall rules that control traffic for your instance. It provides options to 'Create a new security group' or 'Select an existing security group'. The 'Select an existing security group' option is selected. A table lists existing security groups:

| Security Group ID | Name | |
|------------------------------|---|----|
| sg-ab4b66c1 | default | d |
| sg-07604161dd7221f49dlec2-sg | | S |
| sg-01c1d97ca12d1f2e7 | E516-EMR-Master | F |
| sg-08c0412b7f2a2edd8 | ElasticMapReduce-master | M |
| sg-06b531ff6a94a318c | ElasticMapReduce-slave | S |
| sg-b25479d8 | launch-wizard-1 | la |
| sg-2b567b41 | launch-wizard-2 | la |
| sg-044b666e | Neo4j Graph Database - Community Edition-3-4-1-AutogenByAWSMP-T | |
| sg-bb2f02d1 | ssh_http | S |
| sg-0727f15331b2dbce9 | Ubuntu-EC2 | U |

Figure 52: AWS EC2 Security Select [38]

The EC2 instance could then be launched.

14.3.3.3 EC2 SSH

We then went into to a local Linux environment and set up a key pair to enable ssh to our EC2 instance. We did this using CLI and the following commands:

```
$ aws ec2 create-key-pair --key-name dlec2-key --query 'KeyMaterial' --output text > dlec2-key.pem
```

Allowed access to the key:

```
$ chmod 400 dlec2-key.pem
```

Then locating the ‘Public DNS’ at: [AWS EC2](#), we connected to the EC2 instance with the following command:

```
$ ssh -i "dlec2-key.pem" ubuntu@ec2-18-191-50-79.us-east-2.compute.amazonaws.com
```

14.3.3.4 EC2 Set Up

We then set up a Python virtual environment for our rest services:

```
$ pyenv install -1  
$ pyenv install 3.6.6  
$ pyenv virtualenv 3.6.6 RestService  
$ pyenv activate RestService
```

Installed AWS CLI

```
$ pip install awscli
```

The following items had to be configured for CLI:

- AWS Access Key ID
- AWS Secret Access Key
- Default region name (this is the default region that will be used when you create EC2 instances)
- Default output format (the default format is json)

14.3.4 Create S3 Storage

In order to store analytical data and to backup our Jupyter notebooks We created two S3 buckets using the following commands:

```
$ aws s3 mb s3://e516-analytical-datasets --region us-east-2  
$ aws s3 mb s3://e516-jupyter-backup --region us-east-2
```

14.3.5 Codegen Set Up

14.3.5.1 Install Java

We used Codegen to create our rest services and Java is a requirement. We installed Java using the following commands:

```
$ sudo apt update
$ sudo apt install default-jre
$ sudo apt install default-jdk
```

14.3.5.2 Install Codegen

We ran the following commands for installation:

```
$ mkdir ~/e516/swagger
$ cd ~/e516/swagger
$ wget https://oss.sonatype.org/content/repositories/releases/io/swagger/swagger-codegen-cli-2.3.1.jar
```

We then opened the .bashrc file and added an alias for codegen:

```
alias swagger-codegen="java -jar ~/e516/swagger/swagger-codegen-cli-2.3.1.jar"
```

14.3.6 Building the EMR Rest Service

14.3.6.1 Swagger YAML Specs

Using Swagger we built my API specs. This API has POST, DELETE, and GET methods. The POST method will create an AWS EMR cluster and install JupyterHub. The DELETE method allows for the termination of the cluster. The GET method retrieves information about the cluster including the status and a link to the Jupyter Hub web ui.

```
swagger: "2.0"
info:
  version: "0.0.1"
  title: "emrinfo"
  description: "API to spin up an AWS EMR cluster, to check status, and to terminate."
  termsOfService: "http://swagger.io/terms/"
  contact:
    name: "EMR Rest Service"
  license:
    name: "Apache"
host: 18.191.50.79:8080
basePath: /api
schemes:
  - http
consumes:
  - "application/json"
produces:
  - "application/json"
paths:
  /emr/create/{num_of_nodes}:
    post:
```

```

summary: Create EMR cluster.
parameters:
  - in: path
    name: num_of_nodes
    required: true
    type: integer
    minimum: 1
    description: The number of nodes in the cluster
responses:
  200:
    description: OK
/emr/info/{cluster_id}:
get:
  summary: Returns EMR cluster Info.
  parameters:
    - in: path
      name: cluster_id
      required: true
      type: string
      minimum: 1
      description: The cluster id for EMR
  responses:
    200:
      description: OK
/emr/terminate/{cluster_id}:
delete:
  summary: Deletes EMR cluster.
  parameters:
    - in: path
      name: cluster_id
      required: true
      type: string
      minimum: 1
      description: The cluster id for EMR
  responses:
    200:
      description: OK
definitions:
  EMR:
    type: "object"
    required:
      - "model"
    properties:
      model:
        type: "string"

```

14.3.6.2 Deploy EMR Rest Service

As mentioned, we used Swagger Codegen for the creation of our APIs. Once our swagger yaml files was cloned to our EC2 instance we ran the following code to create the needed files:

```
$ swagger-codegen generate \
-i ~/fa18-516-22/project-code/emr-api.yaml \
-l python-flask \
```

```
-o ~/emr-api/server/emr/flaskConnexion \
-D supportPython3=true
```

We then navigated to our controller file and edited it using nano:

```
$ cd ~/emr-api/server/emr/flaskConnexion/swagger_server/controllers
$ nano default_controller.py
```

We then updated our POST, DELETE, and GET methods with the Python functions we created. The POST method accepts a parameter indicating the number of instances desired in a cluster and will create an AWS EMR cluster, configure it and install JupyterHub. This includes setting up the security for the EMR cluster and specifying S3 buckets for it to interact with. It returns a dictionary that includes the created cluster's ID, a link to the GET method for checking the status of the cluster, and a curl command for executing the DELETE method for terminating the cluster.

```
import subprocess

def emr_post(num_nodes):

    aws_cmd = "aws emr create-cluster --name='E516-JupyterHub-Cluster'"
    aws_cmd = aws_cmd + " --release-label emr-5.19.0"
    aws_cmd = aws_cmd + " --applications Name=JupyterHub"
    aws_cmd = aws_cmd + " --log-uri s3://e516-jupyterhub-backup/JupyterClusterLogs"
    aws_cmd = aws_cmd + " --use-default-roles"
    aws_cmd = aws_cmd + " --ec2-attributes SubnetIds=subnet-d0169eaa,KeyName=dlec2-key,AdditionalInstanceTypes=m4.xlarge"
    aws_cmd = aws_cmd + " --instance-count " + str(num_nodes)
    aws_cmd = aws_cmd + " --instance-type m4.large"
    aws_cmd = aws_cmd + " --configurations '[{\"Classification\":\"jupyter-s3-conf\", \"Properties\": {}}]"
    aws_cmd = aws_cmd + " --output text"

    c_id = subprocess.run(aws_cmd, shell=True, stdout=subprocess.PIPE)

    cid = c_id.stdout.decode('utf-8')
    cid = cid.rstrip()

    c_status = "http://ec2-18-191-50-79.us-east-2.compute.amazonaws.com:8080/api/emr/info/"
    t_clstr = 'curl -X "DELETE" http://ec2-18-191-50-79.us-east-2.compute.amazonaws.com:8080' + c_status

    rtn_dict = {
        "ClusterId": cid,
        "CheckClusterStatus": c_status,
        "TerminateCluster": t_clstr
    }

    return rtn_dict
```

The DELETE method accepts an EMR cluster ID as an input and then terminates the specified EMR cluster. It returns a dictionary containing the cluster ID, the current status of the cluster, and the url for using the GET method to check the status of the cluster.

```
import subprocess

def emr_delete(cid):

    subprocess.run("aws emr terminate-clusters --cluster-ids " + cid, shell=True)

    c_status = ("http://ec2-18-191-50-79.us-east-2.compute.amazonaws.com:8080/api/emr/info")

    rtn_dict = {
        "ClusterId": cid,
        "Status": "TERMINATING",
        "CheckClusterStatus": c_status
    }

    return rtn_dict
```

The GET method allows a user to query information about the EMR cluster. The method accepts a cluster ID as an input and returns a dictionary containing the cluster ID, the cluster status, a link to access the JupyterHub UI, as well as the default username and password for JupyterHub.

This function utilizes the Python library ‘boto3’. This is a library created by Amazon to interact with AWS products with Python directly.

```
import boto3

def emr_get(cid):

    client = boto3.client('emr')
    c_info = client.describe_cluster(ClusterId=cid)

    c_status = c_info["Cluster"]["Status"]["State"]
    if "MasterPublicDnsName" in c_info["Cluster"]:
        j_hub = ("https://" + c_info["Cluster"]["MasterPublicDnsName"] + ":9443")
        j_un = "jovyan"
        j_pw = "jupyter"
    else:
        j_hub = ""
        j_un = ""
        j_pw = ""
```

```

rtn_dict = {
    "ClusterId": cid,
    "ClusterStatus": c_status,
    "JupyterHub": j_hub,
    "JupyterUN": j_un,
    "JupyterPW": j_pw
}

return rtn_dict

```

Once the default_controller.py file was updated, we activated this rest service using the following commands.

```

$ cd ~/emr-api/server/emr/flaskConnexion
$ pip install -r requirements.txt
$ python setup.py install
$ python -m swagger_server

```

The POST method could then be accessed using a curl command. In this case we will only specify two instances to be included in the cluster (the example output is incuded)

```

$ curl -X POST http://ec2-18-191-50-79.us-east-2.compute.amazonaws.com:8080/api/emr/create,
{
  "CheckClusterStatus": "http://ec2-18-191-50-79.us-east-2.compute.amazonaws.com:8080/api/e
  "ClusterId": "j-3A70IXQPD60HK",
  "TerminateCluster": "curl -X DELETE http://ec2-18-191-50-79.us-east-2.compute.amazonaws.co
}

```

The CheckClusterStatus url can then be used in a web browser to check the cluster's status.

Figure 53 shows the results of executing the GET method for the EMR API

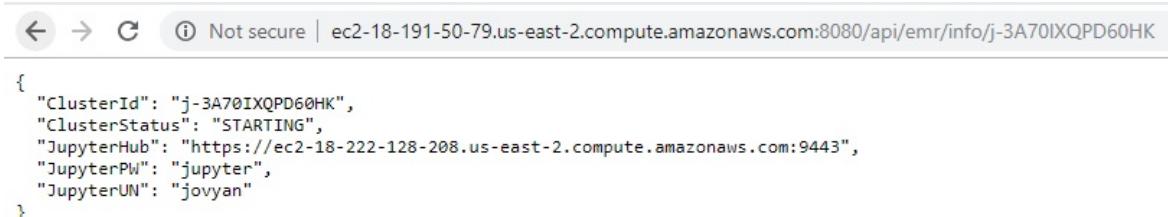


Figure 53: EMR API GET

Once the cluster is started, the JupyterHub web interface can be accessed through the given url. The EMR API configured the cluster to store created notebook to S3, so when the cluster is terminated the

notebooks will still be persisted. In addition, each time the POST method is used to create an EMR cluster any persisted notebooks will be available to the user in JupyterHub.

The DELETE method can be used in a curl command to terminate the cluster.

```
$ curl -X DELETE http://ec2-18-191-50-79.us-east-2.compute.amazonaws.com:8080/api/emr/term:  
{  
  "CheckClusterStatus": "http://ec2-18-191-50-79.us-east-2.compute.amazonaws.com:8080/api/e  
  "ClusterId": "j-3A70IXQPD60HK",  
  "Status": "TERMINATING"  
}  
[1]
```

Using the GET method shows that the cluster has been terminated

Figure [54](#) shows the results of executing the GET method for the EMR API

```
{  
  "ClusterId": "j-3A70IXQPD60HK",  
  "ClusterStatus": "TERMINATED",  
  "JupyterHub": "https://ec2-18-222-128-208.us-east-2.compute.amazonaws.com:9443",  
  "JupyterPW": "jupyter",  
  "JupyterUN": "jovyan"  
}
```

Figure 54: EMR API GET Results

14.3.7 Deploy S3 Rest Service

This rest service was meant to provide an abstraction layer that can be used by another rest service. This API provides a simple function to query the contents of an AWS S3 bucket and return the keys to any files that meet a specified file extension type. The Swagger specifications for this API are included below.

```
swagger: "2.0"  
info:  
  version: "0.0.1"  
  title: "s3info"  
  description: "API to query S3 bucket by file extension"  
  termsOfService: "http://swagger.io/terms/"  
  contact:  
    name: "S3 Rest Service"  
  license:
```

```

      name: "Apache"
host: 18.191.50.79:8081
basePath: /api
schemes:
  - http
consumes:
  - "application/json"
produces:
  - "application/json"
paths:
  /s3:
    get:
      summary: Get a list of files and paths with the given file extension
      parameters:
        - in: query
          name: bucket
          required: true
          type: string
          minimum: 1
          description: The S3 bucket name
        - in: query
          name: path
          required: true
          type: string
          minimum: 1
          description: The folder path to query
        - in: query
          name: extension
          required: true
          type: string
          minimum: 1
          description: The file extension to search for
      responses:
        200:
          description: OK
definitions:
  S3:
    type: "object"
    required:
      - "model"
    properties:
      model:
        type: "string"

```

Similar to the EMR-API we built this API using Swagger Codegen. The same steps were followed with one exception. The difference was the need to specify a different port to use for the API. There seems to be a bug in Codegen where the specified port in the yaml file is not used. Codegen defaults to always using port 8080. In order to change the port to allow for multiple APIs to run on the same EC2 instance, we ran the following code to replace the port number with our desired port (8081). This was done after Codegen was run the needed files were created.

```
$ cd ~/s3-api/  
$ find . -type f -exec sed -i 's/8080/8081/g' {} +
```

The default_controller.py file was then edited with the following Python code. This function used the Amazon ‘boto3’ package to search a specified S3 bucket and path for a given file extension. It then returns a list of all files that meet the specifications.

```
import boto3  
  
def search_s3_by_ext(bucket, path, extension):  
  
    client = boto3.client('s3')  
    obj = client.list_objects_v2(Bucket=bucket, StartAfter=path )  
  
    rtn_list = []  
    for object in obj['Contents']:  
        if object['Key'][-(len(extension)):] == extension:  
            rtn_list.append(object['Key'])  
  
    return(rtn_list)
```

The API could then be deployed using the following.

```
$ cd ~/s3-api/server/s3/flaskConnexion  
$ pip install -r requirements.txt  
$ python setup.py install  
$ python -m swagger_server
```

The API can be used to query a specified S3 bucket. In the example below we query the S3 bucket ‘e561-jupyter-backup’ and specify a path for a particular user ‘jupyter/jovyan’. We also specify to search for files with an extension of ‘.ipynb’ (notebooks). The service returns a list of files (including the path) that meet the specifications.

Figure 55 shows the results of executing the GET method for the S3 API



Figure 55: S3 API GET

14.3.8 Deploy Notebook Rest Service

Our final API is a service that allows for the searching of Jupyter notebook content. This API also connects to the S3 API as an abstraction layer.

Jupyter notebooks allow for custom tagging on individual 'cells'. These tags are saved with the notebook. An example of a 'tag' applied to an individual cell is given below. In this case the tag is called 'parameters'.

Figure [56](#) shows an example of a Jupyter notebook tag



The screenshot shows a Jupyter Notebook cell labeled 'In [15]:' containing Python code. A tooltip labeled 'parameters' is displayed over the code, highlighting three variables: 'mybucket', 'path', and 'file_extension'. The code is as follows:

```
mybucket = 'e516-jupyter-backup'
path = 'jupyter/'
file_extension = '.ipynb'
```

Figure 56: Notebook Tag

Once again we used Swagger for the API specifications. This service has a GET method that accepts parameters for an S3 bucket, a path, and a text field to search for in the notebooks. The Swagger specifications are provided below.

```
swagger: "2.0"
info:
  version: "0.0.1"
  title: "notebookinfo"
  description: "API to query jupyter notebook parameters"
  termsOfService: "http://swagger.io/terms/"
  contact:
    name: "Notebook Rest Service"
  license:
    name: "Apache"
host: 18.191.50.79:8082
basePath: /api
schemes:
  - http
consumes:
  - "application/json"
produces:
  - "application/json"
paths:
  /notebook:
    get:
```

```

summary: Get a list of files and paths with the given file extension
parameters:
  - in: query
    name: bucket
    required: true
    type: string
    minimum: 1
    description: The S3 bucket name
  - in: query
    name: path
    required: true
    type: string
    minimum: 1
    description: The folder path to query notebooks
  - in: query
    name: search_on
    required: true
    type: string
    minimum: 1
    description: The parameter text to look for
responses:
  200:
    description: OK
definitions:
  NOTEBOOK:
    type: "object"
    required:
      - "model"
    properties:
      model:
        type: "string"

```

Codegen was used to generate the needed files and the default_controller.py file was edited to include the following Python code. This function uses the Python library ‘requests’ to call our S3 API. The S3 API provides a list of .ipynb files in the specified bucket and path. We then loop through that list and search each cell to see if there is a ‘parameter’ tag. If there is a ‘parameter’ tag the specified search value is looked for in the cell’s source code. The function returns a list of all the unique notebooks that contain the specified text in any ‘parameter’ cell.

```

import boto3
import json
import requests

def search_nb_param_inpt_data(bucket, path, search_on):

  #call s3 API to get list of files including paths
  url = 'http://ec2-18-191-50-79.us-east-2.compute.amazonaws.com:8081/api/s3'
  payload = {'bucket': bucket, 'path': path, 'extension': '.ipynb'}
  r = requests.get(url, params=payload)

```

```

s3 = boto3.resource('s3')

#loop through all .ipynb under path
nb_has_val = False
nbs_found = []
for file_path in r.json():

    content_object = s3.Object(bucket, file_path)
    file_content = content_object.get()['Body'].read().decode('utf-8')
    json_content = json.loads(file_content)

    #loop through each cell of nb
    for cell in json_content['cells']:
        if 'source' in cell:
            if 'metadata' in cell:
                if 'tags' in cell['metadata']:
                    if 'parameters' in cell['metadata']['tags']:
                        if search_on in cell['source']:
                            nbs_found.append('s3://' + bucket + '/' + file_path)

rtn_nb_full_path_list = list(set(nbs_found))

return(rtn_nb_full_path_list)

```

Again we needed to specify a different port (8082) with the following code.

```

$ cd ~/s3-api/
$ find . -type f -exec sed -i 's/8080/8082/g' {} +

```

The API could then be deployed using the following.

```

$ cd ~/notebook-api/server/notebook/flaskConnexion
$ pip install -r requirements.txt
$ python setup.py install
$ python -m swagger_server

```

The API can be used to query the contents of Jupyter notebooks. In the example below we query the S3 bucket 'e561-jupyter-backup' and specify a path of 'jupyter'. We also specify to search the parameter cells for 'test1/iris.csv'. This text relates to a dataset that is stored on S3. In this way a user could query a set of analytical notebooks to see who is performing analysis using a particular dataset. The service returns a list of files (including the path) that meet the specifications.

Figure 57 shows the results of executing the GET method for the Notebook API

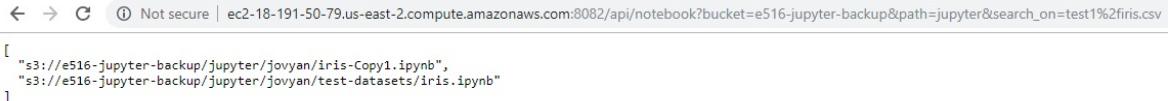
A screenshot of a web browser window. The address bar shows a URL starting with "ec2-18-191-50-79.us-east-2.compute.amazonaws.com:8082/api/notebook?bucket=e516-jupyter-backup&path=jupyter&search_on=test1%2firis.csv". The main content area displays a JSON response: ["s3://e516-jupyter-backup/jupyter/jovyan/iris-Copy1.ipynb", "s3://e516-jupyter-backup/jupyter/jovyan/test-datasets/iris.ipynb"]

Figure 57: Notebook API GET

14.3.9 Running all APIs

In order to have all three of these Rest services running on the same EC2 instance. The following commands were run. The "&" returns to the prompt after each service is deployed.

```
$ cd ~/emr-api/server/emr/flaskConnexion  
$ python -m swagger_server &  
$ cd ~/s3-api/server/s3/flaskConnexion  
$ python -m swagger_server &  
$ cd ~/notebook-api/server/notebook/flaskConnexion  
$ python -m swagger_server &
```

The services can be terminated with the following command.

```
$ pkill -f swagger
```

14.4 CONCLUSION

This project results in a usable Jupyter notebook environment that is scalable over time. The intial APIs lay a foundation for a robust notebook environment in which a user can create analytical processes and search for existing ones.

15 TBD

FA18-516-24

github: [cloud911](#)

16 HADOOP, HIVE AND SPARK MULTI NODE CLUSTER SET UP ON AMAZON EC2 INSTANCES

please see format from our example

references missing

obviously your report is an early draft. You need to explain what you do not in person to me, but in written form in this document

you need to have shell scripts that start the vms, you can leverage cm4 if you like or design your own shell script while for example using the AWS CLI

you need to have a mechanism on telling us how you set up the cloud. Science is about reproducability, If your hadoop and spark implementation can not be deployed by others, they can not be reproduced. Thus we can not replicate your benchmark. This is simple, just write a shell script on how to set up hadoop on your 4 machines.

If this can not be done. Write a manual on how to do this including screenshots where needed.

github: [cloud9](#)

16.1 ABSTRACT

The goal of this project is to demonstrate the steps needed to set up a 4 node Hadoop Cluster with Spark and Hive on Amazon EC2 instances from the scratch and do comparison between the traditional mapreduce and Distributed Computing through Spark. It details the Hadoop and Spark configurations required for a 4GB and 8GB(memory) node Hadoop cluster for developmental and testing purpose. The objective is to do all the installations and configurations from the scratch. This process will be the same as doing the

installation and set up on any 4 unix machines which have a static IP.

This should work on arbitrary numbers of VMs

how does your performance compare to EMR

headings must not be all caps

We will compare the processing times between mapreduce computation and spark computation and prove how spark is much faster than mapreduce.

how does it stack up with EMR?

We show how to integrate spark as a compute engine in hive and bypass mapreduce in hive.

Lastly, we explore a sorting technique called secondary sort where we sort values in large data files by making use of map reduce framework without bringing all the data in memory of one node in a distributed environment.

16.2 KEYWORDS

Amazon EC2, Hadoop, Spark, Hive

16.3 INTRODUCTION

The project describes what are the minimum configurations required for a multi node Hadoop cluster set-up with Spark and Hive in AWS EC2 instances and how to establish a passwordless ssh connection between the instances. This is applicable for any unix/linux instances which can have a static IP and the same steps need to be followed for establishing a passwordless ssh connection and configuring the hadoop and Spark config files. The recommended amount of memory for an instance is 8GB and at least 20GB of physical disk space.

16.4 SOFTWARE VERSIONS

- Hadoop 2.9.1
- Hive 2.3.3
- Spark 2.3.2

16.5 ARTIFACTS

links missing

- Project Proposal
- Project Code

16.6 REFERENCES

17 OPENFAAS & DOCKER ON A RASPBERRY PI MULTI-NODE CLUSTER WITH CM-BURN REPORT FA18-516-23

Anand Sriramulu

asriram@iu.edu

Indiana University

hid: fa18-516-23

github: [cloud](#)

code: [cloud](#)

code: [cloud](#)

Keywords: Cloud, Cluster, Raspberry Pi, Kubernetes

17.1 ABSTRACT

Creating a Multi-Node cluster with 3 Raspberry PIs and run OpenFaas and Docker Swarm

17.2 INTRODUCTION

17.3 REQUIREMENTS

17.4 DESIGN

17.5 ARCHITECTURE

17.6 DATASET

17.7 IMPLEMENTATION

17.8 BENCHMARK

17.9 CONCLUSION

17.10 ACKNOWLEDGEMENT

18 MORPHOLOGICAL IMAGE-BASED PROFILING OF SKIN LESIONS FOR SCIENTIFIC COMMUNITY FA18-523-52

Anna Heine

avheine@iu.edu

Indiana University

hid: fa18-523-52

github: [blue icon](#)

code: [blue icon](#)

Keywords: fa18-532-52, lesion, medical

18.1 ABSTRACT

One major area that is being utilized highly today within big data platforms is that of medical image collection. Medical image datasets are often used as training tools as a source of comparison when analyzing diagnostic images from current or past cases of diseases. The issue in this field is that current datasets lack the diversity and number of samples that is necessary to make sufficient predictions based on analysis. With valuable datasets, medical advances can be made to deliver more personalized medicine, to create predictive diagnostic models, research treatment methods, improve the overall value of healthcare, and much more. The main goal of medical big data analysis is to find associations and correlations within complex data. The HAM10000 dataset is a significant resource for analysis because it includes over 10,000 dermatoscopic images that have been carefully stained and optimized to display skin-lesion biopsies. The grafts entered into this dataset were verified and therefore proven worthy based on expert consensus, a follow-up, and in-vivo microscopy. Each individual donation is tracked by its patient identification number, image identification number, donor age, donor sex, localization on the body, and a final inclusion reasoning.

18.2 INTRODUCTION

The big data revolution has changed multiple industries around the world, one of which largely includes the field of medicine. The role of big data in the medical sector is high-grade as it aims to be predictive in order to diagnose patients and even discover new treatment methods. This means that the data obtained and used in these models must be wide enough to include a variety of patients and diseases. Disease is a major unknown for many reasons in medicine. Big data has recently been shown to be beneficial in disease management as it provides aggregate information around multiple aspects of a disease. For example, some datasets include functionality and characteristics of DNA and RNA, proteins, cell types, tissues, organs, and more [39]. The ability for these models to evolve and grow are what will advance the predictive analysis so desired in the field today. The sources of big data are varied. Some of which include administrative claim records, health records, the internet, medical imaging, and clinical trials [40].

Despite the vast number of sources, medical big datasets can be quite different than other types of big data. The main difference is located in the form of accessing the data. Because of privacy laws and ethical stances, some medical data is hard to come by. The fact that data has been breached in other industries such as in business and consumerism models, most people are hesitant to include large amounts of people's private health records in one location. That being said, medical datasets have to be managed so that the records being added follow a specific, structured protocol. These datasets are also costly, as they may need to be examined and measured multiple times by trained personnel to ensure correctness. The nature of the data is therefore, dangerously susceptible to human error, as most of it usually contains information we are not very sure about to begin with. Although data scientists have found methods and reasonings that allow for the production of medical datasets, they are often unreliable because the amount of data contained is just too small to make any valuable inferences or claims.

Though there are many challenges medical big data will need to dissolve in order to gain popularity and trust, the field is already proven to hold valuable insights. For example, the field will expand the use of big data into other industries by providing a basis for accumulating data of various sources and materials. Medical big data can provide answers to uncertainties even when lacking substantial evidence. The influx of continued large medical datasets is contributing to the advance of trusted future predictive healthcare learning models. The overall goal is not to automate the position of a trained physician, but to make the diagnostic process much easier and more efficient as well as aid in several areas of medical research.

The realm of skin cancer research is more or less crying out for big data analysis. Melanoma, specifically, affects around 73,000 new people each year which will result in about 9,000 deaths [41]. There is no general biomarker for melanoma, which causes for imprecise diagnostic margins. With this prevalence of disease in society, the amount diagnoses per melanoma skin lesion sometimes reaches up to 36 because of false-negative uncertainties [41]. However, with big datasets and the use of computer algorithms, there has been a significant increase in diagnostic accuracy -less than 5% error rates.

18.3 REQUIREMENTS

This project requires Python 2.7 or greater to integrate the Pandas and NumPy packages. It requires at least 420 KB of storage to hold the HAM10000 dataset. The project also requires the Anaconda platform to access Jupyter Notebook with Numpy and Pandas packages installed. The cloud service analytics were performed on KNIME's Cloud Analytics Platform.

18.4 DESIGN

The design of the project was to obtain the dataset and test it on a web services. First, we tested the dataset on Jupyter Notebook to get a baseline of the components. After creating some visualizations, we

then incorporated the data into KNIME's Analytics Cloud Platform.

18.5 ARCHITECTURE

Before we made any implementations on this project, we realized that we would be using multiple software packages. Therefore, we wanted to ensure that they were all in the same place. To do this, we installed Anaconda. Anaconda is an open-source, free distribution software with Python and R available packages already installed. Its main purpose is to provide simple tracking of these packages with an easy, user-friendly dashboard design[42]. Each version of its packages are managed by its package management system, Conda. In Anaconda, we used Jupyter and Jupyter Notebook to perform most of my data analysis. Jupyter allows for data manipulation across many programming languages. Jupyter is also an open-source web application that allows for distribution of documents, code, and other projects for collaboration. It has many uses: data cleansing, statistical data modeling, visualization, machine and deep learning, data transformation, and more. Once a project has been created, your work can be output as HTML, images, video, or LaTeX[43]. The Notebook specifically can contain both code and text. These formats have the ability to create descriptions and visible output for graphs of many types. Jupyter runs via client-server, which means it does not need Internet access to be run. Each notebook contains a kernel which controls the execution of the code inside it. For example, if you wanted to execute Python code, the notebook would execute via the ipython kernel. To manually execute a file, the user must either choose to run each cell one at a time by clicking Run on the left of each line or by clicking on Run All in the Cell menu[44].

NumPy is a package that interacts with Python to provide numerical operations. NumPy uses its own defined library to make things like arrays and matrices and invoke operations on them. The most basic object in NumPy is the ndarray object. The ndarray holds arrays of homogeneous data types which are compiled for efficiency. NumPy arrays have a fixed size array and are homogeneous, which is different from regular Python arrays. NumPy also has vectorized and

broadcasting behavior which both work to amplify performance and decrease compile time. Vectorized code is simple and easy to read code that typically is prone to less bugs. The code sometimes is mistaken for mathematical notations, but of course, results in Python-looking code. Broadcasting describes the step-by-step behavior of operations. It is beneficial in taking the outer operation of two arrays to make a combined array. However, both arrays must be of same dimension[45].

Pandas is a Python package also automatically downloaded with Anaconda. It provides many data analysis features that are widely used in data visualization. Pandas is able to incorporate many types of data formats as well. For example, Pandas can read in tabular data from SQL or Excel, it can obtain ordered or unordered data, arbitrarily matrixed data, and it can even read in data that has no labels. Pandas is able to handle missing values and also non-floating point data by labeling it as NaN. One important feature we specifically used was to convert data that incorporated NumPy structures into DataFrame objects. This feature allowed Pandas to easily read and control the data into an acceptable format that was able to successfully create a readable graph. Another convenient tool we used while manipulating my dataset was Pandas ability to slice and create subsets of my data. This allowed me to create new tables and graphs from carefully selected data where we saw possible correlations. To create detailed visualizations, Pandas incorporates matplotlib API [46]. Matplotlib allows the import of visualization libraries that can be read by Pandas. Matplotlib is a 2D plotting library that can be used in Python scripts and the Jupyter notebook, which we have previously mentioned. Some examples of the types of plots matplotlib includes is histograms, bar charts, scatterplots, errorcharts, power spectra, and more[47].

18.6 INSTALLATION

To install Jupyter itself you must already have Python 2.7 or Python 3.3 or greater. It is recommended to go ahead and download Anaconda, like we did, so that all of your packages are in one place.

To download the latest version of Anaconda, follow the code below:

```
import webbrowser  
webbrowser.open('https://www.anaconda.com/download/')
```

From there, you will have to choose which operating system to download from. That is all you have to do to install Anaconda. You know have an open platform with many packages for use. One of these packages is Jupyter Notebook. To run Jupyter Notebook the following line into the command prompt:

```
jupyter notebook
```

Now you can begin using Jupyter's Notebook to create visuals and write code for that manipulates your data. NumPy and Pandas are also automatically downloaded with the latest version on Anaconda.

18.7 DATASET

The dataset we have chosen is often used in training tools for medical professionals and is one of the only few available skin lesion datasets. The HAM10000 (Human Against Machine with 10000 training images) dataset contains dermatoscopic images from different populations that include all general diagnostic categories that have been discovered in this type of medicine. The diagnostic categories in this dataset include diseases such as: Bowen's disease (akiec), basal cell carcinoma (bcc), benign keratosis- like lesions (bkl), dermatofibroma (df), melanoma (mel), melanocytic nevi (nv), and vascular regions (vasc). The confirmation of the samples that were entered into the dataset are given: histopathology (histo), follow-up examination (follow_up), expert consensus (consensus), or confirmation via in-vivo confocal microscopy (confocal). Each image within the dataset can be tracked by their lesion-id [48].

18.8 IMPLEMENTATION

The first part in analyzing the HAM10000 dataset is to acquire it as well as the Anaconda platform. Once you access the Jupyter

Notebook and import the necessary Python packages, you are ready to begin analyzing the data. Jupyter is a great tool to use in data analysis because you can easily manipulate your code line-by-line. By performing multiple plotting algorithms, you get a great visualization of relationships amongst the dataset. Of course, there are simple methods to analyze singular features and columns in your dataset. The method below shows multiple statistical analyses on the age feature.

```
db['age'].describe()

count    9919.000000
mean     52.067749
std      16.686741
min      5.000000
25%     40.000000
50%     50.000000
75%     65.000000
max     85.000000
```

The first algorithm that was used in my Jupyter-based script was a DataFrame comparison between the localization and age features. This comparison shows the number of patients who recorded diagnosed skin lesions for different locations in addition to their differing ages. The Pandas DataFrame is a class that has the ability to take in a mutable, two-dimensional data structure that contains labeled axes[46]. It is known as the primary Pandas data structure. There are many examples of methods that can be used with this structure on Pandas documentation. To configure a visual of this correlation, the following code was imposed:

```
#name the database file to your liking after completed download
db=pd.read_csv('534data.csv')
df3 = pd.DataFrame(np.random.randn(1000, 2), columns=['age', 'localization']).cumsum()
df3['localization'] = pd.Series(list(range(len(db))))
df3.plot(x='localization', y='age')
```

The generated plot is as shown below Figure [58](#) :

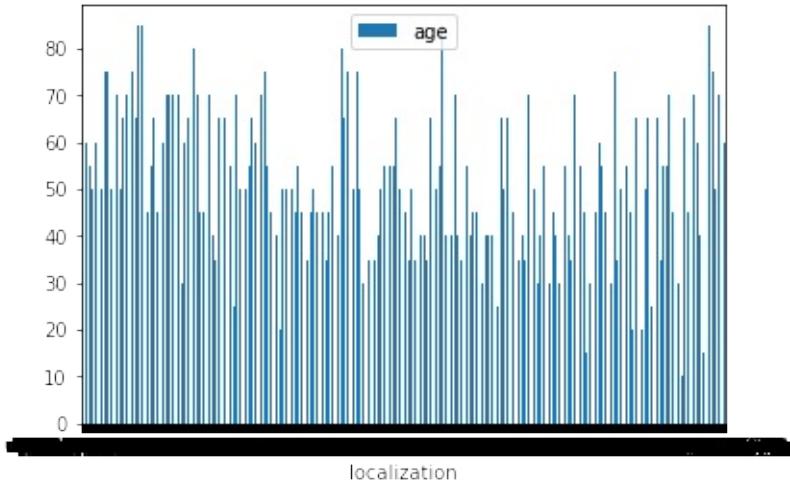


Figure 58: Correlation of localization and age

The visualization makes it obvious that there can be many different localizations per age group. This visualization also supports the need for varied datasets in the clinical domain. It ultimately makes it quite difficult to make assumptions on singular lesion samples because the locations are varied no matter the age.

Another algorithm that was imposed on Jupyter Notebook to create a visual analysis was an autocorrelation analysis. Autocorrelations are important in data analysis because they give important information regarding the quality of your dataset. Specifically, it checks randomness within your values which can also give clues to missing or empty values that you may have missed when cleaning the data. Over time, the data is compared to see if it lies near zero. If the data is considered random, the time series autocorrelation will be near zero for all time-lag separations. If the series is considered non-random, then the autocorrelations will be non-zero. The graph that is generated shows two horizontal lines that indicate 95% and 99% confidence bands. Using the following code, we have generated the following autocorrelation plot Figure 59.

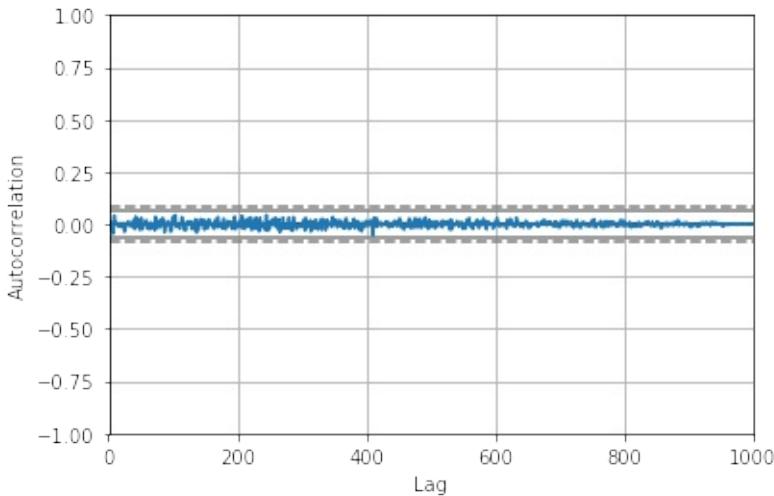


Figure 59: Autocorrelation

Since the values in the above autocorrelation plot are within the 95%, and some very close to the 99% correlation lines, it can be said that the dataset is far from random. This is promising because it reinforces the quality and trust within the dataset.

Another technical tool that we used to analyze my dataset was the KNIME Cloud Analytics Platform [49]. The KNIME cloud can be integrated via the Azure Marketplace or on Amazon AWS. KNIME stands for KoNstanz Information MinEr and is an open source data analytics software that creates services and applications for data science projects. KNIME allows its users to create visual workflows with a user-friendly drag and drop graphical interface that depletes the need for any programming. However, KNIME does allow implementation of other scripting languages such as Python [49] or R [49] that creates connections to abilities within Apache Spark or other machine learning tools. KNIME allows imports of datasets from a variety of formats, some of which include CSV [50], PDF [50], JSON [51] and more. The workflows and visualizations that KNIME produces allows export in many of these formats as well. It also supports several unstructured data types from images, documents, and certain networks. KNIME operates by a node system that includes embedded modules that help its users build their workflow. With this node system, users can make changes at every step of their analysis to ensure the most current version. KNIME also provides

detailed visualizations from a set of defined graphs and charts which can lead to predictive analyses and machine learning implementations. Users can shape their data by a variety of mathematical models such as statistical tests, standard deviations, and means. Users can even select specific features for use in possible machine learning datasets and apply filters to mark out some of the data if needed.

KNIME is a platform that can perform intense data analytics on a graphical user interface and incorporate a user-friendly workflow. It incorporates large or small data sets and even projects as broad as deep learning. KNIME is diverse in that its users do not necessarily need to know any coding languages to use it. KNIME is a process-oriented, single base workflow with basic input/output manipulations. KNIME is an open source platform that uses thousands of its documented nodes within the node repository for use in the KNIME workbench. A node is a single processing point of data manipulations within your workflow. A workflow is described as a sequence of steps a user follows in their platform that is used to complete their final product. The collection of nodes that creates a KNIME workbench is able to be executed locally or within the KNIME web portal on its own server. The workflow that KNIME follows first begins with data collection, data cleaning, data integration, and finally, feature extraction. This workflow allows for large files such as a CSV to be accessed through the web portal and it can therefore be manipulated through several wizards [52].

KNIME has the ability to be integrated with other technologies for larger open-source projects. These cloud services allow for user's projects to be analyzed even further. For example, KNIME can be used with Amazon AWS [53] and Azure [54]. KNIME's platform can be hosted on Microsoft Azure Cloud Services. Azure allows KNIME to perform its analytical, machine learning, and deep learning tasks on its integrated server. This application can be downloaded from Azure's Marketplace. KNIME can also be incorporated with Amazon AWS. When KNIME is connected to AWS resources, users can leverage the memory available while connected to the relational database

service to construct SQL queries visually. KNIME's Analytic Platform is a free service for all who use it. However, if you are using KNIME on a cloud service such as Azure or AWS, there are often subscription fees associated. Students and other organizations can receive discounts or allocated amounts for a specified time of use. KNIME is a data analysis software platform that allows for easy read and manipulation of large datasets that can ultimately be used to make inferences and predictions. Its user-friendly interface allows for a broad integration of users and sometimes more efficient workflows. KNIME has several applications for its users such as data modeling, machine learning, predictive analysis, and more. After visualization, users can extract specific features from their data and implement it into a model of their choice, which can then be exported as a CSV file.

Creating visualizations in the KNIME platform is extremely easy. The steps to create visual analyses on KNIME's analytic platform are to first drag and drop the CSV Reader node into the workflow. Once in the workflow, import the dataset from where you saved it locally by right-clicking on the CSV Reader node, and choosing Configure. Once the dialog box pops up, choose the number of rows and columns to include (we included them all). Then, choose which algorithm you would like to impose on the dataset by searching the node directory again. For my first analysis, we chose to do a basic graph showing the amount of different diagnostic descriptions per area (localization). The figure below shows this correlation, Figure [60](#).

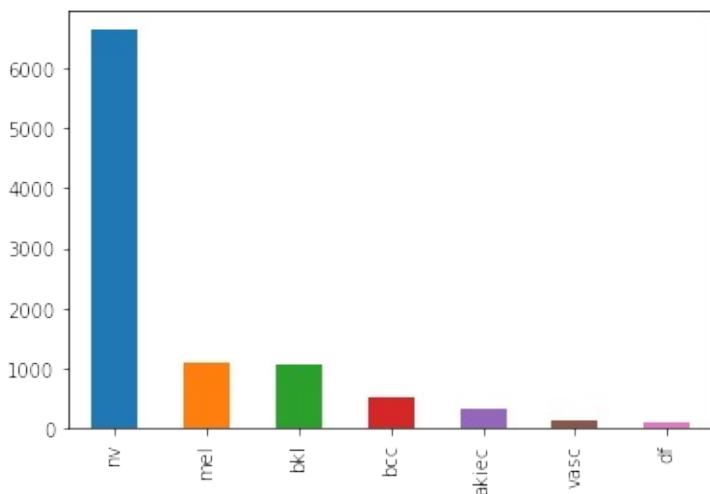


Figure 60: Descriptions by location

The above image shows proof that although some diseases are more relevant than others, it is important to have variable datasets such as these. Having so improves the accuracy of other diagnoses because of the increased comparison amongst individual units of data. It emphasizes that no matter the location, there can be many different types of apparent disease types. Another created image is the visual of density among three data features. We chose to generate this graph with all three features along separate axes. The figure below indicates these results Figure [61](#).

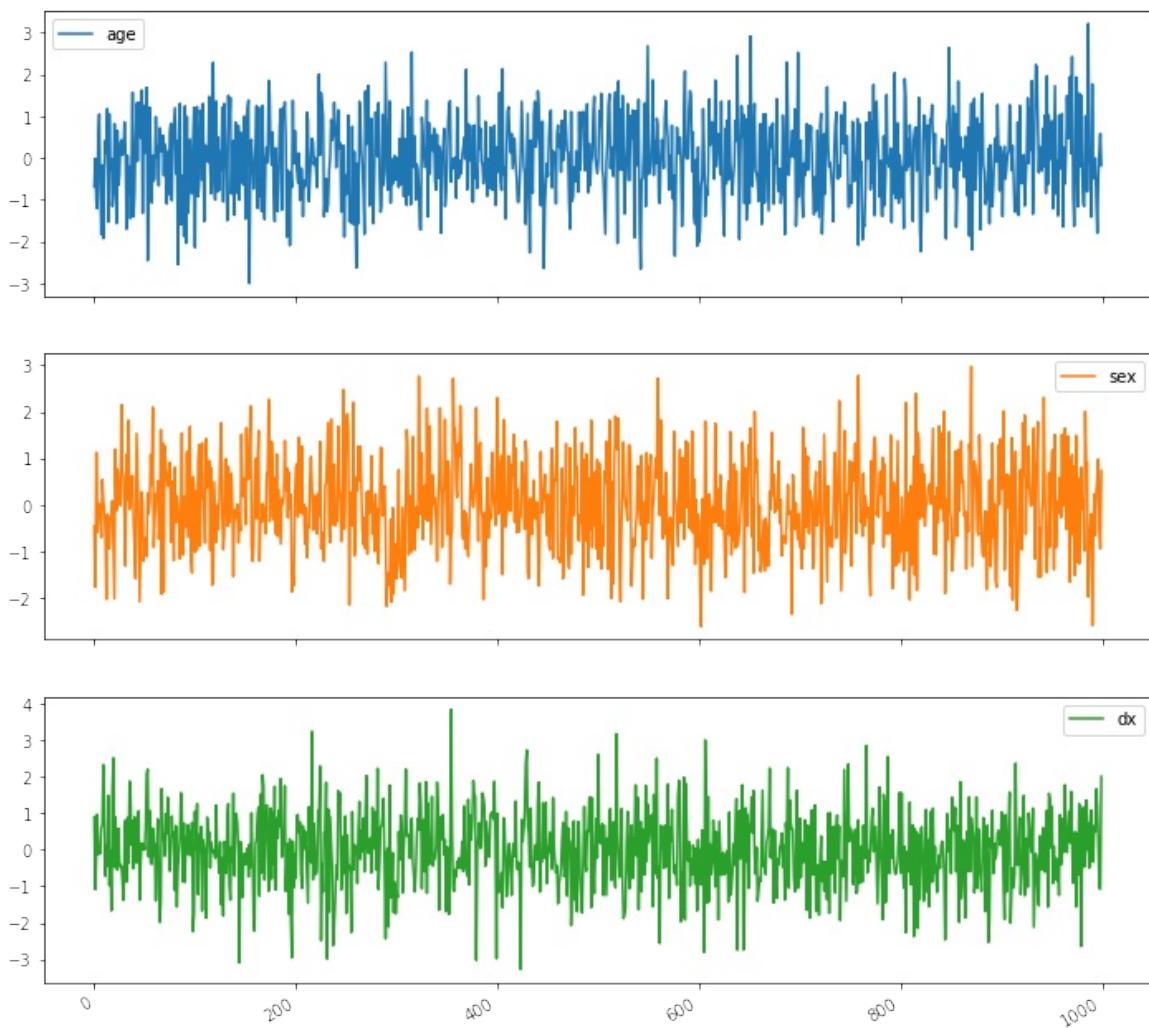


Figure 61: Subplots

Using KNIME's analytical platform, we was able to use defined

algorithms to further expand my visualizations. One algorithm we explored was that of parallel coordinates. Parallel coordinates is a graph that exemplifies the many differences between multiple variables and features. In the graph I've generated, the age and sex features are placed on their own axes. The lines that connect the two features together are the actual values of the two features. Each line can be seen as an accumulation of the points that lie upon each axis [55]. The downside of this algorithm is that sometimes the dataset can make the graph look somewhat dense. The figure below shows the parallel coordinates of age and sex within the HAM10000 dataset, which is also quite dense Figure [62](#).

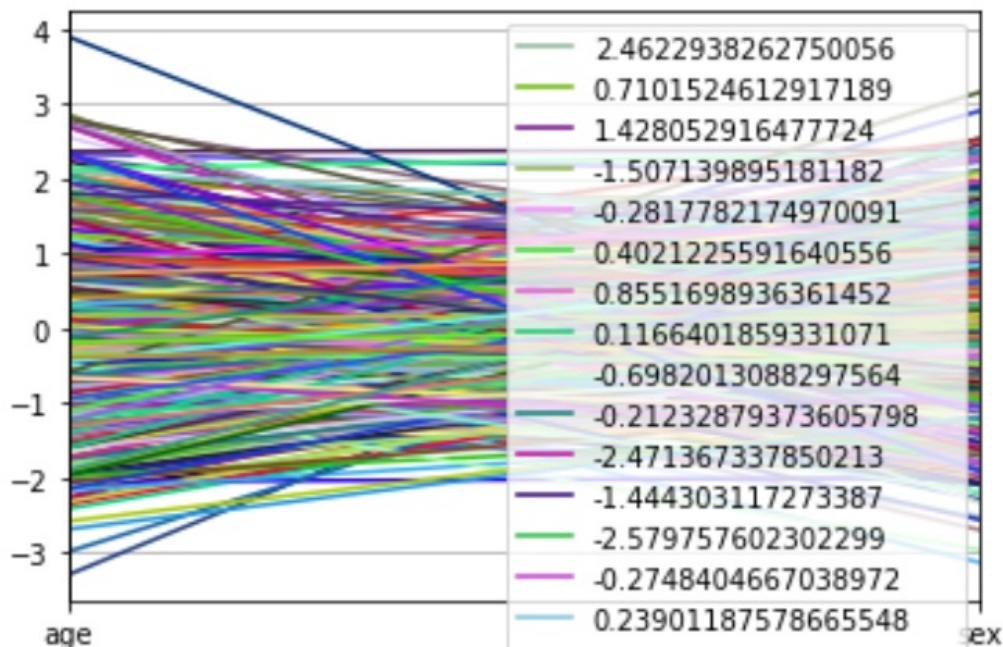


Figure 62: Parallel Coordinates of Age and Sex

KNIME allowed me to create this visualization without the use of any code, as their GUI is simple enough that programming is not always necessary. However, we also created this picture through code in Jupyter Notebook. The following script shows how we was able to do this.

```
from pandas.plotting import parallel_coordinates
df=pd.DataFrame(np.random.randn(1000, 2), columns=['age', 'sex'])
parallel_coordinates(df, 'dx')
```

Another algorithm we incorporated was logistic regression. This type of analysis compares a nominal independent and dependent variable. The regression model is used to predict a possible outcome between the two given variables[56]. The logical question we thought of when performing this analysis was if the location of skin cancer on a person's body had any influence or correlation on which type of cancer it was. The following image Figure [63](#) was created to show the logistic regression in this dataset. It shows that the relationship between cause of diagnosis (description/dx) and location is very much varied. There are, of course, some outliers, but it seems as though one can conclude that many different types of skin cancers can occur in numerous locations. This can be useful for medical professionals or students who are using this dataset as a training tool. It can also be said that conclusions can not automatically be made on what type of cancer is in a given region just by initial glance or statistics.

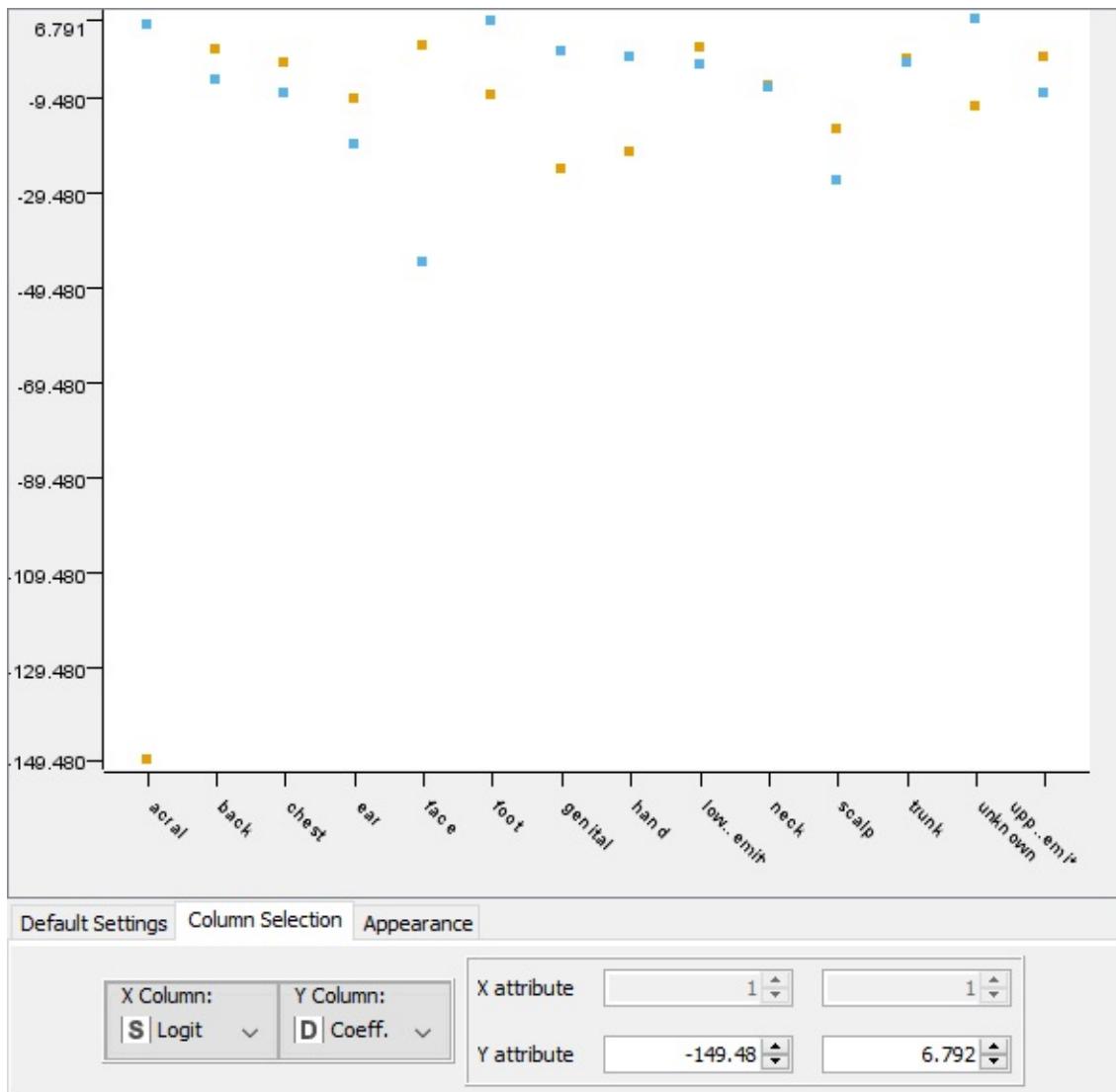


Figure 63: Logistic Regression of dx and localization features

It turns out that our analysis is correct in that different types of cancers can occur in varied locations. The figure below, Figure 64, actually shows the amount of different localizations within the dataset. Therefore, the relationship between these two features is not very strong. Everyone knows that one of the most common risks for skin cancer is sun exposure and ultraviolet light. Melanoma (mel) and basal cell carcinoma (bcc), most often caused by these factors, can occur in areas such as the face, arms, chest, back, scalp, ears, neck, and so on [57]. However, skin cancers can also occur in areas where the sun cannot reach [fa18-523-52-abchealth]. These occurrences can be caused by a genetic mutation or a change in gene regulation that

causes healthy cells to divide with errors. These mutated cells could then invade other parts of the body and spread. It has also been seen that skin cancers can possibly be caused by pollutants or toxins in the environment [fa18-523-52-abchealth].

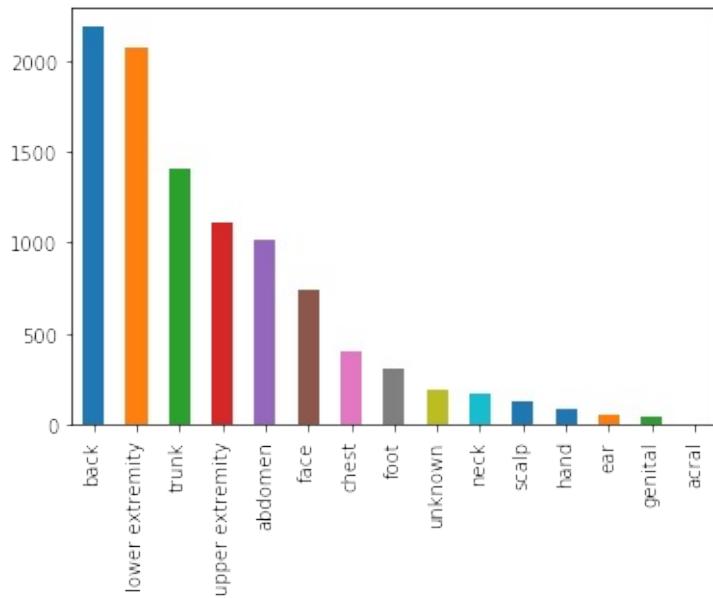


Figure 64: Skin Cancer Localizations Within the HAM10000 Dataset

A good algorithm to test the relationship between diseases and the sexes is a regression tree. A regression tree is an algorithm that predicts the results of two, usually categorical, variables. The regression model is quite easy to interpret. Using KNIME's analytical platform, we conducted a predictive regression tree diagram of the sexes within the HAM10000 dataset. The results show that the dataset includes a higher number of diseased males than females. The figure below shows the results in the model of the actual regression tree Figure 65. This analysis is also important to know because there is no evidence that strongly supports a specific sex to have a greater or lesser chance of developing a disease. It is easy to see that there are 54.7% of males with diseases in the dataset and 48.8% of females with diseases.

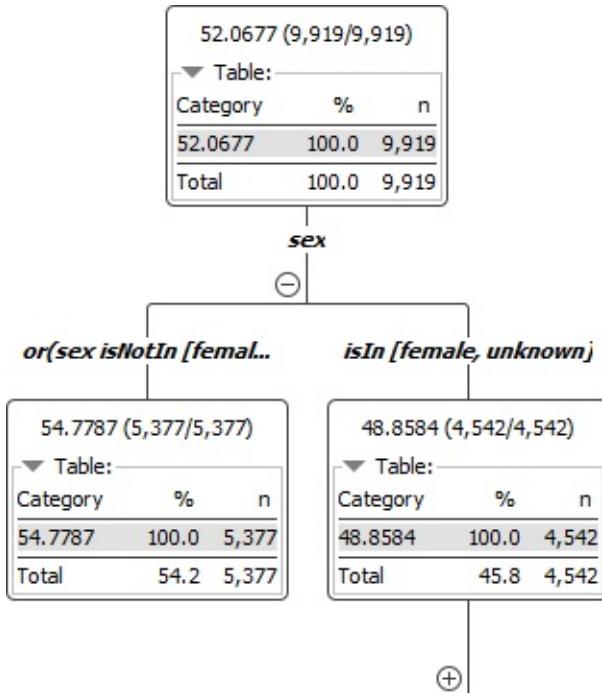


Figure 65: Regression Tree of the Sexes

18.9 BENCHMARK

The presence of big data analysis in healthcare is increasing exponentially. The burden placed on medical professionals to read and understand each patient's medical history as well as handling the current visit is sometimes too much to ask. The use of big data in the healthcare sector would make the time spent analysing each patient's medical history much shorter, resulting in an increased amount of face-to-face communication time between the patient and professional. Predictive analysis is a large advantage taking place in healthcare's use of big data today. Predictive analyses can be used to identify risks for future diseases among patients and therefore, doctors can use that information to prescribe certain medications, health-promoting activities, and other preventative measures. As mentioned earlier, electronic health records are now being used as one of the most reliable forms of big data in healthcare. It is obvious, especially in the modern age, that we realize people's health is ever-changing and there are multiple factors that weigh in for almost every

medical situation. Use of big data in healthcare setting is also being seen in medical research. Analysis of recovery methods for certain drugs and treatments in cases of cancer can be used to discover the best-working plan [58].

18.10 CONCLUSION

The use of big data analytics in healthcare is variable, but has an exciting future. The teaming up of data scientists with medical professionals will expand society's knowledge for determining the identity of a disease or even a patient's future. Big data in healthcare has the ability to give us inferences to outcomes and diseases we may have not even seen coming. The combination of these two fields will definitely speed up research and increase the rate of discovery from both sides. This starry-eyed future, however, comes at a cost. Acquiring reliable datasets are expensive, the data must be kept confidential, and it must also be extremely accurate regarding its inclusion requirements. The HAM10000 dataset, which satisfies all of the above credentials, is one of many big datasets being used for analysis among medical professionals. The analyses performed on this dataset is just the beginning of what can truly be done to explore further information into the relationship between the donated skin lesions and its multiple features.

18.11 ACKNOWLEDGEMENT

Thanks are given to Professor Geoffrey Fox and Professor Gregor von Laszewski for providing the inspiration to create the analyses displayed in this project.

19 ORCHESTRATING MICROSERVICES FOR A CREDIT SCORING APPLICATION IN KAFKA FA18-523-53

Chaitanya Kakarala

ckakara@iu.edu

Indiana University

hid: fa18-523-53

github: [cloud](#)

code: [cloud](#)

Keywords: Kafka, Zookeeper, Microservices, Python.

19.1 ABSTRACT

This Project deals with orchestration of micro services in a Credit Scoring application using a Kafka cluster. A user keys in his personal identification information in a user interface created in Python and upon submitting the same multiple micro services written in python will be executed. These light weighted and autonomous services interact with one another using a Kafka broker which works in a subscribe-publish model. The user will then see his credit report and the factors that impact his credit.

19.2 INTRODUCTION

With the increase in the amount of data being processed there is a great need of developing the applications with better performance. One such technique to make the application perform better is breaking the application into smaller units. These units are light weighted and autonomous in nature. These small applications provides scalability and because of their autonomous nature they can be plugged into any system. These small applications are also known as Microservices. Since we are breaking the big or complex

application to multiple small or light weighted microservices, a mechanism to efficiently communicate between these microservices is required. Apache Kafka is one such mechanism which provides a message streaming platform so that the microservices can either subscribe or publish the data.

19.3 APACHE KAFKA

Apache Kafka [59] is an open source streaming platform that streams the data in the form of messages. The messages are nothing but a collection of bytes and kafka has nothing to do with the content of these messages. Each of these messages will be tagged with a topic name and these messages will be published into a partition (disk space) of the topic it is associated with. These partitions can be made available across different machines which makes kafka a horizontally scalable streaming platform. Optionally each of these messages can be given with a key and whose hash values determines the partition it should be saved in. Hence the messages with the same key value will be stacked together in the same partition. Figure [66](#) describes four partitions of a single topic.

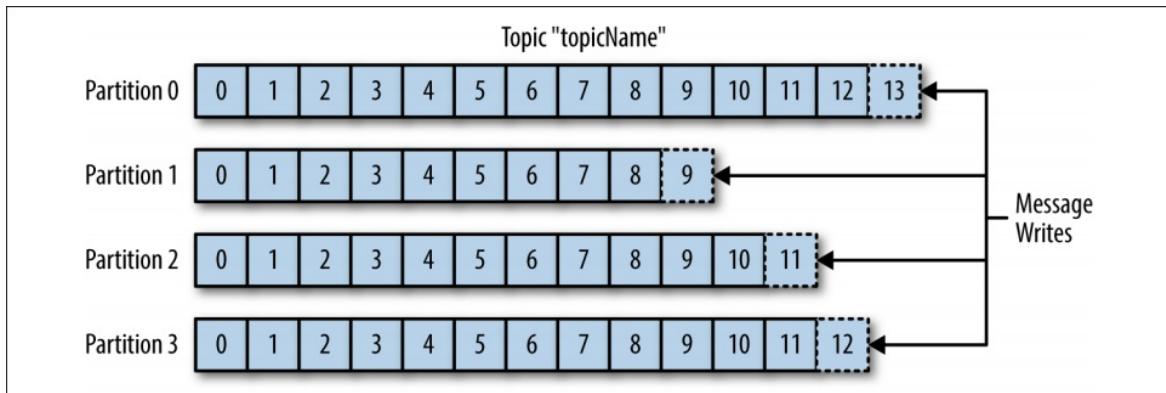


Figure 66: Representation of topic with multiple partitions [60].

There are two users of the kafka system. They are producers and consumers. Producers sends the messages and they are also known as publishers. producers while sending the messages does not care about the partition the message is going to save. However, publishing

the message with a key value ensures all the messages with same key stored in the same partition. On the other hand consumers consumes those messages by the producers. The consumers saves the offset of each message it reads and process the same. Saving the offset helps restarting from the point of failure in case of an issue rather than starting all over again. These consumers can be grouped together called as consumer group and each consumer in the consumer group could be hosted in a different machine which makes the consumer aspect scale horizontally. Consumer groups also restricts the partition to be read by multiple consumers if required. The data retention in each partition can be controlled in different ways. For example the message in a partition can be removed after one month or the partition can always be maintained at the capacity threshold set to 1 GB. Figure 67 illustrates on how consumer group works.

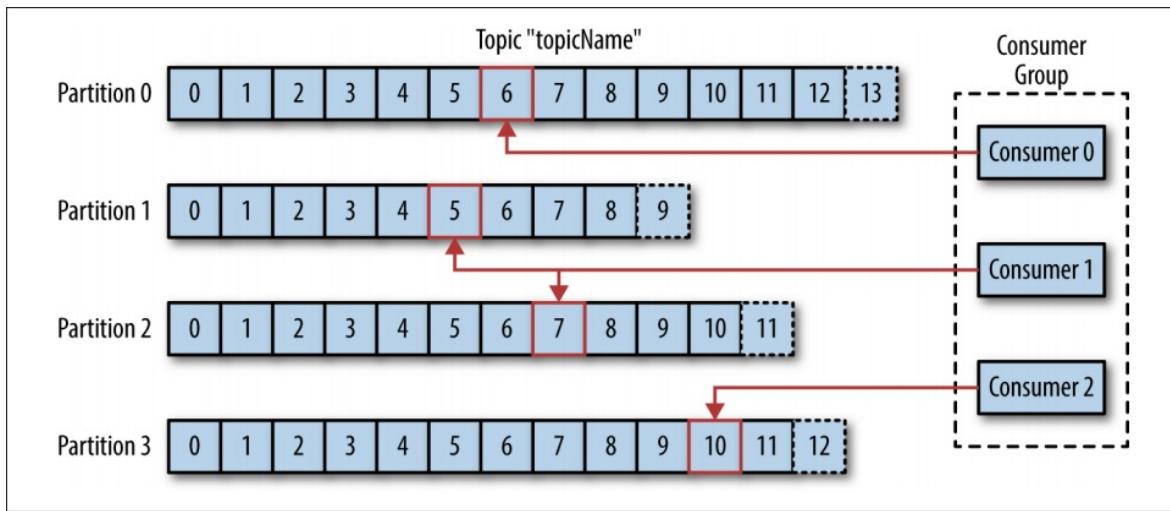


Figure 67: A consumer group reading from a topic [60].

A single kafka server is called as kafka broker. Each broker receives messages from producers and save them into their respective partitions. The broker also responds to the consumer requests and saves the offset of the consumed messages. Kafka is designed to have multiple brokers and collection of all such brokers is called as kafka cluster. A leader broker can be defined in each cluster and the data replicates from leader broker to the other brokers to provide high data availability and persistent data. Kafka also supports the

communication between the clusters in different data centers. Figure [68](#) explains how multiple brokers are replicated in a kafka cluster.

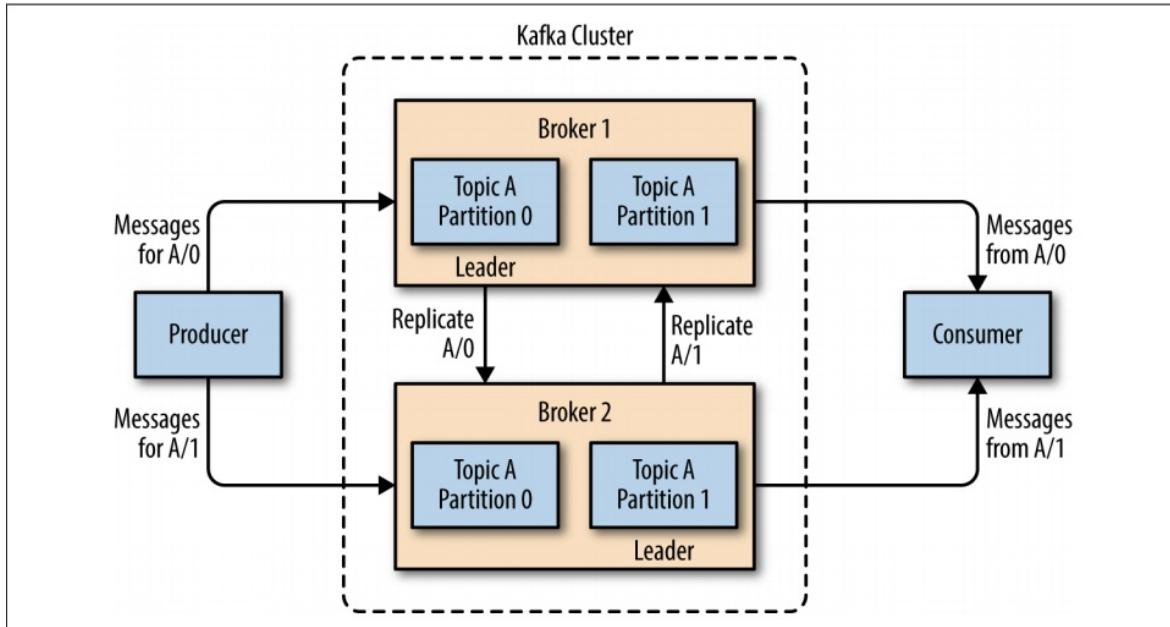


Figure 68: Representation of partitions in a cluster [60].

19.4 REQUIREMENTS

A Credit reporting agency would like to create an application to help their customers in finding their credit scores and the factors influence their score. In order to provide the above information, the application require the below personal identification information from their customers. They are:

- Name Salutation (Optional),
- First name,
- Middle Name (Optional),
- Last name,
- Name Suffix (Optional),
- Address Line 1,
- Address Line 2 (Optional),
- City,
- State,
- Zip Code,

- SSN.

Upon collecting the above information from the user, the below rules need to be applied to cleanse and standardize the user input.

- None of the name related information should contain any integers in them. They should contain only characters.
- Address lines should be standardized such as Lane to Ln and Circle to Cir.
- City and State should be characters.
- Zip code has to be integer.
- SSN should be a 9 digit integer.

After cleansing and standardizing the user input, below logic has to be applied for determining the score.

- The maximum score one can get is 850
- 10 point reduction should be applied for every credit inquiry
- Existence of a public record should result 200 point reduction
- Every missed payment will result in 100 point reduction
- If the available credit to total debt ratio is less than 10%, there will not be any reduction in score
- If the available credit to total debt ratio is between 10% to 20% , there will be a 20 point reduction in score
- If the available credit to total debt ratio is between 20% to 30% , there will be a 30 point reduction in score
- If the available credit to total debt ratio is between 30% to 40% , there will be a 40 point reduction in score
- If the available credit to total debt ratio is between 40% to 50% , there will be a 50 point reduction in score
- If the available credit to total debt ratio is greater than 50% , there will be a 100 point reduction in score
- The minimum score that one can get is 350.

The application has to be designed in such a way that the code can be packaged and implemented in any machine. The services have to be light weighted and autonomous in nature. In case of any issue with the code the user inputs must be guarded and the application should

start from the last point of failure.

19.5 ARCHITECTURE

The application is designed using below technologies:

- Kafka: Kafka is used as a message streaming tool for establishing a data pipeline between multiple microservices.
- Python: Python is the programming language used for creating the microservices and kafka library is used to publish and subscribe messages to kafka clusters. Tkinter library is used for creating user interfaces.
- Zookeeper: Zookeeper is used for maintaining a centralized configuration information by providing distributed synchronization and providing group services. Apache kafka uses zookeeper for maintaining the configurations.

Figure [69](#) describes the high level Kafka Architecture

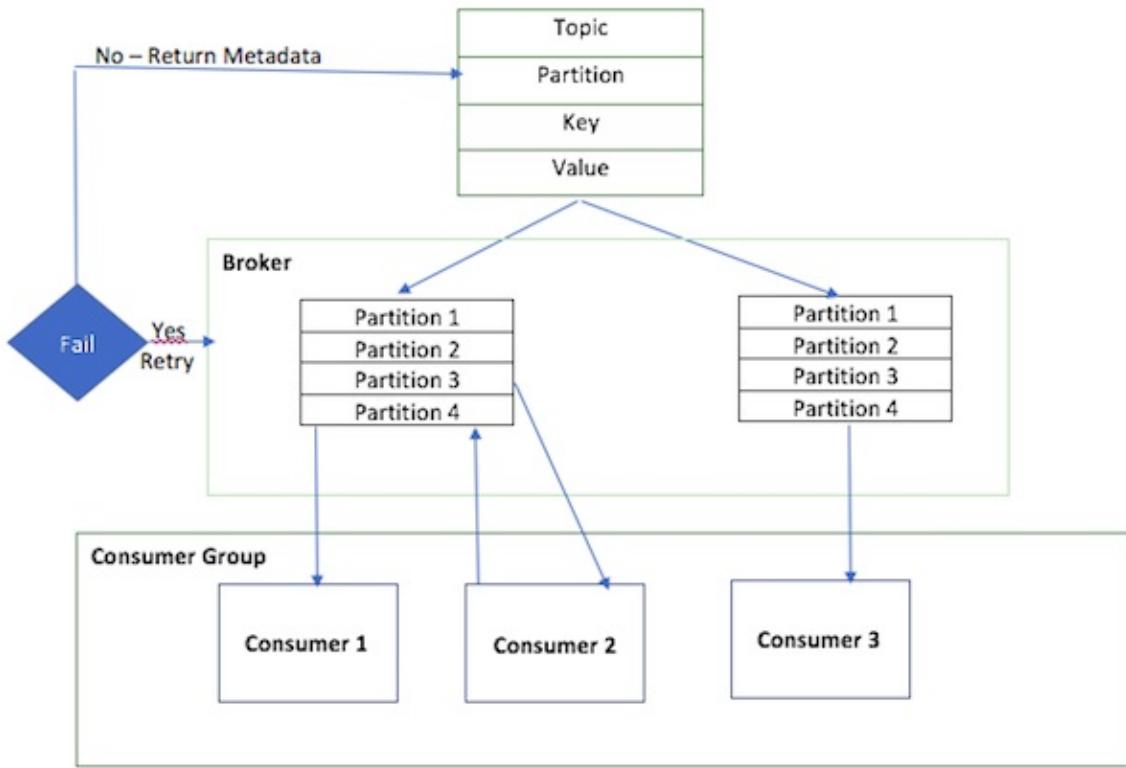
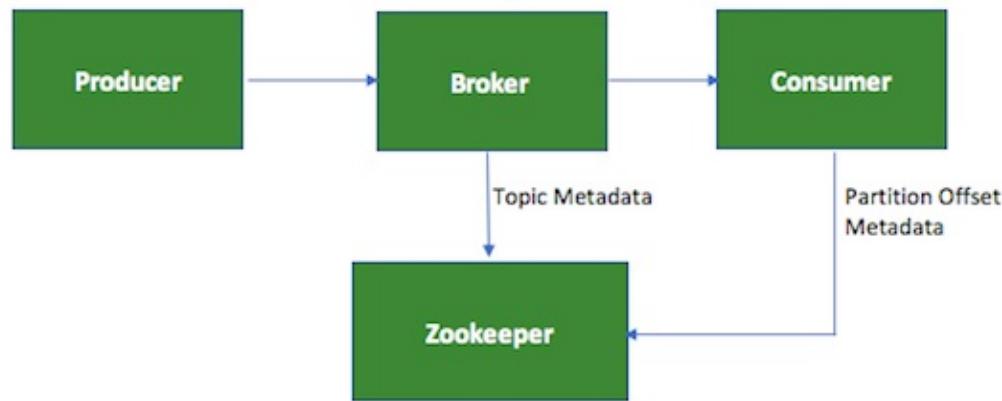


Figure 69: Kafka Architecture

19.6 DESIGN

An application is designed to have different services in python which interacts with one another using kafka streams. Below are the different microservices created as part of this application.

- scoringUI: This microservice launches a user interface for the user to input the personal identification information. The service reads each of the user inputs and create a JSON payload. The JSON payload created is sent to kafka partition under 'nameParsing' topic. The service is coded in python and using TKinter and Kafka libraries.
- nameParsingMicroService: This microservice is a consumes the messages from 'nameParsing' topic and then parse the payload. The service also applies the parsing logic to clean all the name related fields. Upon cleansing the name fields, the service creates a payload with parsed name and all the address related fields. The payload is then sent to kafka partitions under 'addressParsing' topic.
- addressParsingMicroService: This microservice consumes the messages from 'addressParsing' topic and applies parsing logic on the address related fields to define the parsed address. The service then creates a payload with parsed name and address to kafka partitions under 'scoring' topic.
- scoringMicroService: This microservice consumes the messages from 'scoring' topic from kafka cluster and applies scoring logic on the subject. This service reads the dataset called 'creditDatabase.csv' for obtaining the required attributes for a given SSN to calculate the score. The service then launches the user interface for the user to check his score and the factors contributing towards the same. Figure [70](#) describes how the above stated microservices are orchestrated in Kafka.

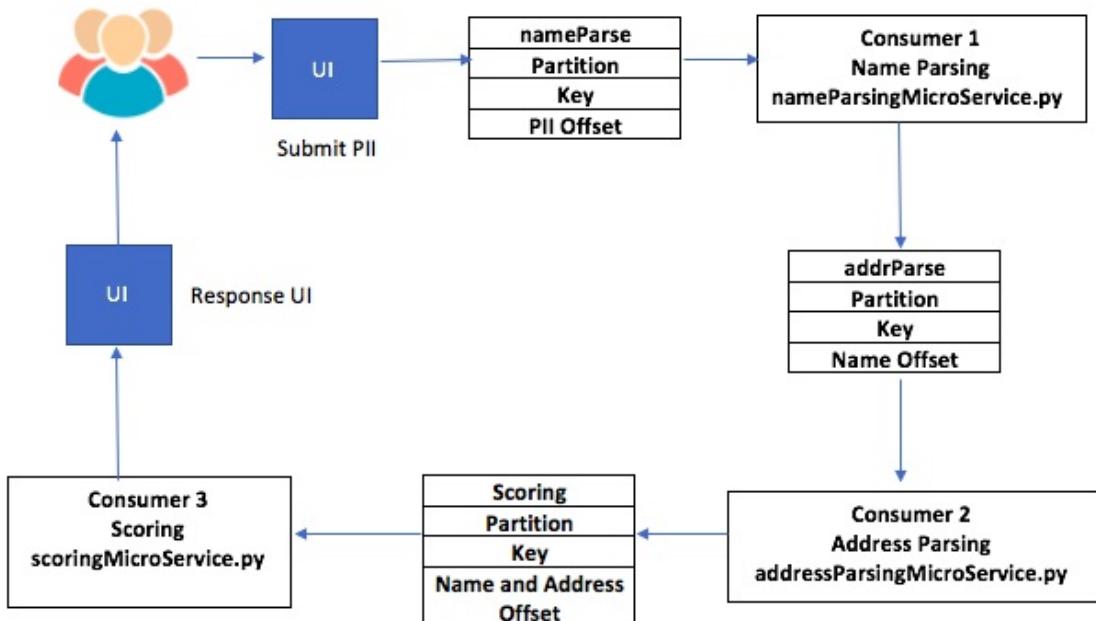


Figure 70: Scoring Application Design

Zookeeper saves the offsets of the messages consumed and produced to maintain the configuration information. Kafka requires this information in the event of restarting the application from a point of failure.

19.7 DATASET

A credit database file in csv format is created for this project. Below are the attributes of the dataset.

- SSN : Key field that uniquely identifies a person
- ACTIVE SINCE : This is the date in YYYY-MM-DD format on which date the credit was established for the person
- INQUIRIES : Number of inquiries on the record in the past 3 years.
- CREDIT LIMIT : Total credit limit of the person across all his accounts.

- TOTAL BALANCE : Sum of all current balances reported by different financial institutions.
- PUBLIC RECORD : This attribute is set to "Y" if there are any public records listed for this person. Default Value is "N"
- MISSED PAYMENTS : Number of missed payments by this person from the date of credit establishment.

19.8 IMPLEMENTATION

- Kafka Installation:
 - Kafka tar file can be obtained from [61]. Please download and save it on the server. Please be aware that kafka requires Java to be installed on the server.
 - Untar the downloaded file using below commands

```
tar -xzf kafka_2.11-1.1.0.tgz
```

- If the java version in your server is having a LTS (Long Time Support) then below fix is needed in kafka-run-class.sh located in bin folder under the kafka home directory. This is a known fix and kafka is working to address this issue for future releases [62].

Change below line

```
JAVA_MAJOR_VERSION=$(($JAVA -version 2>&1 | sed -E -n 's/.*/ version "([^.]*).*/\1/p')
```

to

```
JAVA_MAJOR_VERSION=$(($JAVA -version 2>&1 | sed -E -n 's/.*/ version "([^.]*).*/\1/p')
```

- install below libraries of python
 - json
 - CSV
 - OS
 - re
 - kafka
 - tkinter
 - tk_tools
 - datetime

- time

A python library can be installed using pip install commands. Here is the example command line to install kafka library

```
pip install kafka
```

- Start the zookeeper server using the below command. You need to be in kafka home directory to be able to successfully execute the below command.

```
bin/zookeeper-server-start.sh config/zookeeper.properties
```

- Start the Kafka server using the below command. You need to be in kafka home directory to be able to successfully execute the below command.

```
bin/kafka-server-start.sh config/server.properties
```

- Execute below services in different instances
 - nameParsingMicroService.py
 - addressParsingMicroService.py
 - scoringMicroService.py

```
./nameParsingMicroService.py  
./addressParsingMicroService.py  
./scoringMicroService.py
```

- Execute the UI service

```
./scoringUI.py
```

This command opens a user interface to key in the personal identification information. Provide the details and hit “Check your Score” button. Figure 71 is the sample input screen.

The screenshot shows a user interface for a credit scoring application. At the top right, it says "Know Your Credit Score" and "Credit Scoring Application". On the left, there are form fields for personal information: Salutation (Mr.), First Name (Chaitanya), Middle Name (K), Last Name (Kakarala), Name Suffix (Jr.), Address Line1 (1234 Hollywood St), Address Line2 (empty), City (Chicago), State (IL), and Zip (60504). Below these fields is a note: "SSN-Defaulted to the one available in Dataset" followed by a text input field containing "123-45-6789". At the bottom right is a button labeled "Check your Score".

| | |
|---------------|-------------------|
| Salutation | Mr. |
| First Name | Chaitanya |
| Middle Name | K |
| Last Name | Kakarala |
| Name Suffix | Jr. |
| Address Line1 | 1234 Hollywood St |
| Address Line2 | |
| City | Chicago |
| State | IL |
| Zip | 60504 |

SSN-Defaulted to the one available in Dataset

[Check your Score](#)

Figure 71: Scoring Application User Interface

If the SSN provided in the user input does not found in 'creditDatabase.csv' dataset then you will see a "Credit Record Not Found" exception. If not, a user interface similar to Figure 72 will be opened with the credit information.

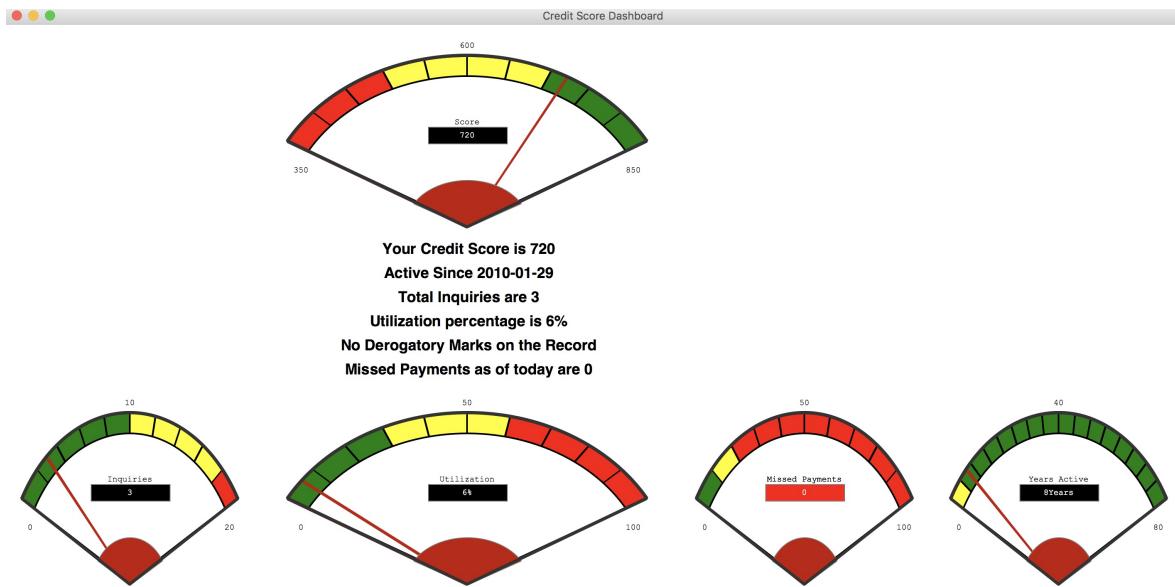


Figure 72: Scoring Application Results Interface

19.9 CONCLUSION

This project successfully demonstrates the orchestration of microservices using a kafka cluster. The four microservices created in python as part of this credit scoring application interacts with one another using kafka cluster.

19.10 ACKNOWLEDGEMENT

The author would like to thank Professor Gregor von Laszewski and associate instructors for their help and guidance.

Daniel Hinders
dhinders@iu.edu
Indiana University
hid: fa18-523-56
github: [cloud](#)
code: [cloud](#)

Learning Objectives

- Learn about NIFI
 - Install NIFI and setup a data stream with various processors
-

Keywords: ETL, Data Stream, NiFi, NSA,

20.1 APACHE NiFi INTRODUCTION

NiFi is a customizable tool for building flexible data flows while preserving data provenance and security [63]. NiFi provides the ability to build or alter an ETL flow with a few clicks. NiFi builds Gets, Converts, and Pulls in a GUI and allows the user to build and customize the flow ???- www-nifi-issartetlsimple. This flexibility and usability is key to NiFi's value in a big data world where stovepipes and inflexibility are frequently challenges.

NiFi is a tool for:

- Moving data between systems, including modern systems such as social media sources, AWS cloud server, Hadoop, MongoDB, and so on [64]
- Delivering data to analytics platforms[64]
- Format Conversion, extracting/parsing data[64]
- Data or files routing decisions[64]
- Real-time data streaming[64]

NiFi is not recommended for:

- Distributed Computation[64]
- Complex Event Processing[64]
- Join/ Aggregated Functions[64]

20.2 NiFi HISTORY

NiFi was first developed at the National Security Agency but was released as an open source project to the public.

“NiFi was submitted to The Apache Software Foundation (ASF) in November 2014 as part of the NSA Technology Transfer Program” [65].

Since then, Apache Foundation has used its volunteer organization to grow and mature the project [66].

20.3 NiFi FEATURES

NiFi incorporates a straightforward User Interface (UI) to engineer traceable data provence with configurable components. NiFi offers up the ability to custom build processors and incorporate them into a highly customizable flows. Through

“data routing, transformation, and system mediation logic” [63],

NiFi seeks to automate data flow in a big data environment and gives architects the ability to keep data flowing between evolving systems quickly. Amongst a host of features NiFi offers, one sticks out as particularly important because of the challenges associated with what the feature addresses: data errors, data inconsistency, and data irregularity handling. NiFi provides users the ability to incorporate in the flow processes to catch these non-happy path realities in big data. As new situations are discovered, a user can quickly build if-then forks in the process to catch, store, or resolve the data issues.

NiFi's main features are:

- Guaranteed delivery: use purpose-built persistent write-ahead log and content repository to ensure guaranteed delivery in an effective way[63] [67].
- Web-based user interface: easy to use web-based GUI with drag and drop features that allows users to build, schedule, control, and monitor data flow[67] [63].
- Provenance: provide the ability to track data flows through the systems with audit trail and traceability functionalities [67] [63].
- Queue Prioritization: provide the ability to configure and prioritize job flow and determine the order of events [67][63].
- Secure: provide and support multiple security protocols and encryptions, as well as authorization management [67][63].
- Extensibility: provide flexibility by allowing pre-built and built-your-own extension the be integrated [67][63].
- Scalability: supports scale-out by clustering architecture as well as scale-up and scale-down [67][63].

20.4 NiFi ARCHITECTURE

The NiFi homepage Figure [73](#) shows the main components in NiFi architecture.

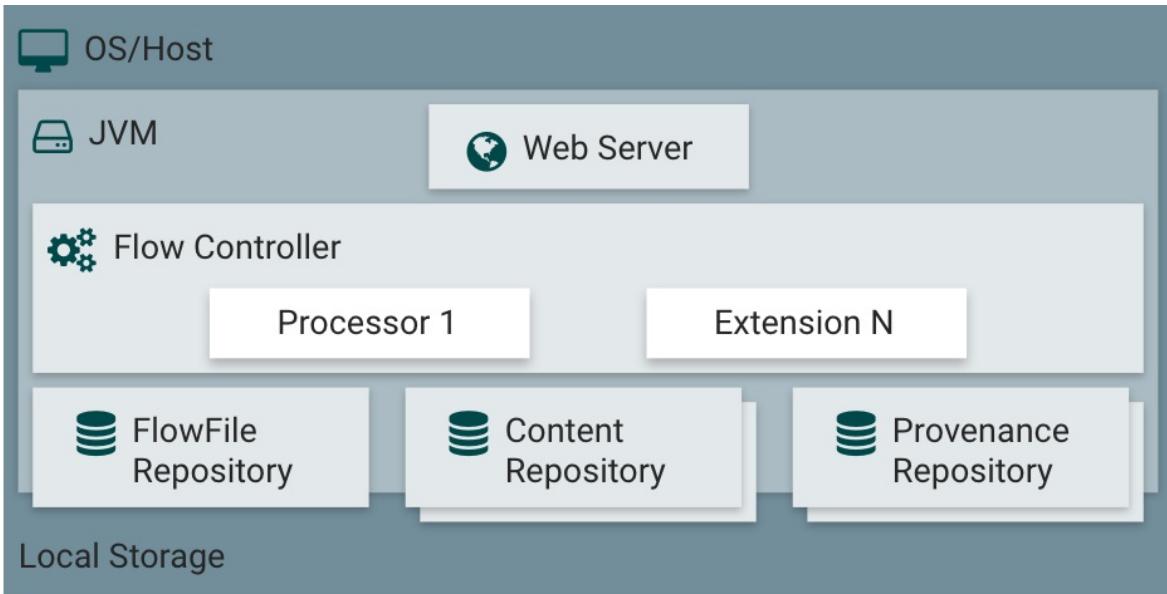


Figure 73: NiFi Architecture [67]

From the top down, NiFi is web browser accessible by a NiFi hosted Web Server. NiFi processor operations are managed through the Flow Controller and the three repositories; FlowFile, Content, and Provenance work to process data on and off disk and in a NiFi flow. NiFi is hosted in the Java Virtual Machine environment or JVM [67].

20.4.1 Web Server

NiFi's easy-to-use graphic user interface(GUI) is hosted on the Web Server within the JVM [67].

20.4.2 Flow Controller

NiFi central operations hub is the Flow Controller. Threads are managed and allocated to the processors and the FlowFiles are passed through and managed through the Flow Controller [68].

20.4.3 FlowFile Repository

Files in an active NiFi flow are tracked in a write-ahead log so that as data moved through the flow NiFi can keep track of what is known about files as they pass through[67].

20.4.4 Content Repository

The real data for a flow file is in the NiFi content repository. NiFi uses simple blocks of data in a file system to store this FlowFile data [67]. Multiple file systems can be used in order to increase speed with multiple volumes being utilized.

20.4.5 Provenance Repository

In NiFi, the provenance repository stores historic event data. The provenance data about flows is indexed to enable search of the records [68].

20.4.6 Processors

NiFi provides more than 260 processors and more than 48 controller services for users to integrate into a flow from the graphic user interface(GUI) of Nifi[69]. Processors are base on underlying controller services in the java virtual machine. Controller services can be centered around a security implementation, database CRUD (create, read, updates, and deletes), and many other foundational areas. Users can create custom processors from existing controller services or create a customer controller service as well [69].

20.4.6.1 Processor Examples

- **Get**
 - Examples: GetFTP, GetMongo, GetTCP, etc [67]
 - Similar input type processors: Consume, Extract, Fetch, Listen, etc

Nifi provides dozens of Get processor options and many other similar input type processors. A Get processor is commonly used to pick up a file or data and launch a FlowFile. The Get file processer setup typically gives configuration options to point to a host, set timing increments for polling and timeouts, set proxy settings, and more

[67].

- **Convert**

- Examples: ConvertJSONToSQL, Convert Record, ConvertExceltoCSVProcessor, etc [67]
- Similar transformation type processors: Evaluate, Merge, Split, etc.

Once data is in the flow, NiFi provides dozens of processors to manipulate or transform data. The Convert processors can be configured to the expected schema or type from the Get processor and transform, edit, thin, enrich, or many other functions on the data in the flow [67].

- **Put**

- Examples: PutFile, PutFTP, PutSQL, PutElasticSearch, PutAzureBlobStorage, etc [67]
- Similar output type processors: Publish, etc.

A critical part of a flow in NiFi is pushing the right data out of the flow into the right spot. There are dozens of Put processors that can be configured to set the directory to write files too. Additional configuration options are specific to the destination type to include SSL configuration, cache options, batching options, and many other configuration options based on the destination type [67].

20.4.7 NiFi Clusters

NiFi can also be integrated with ZooKeeper to operate within a cluster as shown by Figure 74 shows how ZooKeeper manages NiFi's nodes by determining the primary node, Zookeeper Coordinator, and failover node . Each of the nodes performs the same tasks but processes different dataset(s) [fa18-523-56-www-nifi-homepagetechdoc].

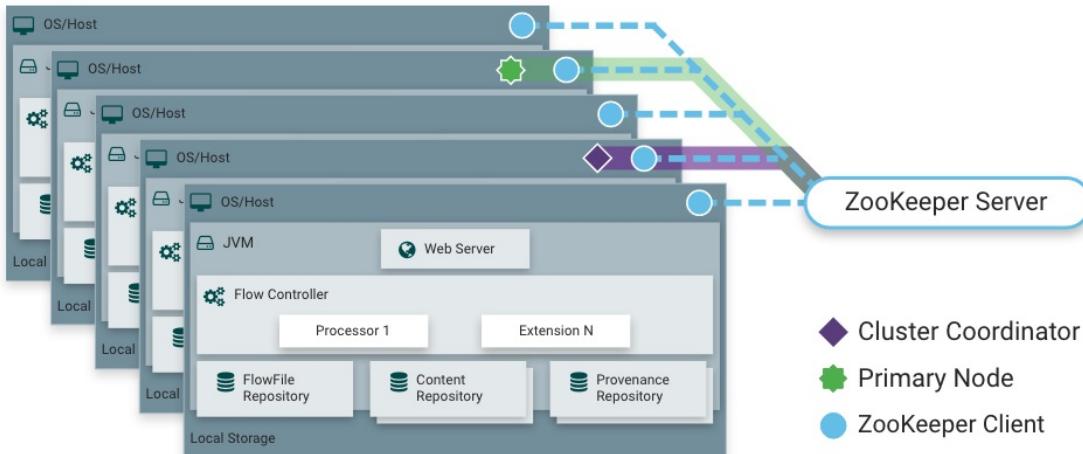


Figure 74: NiFi Cluster Architecture [67]

20.5 INSTALL NiFi

20.5.1 Apache NiFi - Windows

(Note: Assumes Windows OS and recent verison of Java is installed)

1 NiFi can be downloaded from Apache NiFi homepage[70]. Select the latest version and the bin.zip option for the Windows instillation.

Apache nifi Downloads

To verify the downloads please follow these procedures using these KEY!

If a download is not found please allow up to 24 hours for the mirrors to sync.

Releases

• 1.8.0

- Released October 26, 2018
- Sources:
 - [nifi-1.8.0-source-release.zip \[53 MB\]](#) (asc, sha256, sha512)
- Binaries
 - [nifi-1.8.0-bin.tar.gz \[1.2 GB\]](#) (asc, sha256, sha512)
 - [nifi-1.8.0-bin.zip \[1.2 GB\]](#) (asc, sha256, sha512)
 - [nifi-toolkit-1.8.0-bin.tar.gz \[42 MB\]](#) (asc, sha256, sha512)
 - [nifi-toolkit-1.8.0-bin.zip \[42 MB\]](#) (asc, sha256, sha512)
- Release Notes

Figure 75: nifi_download

picture quality bad, convert to ascii, no need for image use text

2 Unzip the install package.

3 Navigate to the configuration directory:

nifi-1.8.0-bin\nifi-1.8.0\conf

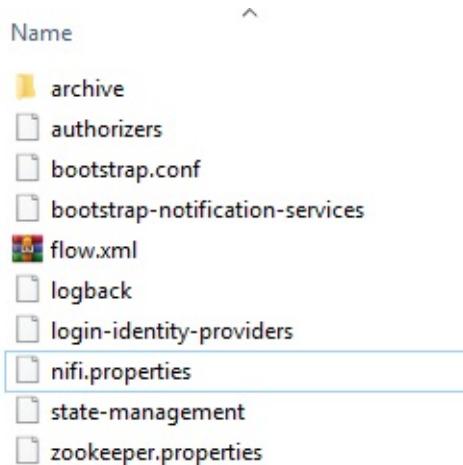


Figure 76: nifi_config

Open nifi.properties file with a text editor and edit nifi.web.http.port= to the desired port

```
# web properties #
nifi.web.war.directory=./lib
nifi.web.http.host=
nifi.web.http.port=9090
nifi.web.http.network.interface.default=
nifi.web.https.host=
nifi.web.https.port=
nifi.web.https.network.interface.default=
nifi.web.jetty.working.directory=./work/jetty
nifi.web.jetty.threads=200
nifi.web.max.header.size=16 KB
nifi.web.proxy.context.path=
nifi.web.proxy.host=
```

4 Start up NiFi by navigating to

nifi-1.8.0-bin\nifi-1.8.0\bin

| Name | Date modified |
|-----------------------|----------------|
| bin | 11/9/2018 4:18 |
| conf | 11/9/2018 5:40 |
| content_repository | 11/9/2018 4:33 |
| database_repository | 11/9/2018 4:33 |
| docs | 11/9/2018 4:18 |
| flowfile_repository | 11/9/2018 5:27 |
| lib | 11/9/2018 4:18 |
| logs | 11/12/2018 5:0 |
| provenance_repository | 11/9/2018 4:33 |
| run | 11/9/2018 4:31 |
| state | 11/9/2018 4:33 |
| work | 11/9/2018 4:32 |
| LICENSE | 11/9/2018 4:17 |
| NOTICE | 11/9/2018 4:17 |
| README | 11/9/2018 4:17 |

{#fig:nifi_bin}

Run the windows batch file

```
run-nifi.bat
```

Wait about 5 minutes for NiFi to load

5 Open the NiFi GUI by opening a browser and navigate to

```
http://localhost:9090/nifi
```

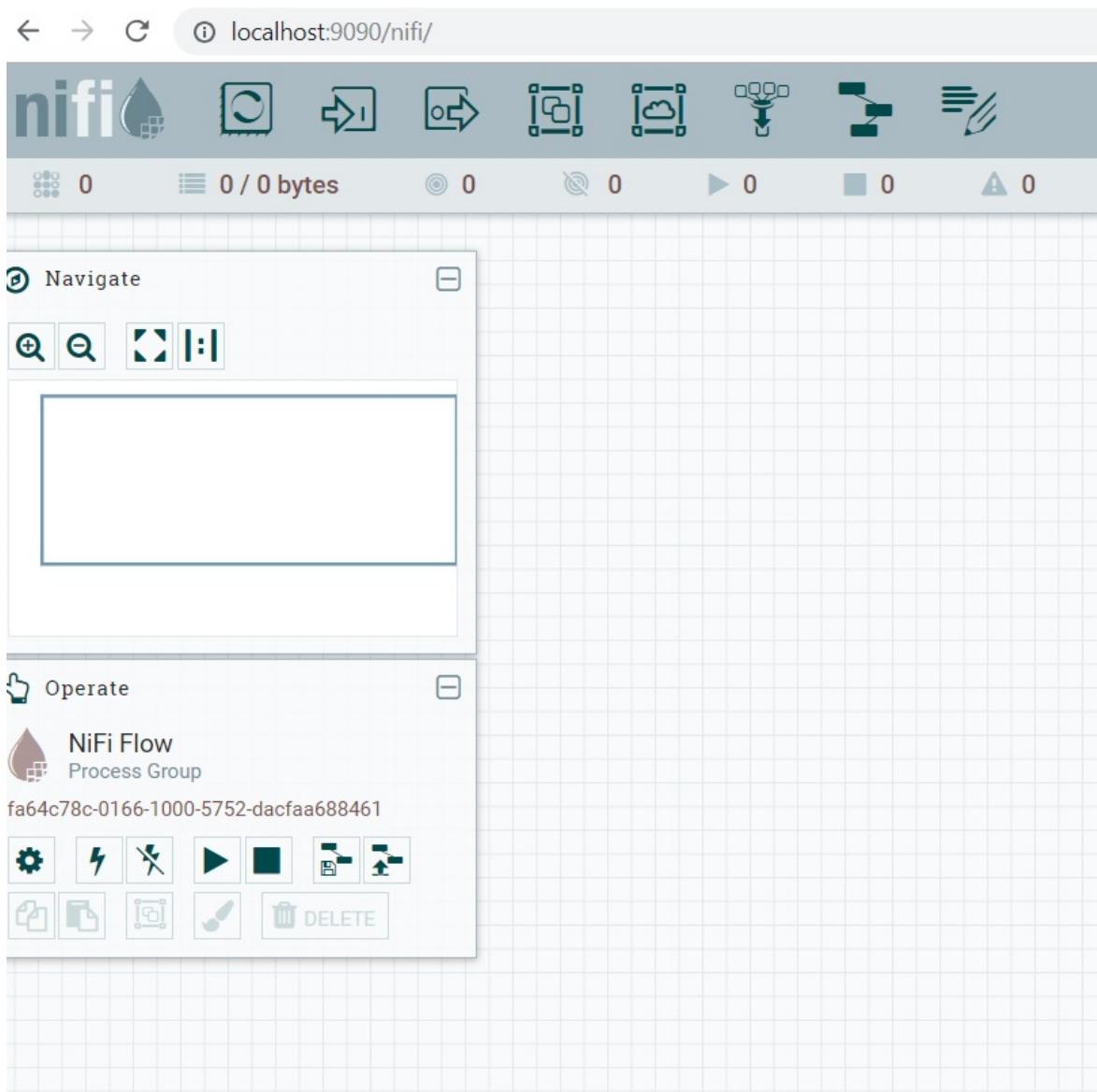


Figure 77: nifi_nifihome

20.6 BUILDING A NIIFI FLOW

1 Add a TailFile Processor by clicking and dragging the processor icon from the top tray to add a processor. Type into the filter "tail" and select the TailFile processor and click ADD

Displaying 1 of 286

| Type | Version | Tags |
|----------|---------|---|
| TailFile | 1.8.0 | file, log, tail, restricted, text, sou... |

Figure 78: nifi_processor_tailfile

2 Configure the TailFile Processor by right-clicking on the process and click configure.

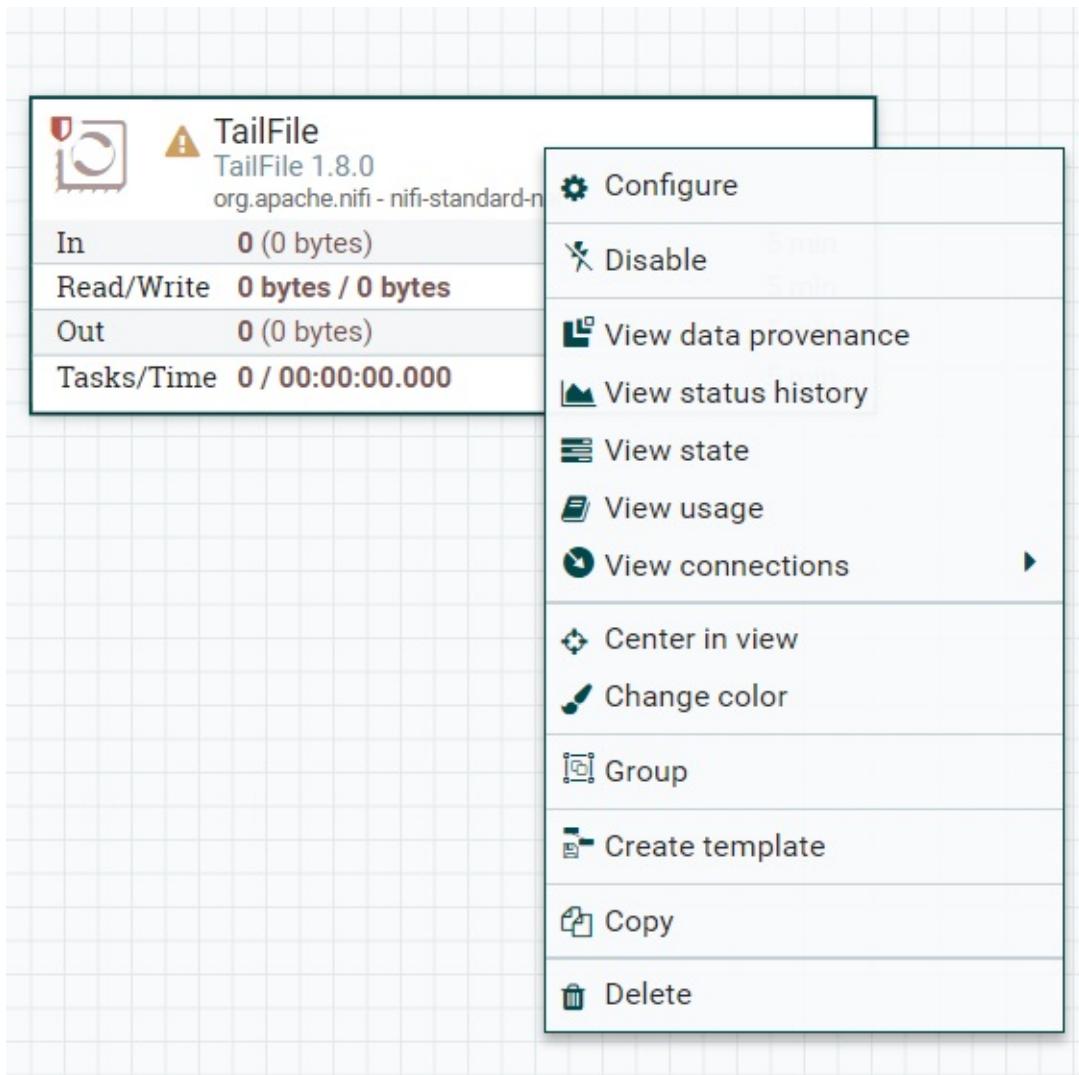


Figure 79: nifi_processor_config

Click the properties tab and click the value for the property "File(s) to

Tail”

A box will appear to paste the location of the file to tail. For this example I will use a log file for a music player because it will provide a lot of data.

Use / slash when inputting file path

```
/AppData/Local/Amazon Music/Logs/AmazonMusic.log
```



{#fig:nifi_tailfile_config}

Click OK and then click APPLY

3 Add a processor called SplitText

Open the configuration options for the processor and on the settings tab in the options for Automatically Terminate Relationships check the boxes “failure” and “original”

Configure Processor

| SETTINGS | SCHEDULING | PROPERTIES | COMMENTS |
|---|---|---|----------|
| Name SplitText | <input checked="" type="checkbox"/> Enabled | Automatically Terminate Relationships ? | |
| Id 190be21a-0167-1000-3657-ee3781173047 | <input checked="" type="checkbox"/> failure If a file cannot be split for some reason, the original file will be routed to this destination and nothing will be routed elsewhere | | |
| Type SplitText 1.8.0 | <input checked="" type="checkbox"/> original The original input file will be routed to this destination when it has been successfully split into 1 or more files | | |
| Bundle org.apache.nifi - nifi-standard-nar | <input type="checkbox"/> splits The split files will be routed to this destination when an input file is successfully split into 1 or more split files | | |
| Penalty Duration ? 30 sec | Yield Duration ? 1 sec | | |
| Bulletin Level ? WARN | | | |

Figure 80: nifi_splittext_config2

NOTE

This provides direction if there is a failure at this step if a file can't be split any what to do with the original file after it is split. This flexibility that NiFi provides requires extra configuration choices but provides the NiFi admin extensive control over every aspect of the flow being built.

Click the properties tab and change the property Line Split Count to a value of "1"

| SETTINGS | SCHEDULING | PROPERTIES | COMMENTS | | | | | | | | | | | | |
|---|--------------|------------|----------|----------|-------|------------------|---|-----------------------|--------------|-------------------|---|-------------------------------|--------------|--------------------------|------|
| Required field | | | | | | | | | | | | | | | |
| <table border="1"> <thead> <tr> <th>Property</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td>Line Split Count</td> <td>1</td> </tr> <tr> <td>Maximum Fragment Size</td> <td>No value set</td> </tr> <tr> <td>Header Line Count</td> <td>0</td> </tr> <tr> <td>Header Line Marker Characters</td> <td>No value set</td> </tr> <tr> <td>Remove Trailing Newlines</td> <td>true</td> </tr> </tbody> </table> | | | | Property | Value | Line Split Count | 1 | Maximum Fragment Size | No value set | Header Line Count | 0 | Header Line Marker Characters | No value set | Remove Trailing Newlines | true |
| Property | Value | | | | | | | | | | | | | | |
| Line Split Count | 1 | | | | | | | | | | | | | | |
| Maximum Fragment Size | No value set | | | | | | | | | | | | | | |
| Header Line Count | 0 | | | | | | | | | | | | | | |
| Header Line Marker Characters | No value set | | | | | | | | | | | | | | |
| Remove Trailing Newlines | true | | | | | | | | | | | | | | |

Figure 81: nifi_splittext_config

This will split each line of the log file into one row that will be processed independently in the rest of the flow.

4 Add a processor called RouteOnContent

Open the configuration options for the processor and on the settings tab in the options for Automatically Terminate Relationships check the box “unmatched”

Click the properties tab and change the property Match Requirement to “content must contain match” click OK

Click the + in the upper right corner to add a property.

NOTE

This property will be used to select a word or phrase from the rows of the log file. When the word is seen in the row the content of the row will be routed down stream. For this example we will use "AddToLibrary" when a user in the music player adds a song to the library. We will use "ClientImplWinHTTP.cpp:525" which is the tag in the log when a song plays in the music player

After naming the new property click OK

Click on the value and use the NiFi expression language insert the tags to use to select the rows for processing.

NOTE

The NiFi expression language can be found on the Apache NiFi website[@fa18-523-56-www-nifi-expressionlanguageguide].

We will use these tags for our new properties:

```
\bAddToLibrary\b  
\bClientImplWinHTTP.cpp:525\b
```

| SETTINGS | SCHEDULING | PROPERTIES | COMMENTS |
|---------------------|------------|-------------------------------|--------------|
| Required field | | | |
| Property | | | Value |
| Match Requirement | ? | content must contain match | |
| Character Set | ? | UTF-8 | |
| Content Buffer Size | ? | 1 MB | |
| addsong | ? | \bAddToLibrary\b | |
| playsong | ? | \bClientImplWinHTTP.cpp:525\b | |

Figure 82: nifi_routeoncontent_config

Once all properties have been added click APPLY

5 Link the processes by hovering over the TailFile processor click on the arrow that appears and drag to connect it to the SplitText Processor

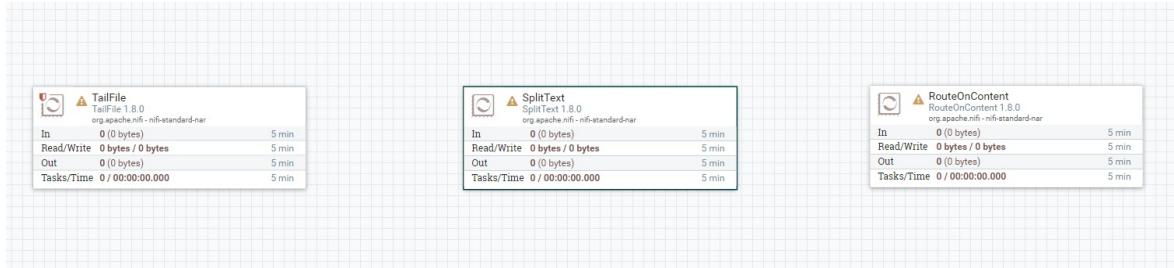


Figure 83: nifi_flow1

A window will appear to create the connection. Click ADD to connect the processors

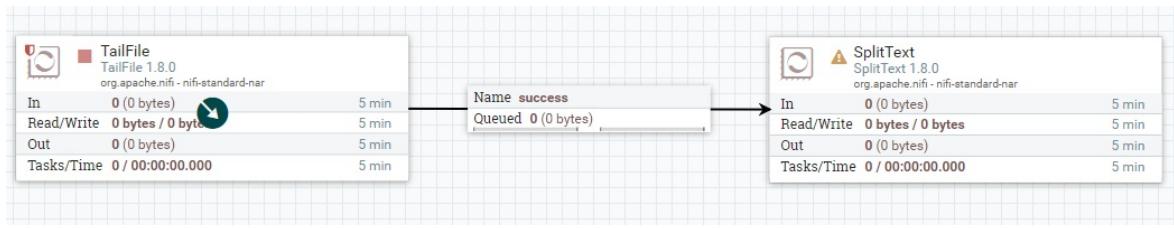


Figure 84: nifi_flow1

Connect the SplitText Processor to the RouteOnContent processor

Configure the connection in the column For Relationships, check the box “splits”

Create Connection

| | |
|--|--|
| DETAILS | SETTINGS |
| From Processor SplitText SplitText Within Group NiFi Flow For Relationships <input type="checkbox"/> failure <input type="checkbox"/> original <input checked="" type="checkbox"/> splits | To Processor RouteOnContent RouteOnContent Within Group NiFi Flow |

Figure 85: nifi_connection1

NOTE

This configuration for this connection will route the rows we selected from the log file that were split out. Another path could be created to handle the orginal files or the failures.

6

Right click on the RouteOnContent processor to open the configuration options for the processor and on the settings tab in the options for Automatically Terminate Relationships check the boxes “playsong” and “addsong” and click APPLY.

The flow is now complete, it will read a log file, select and split inputs into rows based on parameters and routes the selected rows for output. But we have chosen to terminate the output at the RouteOnContent processor so that we run this simple flow first before connecting the flow to an external consumer.

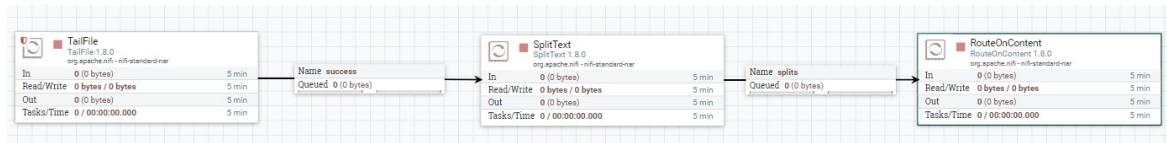


Figure 86: nifi_flow3

Select all five components in the flow with the shift key held down, then right-click on any component and select Create template.

Type a name for the template and click CREATE.

Click and drag the Template icon from the top tray to add to the workspace and a dialogue box will ask what template to add and you can select the template that was just saved.

7 To run our completed flow we need to turn on individual components or start them all at once.

Select all components with the shift key held down and click on the play button on the operate panel on the left side of the workspace.

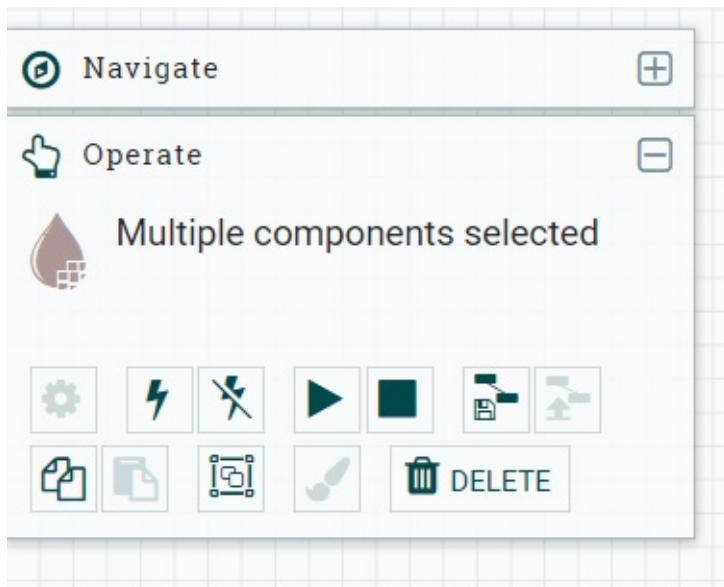


Figure 87: nifi_operate

After a few seconds right-click anywhere in the workspace and click refresh.

There will be statistics on each processor for data flowing through the flow.



Figure 88: nifi_flow4

20.7 LINKING NiFi FLOW TO APACHE KAFKA

NiFi provides numerous endpoint processors pass data out of the NiFi flow to a new host. One example is Apache Kafka, these directions for setting up Kafka are for Windows.

1 Start setting up Apache Kafka by download the lastest version of Kafka from the Apache Kafka website. The download will include Apache ZooKeeper.

2 Configure Zookeeper and Kafka by navigating to the configuration directory:

```
kafka_2.11-2.0.1\kafka_2.11-2.0.1\config
```

Open zookeeper.properties file with a text editor and edit clientPort= to the desired port and the dataDir= to the desired folder for logs

```
clientPort=2181  
clientPortAddress=localhost
```

Open server.properties file with a text editor and edit zookeeper.connect= to the port selected for zookeeper and the dataDir= to the desired folder for logs

```
zookeeper.connect=localhost:2181
```

3 Start the ZooKeeper server by opening cmd prompt and set directory to

```
\kafka_2.11-2.0.1\kafka_2.11-2.0.1\bin\windows
```

type:

```
zookeeper-server-start.bat c:\(path)\kafka_22.11-2.01\config\zookeeper.properties
```

Hit enter and the ZooKeeper server will start up

```
[2018-11-23 12:51:04,692] INFO tickTime set to 3000 (org.apache.zookeeper.server.ZooKeeperServer)  
[2018-11-23 12:51:04,692] INFO minSessionTimeout set to -1 (org.apache.zookeeper.server.ZooKeeperServer)  
[2018-11-23 12:51:04,693] INFO maxSessionTimeout set to -1 (org.apache.zookeeper.server.ZooKeeperServer)  
[2018-11-23 12:51:04,721] INFO Using org.apache.zookeeper.server.NIOServerCnxnFactory as server connection factory (org.apache.zookeeper.server.ServerCnxnFactory)  
[2018-11-23 12:51:04,725] INFO binding to port localhost/127.0.0.1:2181 (org.apache.zookeeper.server.NIOServerCnxnFactory)
```

Figure 89: nifi_zookeeper_startup

4 Start Kafka server by opening another cmd prompt and set directory to

```
\kafka_2.11-2.0.1\kafka_2.11-2.0.1\bin\windows
```

type:

```
kafka-server-start.bat c:\(path)\kafka_2.11-2.0.1\config\server.properties
```

Hit enter and the Kafka server will start up

```
2018-11-23 17:31:48,439] INFO [Partition warn-0 broker=0] warn-0 starts at Leader Epoch 0 from offset 0. Previous Leader Epoch was: -1 (kafka.cluster.Partition)
2018-11-23 17:31:48,452] INFO [ReplicaAlterLogDirsManager on broker 0] Added fetcher for partitions List() (kafka.server.ReplicaAlterLogDirsManager)
```

Figure 90: nifi_kafka_startup

5 Create two Kafka topics to so that we can put files data from the NiFi flow into the Kafka topics

To add the addsong topic open a new cmd prompt and set directory to

```
\kafka_2.11-2.0.1\kafka_2.11-2.0.1\bin\windows
```

type:

```
kafka-topics.bat --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic addsong
```

[71]

Hit enter and the new topic will be created

```
\windows>kafka-topics.bat --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic addsong
Created topic "addsong".
```

Figure 91: nifi_kafka_addtopic

Repeat to add the process type:

```
kafka-topics.bat --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic playsong
```

6 Add NiFi Processors to push data from the NiFi flow to Kafka

Select all three processors with the shift key held down and click on the stop button on the operate panel on the left side of the workspace.

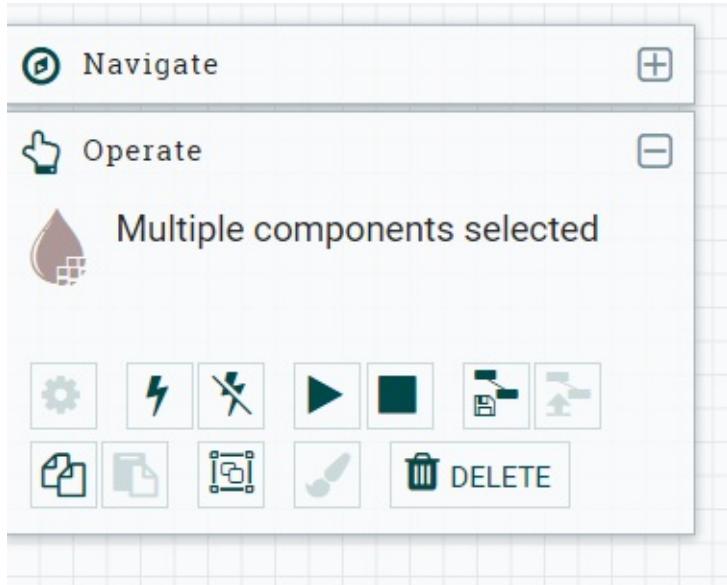


Figure 92: nifi_operate

Add a PublishKafka processor to the flow. On the settings tab check the success check box. On the properties tab provide the port for the Kafka Broker.

localhost:2181

On the properties tab provide the Topic Name: addsong

Repeat this process by adding another PublishKafka processor and configure it the same except set the Topic Name: playsong

| Configure Processor | | | |
|--------------------------|--------------------|------------|--------------|
| SETTINGS | SCHEDULING | PROPERTIES | COMMENTS |
| Required field | | | |
| Property | | | Value |
| Kafka Brokers | localhost:2181 | | |
| Security Protocol | PLAINTEXT | | |
| Kerberos Service Name | No value set | | |
| SSL Context Service | No value set | | |
| Topic Name | playsong | | |
| Delivery Guarantee | Best Effort | | |
| Kafka Key | No value set | | |
| Key Attribute Encoding | UTF-8 Encoded | | |
| Message Demarcator | No value set | | |
| Max Request Size | 1 MB | | |
| Acknowledgment Wait Time | 5 secs | | |
| Max Metadata Wait Time | 5 sec | | |
| Partitioner class | DefaultPartitioner | | |
| Compression Type | none | | |

Figure 93: nifi_publish_kafka_config

6 Configure the RouteOnContent processor, on the settings tab in the options for Automatically Terminate Relationships uncheck the boxes “playsong” and “addsong” and click APPLY.

| Configure Processor | | | |
|---------------------|--------------------------------------|---|--|
| SETTINGS | SCHEDULING | PROPERTIES | COMMENTS |
| Name | RouteOnContent | <input checked="" type="checkbox"/> Enabled | Automatically Terminate Relationships ? |
| | | <input type="checkbox"/> addsong | |
| | | <input type="checkbox"/> playsong | |
| | | <input checked="" type="checkbox"/> unmatched | FlowFiles that do not match any of the user-supplied regular expressions will be routed to this relationship |
| Id | 19715052-0167-1000-08f3-5a1a614d988f | | |
| Type | RouteOnContent 1.8.0 | | |

Figure 94: nifi_routeoncontent_config_kafka

7 Link the RouteOnContent processor and a PublishKafka processor by hovering over the RouteOnContent process and click on the arrow that appears and drag to connect it to the PublishKafka Processor

The Create Connection settings will appear, For Relationships, check

the appropriate topic being used by the PublishKafka processor that is linked, addsong, or playsong. Repeat the link from the RouteOnContent processor and the other PublishKafka processor and check the correct relationship.

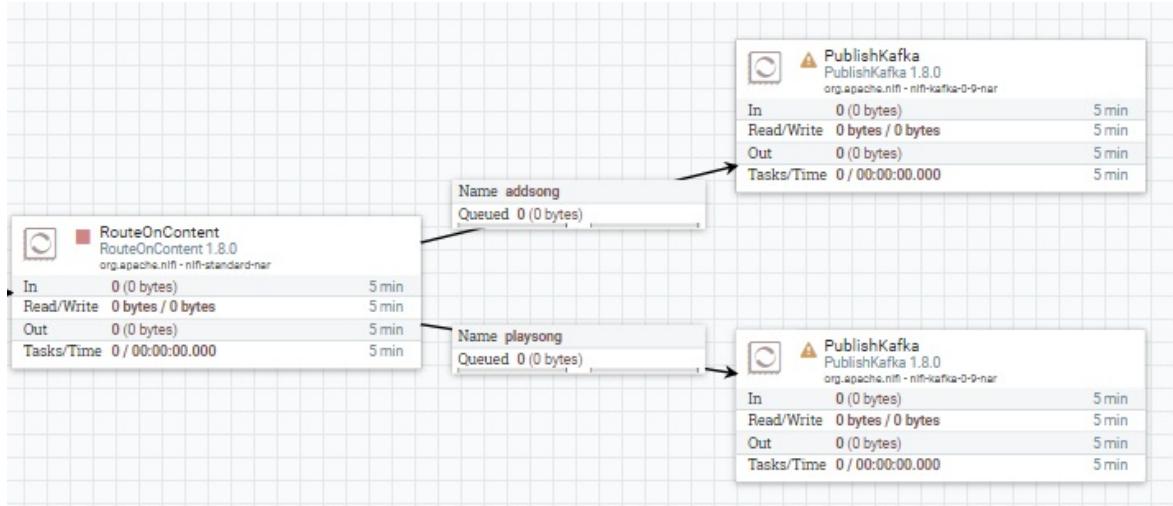


Figure 95: nifi_kafka_linked

NOTE

Hover over the yellow exclamation point on one of the PublishKafka processors. The error will say "'Relationship failure' is invalid because Relationship 'failure' is not connected to any component and is not auto-terminated" This is a good example of the robust validation in NiFi to ensure flows have the proper fail-over properties in place.

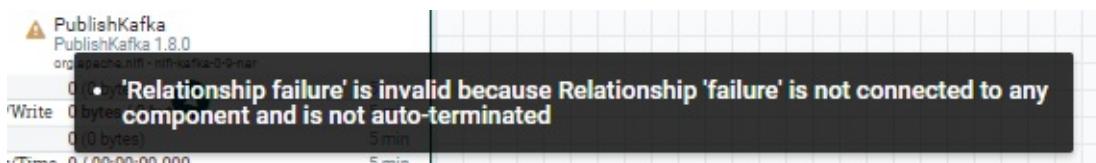


Figure 96: nifi_kafka_error

8 To create a pathway for any data that fails in the PublishKafka processors hover hovering over the PublishKafka processor and click on the arrow that appears and drag to connect it with itself.

The Create Connection settings will appear, For Relationships, check the failure box.

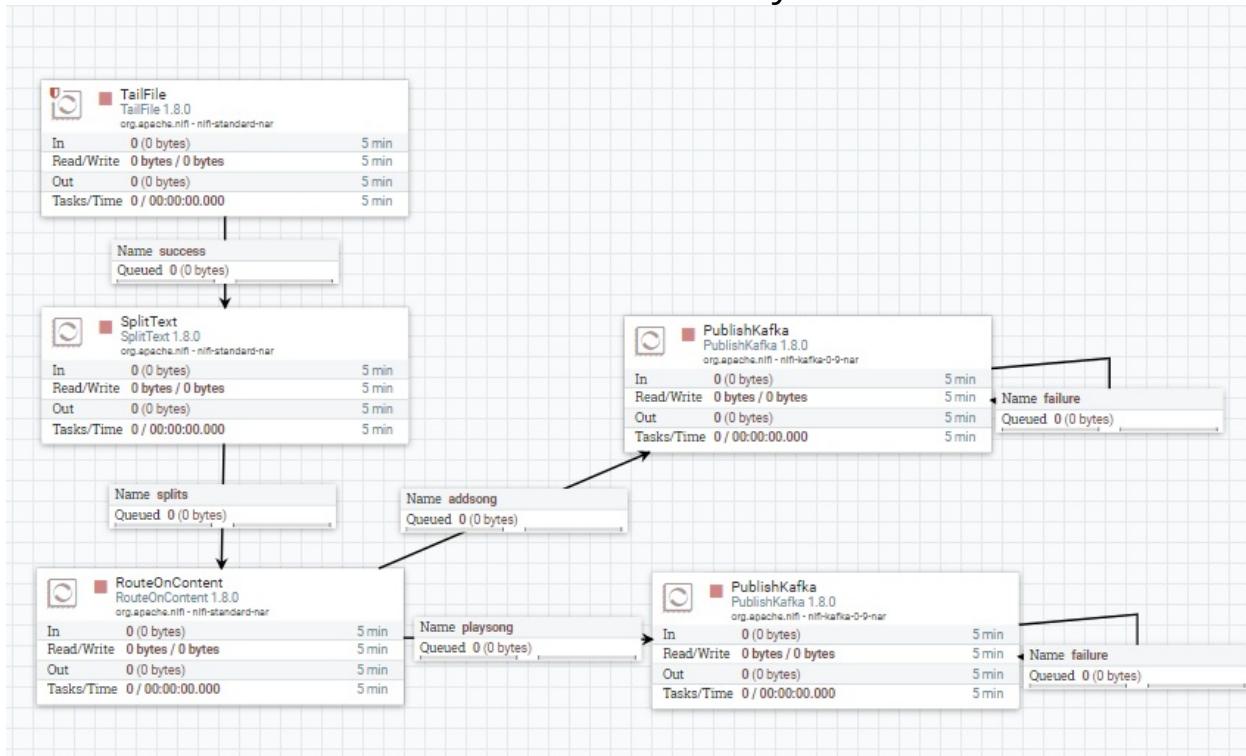
Create Connection

| DETAILS | SETTINGS |
|--|---|
| From Processor PublishKafka PublishKafka | To Processor PublishKafka PublishKafka |
| Within Group NiFi Flow | Within Group NiFi Flow |
| For Relationships <input checked="" type="checkbox"/> failure <input type="checkbox"/> success | |

Figure 97: nifi_kafka_fail

Repeat for the other PublishKafka processor.

The flow is ready to run.



Select all processors while holding down the shift key, right click on any of the processors and click start.

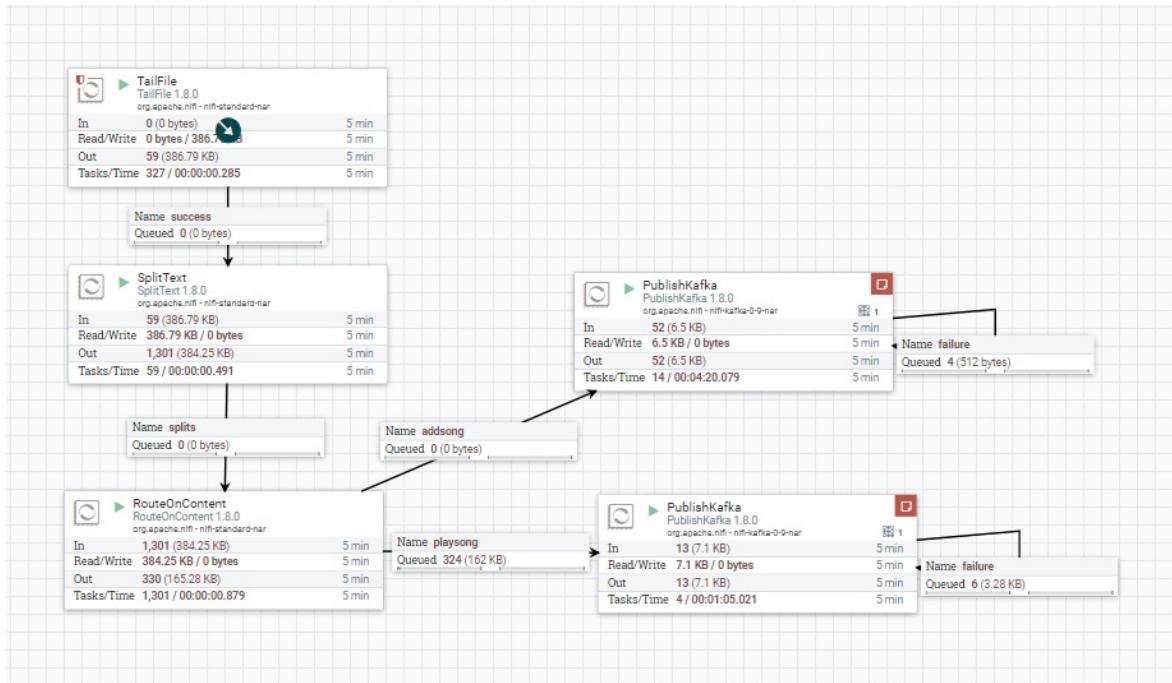


Figure 98: nifi_nifi_kafka_finalflow_run

Data from the selected rows of the log files are flowing all the way through the NiFi flow to the Kafka cluster.

20.8

[link to section within paper](#)

[NiFi Architecture](#)

20.9 ALTERNATIVES TO NIFI

20.10 CONCLUSION

20.11 ACKNOWLEDGEMENT

Batch vs Live stream (Live)

Theory Implementation (e.g. Python) Benchmark A more detailed outline is Paper Title Abstract Requirements

Architecture Implementation Benchmark Conclusions Bibliography
Work breakdown

21 HISTORICAL STORM DATA ANALYSIS WITH COSMOS DB

FA18-523-57 FA18-523-58

Divya Rajendran, Pramod Duvvuri

divrajen@iu.edu, vduvvuri@iu.edu

Indiana University, Indiana University

hid: fa18-523-57 fa18-523-58

github: [cloud](#)

code: [cloud](#)

Keywords: e534, hid fa18-523-57, fa18-523-58, [Data Visualization](#), Data Analysis, Cosmos DB, [Machine Learning](#)

21.1 ABSTRACT

The decisions people make these days are all data driven. The amount of data we collect has increased exponentially and this makes it harder to visualize and understand. We can overcome this hurdle by sampling and visualizing subsets of the data. Sampling refers to the process of collecting random subsets of the data. The only thing one should remember is to pick a good sample of the data or to repeat the process of sampling and visualizing the data subset. In this project, we visualized the historical storm dataset and obtained inferences from our visualizations.

21.2 INTRODUCTION

The disastrous effects of the increasing number of storms all around the world, we thought of taking storm dataset for entire Asia Pacific region and visually analyze the effects of various variables on the storms. We want to check any patterns in the storm data and visualize them geographically so that it might help us to make useful inferences on the storm data. We have identified a data set on

data.world[72] and we aim to identify the various correlations among the attributes and understand if they can help explain the change in frequency of the storms. Data World is a community where we find open datasets from various organizations containing varying data attributes according to the need.

21.3 IMPLEMENTATION

21.3.1 Data set

The data set for the various storms in Asia Pacific region for the years between 1956 and 2017. We have chosen an initial subset of data to visualize locally and then implement the same visualizations on the entire data. The data set contains attributes like region, latitude, longitude, type of category, name, date and hour, speed, and pressure. There are more than 190,000 rows in the dataset. We shall attempt to clean the data before we generate any visualizations.

21.3.2 Related Work

We have identified an earlier work on tropical storm data using R by Stoltzman consulting LLC and have used it as a base reference to visualize our data using Python [73].

21.4 TECHNOLOGIES USED

1. Python is an object-oriented programming language we have utilized in this project [74].
2. The data is being stored on an Microsoft Azure Cosmos DB instance and we will be using Mongo API to connect and retrieve the data.
3. Matplotlib is a 2D visualization library containing plotly which gives us publication ready images and we aim to utilize for identifying correlation between various attributes [75].
4. Seaborn is a visualization tool based on Matplotlib used to draw attractive statistical plots and we aim to show the

- change in number of storms per year [76].
5. Altair is a statistically aimed visualization library which produces output plots which are easily shown on a website [77].
 6. Folium is an interactive mapping tool which plots the data on a map based on the latitude and longitude values [78].

The data is stored in the cloud, we are using and instance of Microsoft Cosmos DB for storage. Cosmos DB is a NoSQL database. The output of our visualization is shown on jupyter notebook.

21.5 VISUALIZATION

This section will contain visualizations of the data and the inferences we draw from them. This section will mostly be data analysis. Here are few questions we hope to answer with our analysis of the data.

1. Category of storms in different latitude and longitude is visualized to identify the variation in category of storms in those location.
2. Category of storms in different regions is visualized to check the correlation between them.
3. Correlation between Speed, pressure and category type is identified by visually analyzing them.
4. Correlation between latitude, longitude and speed, pressure is to be explored.
5. Concentration of storms in different regions should be visually explored.

21.6 MACHINE LEARNING

21.6.1 Multinomial Naive Bayes

Multinomial Naive Bayes (MNB) is a simple classifier that uses the Bayes Theorem and assumes independence between the features and is used for multinomially distributed data, since the data is

represented in sparse matrix of word vectors this would be ideal for such scenarios. We shall use `sklearn.naive_bayes.MultinomialNB` implementation in our project. We have used various train/test splits to analyze the accuracy of our model. The same has been represented in terms of error in the plot below. The highest accuracy we got was when we used the 80/20 train-test split which gave us an accuracy of split gave us an accuracy of 13.4 % (rounded to nearest decimal point)

21.7 SUMMARY

This section will contain the inference and conclusions we draw after completing this project

21.8 FUTURE WORK

In this project, our main aim was to visualize the dataset. After visualizing and making inferences, we would like to propose a model that uses machine learning pattern in our data. We hope to use unsupervised machine learning algorithms to find clusters in our data. This might help us identify further patterns in the data that could not have been possible with just plain visualization of the data. The main problem that we have identified in building such a model is the availability of resources required to run the algorithm without any interruption. Using a local computer resources for running such a model would not be feasible. Hence, we hope to deploy such a model in the cloud and utilize its power and resources for machine learning.

21.9 ACKNOWLEDGEMENT

The authors would like to thank Dr. Gregor von Laszewski for giving the opportunity to work on this project and for providing valuable feedback during the duration of our project. The authors would also like to thank the associate instructors of this class for their help and prompt responses to our questions on Piazza. The authors would also like to thank Microsoft for providing a free trial of Azure Cloud

services.

21.10 WORK BREAKDOWN

The authors Divya Rajendran and Pramod Duvvuri have contributed equally in preparing this report. We shall co-ordinate to equally divide the work that is required to complete the project and the final report before the deadline in December, 2018. Our contributions towards the completion of the project shall be updated in our respective notebook markdown files in our respective git repositories and a summary of our individual contributions shall be added in the final report.

22 KICKSTARTER PROJECTS ANALYSIS

FA18-523-60, FA18-523-64, FA18-523-72

FA18-523-60,

Izolda Fetko, Nishad Tupe, Vishal Bhoyar
ifetko@iu.edu, ntupe@iu.edu, vbhoyar@iu.edu

Indiana University

hid: fa18-523-60, fa18-523-64, fa18-523-72

github: [cloud](#)

code: [cloud](#)

Keywords: MongoDB, PyMongo, Crowdfunding, Virtual Machine, IaaS & DBaaS , Data Analysis,

22.1 ABSTRACT

Crowdfunding is a certain way of raising funds where individuals come together and collectively support projects by investing in them. Kickstarter is one of the leading crowdfunding platforms in the world that helps individuals/businesses in various categories such as art, film, music, theatre, games, design, and other, to raise necessary funds to complete their projects. This paper explores the utilization of big data technologies and cloud services such as MongoDB, virtual machines, MongoDB Atlas, DigitalOcean, AWS, and PyMonogo while analyzing the Kickstarter Projects dataset obtained from the Kaggle datascience platform.

22.2 INTRODUCTION

Many creative individuals and groups around the globe are trying to develop products, projects, and businesses based on their unique ideas and talents. In many cases, an idea or a talent is not enough to accomplish this. Capital funds are one of the main elements of a successful start-up process. Although there is a significant number of

government programs in the United States designed to help small businesses and creative individuals to obtain necessary funds for their projects, a high number of start-ups nonetheless remain capital constrained [79]. This is the reason why the Kickstarter platform was developed and is still successfully operating since 2009 [79].

Kickstarter is a web-based crowdfunding platform that helps support creative arts around the globe [79]. It uses the fundable project model where the would-be business owners or project managers submit their project along with the necessary project information and its funding goal.

"The funding goal establishes a base target for the project and a deadline (generally 30-days) to achieve the funding" [79].

The prospective artists and entrepreneurs typically display their projects in a video form where they have the opportunity to outline the project along with its benefits and funding requirements [79]. The video is shown on the Kickstarter project webpage along with the number of backers, the amount of funds received/pledged as well as the fundraising goal. Since each project is timed, the time remaining in the Kickstarter promotion is shown on the project page as well. The platform is based on an all or nothing funding model which means that the pledged funds are not available to the entrepreneurs until the end of the funding period, which is determined by the goal-setting statement [79]. The financial transactions occur only in cases where projects meet their initial targets and funding expectations [79]. If the initial target is exceeded, the project receives all funding as well as the excess of its goal.

Our project includes several big data technologies while exploring the Kickstarter dataset collected in the 2009-2018 timeframe and made public via Kaggle, a well known datascience platform. The base technology used to store and query the dataset is the widely known NoSQL database called MongoDB. Three cloud services chosen to be benchmarked against each other are MongoAtlas, DigitalOcean, and AWS. Our team had developed a Python program and a packaged

shell script that builds the MongoDB environment on a Unix platform, performs data analysis through visualizations to gain hidden insights and builds a logistic regression model to predict the success/failure of the Kickstarter projects. MongoDB database is hosted on several cloud environments. The time needed to query the dataset was used as a quantitative measure to analyze the relative benchmarking. Moreover, our team provides an ease-of-use review of each service as an addition to the performance benchmarking. This report introduces the reader to the related work in the MongoDB realm, as well as to the chosen dataset prior to presenting the project design and research methods, architecture, technologies, and results.

22.3 LITERATURE REVIEW

MongoDB and Python are both open source technologies. One can get started quickly by building an application on MongoDB using any of the languages that leverage MongoDB's driver. This database offers a native driver called PyMongo to fit Python developer community needs.

"MongoDB stores data in documents, however, they are not like Microsoft Word or Adobe PDF documents but rather JSON documents based on the JSON specification" [80].

There are several advantages of storing data in document format and some of them are flexible schema and ability to store arrays, which are faster to process using native commands of Python scripts [80]. In June 2018, students of the Indiana University, Bloomington Izolda Fetko, Rashmi Ray, and Nishad Tupe explored the France accidents dataset using MongoDB, PyMongo, and Tableau to provide safety recommendations. They used the newest release of the MongoDB driver called the Mongo BI connector that allowed BI tools such as Tableau to interact with MongoDB. The BI connector converts the Tableau's SQL-like commands on structured data into the native MongoDB commands while fetching data [81]. The team also used PyMongo and other web technologies to build the website that could

store, update, process the records live and can be accessed by the global audience. As MongoDB is a NoSQL engine, it scales easily for multiple tables as a single JSON object, and makes query retrieval speed faster than RDBMS, while avoiding complex joins. However, the research also showed that Tableau and MongoDB is a lousy marriage predominantly because Tableau was built before the NoSQL and Big Data were popular and is not yet mature to process relational data [82]. Tableau's slow processing of joins, when connected directly to MongoDB, was one of the main reason authors decided to join the CSV files using Pandas data frame and use MongoDB as a backend tool. This tool allows a stable platform for the user-friendly BI tools such as Tableau to perform analytics on large datasets with millions of rows and also to stand as a robust database on which one can build applications [81]. In the final step, authors created a website and provided links to various dashboards using technologies such as HTML, CSS, JavaScript, Bootstrap, Flask, JQuery, and Chartist [81]. Although the goals of the aforementioned and our current project differ, they do share a lot of similarities such as using the exact same big data technologies MongoDB and Python.

Another paper written by [83], presents an interesting benchmarking of the MongoDB database on several cloud instances. In his report, [83] notes that each virtual instance created for this purpose had MongoDB and PyMongo installed on it. The similarity between this report and our project is that both exploit the benefits of the PyMongo driver and the Amazon Web Services, more specifically the EC2 instances. In addition to the AWS, [83] used Linode, a hosting company that offers a virtual private server (VPS); Rackspace cloud and its open source technology OpenStack; and Windows Azure and its virtual machines [83]. In the final section of his paper, [83] concludes that MongoDB's performance varies from cloud to cloud due to various factors. One of the most important factors that he lists are the fast I/O access and ability of the database to cache all indexes in RAM [83].

The report by [84] shows similarities with our report in the sense that is conducting benchmarking of MongoDB using the Amazon Web

Services. However, in his report, [84] takes a step further and compares the MongoDB performance to other databases such as CouchDB and Apache Spark. He concludes that MongoDB performs well on the cloud with ultra-low latency which makes it a great choice for applications with flexible schema requirements [84].

The article written by [85], presents NoSQL database benchmarking with the use of the Yahoo Cloud Benchmark (YCSB) and Amazon Web Services with an installed Linux operating system [85]. The purpose of his article is to help developers choose the right database for their application. He tested Cassandra 2.0, MongoDB 2.4.6, HBASE 0.92 and concluded that developers need to evaluate different solutions in their search, and test their performance first prior to making any decisions [85]. According to him, all databases are good in some way, but may perform differently in different scenarios, hence, it is important to chose them based on the most needed properties and project requirements [85]. The similarity between this project and our project is the use of the Amazon Web Services as a benchmarking tool for MongoDB.

22.4 DATASET DESCRIPTION

22.4.1 Kaggle API

The technology used to easily obtain the Kickstarter Projects dataset is the newly offered Kaggle Public API. API stands for Application Programming Interface through

“which interactions happen between an enterprise and applications that use its assets” [86].

Kaggle Public API was launched in February 2018 and can be used for creating datasets, kernels, or simply connect with Kaggle [87]. Although still in its beta phase, it allows a more user-friendly data download as well as a seamless workflow for its community members. To be able to use this technology, users need to ensure that they had installed the latest Python 3 version on their machines

as well as the pip package manager [88]. Accessing the Kaggle API is done by using a simple command line; however, this is not possible until a Kaggle account is created [88]. Once the initial step had been completed, an API token can be created, which triggers a download of a JSON file that contains the user credentials necessary to access the API [88]. Once the sign-up had been finalized, various command lines can be used to access the list of competitions along with the files and submissions associated with them [88]. A different set of commands can be used for dataset downloads and dataset creation, while the final group of commands listed on the Kaggle API GitHub page are the commands to manage Kernels, more specifically Kernel pull and push [88].

The importance of the Kaggle's public API is significant. It minimizes the need for its users to manually download large datasets, hence saving them time when working on important projects. It is also helping students in expanding their knowledge and programming experience through practical examples and real-life data that can be later implemented in their professional work.

22.4.2 Kickstarter Projects Dataset

The Kickstarter Projects dataset is publicly available on the Kaggle website and can be accessed using this [path](#). The instructions on how to obtain the dataset using the Kaggle API can be found [here](#). The dataset is available in the CSV format and contains more than 370 thousand projects submitted to Kickstarter between 2009 and 2018 [89]. The dataset variables allow versatile data analysis which is presented in the Observations and Visualizations segment of our project. Other than the project ID, the dataset contains information on the project name; main category and category of campaign; currency used to support the project; fundraising goal (the amount of funds needed to complete the project); project launch (date); project crowdfunding deadline (date); state – current state of the project; actual funds pledged to the project along with the number of backers; the country of origin; and the total amount of funds pledged by currency [89].

22.5 DESIGN AND METHODS

One of the primary aims of the project was to utilize the minimal cost cloud resources. Thankfully, nowadays, every cloud provider provides a free tier service. To leverage this, our team created VM's on Amazon Web Services (AWS), DigitalOcean cloud service with more or less the same basic configuration of memory and hard disk. The AWS and DigitalOcean are primarily an IaaS service, while MongoDB Atlas is a DBaaS . This gave us the opportunity to test the solution on various cloud providers and benchmark their performance with pros and cons. Although the cloud VM's gave us the lot more control and customization over the OS and database level resources, MongoDB Atlas cloud services provided a stable database clustered environment for high availability of data. There was no overhead of configuring MongoDB instance as it is a DBaaS service. We decided to leverage Unix bash script to perform a task automatically. The following bullet points outline the steps taken during our project:

1. Dataset Download
2. Install MongoDB
3. Import MongoDB
4. Conduct Python Analysis

Once the stable infrastructure foundation on the cloud based VM's was achieved, we moved onto loading the data using two methods:

1. Python script based method for DBaaS
2. Mongolimport for cloud VM's

Before beginning the analysis, it is crucial to extract the essential features. We used Python datetime library to get the date, year, month; to clean up the NaN rows carefully and also to calculate the duration of a project. The next step encompassed in data enhancement by extracting the features that can give a foundation to

perform various analysis and develop a machine learning model. During the aforementioned process, our team had utilized the following methods to complete the data analysis and draw insights from the dataset.

1. Cloud and MongoDB set up

The first step was to create a stable infrastructure to perform analysis. This method involved creating the cloud VM's, and use the bash shell scripts to complete the installations. We had also done some testing of the MongoDB connectivity from various machines to the cloud instances. Finally, we had created and prepared the architecture diagram.

2. Exploratory Analysis and Visualization

Data Visualizations that let one discover trends or patterns in a dataset are called the Exploratory Data analysis. Once the data is in a good shape, it is easier to gain insights by using visualizations that often become handy tools for finding interesting patterns.

3. Correlation or Heatmap analysis

The correlation analysis is a statistical method used to evaluate a relationship between two continuous variables [90]. For example, we had used this type of analysis to find relationships between project categories and states.

4. Time-Series Analysis

The time series analysis is a statistical technique which is related to data that is distributed in a series of particular time periods or intervals [91]. We had used it to get a better understanding of the distribution of the projects over years and months.

5. Logistic Regression Model

The logistic regression analysis is a predictive analysis used to describe a relationship between one dependent and one or more nominal, ordinal, interval, or ratio-level independent variables [92]. We had used this model in our machine learning algorithm to predict successful vs. failed project status.

6. Perform MongoDB queries

The gist of our project is to show the MongoDB ability to query in real time, hence we used the MongoDB aggregation framework to analyze the Kickstarter dataset as well.

22.6 TECHNOLOGIES

22.6.1 Technologies and Tools Used

- Python version 3.6 and various libraries such as Seaborn, Matplotlib, Pandas, and Scikit-learn
- PyMongo Driver, Bash Shell
- Cloud service - MongoDB Atlas, 3 node replica cluster
- Cloud service - DigitalOcean, Ubuntu 18.04 ,MongoDB 3.6.3
- Cloud service - AWS , Amazon Elastic Compute Cloud (EC2), Linux, MongoDB 3.6.3

22.7 CODE ORGANIZATION

Our code is checked-in on GitHub and can be accessed by using this [link](#). It is organized as described in the following section.

22.7.1 bin

- main.py
- load_csv.py
- setup.sh
- mongo_uninstall.sh

- BDAA Project - Time Series.ipynb
- MongoDB Queries.txt
- bdaa_project_machine_learning.ipynb

22.8 ARCHITECTURE

Since the beginning of our project, the team aimed to create a scalable Python code that can run seamlessly on different cloud environments. We mainly used cloud computing services from Amazon Web Services (AWS), DigitalOcean as IaaS and MongoDB Atlas as a DBaaS platform. Though AWS is giant a cloud provider with multiple cloud services, we found that DigitalOcean's user-friendly virtual machine (VM's) management interface is equally attractive. The architecture diagram (Figure 99), more specifically the upper left dotted box, shows the client machine or application tier where the source code(.py) was stored and used for performance benchmarking.

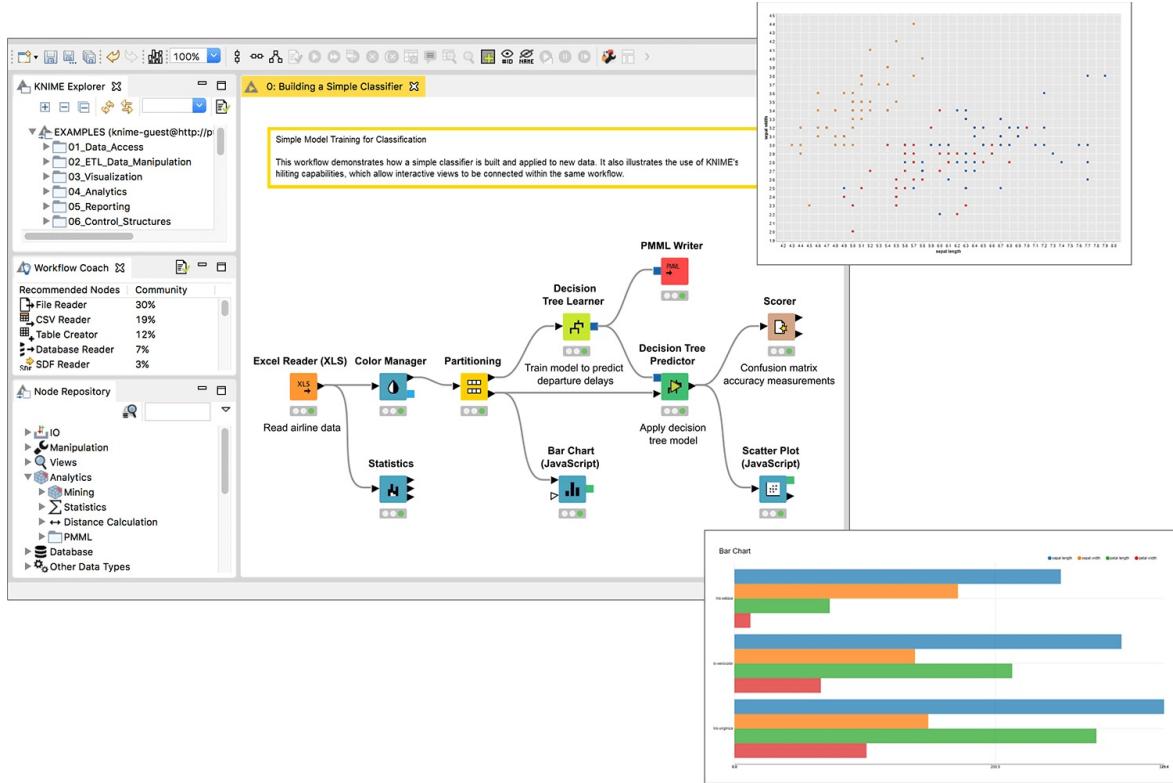


Figure 99: Architecture Diagram

The team also kept a copy of the source code and other scripts on the cloud VM's file systems. We used PyMongo driver as one of the primary components for communicating with the MongoDB database. PyMongo does not only provide the MongoDB driver access to Python libraries, but is also a recommended choice when wrangling data with MongoDB [93].

During the initial loading phase of the Kickstarter dataset, our team imported the raw data using the MongoDB import command line in the DigitalOcean and AWS cloud VM's. A custom bash shell script that installs, configures the MongoDB environment and then imports the CSV data to MongoDB was also written. To load the data in the MongoDB Atlas cluster, we used a simple Python function `load_csv.py`. The primary reason for this is because MongoDB Atlas is a DBaaS cloud service and typically in DBaaS users do not have control or access to OS resources such as a file system. In most basic form, our M0 cluster consists of one primary node and two secondary nodes. The primary nodes are mainly responsible for writing

operations, while the secondary nodes replicate primary's oplog. This way the secondary node's dataset reflects the primary's dataset in cases where the primary node is unavailable [94]. Only the eligible secondary nodes will

"hold an election to elect itself the new primary" [94].

This replica set arrangement ensures high availability of the data. Optionally, one can configure the arbiter node which does not hold any data but keeps the track quorum in the replica set. Since arbiter nodes do not hold any data, they act as a suitable repository to keep the heartbeat information at a cheaper cost [94]. On the contrary, our team had found that the IaaS services provide greater control and customization to the OS resources but add overhead to perform the configuration and other tasks which can be complex and may induce a lag time for writing the code due to incomplete pre-requisites. On the cloud VM's, the team hosted a single instance MongoDB database on Ubuntu 18.04 platform. As post install steps, our team had installed the Python Anaconda Distribution and other necessary libraries essential to complete the analysis. The communication between the MongoDB and Python application happens by connecting string. The connect string, one must have a valid username and password and necessary privileges to access and modify the database. To accept the remote connection, one of the vital steps is to set a value of bind_ip to 0.0.0.0 in the mongodb.conf file that resides on the VM. For all cloud providers, our team had used free-tier services. We observed notable advantage of using MongoDB Atlas free tier service often called as M0 cluster. By default, the M0 cluster comes with three node replica sets and 512 MB storage. The replica set is a group of mongod processes which provide redundancy and high availability to the application while accessing the MongoDB data.

22.9 OBSERVATIONS AND VISUALIZATIONS

Our team completed an exploratory analysis of the Kickstarter dataset. The results of the analysis can be observed in the following

sections.

22.9.1 Exploratory Analysis

Our queries have determined that the Kickstarter Projects data includes six different project states such as failed, successful, canceled, live, suspended and undefined, as shown in Figure 100. Due to the lack of funding, more than 300000 projects in the overall pool of projects had failed, while 200000 projects achieved the funding goal with a successful status. Approximately 70000 projects were canceled while others with less significant counts were marked as live, suspended and undefined.

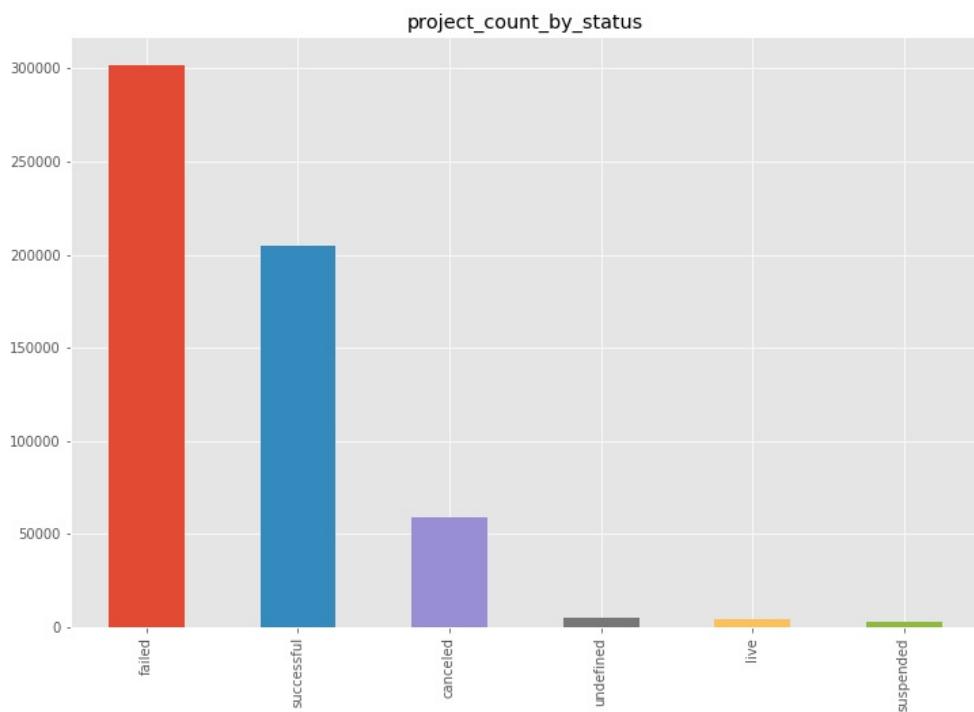


Figure 100: Project States Count By Year

A more detailed view of the project counts was created for the main two state categories - successful and failed projects. The data visualization in Figure 101 shows that the highest number of projects were submitted in 2015. Nearly 70000 projects failed to achieve their funding goal in the targeted time-frame. This was the highest count of

failed projects in the history of Kickstarter. The number of successful projects has steadily increased since 2009 and shows a normal distribution over the years with lot less variance compared to failed projects. Overall, it can be concluded that the number of failed projects was much higher then the number of success projects in each year.

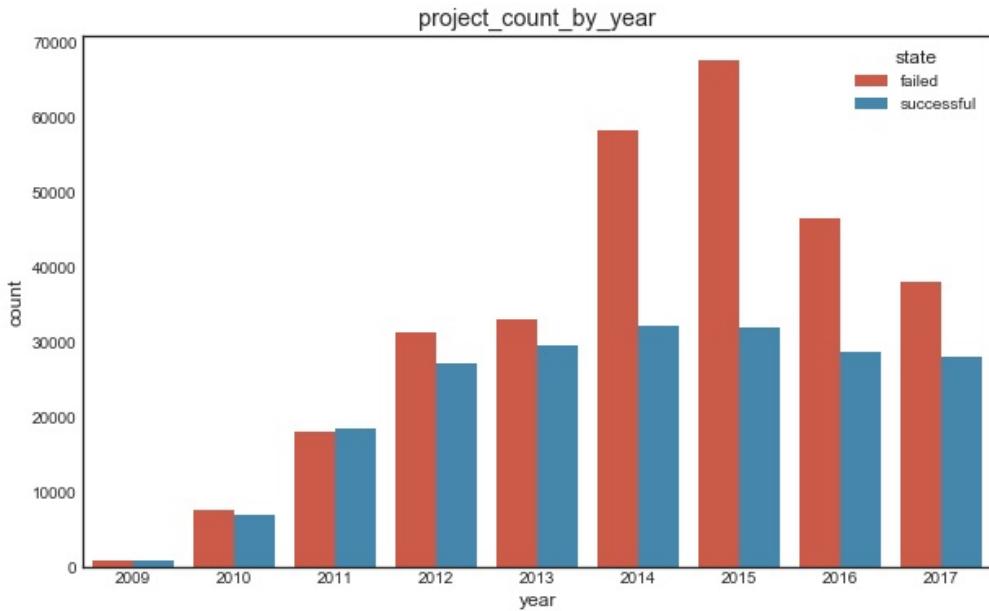


Figure 101: Project Counts By Year(Succces & Failed)

The Figure [102](#) visualization is showing the best markets (countries) for funding projects. Based on this analysis, Austria appears to be the best country for project funding. More than 800000 projects with pledged US dollars launched in Austria achieved their goal within deadline. It is followed by China and New Zealand, which have less successful projects but almost equal amount of failed projects as Austria.

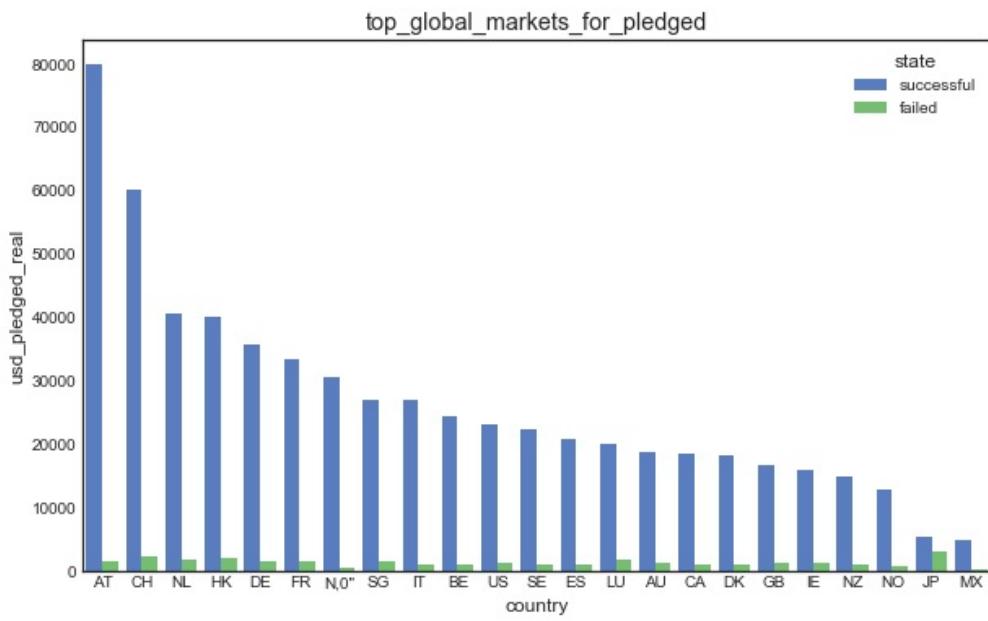


Figure 102: Top Markets For Pledged Funds

The heatmap visualization in Figure 103 is showing project state count against the main category. The scale represents highest count with yellow color and lowest count with dark blue color. The chart is showing highest successful project counts for the Technology main category followed by Dance and Games categories. The count of successful projects in the Technology main category was more than 80000.

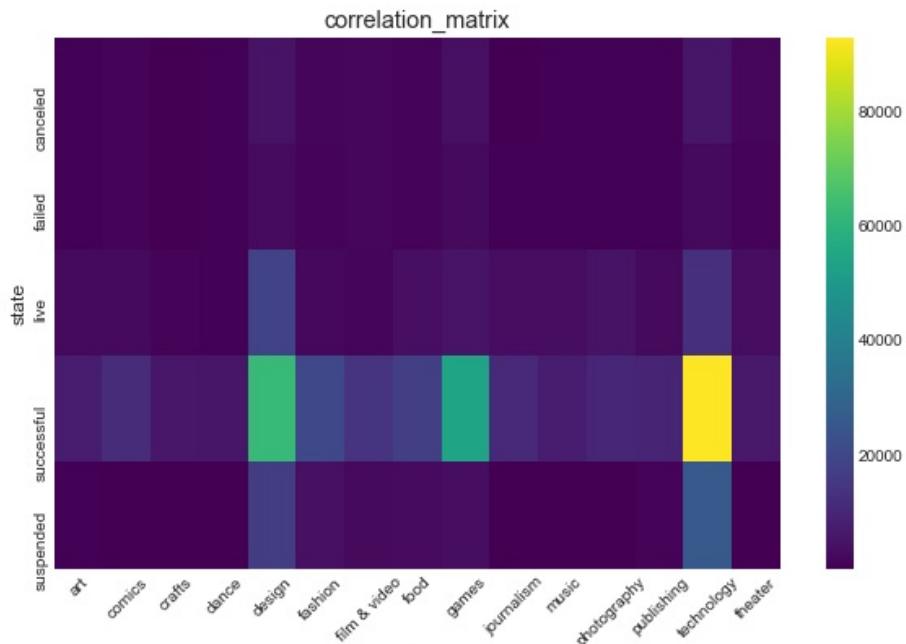


Figure 103: Heatmap Amount Pledged By Category & State

The projects recorded as failed are the ones that were not able to achieve funding goals between the project launched date and the project deadline date. There are some projects which started collecting funding however, were not able to collect the targeted amount. The visualization in Figure [104](#) is showing the targeted funding goal data and actual funding collected for main categories. The Film & Video category has the highest funding goal and only 10% fund achieved by projects. The funding collection of the Technology category was the highest compared to the total fund goal.

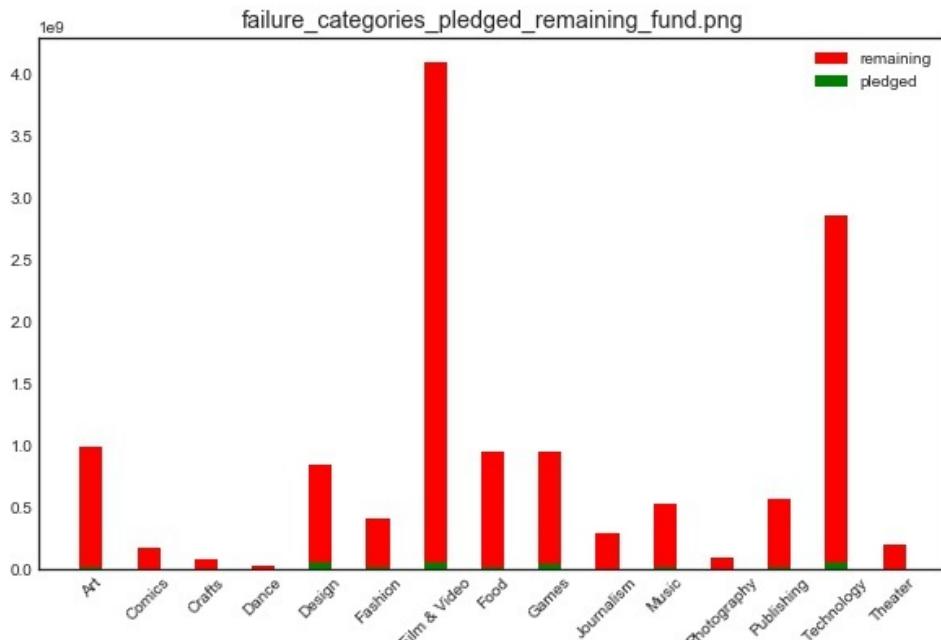


Figure 104: Pledged & Remaining Funds Failed Categories

The project fund duration is a measure that represents the day counts between the project launch date and the project deadline date. The duration box plot in Figure [105](#) shows the average duration for successful and failed projects. The average duration of the successful projects was less than 31 days and the average duration of the failed projects was more than 40 days. The median between successful and failed projects the duration at approximately 36 days.

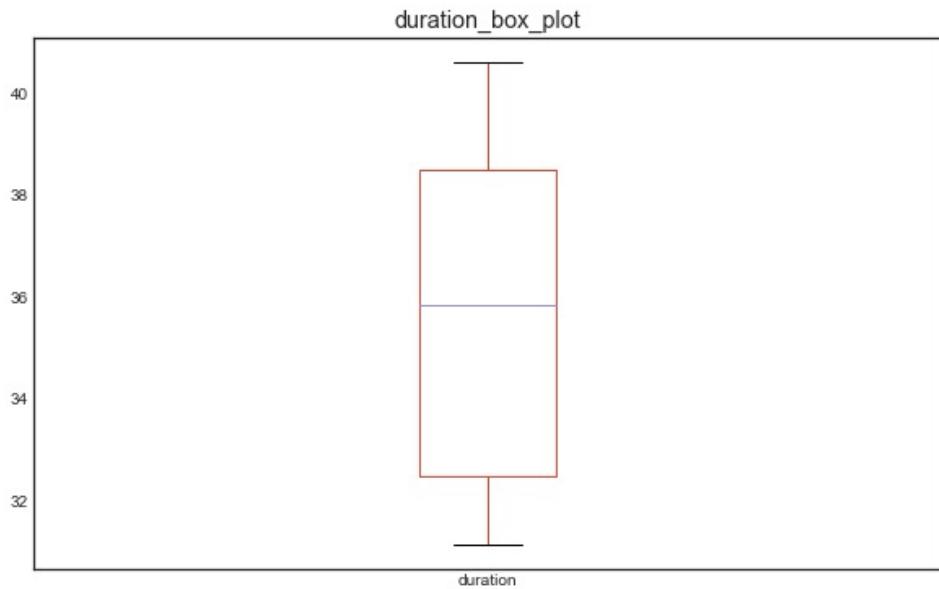


Figure 105: Project Duration Box Plot

22.9.2 Time Series Analysis

The time-series analysis conducted by the team revealed certain trends in the Kickstarter dataset which will be presented in the following segment. One of the trends noticed was that the projects mostly get launched in the warmer months with July being the month with the highest number of projects. The trend of the launched projects by month can be seen in Figure [106](#).

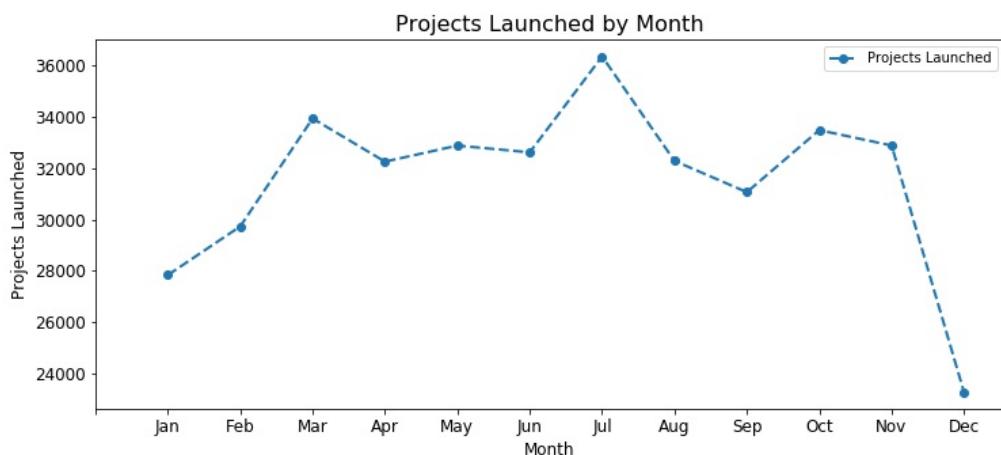


Figure 106: Projects by Month

Drilling further into the project by month data, one can notice that proportionally to the number of launched projects, July is the month with the highest number of cancelled and failed projects. The number of live projects is higher in winter months, mostly in November and December, however, the overall number of projects drastically decreases most likely due to the holidays.

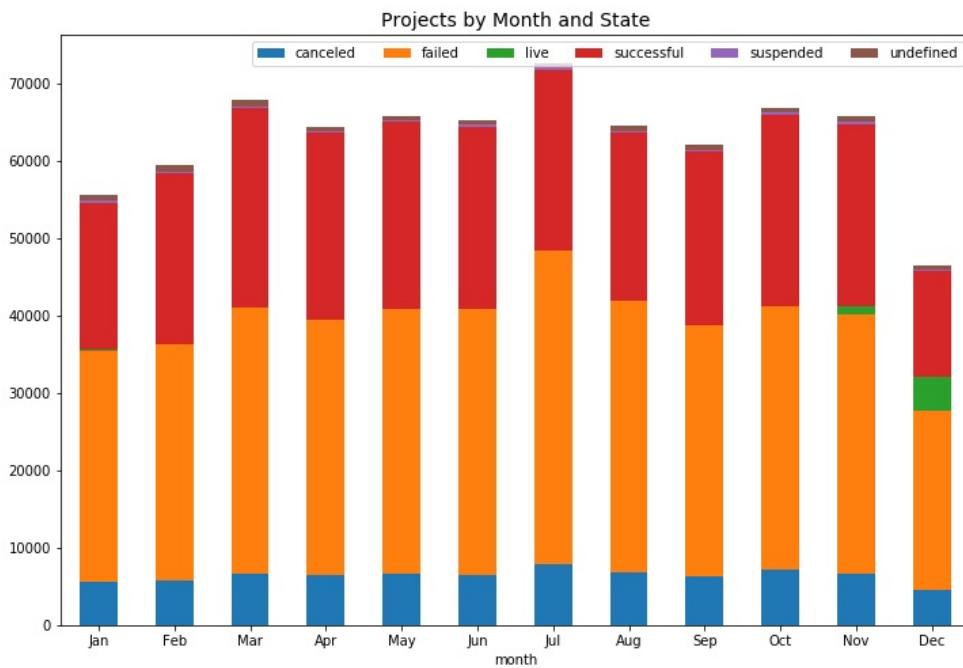


Figure 107: Projects by Month and State

Looking at the same categories over the years time series, from Figure 108 one can conclude that overall, 2015 was a great year for Kickstarter with the highest number of submitted projects, but also, proportionally, one of the highest number of failed projects.

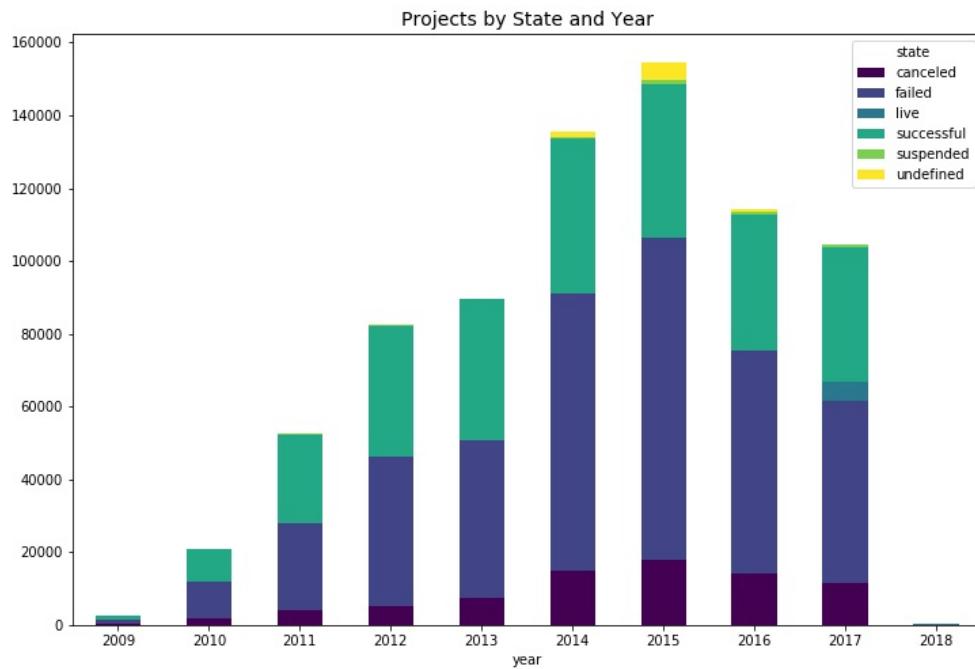


Figure 108: Projects by Year and State

When it comes to the countries from which the projects were submitted in the 2009-2017 timeframe, one can conclude that Kickstarter was popular and available only in the United States in the first four years of its activity. In 2013, for the first time, one can notice other countries making project contributions such as Canada and New Zealand. In the following years, the number of contributing countries rises, however, the leading one still remains the US.

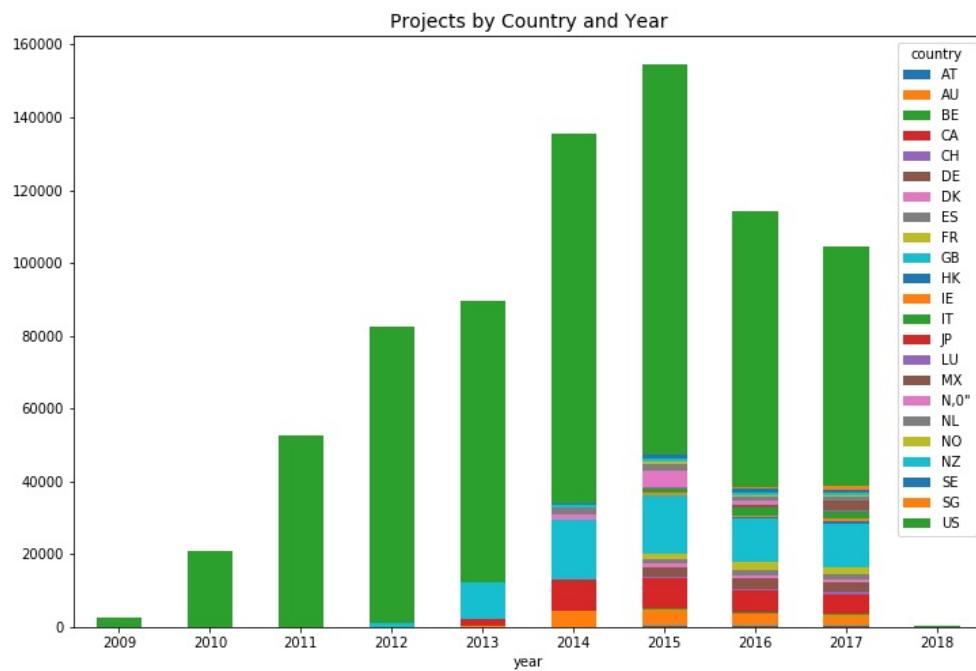


Figure 109: Projects by Year and Country

22.9.3 Logistic Regression

In real life classification problems are more prevalent than regression. Logistic regression helps us solve classification problems by employing the sigmoid function. As shown in Figure 110, logistic regression tells us about the probability as a cut off point[95].

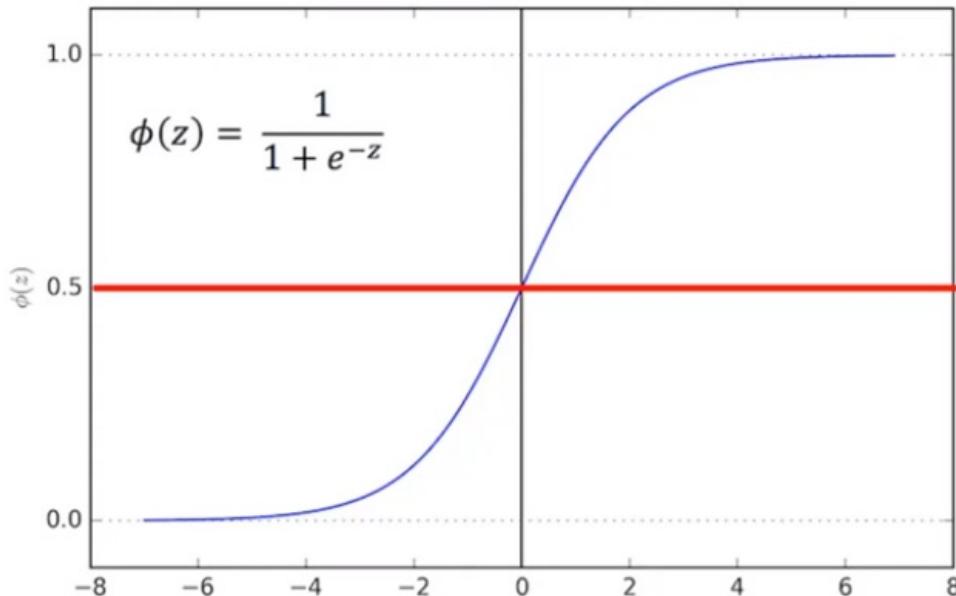


Figure 110: Sigmoid Logistic Graph

The probability of belonging to a class is less than 50%. The values are assigned to class 0 and in our case, the classes were predefined as success or failed. Thanks to the scikit-learn logistic regression model which takes care of the substantial mathematical part, we moved forward with building a model that can predict future projects' states.

To build this model, we used the following steps:

1. Cleaning the dataset.
2. Extract projects with success and failed states only.
3. Explore dataset to predefined the classes
4. Identify top 5 categories
5. Extract features and label data
6. Build the model
7. Predict and measure accuracy

The following figure shows categories with highest success and failed

projects.

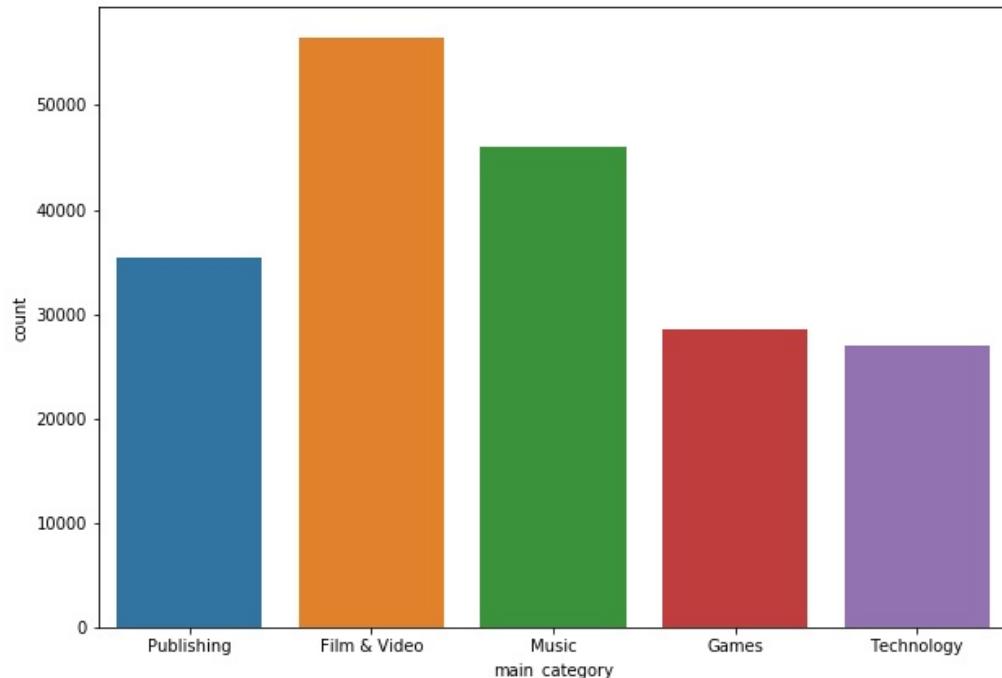


Figure 111: Top Five Categories

We decided to use features such as 'main_category', 'goal', 'backers', 'duration', 'successful' out of which successful feature was generated using pandas get_dummies method.

```
proj_state = pd.get_dummies(data=df_sf_t['state'], drop_first=True)
```

The following figure shows the categorization of our input dataframe into two classes - successful and failed.

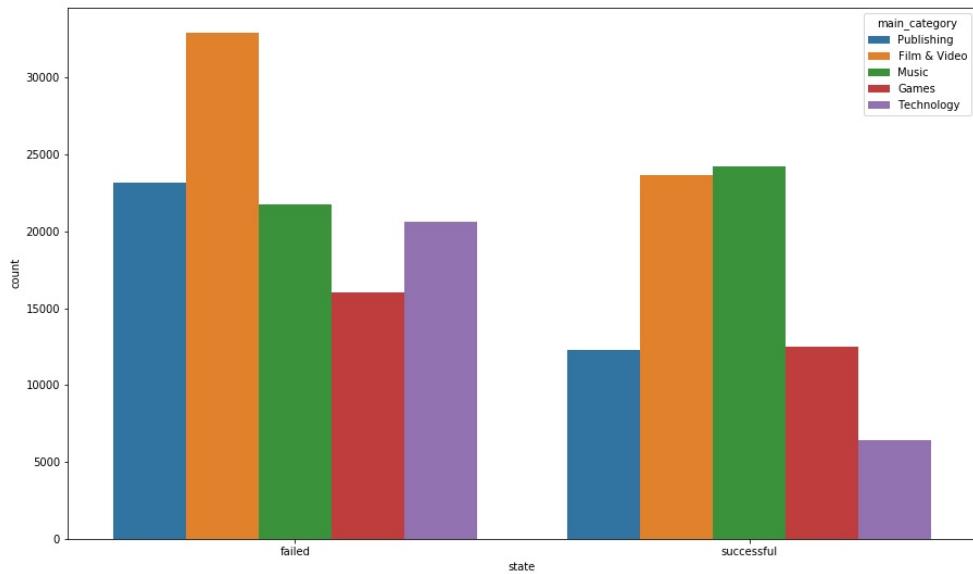


Figure 112: Category Classes

All the main categories were mapped to numerical values as shown in Figure 113 so as to input them as numeric vectors to the machine learning algorithm.

```
cats = {'Publishing':4, 'Film & Video':1, 'Music':2, 'Games':5, 'Technology':3}
```

| main_category | goal | backers | duration | successful |
|---------------|------|---------|----------|------------|
| 0 | 4 | 1000.0 | 0 | 58 |
| 1 | 1 | 30000.0 | 15 | 59 |

Figure 113: Labeled Data

Once the data frame with labeled data was ready we followed the standardized machine learning steps to build the model which involved creating the classifier object; splitting the train test dataset; fitting the model; predicting values; and measure model accuracy. The following figure shows the confusion matrix of our test dataset.

| confusion matrix | | |
|------------------|--------------|---------------|
| y_test(n)= 58048 | Predicted NO | Predicted Yes |
| Actual No | TN = 32970 | FP = 1545 |
| Actual Yes | FN = 4550 | TP = 18983 |

Figure 114: Confusion Matrix

We used the five fold cross-validation method to measure the accuracy of the algorithm. We had found the average accuracy score was close to 90%. To get more accuracy, this model can be extended to additional features and also a larger set of classes.

| Cross Validations | Accuracy | Avg Accuracy |
|-------------------|----------|--------------|
| CV-1 | 89.45 | 89.46% |
| CV-2 | 89.36 | |
| CV-3 | 89.38 | |
| CV-4 | 90.59 | |
| CV-5 | 89.45 | |

Figure 115: Accuracy Scores

22.9.4 MongoDB Queries

Our team had also had the opportunity to write MongoDB queries directly working in Mongo Atlas with a purpose of comparing the ease-of-use related to querying of the Kickstarter data between MongoDB and Python. One of the queries included the count of projects by category.

```
> db.project.aggregate([
    {$group:{_id:{category:"$main_category"}, count:{$sum:1}}},
    {$sort:{count:1}}
])
```

The output of this query can be seen in Figure [116](#).

```

{
  "_id" : { "category" : "Dance" }, "count" : 3768 }
{
  "_id" : { "category" : "Journalism" }, "count" : 4755 }
{
  "_id" : { "category" : "Crafts" }, "count" : 8809 }
{
  "_id" : { "category" : "Photography" }, "count" : 10779 }
{
  "_id" : { "category" : "Comics" }, "count" : 10819 }
{
  "_id" : { "category" : "Theater" }, "count" : 10913 }
{
  "_id" : { "category" : "Fashion" }, "count" : 22816 }
{
  "_id" : { "category" : "Food" }, "count" : 24602 }
{
  "_id" : { "category" : "Art" }, "count" : 28153 }
{
  "_id" : { "category" : "Design" }, "count" : 30070 }
{
  "_id" : { "category" : "Technology" }, "count" : 32569 }
{
  "_id" : { "category" : "Games" }, "count" : 35231 }
{
  "_id" : { "category" : "Publishing" }, "count" : 39874 }
{
  "_id" : { "category" : "Music" }, "count" : 51918 }
{
  "_id" : { "category" : "Film & Video" }, "count" : 63585 }
>

```

Figure 116: Projects by Category

From the results we can see that the highest number of the overall submitted projects was in the Film and Video category. This makes sense when we take into consideration the number of smart phones and camera devices in the world. These technologies have made this art more available to amateurs. In a similar manner, the team had written a query to count the number of projects by year.

```

> db.project.aggregate([
  {$group:{_id:{year:"$year"}, total_number_of_projects:
  {$sum:1}}},
  {$sort:{_id:1}}
])

```

The results in this query have shown that the number of projects varied through out the years, and that the highest number of projects was submitted in 2015. Finally, a query that resembles the previous two is the total count of projects by project state, which revealed that the vast majority of projects that get submitted also get funded.

```

> db.project.aggregate([
  {$group:{_id:{state:"$state"}, count:{$sum:1}}},
  {$sort:{count: -1}}
])

```

A slightly more complex query was used to compute the total amount of the pledged funds, number of backers (investors), and total

funding goal by project category. Our team had determined that with more complex queries, the output appeared less attractive and more difficult to read.

```
> db.project.aggregate([
    {$group:{_id:{category:"$main_category"}, tot_amt_pledged:
        {$sum:"$pledged"}, tot_backers:{$sum:"$backers"}, tot_goal:
        {$sum:"$goal"}}}
])

{
    "_id": {
        "category": "Journalism",
        "tot_amt_pledged": 15301995.2,
        "tot_backers": 182091,
        "tot_goal": 422165908.28
    },
    "_id": {
        "category": "Food",
        "tot_amt_pledged": 131378696.71,
        "tot_backers": 1332751,
        "tot_goal": 1197963524.6
    },
    "_id": {
        "category": "Technology",
        "tot_amt_pledged": 735608801.82,
        "tot_backers": 5356513,
        "tot_goal": 3898910445.2
    },
    "_id": {
        "category": "Design",
        "tot_amt_pledged": 815490920.96,
        "tot_backers": 7255880,
        "tot_goal": 1405280261.92
    },
    "_id": {
        "category": "Theater",
        "tot_amt_pledged": 44713012.92,
        "tot_backers": 513536,
        "tot_goal": 300569615.61
    },
    "_id": {
        "category": "Dance",
        "tot_amt_pledged": 13906929.44,
        "tot_backers": 161275,
        "tot_goal": 38890776.3
    },
    "_id": {
        "category": "Music",
        "tot_amt_pledged": 207294846.96,
        "tot_backers": 2708475,
        "tot_goal": 833613962.47
    },
    "_id": {
        "category": "Film & Video",
        "tot_amt_pledged": 404574432.02,
        "tot_backers": 4197577,
        "tot_goal": 5362378004.68
    },
    "_id": {
        "category": "Photography",
        "tot_amt_pledged": 39501225.45,
        "tot_backers": 428078,
        "tot_goal": 140161865.51
    },
    "_id": {
        "category": "Publishing",
        "tot_amt_pledged": 145090176.71,
        "tot_backers": 2231589,
        "tot_goal": 1161588016.96
    },
    "_id": {
        "category": "Crafts",
        "tot_amt_pledged": 17760300.12,
        "tot_backers": 240342,
        "tot_goal": 102116446.5
    },
    "_id": {
        "category": "Games",
        "tot_amt_pledged": 770331916.1,
        "tot_backers": 11336829,
        "tot_goal": 1786609751.2
    },
    "_id": {
        "category": "Comics",
        "tot_amt_pledged": 74643647.75,
        "tot_backers": 1458090,
        "tot_goal": 219016009.29
    },
    "_id": {
        "category": "Art",
        "tot_amt_pledged": 101547027.64,
        "tot_backers": 1188200,
        "tot_goal": 1149463908.6
    },
    "_id": {
        "category": "Fashion",
        "tot_amt_pledged": 149422709.86,
        "tot_backers": 1401993,
        "tot_goal": 566253100.85
    }
}
> █
```

Figure 117: Total Metrics by Category

In a similar fashion, the average metrics by category were obtained thanks to the following query:

```
> db.project.aggregate([
    {$group:{_id:{category:"$main_category"}, avg_amt_pled:
        {$avg:"$pledged"}, avg_backers:{$avg:"$backers"}, avg_goal:
        {$avg:"$goal"}},{$sort:{avg_amt_pled: -1}}}
])

{
    "_id": {
        "category": "Design",
        "avg_amt_pled": 27119.751279015632,
        "avg_backers": 241.29963418689724,
        "avg_goal": 46733.630260059865
    },
    "_id": {
        "category": "Technology",
        "avg_amt_pled": 22586.164813779975,
        "avg_backers": 164.4666093524517,
        "avg_goal": 119712.31677975989
    },
    "_id": {
        "category": "Comics",
        "avg_amt_pled": 6899.311188649598,
        "avg_backers": 134.7123578888992,
        "avg_goal": 50711.29832250007
    },
    "_id": {
        "category": "Games",
        "avg_amt_pled": 321.7856149413868,
        "avg_backers": 60.01520798930565,
        "avg_goal": 50711.29832250007
    },
    "_id": {
        "category": "Fashion",
        "avg_amt_pled": 6549.031813639552,
        "avg_backers": 61.44779978962132,
        "avg_goal": 24818.246004996494
    },
    "_id": {
        "category": "Film & Video",
        "avg_amt_pled": 6362.733852638201,
        "avg_backers": 66.01520798930565,
        "avg_goal": 84334.00966705984
    },
    "_id": {
        "category": "Food",
        "avg_amt_pled": 5340.163267620518,
        "avg_backers": 54.17246565319893,
        "avg_goal": 48693.745410942196
    },
    "_id": {
        "category": "Theater",
        "avg_amt_pled": 4097.2246788234215,
        "avg_backers": 47.0527114450655,
        "avg_goal": 27542.34542380647
    },
    "_id": {
        "category": "Music",
        "avg_amt_pled": 3992.7356015254827,
        "avg_backers": 52.168323124927774,
        "avg_goal": 16056.357380292
    },
    "_id": {
        "category": "Dance",
        "avg_amt_pled": 3690.7986836518044,
        "avg_backers": 42.80122080794056,
        "avg_goal": 10321.331289808917
    },
    "_id": {
        "category": "Photography",
        "avg_amt_pled": 3664.646576676872,
        "avg_backers": 39.7140736617497,
        "avg_goal": 13003.234577419054
    },
    "_id": {
        "category": "Publishing",
        "avg_amt_pled": 3638.716374329137,
        "avg_backers": 55.96601795656317,
        "avg_goal": 29131.46453729247
    },
    "_id": {
        "category": "Art",
        "avg_amt_pled": 3606.970043609838,
        "avg_backers": 42.20509359570916,
        "avg_goal": 40829.18014421198
    },
    "_id": {
        "category": "Journalism",
        "avg_amt_pled": 3218.0852155625657,
        "avg_backers": 38.29463722397476,
        "avg_goal": 8873.57692534174
    },
    "_id": {
        "category": "Crafts",
        "avg_amt_pled": 2016.1539470995574,
        "avg_backers": 27.28368713815416,
        "avg_goal": 11592.28590078329
    }
}
> █
```

Figure 118: Average Metrics by Category

Although querying in MongoDB is relatively simple, the team had concluded that compared to Python the output results are less attractive and the overall usage is less friendly.

22.10 CONCLUSION

In this project, our team was able to deploy a successful Python application that has the capability of running on various cloud services using MongoDB as a stable backend. The team had observed that the performance was better on DigitalOcean and Amazon platforms (44 seconds) which have a dedicated memory and CPU, as well as more customization options, compared to the MongoDB Atlas cluster where it took 96 seconds to run the aforementioned Python script. Various factors need to be taken into consideration when benchmarking performance on different services. Some of them include load and dedicated resources such as CPU and memory, which play a very important role. Overall, we have concluded that each platform has its own advantages such as ease of use, customization of the environments, however, one of the very important aspects also includes the cost of the offered services. Our team had used the minimal cost approach in this project, and ran the code on the available free tier level services, which could have also affected the performance. When it comes to the crowdfunding data, we have found that the overall success rate of the Kickstarter projects in the 2009-2018 time frame was 36%, and that each year, the number of failed projects was much higher compared to their successful counterparts. We had also identified the most successful categories - Technology, Games, Publishing, Music, Film & Video. From those categories, Technology and Games were among the categories with the highest amount of pledged funds, including Design. The highest number of successful projects was achieved in 2014 while the number of suspended projects was the highest in 2015. Overall, 2011 was the year of the highest success rate. By conducting the monthly time series analysis, we had identified certain trends where we could see that warmer months yielded more project

launches while the colder months were sub-sequentially less project heavy. Although the platform was created in the United States, countries such as Austria , China, and Netherland shown to be the top countries for generating pledged funds. Project duration was also analyzed and the team had concluded that the average duration of the successful projects was less than 30 days, while the upper limit for the maximum project length was 45 days. Moreover, thanks to our implemented machine learning methods, we were able to predict the success and failer of the projects based on the extracted features such as goal, backers, categories, and duration. However, we also believe this model can be expanded with additional feature engineering and stimulate sponsors to fund projects that are more likely to be successful with the sole purpose of keep encouraging the start-ups.

22.11 ACKNOWLEDGEMENT

The authors would like to thank the Big Data Applications and Analytics (I-523) course teaching staff, mainly professor Gregor von Laszewski for their support and guidance during this project. Also, we would also like to extend our appreciation to Kaggle for providing us with the Kickstarter Projects dataset, as well as to other online sources for allowing us to gather meaningful insights and programming support.

22.12 WORKBREAKDOWN

22.12.1 Nishad Tupe

- Project cloud architecture and implementation research
- MongoDB Atlas, Digital Ocean Cloud VM set ups
- Bash script for Dataset download, installation & import of MongoDB
- Python data loading scripts and exploratory visualizations
- Python Logistic regression implementation

22.12.2 Vishal Bhoyar

- Project requirements and design research
- Exploratory Analysis of data
- Amazon Cloud VM set up
- PyMongo research

22.12.3 Izolda Fetko

- Project dataset and literature research
- Testing MongoDB Queries on Cloud VM's
- MongoDB Aggregation framework research
- Python Time Series Analysis

22.13 NISHAD TUPE, VISHAL BHOYAR, IZOLDA FETKO

- Performance benchmarking
- Project Papers

23 TWITTER TEXT MINING USING PYTHON, MONGODB, AND AMAZON WEB SERVICES - FA18-523-61

finalize title

refs missing

please remove first person writing style

please remove phrases such as in this paper, in this term paper ...

- Still a draft. Images and references will be inserted/updated soon.

Jay Stockwell

jaystock@iu.edu

Indiana University

hid: fa18-523-61

github: [blue icon](#)

23.1 ABSTRACT

This term paper will provide a thorough analysis, and detailed, informative document pertaining how the Twitter API, in conjunction with Python and MongoDB Atlas can provide an effective solution for collecting and analyzing tweets. The paper's objectives are to provide a thorough understanding pertaining to how the twitter API works, how twitter data is constructed, how Python's tweepy library works to collect data, and how MongoDB Atlas cloud service provides a robust environment to store twitter data for future analysis. The paper will also discuss the implications of using the Twitter app and its data within the realm of Big Data and Natural Language Processing.

23.2 INTRODUCTION

Twitter allows individuals and organizations to post short messages

called tweets. Individuals can follow other sites within Twitter to receive their tweets and to stay informed in subjects that appeal to them. Twitter is also thought of as an instant messaging and micro-blogging application that allows users to communicate their opinions and thoughts in a completely open and uninhibited manner. One major component of Twitter is the hashtag. Users can append a hashtag '#' in front of a word or phrase within their tweet in order to categorize them so other users can find and contribute to your tweet [96]. The hashtag can quickly draw attention to, and promote your tweet very effectively. The implications of how the data on Twitter can be used are simply astounding. In particular, Twitter is used by businesses to find new customers, market new products, find marketplace trends, and many other uses.

Twitter data, and social media data in general, represent a huge component of Natural Language Processing (NLP). NLP in a nutshell is the ability of machines to decipher and understand human language ????. This can sometimes pose a challenge because language can tend to have multiple meanings and contexts, and can prove to be very ambiguous to machines which tend to rely on highly structured and precise programming languages ????. Recent advances in NLP technology have eliminated some of the challenges facing NLP, specifically with part-of-speech tagging and Natural Language Understanding, and have greatly improved how semi-structured and unstructured text data is analyzed and leveraged today. NLP is largely based on the deep learning concept, which is prevalent in the emerging world of Artificial Intelligence. Deep learning algorithms look for patterns in data and use those identified patterns to help machines derive understanding from textual data. Another aspect of NLP that has major implications for Twitter is Sentient Analysis. Sentient Analysis can help organizations ascertain whether a particular product is seen in a positive or negative way based on user comments or reviews. Sentient models can analyze the language components and provide valuable data about products and/or services that can affect corporate decision making and product marking and placement strategies.

Data Scientists can mine the data on Twitter and discover new insights and uses of the data through the applications of machine learning algorithms, such as the deep learning algorithms mentioned earlier. I will be examining through the use of the Twitter API, how to mine for tweets that contain the hashtags #dog, #cat, or both. I will then examine how to clean up the data in python and I'll also examine and contrast two different machine learning classification algorithms to ascertain just how effective it is to classify tweets into specific categories. This type of analysis can have far-reaching implications as aforementioned previously, and specifically we can try to answer many questions about an individuals tastes, interests, and hobbies based solely on the information in their tweets. I also want to examine how effective twitter data is being stored within cloud database, in particular MongoDB Atlas to determine if that platform provides an effective storage infrastructure for json-styled, semi-structured data.

23.3 TWITTER API AND PYTHON IMPLEMENTATION

Twitter provides a dedicated API platform for developers to allow for the development of custom applications to collect and use tweets. This API can be used for both personal and business purposes. Users can tap into the vast social network to for numerous reasons such as collecting specific tweets and placing in a datastore, integrating tweets from twitter within your own website or application, monitor your own twitter accounts to see how individuals are engaging ??. Twitter provides several different API's for individuals to use based on their final goals of how they intend to leverage the data. For most basic needs, the standard API will work. However, twitter offers an Enterprise API for organizations that depend on Twitter for their day-to-day business.

In order to gain access to Twitter data through the API, users are required to create a Twitter account. The signup process requires users to provide basic user information, as well information about your purpose and intent of your objectives with Twitter. Upon completing the sign up process, Twitter reviews your information and

then assigns 4 keys that will be required for authentication between your application and Twitter. These 4 keys consist of consumer tokens and secrets that provide application authentication, and access tokens and secrets that provide the actual user or account authentication ???.

One method of connecting to, and collecting Tweets is through Python. Python is an open source, general purpose programming language that has gained tremendous traction in the data science field recently due to its ease of use and the availability of many different toolkits that can be implemented to complete data science related tasks. Python toolkits such as numpy, sklearn, pandas, and matplotlib can be leveraged to apply statistical analysis, machine learning algorithms, and quick visualizations using your data ???.

As Python has gained momentum with the Data Science field, another field in which the language is gaining ground is Big Data. Python is now compatible to work with the popular big data tool Hadoop ???.

"Python consists of Pydoop package which helps in accessing HDFS API and also writing Hadoop MapReduce programming. Besides that Pydoop enables MapReduce programming to solve complex big data problems with minimal effort ???".

Python is also very scaleable within this environment. Another great component is that Python has a very large, robust user community that users can reach out to for guidance on all sorts of coding related topics. Due to the popularity of the language, experienced users are extremely willing to participate in the community to facilitate its growth and continued usage within the programming world.

Python can be leveraged using a basic shell program, which can accomplish basic tasks, or with an interpreter or IDE (Integrated Developer Environments). An IDE is a dedicated program, integrated with several tools such as debuggers, build and execution tools, syntax highlighting, and source control tools, that are specifically used to writing software code ??? . Some examples pf well known Python

specific IDE programs are PyCharm, Spyder, and Thonny. When tasked with developing more sophisticated Python code for data science purposes, it is a good practice to use a dedicated code editor well suited to the task.

In regards to working with Twitter, there is a Python library entitled Tweepy that can be installed along with any other required libraries and toolkits. Users will need to use the pip command at a terminal screen in order to install tweepy. PIP is a package management system used by Python to install and manage larger software packages ???.

```
pip install tweepy
```

After installation is complete, users will need to use the import function to fully load the tweepy program into the python script that's in development.

```
import tweepy
```

The next step is authentication between the Twitter API and python. As mentioned earlier, there are 4 required keys needed for this step for which Tweepy is responsible for completing. Tweepy currently supports oauth authentication and authentication is handled via the tweepy AuthHandler class in python ??? . Oauth authentication is considered the standard method for token authentication by 3rd-party applications such as Facebook and Twitter. Oauth works on behalf of the end user to provide a token which authorizes specific information to be shared between applications ??? . There are four variables in the python code that will be used to hold the security information

```
consumer_key=YOUR_CONSUMER_KEY  
consumer_secret=YOUR_CONSUMER_SECRET  
access_token_key=YOUR_ACCESS_TOKEN  
access_token_secret=YOUR_ACCESS_TOKEN_SECRET
```

Once tweepy is set up and the user passes through the authentication step, users can now begin further development on python code that can be implemented against Twitter data.. Tweepy

collects tweets in real-time, so the script will be collecting tweets that were released just prior, or during the implementation of the script. The tweepy program in Python can only collect 100 tweets at a time, so the script can be built to contain a function which runs in an iterative loop to collect a set number of tweets as defined by a 'MaxTweets' parameter. Once that parameter value is met, the function terminates.

The requirements for this project are to run a python script to collect twitter data, specifically tweets with the hashtags #cats and #dogs. Next, is to store the twitter data within a MongoDB database using my local computer and also in the cloud using MongoDB. In order to pull twitter data, users must create a Twitter API account through the Twitter developers website. After creating an account, users wil be provided with four distinct security tokens that users will need in order to connect to the Twitter API through Python.

Next, I installed the latest version of Python, which is version 3.7.1. Python is an open source general purpose programming language that is very useful for data science projects. There are numerous libraries available to install within Python that will help you accomplish your end goal. For this project, the primary library that I want to use for collecting twitter data is tweepy. Tweepy is designed to handle multiple aspects of twitter tweet collection including authentication, connection, session management, and reading and routing incoming messages ???.

23.4 DATA

Twitter data, when extracted, is in JSON format. JSON (Javascript Object Notation) is a data interchange format that is both easy for humans to read and write, and easy for machines to parse and implement ??? . A json object is basically a set of key value pairs ending contained within two braces. This format borrows heavily from the Javascript programming language, hence the Javascript component of the term [???. The format is considered to be structured, as you would expect to see relational data within a table,

but it's considered to be thought of as semi-structured which "...the structure is either flexible or not fully predefined. It is sometimes also referred to as a self-describing structure ???." The JSON datatype has proven to be very popular and has garnered wide usages with websites due to the effective way in which JSON data can be exchanged between client and server web applications ???.

Below is sample JSON object:

```
{"first name": "Jay", "last name": "Stockwell", "hometown": "Louisville", "hobby": "playing guitar"}  
[<| |>]
```

A json array is essentially a collection of key values contained within two brackets

```
{  
    "first name": "Jay",  
    "last name": "Stockwell",  
    "hometown": "Louisville",  
    "hobby": "playing guitar"  
    "favorite cars": ["Porsche", "Lamborghini", "Jeep"]  
}
```

Twitter JSON data is comprised of many different components, all of which are used to describe individual tweets.

"At Twitter we serve many objects as JSON, including Tweets and Users. These objects all encapsulate core attributes that describe the object. Each Tweet has an author, a message, a unique ID, a timestamp of when it was posted, and sometimes geo metadata shared by the user. Each User has a Twitter name, an ID, a number of followers, and most often an account bio ???".

Below is an example of a tweet and a description of the content:

- insert image of sample json twitter record

The tweets follow a parent-child construction. All tweets contain a user object which can also contain a geo-tagged child object describing the geographic location of where the tweet originated. The tweet also contains an entities object that consists of information

such as assigned hashtags, URLs, user mentions, and any sort of media material ????. The data record also contains a flag to indicate if the tweet has been retweeted, or has been forwarded by someone else to another person. A retweet is typically comprised of someone else's comments that a user would like to share.

For this term paper, I created a Twitter API account and used Python to connect to Twitter directly using a set of credentials that were supplied to me. Using a python script, I created two sets of data; one for tweets that contained the hashtag #dog and a second dataset that contained tweets with the hashtag #c .at. After the data is pulled, the json data is converted into dataframes using the pandas package.

23.5 TWITTER CLOUD STORAGE

There are many options available today to store twitter data within a cloud storage platform. Cloud storage consists of storing data within logical pools that can span multiple servers and multiple locations throughout many locations ????. Cloud services can be accessed through a dedicated cloud service platform, and website APIs such as cloud desktop storage and gateways. ????. Some of the main players in today's cloud computing market are Amazon Web Services, Microsoft Azure, and Google iCloud. The focus of this paper will be on the Amazon Web Services platform.

Amazon Web Services (AWS) provides a portfolio of services that can help organizations deal with the voluminous amounts of data that's available in Twitter. AWS provides a cloud storage service called S3 which is capable of storing data of any type from a variety of sources including web sites, mobile apps, and even IoT sensors ????. Used in conjunction with Amazon Glacier, and Amazon Glue, one could build a secure data lake in the cloud that could be set up to contain streaming twitter data. Once the data is in the data lake, one can take advantage of cutting-edge advanced machine learning and analytics capabilities available through additional Amazon services such as AWS Athena (Interactive analytics), AWS Kinesis (Real-Time Analytics), and AWS Sagemaker and Deep Learning AMIs ????.

- insert AWS image here

A lot of organizations today leverage the power and scalability of the S3 platform and come to rely on the system for its day-to-day data needs. That reliability was tested during an S3 outage that occurred on February 27th, 2017 that effected Amazon's entire US-EAST-1 region. This outage caused widespread website outages and vast disruptions to Amazon's clients. The issue was caused by an Amazon employee typing an incorrect command which caused several key subsystems to go offline ????. This was an embarrassing incident for Amazon that just goes to show how important cloud services have become in recent years.

In addition to the S3 platform, Amazon includes other services that allow users to set up and manage virtual machines and virtual cloud platforms. Users can create a cloud cluster leveraging one of these services and run a different database application that is more suitable to their data needs. MongoDB Atlas, an open source database storage system that provides support for JSON document style data, is one of those types of applications that can be stood up and managed on an Amazon Virtual Private Cloud (VPC). Mongo Atlas provides a platform from which users can create and manage their own clusters, as well as set up automation jobs to manage provisioning and deployment tasks ????. Users are required to create an account with Mongo DB Atlas, which provides a minimum of 512 MB of storage for a free cloud database, but can be updated to 10 GB or more along with dedicated RAM, backups, and performance optimization with the creation of a dedicated cluster ????. In addition to running on Amazon VPC, MongoDB Atlas can also run on Microsoft Azure and Google Cloud.

Connecting to MongoDB Atlas from python is fairly straightforward and requires the pymongo python library to be installed. The pymongo library contains several tools that enables the user to connect to a MongoDB data, either on their local machine, or a database hosted on a MongoDB Atlas cluster. Users must use the pip method from a command prompt or terminal screen and run the

following command:

```
python -m pip install pymongo
```

After the pip command is completed and pymongo is installed, the library must be imported into python using the following simple command:

```
import pymongo
```

The Mongo Client component is one of the pymongo tools allowing for python to connect to a MongoDB database. Connecting to a local instance of MongoDB is simple and requires that the software is fully installed and the MongoDB service is up and running. Also, port 27017 must be available on your machine in order for the two applications to communicate properly. The python command for connecting to a local MongoDB instance is as follows:

```
client = pymongo.MongoClient("localhost", 27017)
```

Connecting to a MongoDB Atlas cluster involves a few more steps. First, before you can start up a cluster, there are a few requirements that need to be met. Users must have the correct TLS/SSL support in place and be able to have the correct SNI driver installed in order to connect **???**. Users must also add their IP address to the MongoDB Atlas Whitelist. This ensures that the only connections allowed will be listed in MongoDB's whitelist **???**. Next, users must create a user account with Admin access in order to be able to administer the cluster. After those steps are complete, and the cluster has been started up, users must choose a connection method to use in order to connect to another application. The connection dialog contains different connection strings for different versions of applications including Java, C, C++, Perl, Ruby, and Python. To set up a connection between MongoDB Atlas and Python, users will need to select a URI connection string based on their version of Python. Next, they will need to paste that string within their python script within the following pymongo command:

```
client = pymongo.MongoDB(mongodb://USERNAME:PASSWORD@cluster0-sample-mongodb.net:27017/<DATA>)
```

After this is complete, and you're able to connect without any issues, you can begin to create databases, collect and insert data into the MongoDB database, as well as other data related tasks. Just a note on connecting to MongoDB, users will sometimes receive error messages pertaining to DNS connection errors. To resolve this, users need to install the `dnsPYTHON` library using the pip method and importing the resolver tool from the `dnsPYTHON` library within python.

```
python -m pip install dnsPYTHON
```

```
from dnsPYTHON import resolver
```

Once fully connected to MongoDB, you begin to 'listen' or search for tweets that contain a specific word, phrase, user, #hashtag, or any other piece of information that you're interested in mining for. There's a couple of ways to set up the search criteria, but typically a variable is created that contains the search words such as listed below:

- insert code samples for setting up variables, search words, count, periods.

You can configure tweepy to work in a search or streaming manner. Tweepy has a class entitled StreamListener that will access the Twitter API and pull all tweets are created using the specified criteria **???**. Once executed, the script will continue streaming until the script is stopped. The `api.search` method will provide a collection of tweets based upon a count variable specified earlier. One thing of note is the Tweepy will only return 100 tweets at a time, so the script will need to contain an iterative loop function to run continuously until the value specified in the count variable is met.

After the script is run, the data can now be inserted into the MongoDB Atlas database. The `pymongo` library includes an `insert` function that will allow the user to insert either one record, or many records into the database. In order to use the `insert many` function, you will need to create a collection database object. A MongoDB collection is essentially a collection of documents, or in relation to Twitter data, a collection of json documents **???**. A collection can be

thought of as being similar to a SQL relational table **???**. The collection, if it doesn't already exist, can be created within the insert statement itself. Below are two examples:

- insert sample insert statements

The python script can be run multiple time to continuously gather tweets. However, it is important that we don't collect the same tweets every time we run the script. Therefore, it is adviseable to create an index object after the initial connection to the MongoDb database and database table. This will ensure that each object is properly indexed and contains a unique ID field. Once the script is run subsequent times, there is an argument in place to check the ID field and ensure that duplicate tweets are not being added.

- insert sinceID = 1 argument

Twitter Data contained within the MongoDB database can be queried from within Python using specific command line arguments and functions from the pymongo library. The argument findone() will return the very first record in the database and will include every item associated with that record. With Twitter data, the findone() argument will return all of the components of the first tweet collected such as key, timestamp, description, user, etc. If you want to zero in on one specific item within the tweet, you can add some additional information to the findone() argument. For example, if I wanted to see the key of the first tweet, I would add 'key' to the findone() argument as findone.key(). The argument needs to also contain the data collection that we are querying. The full line is below:

- insert findone command image.

To return all of the records in the database, a cursor object must be created in the python script that will allow the user to read and analyze all of the records within the collection **???**. A cursor can be created by setting up a find() argument within a collection. You can also limit the results by including a limit argument set to the desired number of records.

- insert image of find() command with limit argument.

MongoDB Atlas was designed to handle large datasets by spreading the data among many servers with the cloud computing platform ????. MongoDB can also be connected to a Hadoop instance to deal with the upcoming challenges that are presented by Big Data. This creates an ideal environment to house streaming Twitter data as MongoDB is perfect for document data types such as JSON. MongoDB is also thought of as being a great platform for working with Big Data because of the high scalability offered by MongoDB ????. However, there have been some points of contention that have arisen with NoSQL databases such as MongoDB. Some users that have used MongoDB to store twitter data have reported issues related to duplicate data, which was addressed earlier with the creation of an index. Another issue is that because of the structure of Twitter data and the document storage type, it is very hard, if not impossible to do SQL style joins within the data ????. Social media data is highly unstructured, and individuals that are accustomed to working with relational SQL based databases may find this type of document styled data very hard to understand, analyze, and construct effective queries against.

Conversely, querying and analyzing twitter data in python is an compelling and direct way to open many new doors to insights.

23.6 DATA SCIENCE ALGORITHMS FOR TWITTER DATA

For this paper, I decided to perform an analytical exercise against a twitter data set to investigate and compare two different algorithms and how effective they are for analyzing and classifying tweets. I've determined from my research that Deep Learning algorithms such as Neural Networks are very effective when implemented against twitter data, but I wanted to also compare the accuracy scores and performance to that of a the Naive Bayes classification model. I will be collecting two datasets in python using the tweepy tool; one containing tweets with the hashtag #cats, and a second dataset containing tweets with the hashtag #dogs.

The Naïve Bayes classification method is based on Bayes Theorem probability theory to classify data into distinct classes. The method assumes independence between all of the attributes used in the algorithm and it works on the assumption that a particular feature in a class is unrelated to the rest of the features included in the analysis ????. Naive Bayes is a fairly simple algorithm to set up, and it runs very effectively against very large datasets such as those used in Big Data.

Below is the mathematical formula that comprises the algorithm. The formula follows the Bayes Rule of Probability:

- insert image of mathematical formula
- insert information about what the variables mean

The foundation of the formula is that it calculates the posterior probability $P(c|x)$ from the other variables in the equation, $P(c)$, $P(x)$, and $P(x|c)$???.

The Naive Bayes classification method has gained popularity recently with usage as an effective spam filter, text analysis, and medical diagnosis tool. I wanted to utilize the multinomial Naïve Bayes model as it appears to be more suitable for text classification by explicitly modeling the word counts and makes adjustments to any underlying calculations involved.

The second algorithm I wanted to experiment with is Neural Networks. Neural networks function as an information processing model that closely mirrors how biological nervous systems , specifically neurons, process information ????. Neural networks make predictions by learning the relationships between your data features and other previous observations. This approach works by feeding data into an input layer that goes through a series of transformations within hidden layers until a final transformation occurs and a final output is produced. Deep neural networks go a step further due to the larger number of nodes or layers that the data passes through in a complex, multi-step process of pattern and correlation recognition ????. Working with more layers provides more opportunities for the

data to train on the features abd patterns recognized in the previous layer, which in turn produces a more and more complex hierarchy know as a feature hierarchy ???.

Neural networks are popular in natural language classification tasks because of the ability for neural networks to create an embedded layer. Through vectorization, the words in the embedded layer become mathematical arrays. These vectors can be very useful in text classification. Deep neural networks, due to their extensive feature training, have gained traction with sentiment analysis in recent years due to the improvements with deep learning models and how they can handle increasing levels of complexity with very large data sets ??? . The feature generation aspect of the deep neural network process is also what makes this algorithm very flexible and adaptable to various datasets ???.

23.7 RUNNING A TWITTER SCRIPT IN PYTHON

As mentioned earlier, one of my objectives of this paper is to examine how a basic python scrip is setup and implemented using using data mined from Twitter. The scenario is to design a script that pulls tweets with the hashtags #cats and #dogs. Working with these hashtags, I wanted to generate a simple script that can be used to classify if a person is a cat or dog person based on their tweets. I wanted to evaluate the performance and accuracy of the two classification algorithms used, as well as evaluate any issues and challenges that arose during my experiment.

23.7.1 Python Libraries

Below is the full list of libraries that I imported:

```
import tweepy
import matplotlib.pyplot as plt
import matplotlib
!pip install tweepy
import sys
import jsonpickle
!pip install jsonpickle
import os, json
```

```

import pandas as pd
import numpy
import re
from wordcloud import WordCloud, STOPWORDS
import warnings
import sklearn
import sklearn.metrics
from sklearn import metrics
import sklearn.naive_bayes
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
import sklearn.svm
import sklearn.neighbors
import sklearn.neural_network
from sklearn.feature_extraction.text import CountVectorizer
import seaborn as sns

```

From the sklearn library, I created a function to gauge the precision scores from the two algorithms that will be used later in the script. The precision scores will help me assess whether a record was classified incorrectly, for example a record that is labeled a positive when it should have been labeled negative ???- scikit-learn-ps.

```

def print_score(Ytrue,Ypred):
    s = (sklearn.metrics.precision_score(Ytrue,Ypred),
         sklearn.metrics.recall_score(Ytrue,Ypred),
         sklearn.metrics.f1_score(Ytrue,Ypred))
    print('Precision: {:.3}\nRecall: {:.3}\nF-Score: {:.3}'.format(*s))

```

The next step in the code is authentication with the Twitter API. The concepts behind the authentication process were discussed earlier in the paper. If Twitter is unable to authenticate your account with the provided credentials, a “Can’t Authenticate” message is display and the script terminates.

```

"""twitter credentials. Removed for security reasons."""
Credentials file format:
consumer_key=YOUR_CONSUMER_KEY
consumer_secret=YOUR_CONSUMER_SECRET
access_token_key=YOUR_ACCESS_TOKEN
access_token_secret=YOUR_ACCESS_TOKEN_SECRET

"""Replace the API_KEY and API_SECRET with your application's key and secret."""
auth = tweepy.AppAuthHandler(consumer_key, consumer_secret)

api = tweepy.API(auth, wait_on_rate_limit=True,
                 wait_on_rate_limit_notify=True)

if (not api):
    print ("Can't Authenticate")
    sys.exit(-1)

```

Now that I'm authenticated, I can specify my search criteria for the specific tweets I want to identify and collect. A variable called 'SearchQuery' contains the #cats hashtag that I want to search on, the maxTweets variable contains the maximum number of tweets that I would like to collect, the 'tweetsperQry' is set to 100 as this is the maximum amount of tweets the Tweepy API is allow to gather during one iteration using the search function of the API. The 'fname' contains the name of the file in which the tweets will be collected.

```
"""Information about the type of tweets we want to find as well as how many"""
searchQuery = '#cat' # this is what we're searching for
maxTweets = 2000 # Some arbitrary large number
tweetsPerQry = 100 # this is the max the API permits
fName = 'cat_tweets.txt' # We'll store the tweets in a text file.
```

Since the Tweepy API only allows 100 tweets per iteration, the script contains a functions that runs in a loop until the maxTweets value has been reached. If any issues arise, the function will terminate.

```
SinceID = None
max_id = 1
tweetCount = 0
print("Downloading max {0} tweets".format(maxTweets))
with open(fName, 'w') as f:
    while tweetCount < maxTweets:
        try:
            if (max_id <= 0):
                if (not sinceId):
                    new_tweets = api.search(q=searchQuery, count=tweetsPerQry)
                else:
                    new_tweets = api.search(q=searchQuery, count=tweetsPerQry,
                                            since_id=sinceId)
            else:
                if (not sinceId):
                    new_tweets = api.search(q=searchQuery, count=tweetsPerQry,
                                            max_id=str(max_id - 1))
                else:
                    new_tweets = api.search(q=searchQuery, count=tweetsPerQry,
                                            max_id=str(max_id - 1),
                                            since_id=sinceId)
            if not new_tweets:
                print("No more tweets found")
                break
            for tweet in new_tweets:
                f.write(jsonpickle.encode(tweet._json, unpicklable=False) +
                       '\n')
            tweetCount += len(new_tweets)
            print("Downloaded {0} tweets".format(tweetCount))
            max_id = new_tweets[-1].id
        except tweepy.TweepError as e:
            # Just exit if any error
            print("some error : " + str(e))
```

```

        break

print ("Downloaded {0} tweets, Saved to {1}".format(tweetCount, fName))

```

The iteration loop has completed, and I've created two dataset that contains data for the hashtags of #cats and #dogs. For this paper, I found that working with twitter data that was contained in dataframes was easier to analyze. I used the pandas dataframe function to take the twitter data from the iteration script and load into two separate dataframes.

```

dogs = pd.read_json("C:/Users/Jay/PycharmProjects/ADS_Assignments/FinalExercise/dog_tweets.json")
cats = pd.read_json("C:/Users/Jay/PycharmProjects/ADS_Assignments/FinalExercise/cat_tweets.json")

```

A crucial step when working on a new set of data is to perform some type of data preprocessing and exploration. This gives you a good sense of that data contents, and if you need to perform some cleansing of the data such as removing null values and incorrect characters or words.

In this example, the first two lines provide a record count. Next, using the dogs dataset, I created a new column entitled tweety and extracted all the URLs and usernames from the collected tweets into that new column. I decided to remove all URLs, RT's (retweets), and usernames or twitter handles.

```

dogs['tweety'] = ''

#add tweety first part
for i in range(len(dogs['text'])):
    try:
        dogs['tweety'][i] = dogs['text'].str.split(' ')[i][0]
    except AttributeError:
        dogs['tweety'][i] = 'other'

#Preprocessing tweety. select tweety contains 'RT @'
for i in range(len(dogs['text'])):
    if dogs['tweety'].str.contains('@')[i] == False:
        dogs['tweety'][i] = 'other'

# remove URLs, RTs, and twitter handles
for i in range(len(dogs['text'])):
    dogs['text'][i] = " ".join([word for word in dogs['text'][i].split()
                               if 'http' not in word and '@' not in word and '<' not in word])
dogs['followers_count'][1]

```

I performed the aforementioned step again with the #cats dataset.

Within Natural Language Processing, there are textual components that do not provide any significance to the analysis. These are referred to as stopwords. Some examples of stopwords are is, the, and is. Stopwords should be removed during the processing step to eliminate unnecessary information from being included in the analysis **???**. The nltk python library has contains a corpus library of which contains a comprehensive list of stopwords.

```
import nltk
import nltk.corpus
nltk.download('stopwords')
stopwds = list(nltk.corpus.stopwords.words('english'))
## stopwds
```

23.7.2 Visualizations

Visualizing twitter data in python can be accomplished with the use of a variety of different python packages and tools. One of the more popular and useful libraries for creating plots, charts, and other visualizations is Matplotlib. The Matplotlib python library is a 2D graphical tool that creates publication-like visualizations within a variety of platforms including python, ipython, Jupyter notebooks, and other web applications **???**. “Matplotlib makes easy things easy and hard things possible” **???**. Matplotlib installation is straightforward and can be completed with the following code:

```
import matplotlib
import matplotlib.pyplot as plt
```

Matplotlib contains a tool called pyplot which allows for very simple plotting as well as creating bar charts, line charts, and histograms. The pyplot tool can be utilized to create line and bar charts to show the distribution of twitter users by source. Below is a bar graph depicting the number of followers by source for the #dogs twitter dataset:

- image for bar chart

From looking at the bar chart, one could surmise that the top source

for all of the tweets is the iPhone, followed by Twitter for Android, and then the Twitter Web Client application. This is no surprise given the ubiquitous nature of smart phones today.

The python code used to generate the bar chart is below:

```
dogs['source_new'] = ''  
  
for i in range(len(dogs['source'])):  
    m = re.search('>(.*)</a', dogs['source'][i])  
    try:  
        dogs['source_new'][i]=m.group(0)  
    except AttributeError:  
        dogs['source_new'][i]=dogs['source'][i]  
  
dogs['source_new'] = dogs['source_new'].str.replace(' ', ' ', case=False)  
  
dogs['source_new'].head()  
  
tweets_by_type = dogs.groupby(['source_new'])['favorite_count'].sum()  
plt.title('Number of followers by Source', bbox={'facecolor': '0.8', 'pad': 0})  
tweets_by_type.transpose().plot(kind='bar', figsize=(20, 10))
```

Another effective visualization for analyzing and visualizing twitter datasets are wordclouds. Wordclouds depict groupings of words in different sizes depending on how frequently they are displayed within tweets. The Wordcloud python library can be installed using the pip method and imported into the python script with the following commands:

```
python -m pip install wordcloud  
  
from wordcloud import WordCloud, STOPWORDS
```

Below is the python code to generate two different wordcloud visualizations; one depicting text and another showing language.

```
""" Dogs Text Wordcloud"""
def wordcloud(tweets,col):
    stopwords = set(stopwds)
    wordcloud = WordCloud(background_color="white",stopwords=stopwords,random_state = 2016)
    plt.figure( figsize=(20,10), facecolor='k')
    plt.imshow(wordcloud)
    plt.axis("off")
    plt.title("Dog DataSet")
wordcloud(dogs, 'text')

""" Dogs Language Wordcloud"""
def wordcloud(tweets,col):
```

```
stopwords = set(stopwds)
wordcloud = WordCloud(background_color="white", stopwords=stopwords, random_state = 2016).generate(
plt.figure( figsize=(20,10), facecolor='k')
plt.imshow(wordcloud)
plt.axis("off")
plt.title("Dog DataSet")
wordcloud(dogs, 'lang')
```

Below is a wordcloud chart illustrating the top words or phrases seen within the Twitter #dogs dataset.

- wordcloud text image - wordcloud language image

The word dog is displayed prominently as expected, and there are also some other interesting words such as love, pet, first displayed as well. There are some other words that were not expected such as millionaire, crypto, and trading binance. This could lead to further research as to how these terms are related to tweets containing the #dog hashtag.

23.7.3 Machine Learning Algorithms

The Naive Bayes and Neural Networks algorithms can be set up and executed veru seamlessly in Python using a variety of libraries. Users can import and implement the Naive Bayes and Neural Network algorithms from the sklearn library. Sklearn contains numerous machine learning algorithmic functions and toolkits, and is a manageable task as far as setup and implementation is concerned. After creating the train and test datasets in python with the desired twitter data, setting up and running the algorithm in Python is straightforward.

In order to effectively run the algorithms, there are a few model preparation steps needed to be completed. For the cat and dog datasets, the hashtags should be removed and replaced with a filler, non-significant value.

```
cats_txt = [x.replace('#cat', "BLAH") for x in cats['text']]
dogs_txt = [x.replace('#dog', "BLAH") for x in dogs['text']]
```

In addition to setting up the algorithm, it is advisable to add in a

vectorization component from the sklearn library to incorporate within the algorithm. The vectorization process is used to transform textual information into numerical information that machine learning algorithms such as Naive Bayes and Neural Network can understand and process more efficiently ??? . the stopwords component can also be pulled into play at this step to ensure these elements are excluded from this step. Below is the python code to accomplish this:

```
vectorizer = sklearn.feature_extraction.text.CountVectorizer(cats_txt+dogs_txt, analyzer='w')
vectorizer.fit(cats_txt+dogs_txt)
cat_tdm = vectorizer.transform(cats_txt).toarray()
dog_tdm = vectorizer.transform(dogs_txt).toarray()
vectorizer.get_feature_names()
```

The next step would entail combining the arrays into one matrix, and then assigning adding in values the class labels.

```
catdog_tdm = numpy.concatenate((cat_tdm, dog_tdm), axis=0)
Ycat = numpy.array([0 for i in range(len(cats_txt))])
Ydog = numpy.array([1 for i in range(len(dogs_txt))])
Y = numpy.concatenate((Ycat, Ydog), axis=0)
```

At this point, the data must be split into two distinct data sets; one for training and another to test the algorithm. The training dataset is used by the machine learning algorithm to learn about the data and its possible outcomes. The test data is used to test the algorithm after it has run through the training phase. Typically the training data set is larger in size. Below is the python code for splitting data into train and test datasets:

```
xtrain,Xtest,Ytrain,Ytest = sklearn.model_selection.train_test_split(catdog_tdm, Y, test_size=0.2)
```

In this case, the test data comprised 20% of the dataset while the train data is 80%.

As mentioned earlier, the Naive Bayes Multinomial algorithm was determined to be one of the more effective algorithms for tweet classification. The python code is below.

```
mnb = sklearn.naive_bayes.MultinomialNB()
mnb.fit(Xtrain, Ytrain)
Ypred = nb.predict(Xtest)
```

```

print("\nNaive Bayes Performance")
print_score(Ytest, Ypred)
metrics.accuracy_score(Ytest, Ypred)
metrics.confusion_matrix(Ytest, Ypred)
print(Ytest.value_counts())

```

The mnb.fit portion of the code above is fitting the data into the Naive Bayes Algorithm, and the nb.predict function is running the algorithm against the test data. There are some accuracy scores generated as well as a confusion matrix to help gauge whether or not the model was fitted properly.

The Neural Networks algorithm is another algorithm that has been linked to high performance and results when used with Twitter data. The python code is also straightforward in setting up and requires network hidden layers to be defined as well as the iterations. The python code is below:

```

nn = sklearn.neural_network.MLPClassifier(hidden_layer_sizes=(30, 30, 30), max_iter = 500)
nn.fit(Xtrain, Ytrain)
Ypred = nn.predict(Xtest)

print("\nNeural Network Performance")
print_score(Ytest, Ypred)
metrics.accuracy_score(Ytest, Ypred)
cm = confusion_matrix(Ytest, Ypred)

```

23.8 RESULTS

| Tables are cool | | |
|------------------------|-------|--------|
| Col 3 | right | \$1600 |

23.9 CONCLUSION

23.10 FUTURE RESEARCH

23.11 ACKNOWLEDGEMENTS

24 PREDICT EPL RESULTS USING TWEETS

FA18-523-62

FA18-523-69

Manek Bahl, Sohan Udupi Rai
mbahl@iu.edu, surai@iu.edu
Indiana University
hid: fa18-523-62, fa18-523-69
github: [cloudycode](#)
code: [cloudycode](#)

Learning Objectives

- Learn about extracting Tweets and storing them on MongoDB
 - Run Machine Learning and NLP algorithms on the data to predict soccer results
-

Keywords: Twitter API, MongoDB, Machine Learning, NLP

24.1 ABSTRACT

There are a lot of factors that influence the outcome of a football game. Team statistics such as recent form, win-loss ratio, goals scored, goals conceded etc have been proven to be useful in dictating the results of a game. These statistics can be directly obtained from the Official EPL website, however, there are certain other factors such as injuries, general mood of the fanbase and sentiment of the fanbase, preparation by the team etc. which aren't documented in the official website. Analysis of Twitter data could prove useful in extracting these undocumented features, using NLP models. In this paper, we analyze the usefulness of Tweets in predicting the outcomes of games in the English Premier League. We create three different models to predict the outcomes, the first using the statistics available from the EPL website, the second using only the twitter data and the third with combination of both. If the model improves with

the addition of features obtained from the twitter data, we would be able to prove that Twitter data does increase the predictive power of the model.

24.2 INTRODUCTION

The English Premier League is one of the most famous and competitive football leagues in the world. The season begins in August and ends in May in which 20 teams compete for the championship. Each team plays each of the other teams exactly twice, Home and Away, over the course of the season. A win awards the team with 3 points while a draw earns 1 point and a loss gives 0 points. At the end of the season, the team with the maximum points is crowned as the champion.

Twitter is a social media website where people express their opinions about a wide range of topics such as Politics, Sports, Religion, Entertainment etc. It was launched in 2006 and currently has 335 Million users worldwide. Due to the large number of users, Twitter has proven to be a reliable source of the general opinion of the public regarding any matter. When it comes to tweets regarding the Premier League, Twitter is being used not only by the public, but also by the players and the club officials. Therefore, we think that tweets could contain valuable information which could be used to improve the prediction models.

We are choosing the English Premier League due to its globally widespread fanbase which is active on Twitter. This provides us with a rich corpus of data regarding the team which could be used to analyze and interpret the overall fan sentiment for a team and other factors such as player unavailability, current coaching style etc. which are subjective and hence not obtainable from any other statistical sources.

For each team playing in the upcoming weekend fixtures, we are using Twitter streaming API in Python, to monitor and extract all tweets over the course of the entire week. This is repeated for 4-6

weeks in succession, to build the corpus of tweets. These tweets are fed to Natural Language Processing models to create features which serve as the predictors in the dataset. The corresponding results of the matches will serve as the class labels. The results are only needed for the first 4 weeks which serve as the training dataset. Supervised Machine Learning algorithms are then trained on the training dataset. The trained algorithms are then used to predict the unseen results of the last week of the dataset which serves as the test data.

24.3 RELATED WORK

24.4 IMPLEMENTATION

24.4.1 Data

The tweets were extracted using web scraping tool with Python called Selenium and were stored in MongoDB over the course of almost 3 months from September 2018 to November 2018. We were able to collect more than 100,000 tweets combined for 20 premier league teams using the team specific hashtags. Examples of hastags used are: #AFCB for Bournemouth, #Arsenalfc and #Gunners for Arsenal, #MANU and #RedDevils for Manchester United etc. In total we had 50 hashtags overall for different teams.

The statistical football data for this season was taken from -. The data contains all individual match results for this season with number of goals scored by each team, number of shots, number of shots on target, corners etc. The data also contained win-loss odd prediction for each time according to various betting websites. The results were given as Home team Win, Away Team Win or Draw.

This data however was match wise which isn't useful for our machine learning algorithm. So there was a lot of data manipulation required to get the data in the format we required. ——

24.4.2 MongoDB

Developed by MongoDB Inc., MongoDB is a NoSQL Database system which stores data as documents. The documents are stored in JSON format which means that various documents can have their own different format and data structure. The structure of the existing documents as well can be changed later in time. Another advantage that this structure provides is that, it makes mapping of different objects in the application code much easier. High availability and horizontal scaling are in built features of MongoDB due to it being a distributed database. It was written in C, C++ and Javascript and can be used on various operating systems such as Windows, iOS, Linux and Solaris.

However, owing to default security configuration, MongoDB has been rendered to a lot of data thefts as it allows full database access to all.
[Wikipedia]

24.4.2.1 Local Installation

We used the MongoDB Community Edition on Windows and Mac OS systems. The below steps were followed for Mac OS using Homebrew:

Updated the HomeBrew's version by the following shell command

```
brew update
```

Then use the below command to install the mongodb binary files

```
brew install mongodb
```

To run MongoDB the below command was used

```
<path to the binary files>mongod
```

Once MongoDB is up and running we should be able to see the following line in the shell or terminal window

```
[initandlisten] waiting for connections on port 27017
```

24.4.2.2 Interaction with Python

24.4.3 Tweet Extraction

24.4.3.1 Tweepy and its challenges

24.4.3.2 Selenium

24.4 Machine Learning Approaches

24.5 RESULTS

24.6 CONCLUSION

24.7 ACKNOWLEDGEMENT

24.8 WORK BREAKDOWN

25 ANALYZING BIG DATA SORTING ALGORITHMS

523-63

FA18-

Mark Miller
mgm3@indiana.edu
Indiana University
hid: fa18-523-63
github: [cloud](#)
code: [cloud](#)

fix format errors

comment: sorting is a very complex task, although the algorithms are relatively straight forward. For example the papers that you selected based on the basic algorithm do not actually show you the best sorting algorithms, which depend on lots of hardware related issues surrounding core amnagement, bandwidth, and page and cache sizes. All of which should be dynamically discoverable, but actually are not as its in the cloud. this it makes it hard to develop a good sorting algorithm.

whay are you not running python 3.7

Keywords: Sorting Algorithms, Python, Big Data Sorting, Computational Efficiency

25.1 ABSTRACT

There are many different sorting algorithms that vary vastly in terms of efficiency, memory usage (depending on the programming language), and time to completion. It is simply not effective to identify a specific data point and find it's place in comparison to the rest of the data. There are many methods in place to help computational efficiency when needing to get data sorted quickly and in a light-

weight fashion. The purpose of this project is to provide code for many commonly used sorting algorithms while analyzing their efficiency. This depends highly on the way that the data stored, the initialization of the data, and the computational power inherent to the machine which is performing the operations. We will restrict to identical machine specifics and reset memory usage after each algorithm to ensure balanced and stable results. Not all popular sorting algorithms are used in this project, but many are. A simple timing mechanism, inherent to the programming language in use will be used to time each operation that is performed.

25.2 INTRODUCTION

There are many mathematical principles that contribute to efficient computing. There is a common problem, as defined by Pointless Programming [97], which presents the following scenario:

"You have two light bulbs and 100 story building. You must determine the minimal floor such that if you drop the light bulb from that floor it breaks. Once you break a bulb, it can't be reused. Question: What's the smallest number of drops required in the WORST case to determine the minimal floor." [97]

Assuming that no damage is done to the light bulb after preliminary drops, one method to this approach is to start at the bottom floor and increment the floor until the light bulb breaks. This way, the answer is obtained with just one lightbulb but it takes a long time to climb the stairs and drop the bulb from each individual floor, especially if the lightbulb can resist 99 floors of drops.

Another method is to begin from the fiftieth floor. This way, half of the floors will be removed at once.

The mathematical answer benefits from the "square root law" [98]. Finding the square root of the length of the data, (ten, in this case), provides the segmentation required to specify that it can take no

more than 19 drops in order to determine the number of floors which can be endured for the light bulb.

Similar principles can be used in determining optimal methods for sorting big data algorithms. While not as simple as the problem described above, there are many algorithms that vary in efficiency that will be analyzed in this project. These algorithms include the following: Bubble sort [99], Merge sort [100], Insertion sort [101], Shell sort[99], Selection sort [101], Strand sort [102], Python's sorting algorithm, [103], and Heap sort [104].

Many companies that utilize large datasets, do this through various Structured Query Language (SQL) databases [105]. These can contain far more rows and columns than other, more user-friendly software, like Microsoft Excel, can, by millions of rows and columns [106] [107]. This project is related to big data because in most algorithms, time complexity and operational complexity is more than most computer resources are able to handle for the tasks that are given to them. Big data often needs to be sorted, which causes problems for many database administrators, programmers, data scientists, and more.

25.3 REQUIREMENTS

Due to the random number generation of a ten-million numbers, it is recommended that these steps be completed on a compute system with at least 16 GB of RAM. Python 3.6 or higher.

Here is a summary of the hardware and software requirements of this project:

- * 16 GB of RAM (Due to the generation of 10 million random numbers)
- * Intel(R) Core(TM) i5-4690K CPU @ 3.50GHz (This level of CPU is sufficient)
- * Python 3.6 [108] or higher (Performed using the Spyder IDE [109])
- * Package Inclusion:
 - * Python Random [110]
 - * Python Time [111]
 - * Python Standard Library [112]
 - * TBD

25.4 DESIGN

Contained in this section is a description of each of the sorting

algorithms that are used along with pseudocode of the algorithms.

25.4.1 Bubble Sort

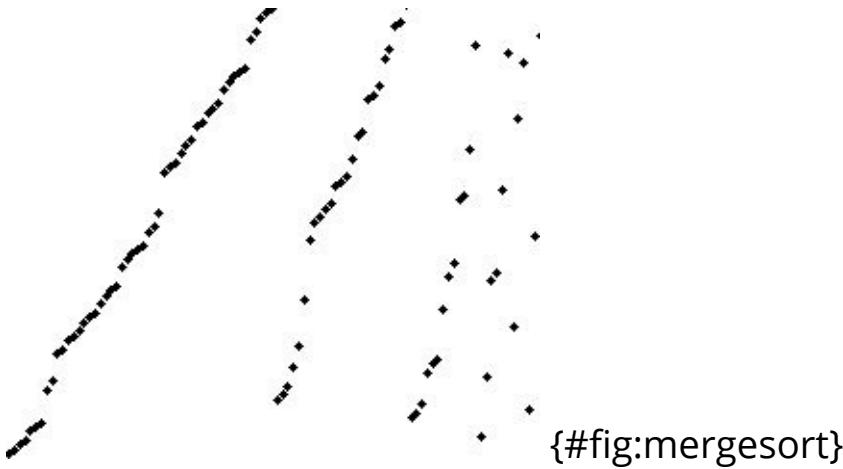
5  3 1 8 7 2 4 {#fig:bubblesort}

The Bubble sort algorithm is known to be one of the slower algorithms [99]. The basic process takes two adjacent data points and if they are in the wrong order it will simply change the order of the two pairs. This is computationally expensive and requires considerable time to compare each individual element to all other elements in the list. At worst case scenario, the amount of operations required are at the order of the number of elements in the list squared [114].

Pseudo-code [113]:

```
procedure bubbleSort( A : list of sortable items )
    n = length(A)
    repeat
        swapped = false
        for i = 1 to n-1 inclusive do
            /* if this pair is out of order */
            if A[i-1] > A[i] then
                /* swap them and remember something changed */
                swap( A[i-1], A[i] )
                swapped = true
            end if
        end for
    until not swapped
end procedure
```

25.4.2 Merge Sort



The Merge sort algorithm is a vast improvement on the bubble sort algorithm. It was originally developed by the popular mathematician and computer scientist John von Neumann [116]. This method segments the data intelligently and sorts in the segmented groups, reducing the computational requirements dramatically [100].

Pseudo-code [115]:

```

function merge_sort(list m)
    // Base case. A list of zero or one elements is sorted, by definition.
    if length of m ≤ 1 then
        return m

    // Recursive case. First, divide the list into equal-sized sublists
    // consisting of the first half and second half of the list.
    // This assumes lists start at index 0.
    var left := empty list
    var right := empty list
    for each x with index i in m do
        if i < (length of m)/2 then
            add x to left
        else
            add x to right

    // Recursively sort both sublists.
    left := merge_sort(left)
    right := merge_sort(right)

    // Then merge the now-sorted sublists.
    return merge(left, right)

```

In this example, the merge function merges the left and right sublists.

```

function merge(left, right)
    var result := empty list

    while left is not empty and right is not empty do
        if first(left) ≤ first(right) then
            append first(left) to result
            left := rest(left)
        else

```

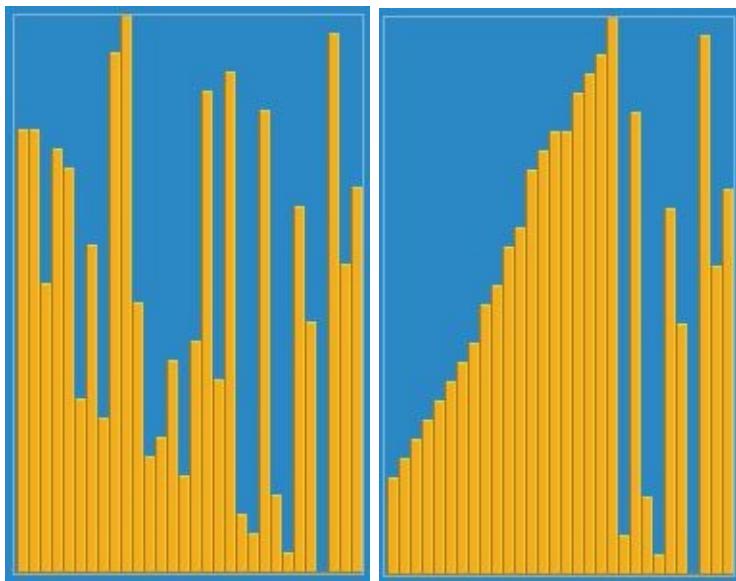
```

        append first(right) to result
        right := rest(right)

// Either left or right may have elements left; consume them.
// (Only one of the following loops will actually be entered.)
while left is not empty do
    append first(left) to result
    left := rest(left)
while right is not empty do
    append first(right) to result
    right := rest(right)
return result

```

25.4.3 Insertion Sort



The insertion sort, while far more efficient than the Bubble sort, is inefficient for large lists, because it builds the final sorted list one element at a time [101]. The Insertion sort remains popular for its simple implementation, efficiency for small data sets, its light-weight use of memory, and its stability [117].

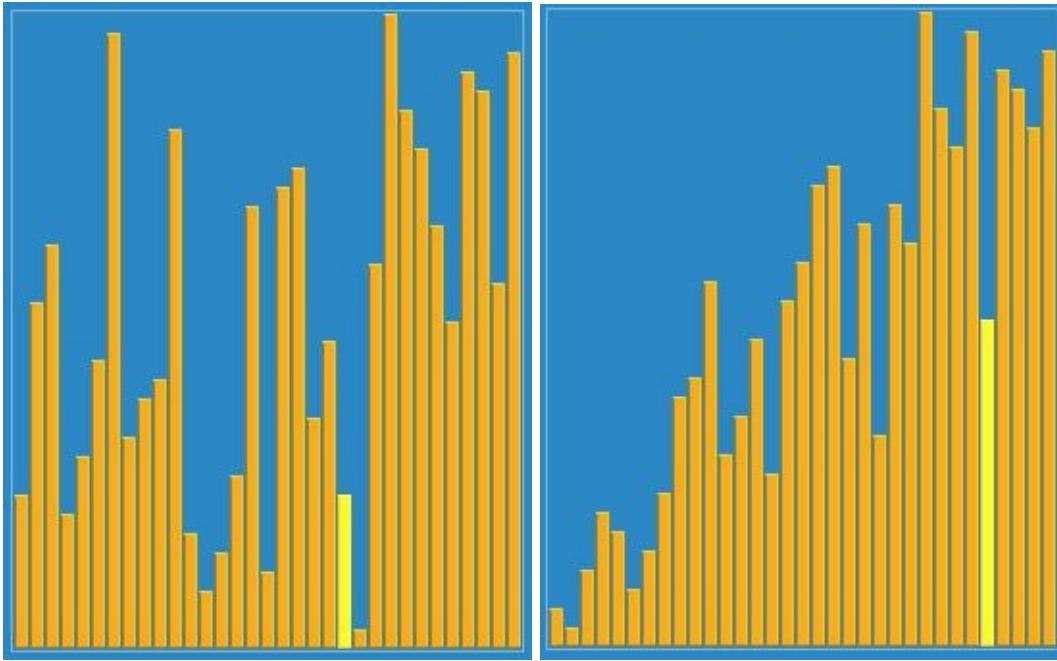
Pseudo-code [118]:

```

i ← 1
while i < length(A)
    j ← i
    while j > 0 and A[j-1] > A[j]
        swap A[j] and A[j-1]
        j ← j - 1
    end while
    i ← i + 1
end while

```

25.4.4 Shell Sort



The Shell sort is essentially a combination of the Bubble sort and the Insertion sort, in that it uses both exchanges and insertions [117]. It uses these two different methods to provide lighter operations and order of operations. It begins with large values and proceeds to smaller values in order to result in a fully sorted list, after the proper exchanges and computations [99].

Pseudo-code [119]:

```
# Sort an array a[0...n-1].
gaps = [701, 301, 132, 57, 23, 10, 4, 1]

# Start with the largest gap and work down to a gap of 1
foreach (gap in gaps)
{
    # Do a gapped insertion sort for this gap size.
    # The first gap elements a[0..gap-1] are already in gapped order
    # keep adding one more element until the entire array is gap sorted
    for (i = gap; i < n; i += 1)
    {
        # add a[i] to the elements that have been gap sorted
        # save a[i] in temp and make a hole at position i
        temp = a[i]
        # shift earlier gap-sorted elements up until the correct location for a[
        i] is found
        for (j = i; j >= gap and a[j - gap] > temp; j -= gap)
        {
            a[j] = a[j - gap]
```

```

    }
    # put temp (the original a[i]) in its correct location
    a[j] = temp
}
}

```

25.4.5 Selection Sort

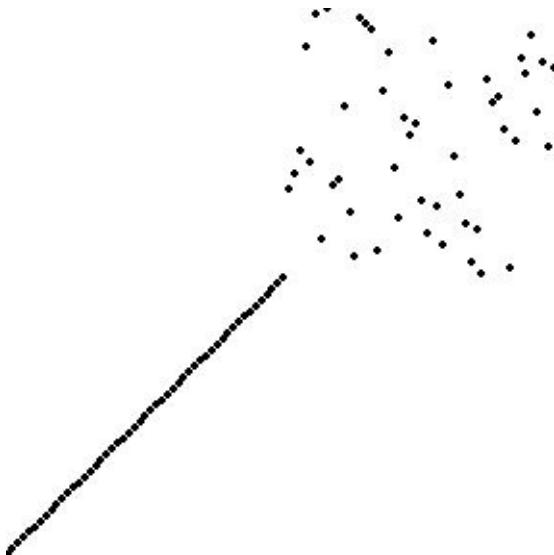


Figure 119: Selection Sort[120]

The Selection sort is not efficient for large data sets because of its time complexity and requirements on computational resources [121]. In this algorithm, two different lists are generated, a list of points that are already sorted and a list of points that are yet to be sorted. The algorithm will search through the entire list of unsorted points and find the smallest in that list and append it to the list that is sorted, as it will be larger than any other element in that list [101].

Pseudocode [120]:

```

bingo(array A)
{ This procedure sorts in ascending order. }
begin
    max := length(A)-1;

    { The first iteration is written to look very similar to the subsequent
      ones, but
      without swaps. }
    nextValue := A[max];
    for i := max - 1 downto 0 do
        if A[i] > nextValue then
            nextValue := A[i];

```

```

while (max > 0) and (A[max] = nextValue) do
    max := max - 1;

while max > 0 do begin
    value := nextValue;
    nextValue := A[max];
    for i := max - 1 downto 0 do
        if A[i] = value then begin
            swap(A[i], A[max]);
            max := max - 1;
        end else if A[i] > nextValue then
            nextValue := A[i];
    while (max > 0) and (A[max] = nextValue) do
        max := max - 1;
    end;
end;

```

25.4.6 Strand Sort

The Strand sort algorithm is a method that uses logic to determine if sublists have elements greater than other lists and slowly appends greater elements to other lists as the algorithm iterates through datasets [122].

Pseudocode [123]:

```

procedure strandSort( A : list of sortable items ) defined as:
    while length( A ) > 0
        clear sublist
        sublist[ 0 ] := A[ 0 ]
        remove A[ 0 ]
        for each i in 0 to length( A ) do:
            if A[ i ] > sublist[ last ] then
                append A[ i ] to sublist
                remove A[ i ]
            end if
        end for
        merge sublist into results
    end while
    return results
end procedure

```

25.4.7 Python Sort (Timsort)

The Timsort algorithm is the way that Python sorts lists. It is an extremely efficient algorithm, utilizing the benefits of many other algorithms. A min function is utilized heavily to expedite the processes [124]. It is difficult to identify weaknesses in this algorithm but if there are any it is that it can struggle with memory usage, which tends to be less than other algorithms still, and stability. It utilizes

adaptive sorting to find optimal sizes for each step of the process.
size sorting,

Pseudocode is intentionally not provided here.

25.4.8 Heap Sort

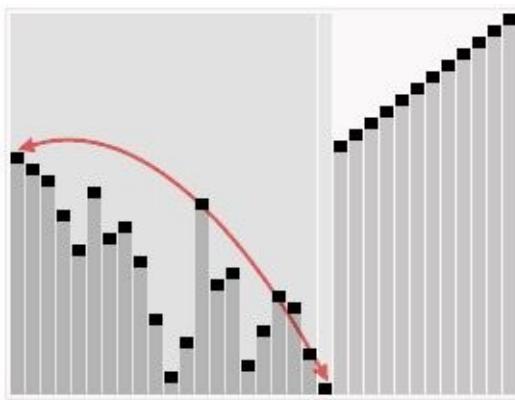


Figure 120: Heap Sort[125]

Heap sort is a method that compares some values to other values by dividing the list of numbers into an unsorted and sorted list (like the Selection sort) with the goal of shrinking the unsorted list based on comparison of a value to the list that is already sorted. It can have high time-complexity, and struggles with larger data sets, do to the number of comparisons that are required at each step of the algorithm.

Pseudocode [www-fa18-523-63-heap-wiki]:

25.4.8.1 Main Routine:

```
procedure heapsort(a, count) is
    input: an unordered array a of length count

    (Build the heap in array a so that largest value is at the root)
    heapify(a, count)

    (The following loop maintains the invariants that a[0:end] is a heap and
     every element
     beyond end is greater than everything before it (so a[end:count] is in
     sorted order))
    end ← count - 1
    while end > 0 do
        (a[0] is the root and largest value. The swap moves it in front of the
```

```

sorted elements.)
swap(a[end], a[0])
(the heap size is reduced by one)
end ← end - 1
(the swap ruined the heap property, so restore it)
siftDown(a, 0, end)

```

25.4.8.2 Sub-routines:

```

(Put elements of 'a' in heap order, in-place)
procedure heapify(a, count) is
    (start is assigned the index in 'a' of the last parent node)
    (the last element in a 0-based array is at index count-1; find the parent
     of that element)
    start ← iParent(count-1)

    while start ≥ 0 do
        (sift down the node at index 'start' to the proper place such that all
         nodes below
         the start index are in heap order)
        siftDown(a, start, count - 1)
        (go to the next parent node)
        start ← start - 1
    (after sifting down the root all nodes/elements are in heap order)

(Repair the heap whose root element is at index 'start', assuming the heaps
rooted at its children are valid)
procedure siftDown(a, start, end) is
    root ← start

    while iLeftChild(root) ≤ end do      (While the root has at least one child)
        child ← iLeftChild(root)      (Left child of root)
        swap ← root                  (Keeps track of child to swap with)

        if a[swap] < a[child]
            swap ← child
        (If there is a right child and that child is greater)
        if child+1 ≤ end and a[swap] < a[child+1]
            swap ← child + 1
        if swap = root
            (The root holds the largest element. Since we assume the heaps
             rooted at the
             children are valid, this means that we are done.)
            return
        else
            swap(a[root], a[swap])
            root ← swap                (repeat to continue sifting down the child
                                         now)

```

25.5 ARCHITECTURE

The architecture for this project is a `main.py` file which is the master python file, calling the other functions from other files, stored in the same directory. The other files will each be a Python 3.6 [108] implementation, generally performed in a rather procedural

programming methodology. The following files are called from main.py in the working directory: * bubble.py * merge.py * insertion.py * shell.py * selection.py * squareroot.py * strand.py * pysort.py * heap.py

In the call of the each of the above files, in main.py there is a timing mechanism, using Python's Time library [111].

25.6 DATASET

The only dataset required for this project is a list of 100 randomly generated numbers, using the Python random library [110]. Using this library, ten-million Gaussian Random Numbers [126] will be generated, using the seed of “one”. This data will only be stored for the duration of the program’s run-time and will not be committed to disk, unless memory is usurped , in which case, temporary disk usage will occur.

25.7 IMPLEMENTATION

The implementation is performed using the Spyder terminal and running the Python 3.6 script via Spyder’s command line interface (CLI). Provided that all of the files are stored in a single directory, the files as listed in the code folder for this project should run flawlessly, with the correct Python version and with all of the hardware and software requirements met. I purposely kept the design very simple, for ease of implementation, and ease of programming.

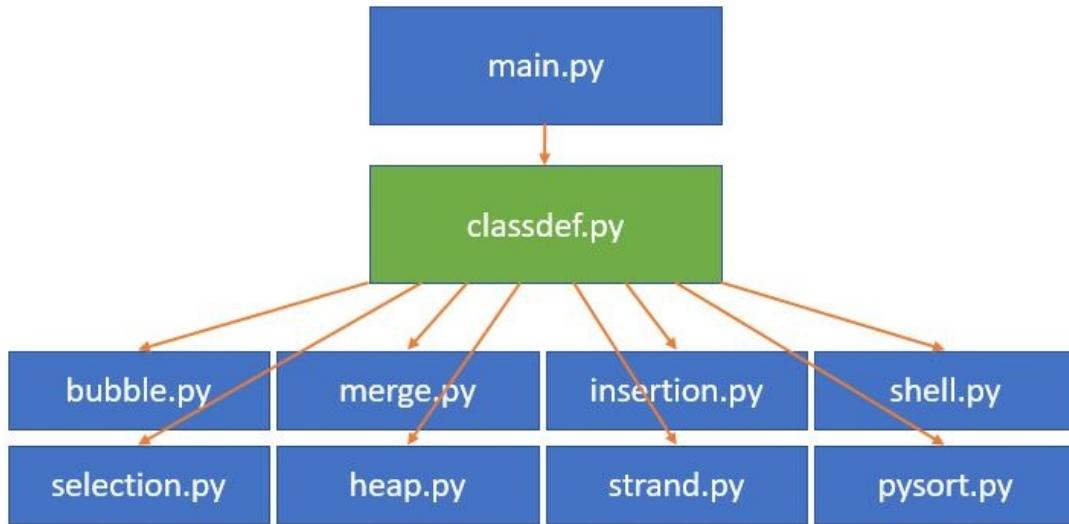


Figure 121: Design Doc

25.8 BENCHMARK

Many different sizes of lists of random numbers were tested. In some cases, smaller lists benefitted from specific algorithms, while others were nearly inoperational for larger data sets. The times for different sizes of lists of random numbers are listed here:

Here is the list of the results, along with some visualizations to help interpret the information that is being presented

```

Python 100 time: 1.114347469410859e-05
Bubble 100 time: 0.0005601062985078897
Select 100 time: 0.0003020468502654694
Merge 100 time: 0.00021025979731348343
Shell 100 time: 3.313718116260134e-05
Heap 100 time: 0.000275947659247322
Strand 100 time: 3.841566649498418e-05

Python 1,000 time: 0.0001316689667874016
Bubble 1,000 time: 0.07056049045422697
Select 1,000 time: 0.029865982105548028
Merge 1,000 time: 0.0027034659433411434
Shell 1,000 time: 0.0002888506314775441
Heap 1,000 time: 0.0045884728933742736
Strand 1,000 time: 0.00035717773062060587

Python 10,000 time: 0.001863013250840595
Bubble 10,000 time: 7.700457498340256
Select 10,000 time: 3.177057224973396
Merge 10,000 time: 0.036345033688121475
Shell 10,000 time: 0.003031025487871375
Heap 10,000 time: 0.061194692068966106

```

```

Strand 10,000 time: 0.01295399768059724
Python 100,000 time: 0.026999469511792995
Bubble 100,000 time: 925.2644678242395
Select 100,000 time: 528.98660055565309
Merge 100,000 time: 0.5586767063505249
Shell 100,000 time: 0.0392740083989338
Heap 100,000 time: 0.8638226169750851
Strand 100,000 time: 1.5463215238887642

```

From this, it is simply determined that for all amounts of randomly generated numerical data, the Selection and Bubble methods are extremely inefficient. These two methods are the bottom scorers for each of the datasets, which means that they do not out perform the other algorithms for either small or large data, at least with these implementations, in this environment.

All of these tests were performed on a single core, to ensure accurate timing for each of the individual algorithms. The memory buffers were never filling, and the core of the CPU which was processing this information was pegged at 100-percent.

To make the visualizations more aesthetic I have excluded the two slowest algorithms (Bubble and Select).

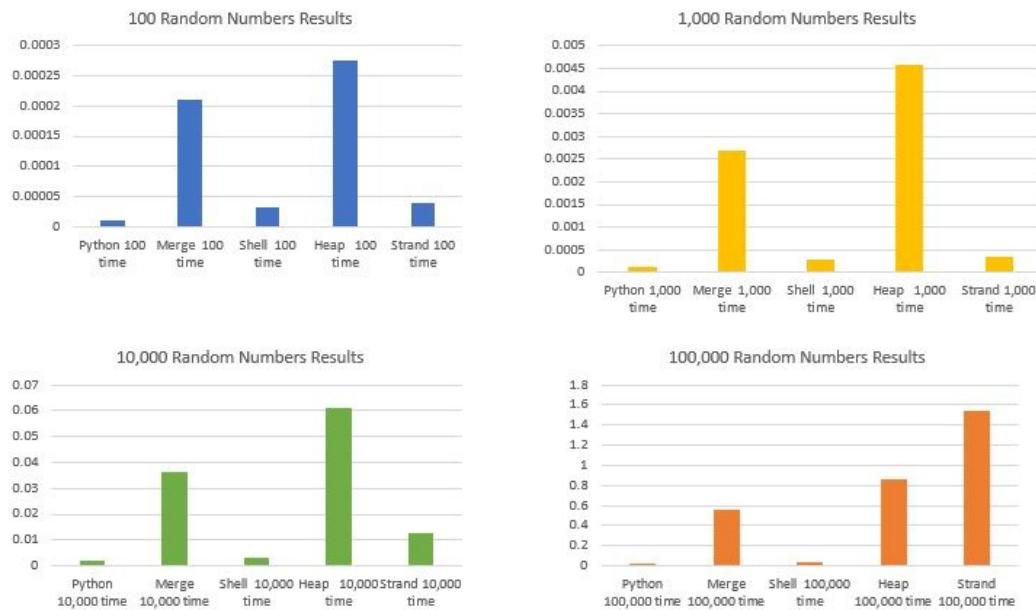
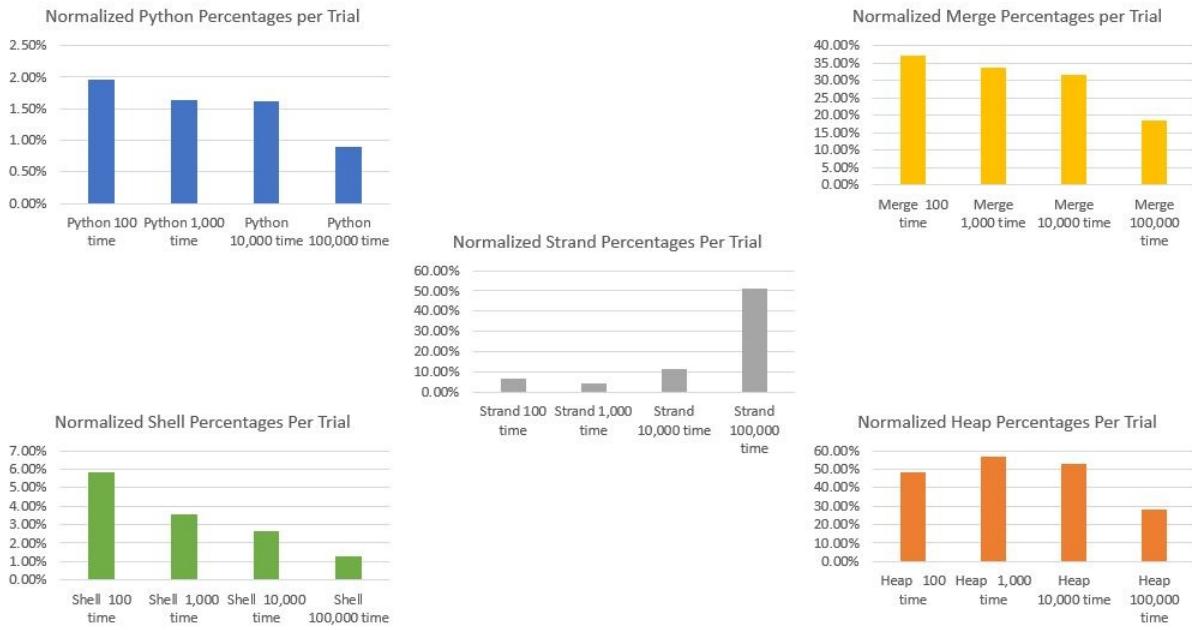


Figure 122: Chart By Size

As seen in the figure above, Among the fize fastest algorithms that

were tested, Python's Timsort algorithm out-performs each other algorithm. The Merge sort and Heap sort algorithms are outperformed until that data gets larger, during which the Strand sort algorithm loses its efficiency that it had on the smaller data sets.



{#fig:chartbymethod}

In this set of graphs, a downward trend of the data indicates that the algorithm outperforms the other algorithms as the datasets get larger. As the data sets get larger, Python's Timsort algorithm, and the Shell sort algorithm decrease its time taken, relative to the other algorithms. The Strand sort algorithm appears to perform well for intermediate data sets but as the data gets larger, on either end, it loses its efficiency, at least as it is implemented in this project.

The altogether run-time for this program, on my computer is approximately twenty minutes.

25.9 CONCLUSION

As data gets larger, certain sorting algorithms, such as the Strand sort algorithm, become nearly unfunctional, due to time and operational complexity of the algorithm. In this case, the old saying of "the

simpler, the better" does not seem to apply. Python's inherent sorting algorithm, Timsort, outperforms all of the other algorithms that were tested. The Shell sort algorithm is a close second. The Bubble and Select algorithms are very computationally inefficient and should not be implemented unless there are very specific circumstances, which are not addressed in this project. The Select and Bubble sort algorithms tend to be what humans use, but they are not computationally sufficient for computer resources to use to sort big data.

One metric that could be used to determine the effectiveness of an algorithm could be to see how that algorithm performs as the datasets get larger. This shows that it can handle the growing needs of big data better than other algorithms, like Python's Timsort and the Shell sorting algorithms showed.

Will include graphs, and all results, barring completion of code.

25.10 ACKNOWLEDGEMENTS

Plenty of research, all which is cited here in references.bib, was used in this analysis of different sorting algorithms. Credit goes to each author of all articles and web pages listed for their brilliance and efforts in furthering the world of data science, computer science, and software development.

###Code Acknowledgements: * [122] * [127] * [128] * [129] * [130] * [131]

26 DATA ANALYSIS OF YELP REVIEWS

FA18-523-65, FA18-523-67

Prajakta Patil, Sahithya Sridhar
patilpr@iu.edu, sahsrid@iu.edu
Indiana University, Bloomington
hid: fa18-523-65, fa18-523-67
github: [github](#)

please fix format, figure labels must not have spaces, repeatedly mentioned.

Keywords: Yelp, Natural language processing (NLP), Sentiment analysis, Scikit-learn

26.1 ABSTRACT

Yelp is a search service with website that hosts reviews for businesses, mainly restaurants, that are useful to people looking for one in a city. Our project performs analysis on text reviews to find how well they correlate with star ratings given by users to restaurants. It is found that star rating of 1 and 5 are well correlated with textual reviews. Star rating of 2, 3 and 4 are not so well correlated with textual reviews. This analysis can help users make better decisions by suggesting them to either just look at star ratings or go through actual text reviews [132].

26.2 INTRODUCTION

Users visit Yelp website to either look for a restaurant or write a review for one after their visit. Yelp had 148 million reviews on its website at the end of 2017. Reviews are in both text and star rating format. Users also post pictures of food and restaurant. This helps

users looking for restaurants make decisions. This is also useful to businesses to improve their services. When a restaurant is very popular, it obviously has more reviews and high star rating. In such case it is obvious for someone to choose that restaurant if it fits their choice of cuisine, budget and other parameters they might have in mind. However, when there are multiple restaurants that are similar; it becomes little difficult to choose one. We took a dataset of Yelp reviews and tried to find correlations between text reviews and star ratings. We perform additional analysis to show additional patterns in data. Technologies used to perform analysis is Natural Language Processing and Sentiment Analysis [132].

26.3 LITERATURE REVIEW

Li et al [133] performed text analysis on Yelp reviews to study if they can be categorized as Useful. This analysis was needed for businesses that don't have many reviews and hence no reviews categorized as either useful /funny/cool. Researchers used SVM and Random Forest models to perform this analysis. They extracted features such as number of words, sentences, average sentence length,TFIDF, star rating, sentiment of review as positive/negative, number of votes received by a review, how long has the user been providing Yelp reviews etc. Using SVM, researchers were able to achieve accuracy of 0.67 and 0.69 in case of Random Forest model for predicting if review is Useful [133].

Hajas et al [134] tried to understand how external factors such as changing taste result in cyclic behavior for user reviews for restaurants. They took Yelp reviews from 11 college campuses across USA and modelled restaurant rating as function of restaurant quality using a second-degree liner differential function. They were able to show how restaurant quality results in cyclic behavior for ratings. This means that restaurants that have good quality, may drop their investment in keeping quality constant which results in bad reviews. This in turn forces them to invest back in quality to gain back lost business [134].

"We provide heat maps of where most reviewed restaurants are located. The aim of this is to show that most reviewed restaurants are usually clustered together. This is an intuitive result, given that consumers have the option of choosing another (equally good) restaurant if the originally chosen restaurant is crowded [134]".

Koven et al [135] focused on methods to predict useful reviews. Their motivation was that there is lot of work done in trying to find bad/fake reviews. But, it was useful to find genuine and useful reviews. They created various attributes for Yelp reviews such as reviewer's average star rating, relationship between reviewers, topic and personality analysis, geographic distribution of reviews etc. They used J48 algorithm and reached accuracy of 79.8 in predicting if a review is useful or not. They were also able to predict bad reviews with roughly 5% false positives [135].

26.4 DATASET

Dataset used for analysis is taken from Kaggle website. Data contains 5.2 million user reviews for 174k businesses from 11 metropolitan areas. Data is in CSV and JSON format. It has following features business_id, name, neighborhood, address, city, state, postal_code, latitude, longitude, stars, review_count, attributes, categories about businesses across 11 metropolitan areas in four countries. It was originally put together for the Yelp Dataset Challenge to perform analysis on Yelp's data [132].

26.5 DATA PROCESSING

Data is processed to get it ready for analysis. Using NLTK library we removed stop-words. We also removed punctuations. All the words are converted to lowercase. We also removed unnecessary symbols and spaces. We didn't remove numbers as it might help users with prices which is an important factor while choosing a restaurant. All

these operations helped reduce size of data by getting rid of unnecessary data while retaining useful information [132].

26.6 ANALYSIS METHODS

Following methods are used for analysis to make observations and conclusions from data:

- Linear Regression is performed to find if there is any correlation between reviews and rating (stars). Reviews were characterized into polarity. This is done using python library called TextBlob. Polarity value is between -1 to 1. Polarity above 0 means text emotion is positive while, polarity below 0 means text emotions are negative. 0 polarity means neutral emotions. We also calculated another parameter called subjectivity. Subjectivity reflects user's personal emotions, opinions. Subjectivity is between 0 to 1. Subjectivity of 0 means fact-based opinion and 1 means subjective opinion. We also used this method to find if there are correlations between reviews and classification of review i.e. whether it is categorized as useful, cool or funny. As data we worked with was large, we used sampling method to use only 10% of all data for this analysis.
- Logistic Regression, Multinomial Naïve Bayes and Random Forest algorithm These are used to predict rating based on text review. This will help understand and predict how well text review and ratings match. Low accuracy means there is more to ratings than just text review. High accuracy means ratings accurately capture sentiments in text review.
- Count, Term Frequency Inverse Document Frequency (tfidf) vectorizer, and Linear SVC. These help with understanding which words are most commonly used in reviews for a restaurant and help gain insight on why certain place might be famous or infamous for.

26.7 RESULTS

Linear regression between polarity and star ratings showed correlation coefficient is 0.61. This means review emotions and star ratings are moderately correlated. This is shown in Figure 123. We can see regression line graphically represents this correlation between review text and star rating given by user. One important result is that we got p value less than 0.001. Meaning, the results are highly significant and that they are very unlikely to have occurred by chance. Review classification such as useful, cool and funny has negative correlation with review emotions. This is likely because people might have used ironic language while providing review for a business. All results for linear regression analysis are shown in Table. 1.

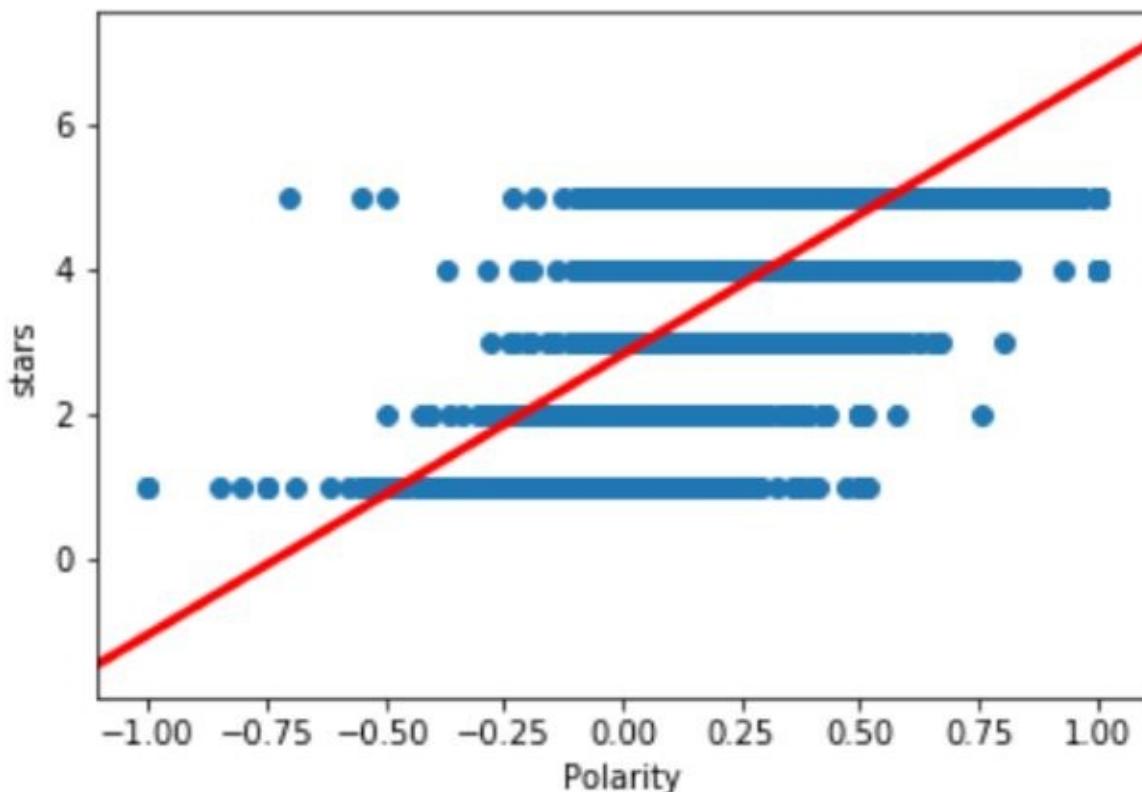


Figure 123: Fig.1

Table. 1 Correlation between review sentiment and star rating and review class

| Polarity (Review Sentiment) | Correlation coefficient-r value | p value |
|-----------------------------|---------------------------------|---------|
|-----------------------------|---------------------------------|---------|

| | | |
|-------------|--------|------------|
| Star Rating | 0.61 | < 0.001 |
| Useful | -0.064 | < 0.001 |
| Funny | -0.043 | 3.97 |
| Cool | -0.011 | 3.66 |

We used 3 different machine learning algorithms i.e. Logistic Regression, Multinomial Naïve Bayes and Random Forest, to check for accuracy in classifying review into 5 star-ratings from 1 to 5. Count vectorizer was used to encode data features in a matrix which these algorithms can learn in scikit-learn. We trained all 3 models on data set and checked their performance for guessing rank on test data. Highest overall precision of 0.55 was achieved with Logistic Regression. All 3 models had higher precision in predicting 1 and 5 star ratings and lowest precision in predicting star rating 3 in general.

Table. 2 Logistic Regression

| Star Rating | Precision | Recall | F1-Score |
|-------------|-----------|--------|----------|
| 1 | 0.78 | 0.72 | 0.75 |
| 3 | 0.60 | 0.43 | 0.50 |
| 5 | 0.87 | 0.94 | 0.90 |
| Average | 0.55 | 0.82 | 0.81 |

Table. 3 Multinomial Naïve Bayes

| Star Rating | Precision | Recall | F1-Score |
|-------------|-----------|--------|----------|
| 1 | 0.71 | 0.74 | 0.73 |
| 3 | 0.56 | 0.34 | 0.42 |
| 5 | 0.85 | 0.93 | 0.89 |

| | | | |
|---------|------|------|------|
| Average | 0.78 | 0.80 | 0.78 |
|---------|------|------|------|

Table. 4 Random Forest

| Star Rating | Precision | Recall | F1-Score |
|-------------|-----------|--------|----------|
| 1 | 0.70 | 0.60 | 0.65 |
| 3 | 0.46 | 0.16 | 0.23 |
| 5 | 0.78 | 0.94 | 0.86 |
| Average | 0.72 | 0.75 | 0.72 |

We created confusion matrix to understand these correlations in simpler way. We grouped our data by star rating. Confusion matrix thus created is shown in form of heatmap in Figure 124. We can see that useful and funny is correlated with text length. cool and subjectivity are correlated to polarity.

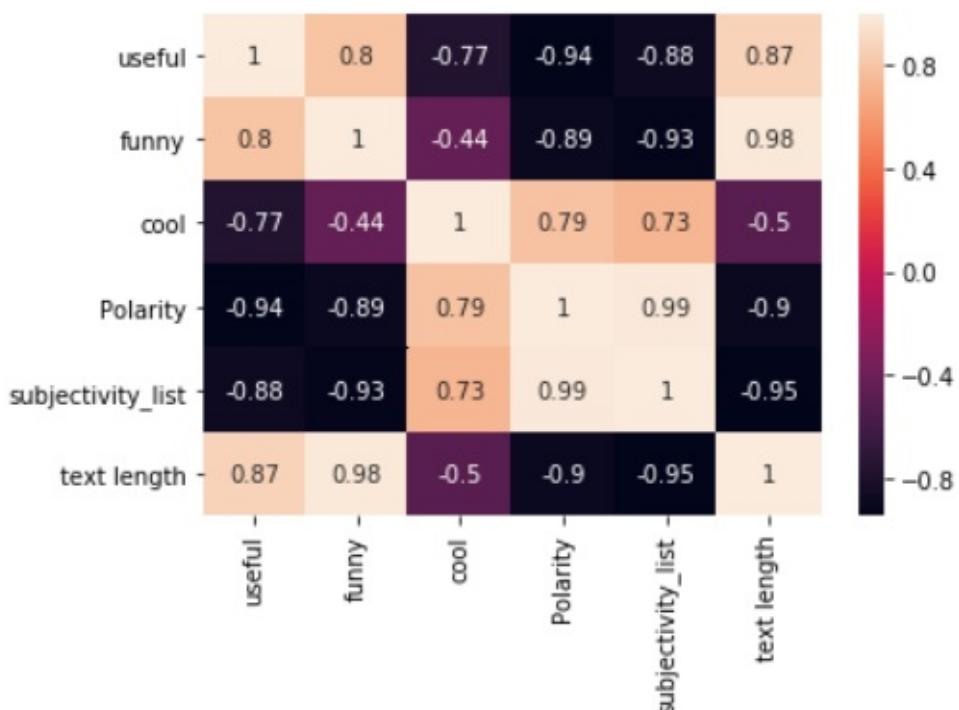


Figure 124: Fig.2

We wanted to understand patterns from this data. First, we looked at

reviews by states. Shown in Figure 125, we see Arizona is the state with most reviews for businesses followed by Nevada and California. We then looked at number of reviews by cities. As seen in +???- user-reviews, Las Vegas is the city with most number of reviews with 26775 reviews followed by Phoenix and Toronto. Figure 126 shows only top 20 cities but, our dataset has reviews for 11 metropolitan areas and had reviews for 1093 cities. We also looked at what type of businesses are present on Yelp. In Figure 127 we can see that various types of restaurants receive most reviews followed by shopping and home services. Another interesting observation we wanted to make was which weekday was preferred by users to go out and hence provide reviews for these businesses. Figure 128 shows that most users liked to go out on Saturday followed by Sunday and Friday. Monday and Tuesday saw lowest number of checkins for businesses.

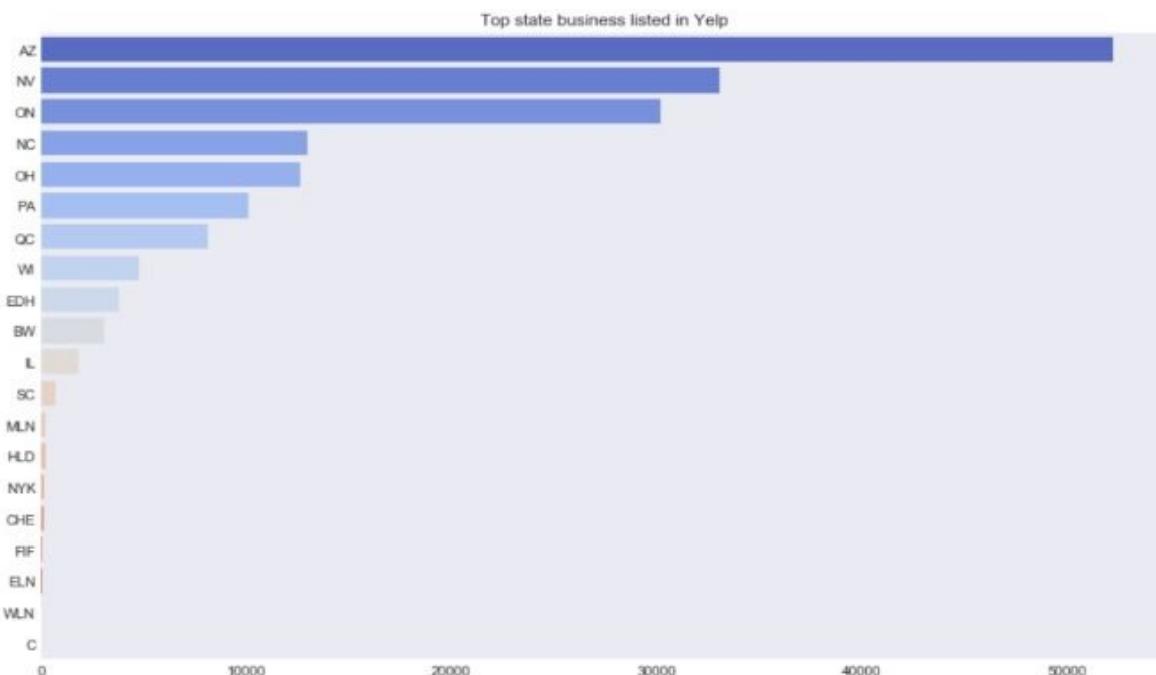


Figure 125: Fig.3

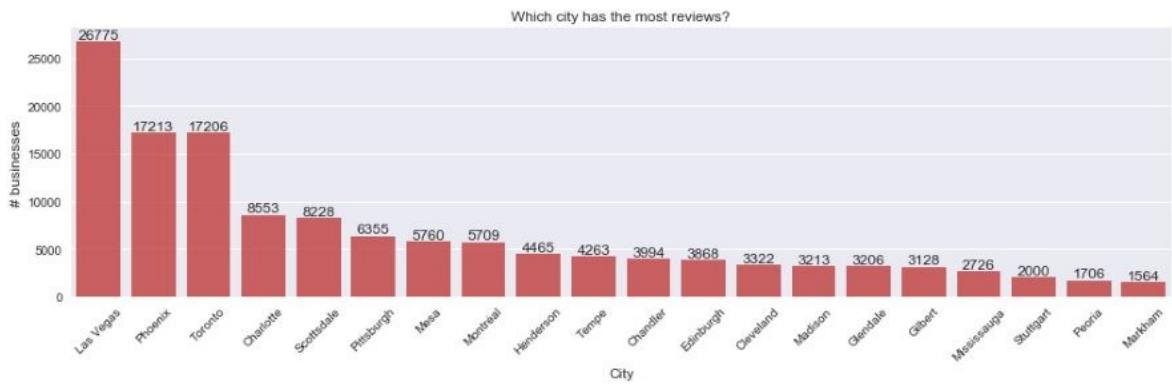


Figure 126: Fig.4

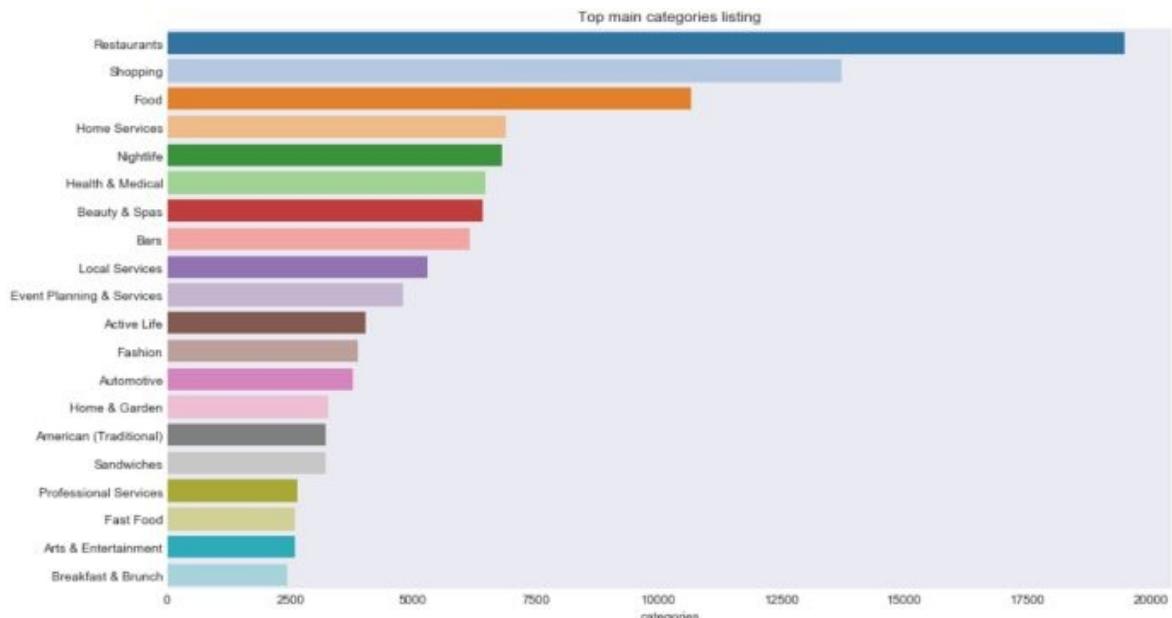


Figure 127: Fig.5

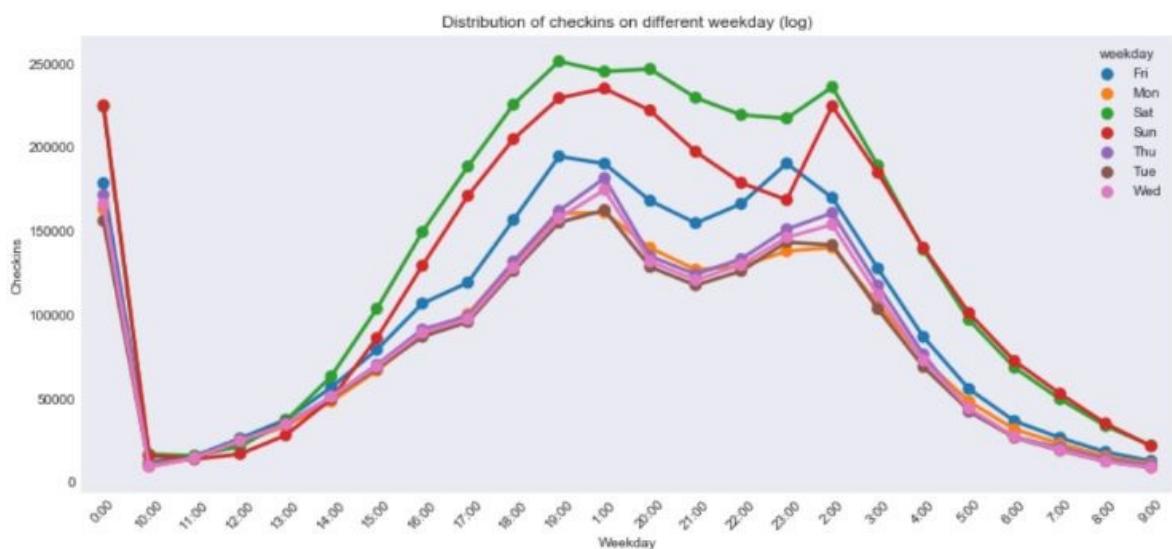


Figure 128: Fig.6

We used pandas groupby function to calculate mean star ratings for businesses. For purpose of this project we decided to look at details of restaurants only. Figure [129](#) shows top rated restaurants with descending average descending star rating. We are showing top 20 businesses only as data has more than 1000 places. Earl of Sandwich was top rated restaurant with average rating of 4.25. Top 10 places are dominated by restaurants that serve pizza and burger. One other thing to notice is that most of these restaurants are franchises. However, regional franchises have higher star ratings than ones that have pan US presence.

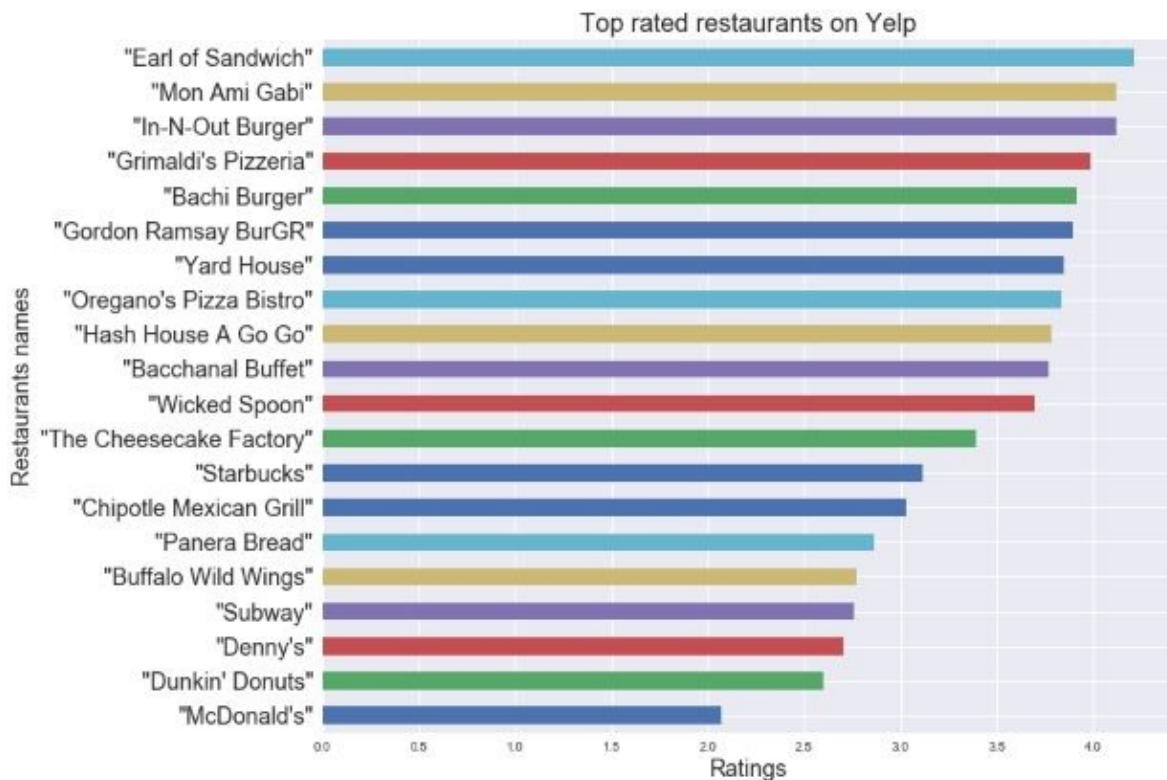


Figure 129: Fig.7

+[???](#) shows distribution of ratings offered to businesses. We see that rating 5 was most used rating by reviewers for describing their experience at given restaurant. Rating 2 was least used rating. Looking at distribution of review length for each star rating shows there was no significant difference in review length distribution. However, comparing Figure [131](#) with Figure [130](#) shows same trend.

Meaning more restaurants got 5-star rating but they also got some textual review. We also looked average number of reviews given by users. We can see in +???- by-users that most users provided less than 5 reviews. There is small percentage of users who have provided more than 30 reviews.



Figure 130: Fig.8

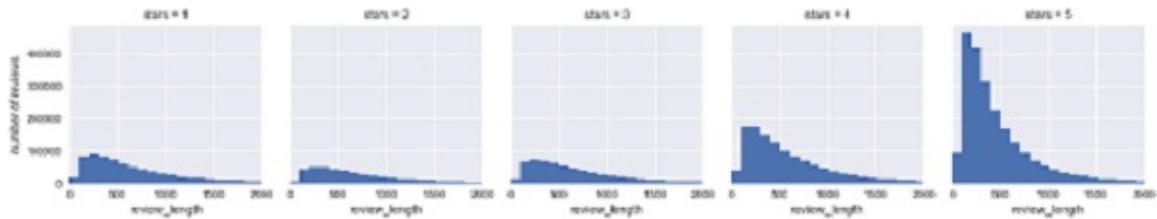


Figure 131: Fig.9

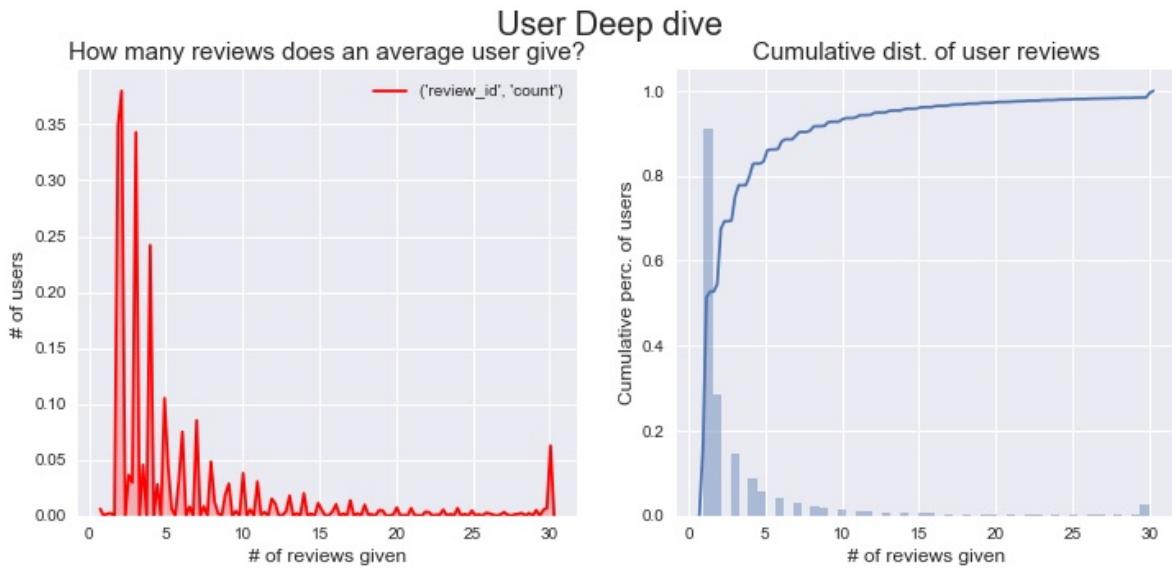


Figure 132: Fig.10

We were interested in seeing usage of words to describe user experience about a certain place. We chose 11 words that we thought could have been most used in reviews. This list covered words that help express both positive and negative emotions. Figure 133 shows that word great was used a lot by users to describe positive experience. Word bad was used mostly to describe negative experience.



Figure 133: Fig.11

26.8 DISCUSSION

We observed higher accuracy in predicting star rating of 1 and 5 than all other ratings i.e. 2, 3 and 4. This was one limitation that we think is result of understanding user emotions and applying language model to decode it. It is difficult to find how many users write a very objective review and give star rating accordingly. It is possible that someone has either a better or worse than expected experience and provides a review based purely on emotions at that point of time. Additional analysis on understanding how a person from certain cultural background finds food from other cultures.

26.9 CONCLUSIONS

We have been able to show textual review and star rating are generally moderately correlated. If a user wishes to choose a business for visiting such as choosing a restaurant for eating out, they can look at rely more on star rating 4 and to find a good place and star rating 1 to know that a certain place is bad. If the restaurant has star rating of 2 or 3, they will have to spend more time in reading the actual textual reviews to make their decision. A good business has lot more reviews than a bad business. Most users give very few reviews and they do so more in case if their experience was good. Businesses serving pizzas and burgers are one of most liked places.

26.10 PROJECT MEMBERS AND WORK BREAKDOWN

- Prajakta Patil - fa18-523-65: Introduction, Literature review, Dataset, Data Analysis using Python, Results, Discussion
- Sahithya Sridhar- fa18-523-67: Abstract, Data processing, Analysis methods, Data Analysis using Python, Discussion, Conclusions

27 Do I BUY OR SELL? - USING BIG DATA TO PREDICT STOCK PRICES FA18-523-66 (...EDITING)

Ritu Susan Sanjay

[rssanjay@iu.edu](mailto:russanjay@iu.edu)

Indiana University, Bloomington

hid : fa18-523-66

github: [cloud](#)

code: [cloud](#)

Keywords

Stock prediction, SciPy, Numpy, Pandas, Python, Regression analysis, Classification, Quandl, Unsupervised Learning, Dow Jones

27.1 ABSTRACT

Big data is everywhere. Newspapers and magazines are teeming with details on how Company A implemented the perfect business plan or how XYZ pharmaceuticals developed the drug to reverse some previously uncurable disease. Better still is the internet with more resources to utilize big data. Numerous copies of books have been published on this topic and Forbes rates big data related jobs as the most desirable. Every single industry today has finally turned its attentions to better understanding how to use data to serve customers better and reap profits. In light of this, there are two industries which have the most to benefit from data science : a) the finance and investment industry and b) the medical industry. This paper focuses on how big data can be used to predict stock the rise and fall in stock prices.

27.2 INTRODUCTION

The financial sector is deemed to be one of the largest producers of data - roughly 2.5 quantillion bytes everyday[136]. However, many reports state how the investment sector could always make better use of their massive databank. Analytics has been widely adopted in the financial services industry and has helped traders make better decisions and thus see more favorable returns.

"Algorithmic trading now uses a great deal of historical data in conjunction with big data and complex mathematical formulas to help investors maximize the returns in their investment portfolios" [137].

In years past, traders and banks made decisions based on market trends and calculated risks on each investment. Today, however, they are at an advantage : computers can do the same things, only it is done on a massive scale and is far more accurate. The algorithm developed makes use of the vast amount of resources it has at its disposal. The accuracy of the predictions made by a computer depend on the input data i.e. the closer the data is to the real-time the more accurate its analysis of the current market. In his book, Our Final Invention, author James Barrat mentions that, if humans are ready to tackle AI and develop it to the ultimate Artificial Super Intelligence, then the Wall Street might be the first place to look for such an advancement [138]. Coming back to the point made earlier : the banking and investment sector holds some very promising advancements for data science and artificial intelligence.

On this note, it is then necessary to state that the stock prediction can be a very tempting prospect for most data scientists. The financial market has a very short feedback cycle and hence quick validation of predictions is possible. However, it is extremely difficult to develop a model that can provide higher levels of accuracy in most of its predictions. It is very much possible that the model developed would make predictions for a maximum of one day after which you might have to revamp it all over again [139]. Nonetheless, this paper will attempt to develop a model that predicts the trends for stock prices over time.

27.3 IMPLEMENTATION

27.3.1 Data

Knowledge Discovery of Databases is the sequential process of transforming raw data into actionable insights **???**. The most important step in the KDD lifecycle is: preprocessing the data. A model applied to a dataset is only as good as the data it is applied to. In this regard, most times raw data is available in the most crude formats, with noise, duplicates, missing values or at times even necessitate parsing of large webpages and documents.

Data for this project was derived from Quandl and Yahoo Finance. The stocks considered are those listed on the Dow Jones Index: 3M (MMM), American Express (AXP), Apple (AAPL), Boeing (BA), Caterpillar (CAT), Chevron (CVX), Cisco (CSCO), Coca-Cola (KO), DowDuPont Inc (DWDP), Exxon Mobil (XOM), Goldman Sachs (GS), Home Depot (HD), IBM (IBM), Intel (INTC), Johnson & Johnson (JNJ), JPMorgan Chase (JPM), McDonald's (MCD), Merck (MRK), Microsoft (MSFT), Nike (NKE), Pfizer (PFE), Procter & Gamble (PG), Travelers Companies Inc (TRV), United Health (UNH), United Technologies (UTX), Verizon (VZ), Visa (V), Walmart (WMT), Walgreens Boots Alliance (WBA), Walt Disney (DIS) **???**.

The historical for the stocks are available to be downloaded via an API on Quandl. The Quandl platform makes it easy to download financial and alternative data for analysis **???**. A python script can easily download allthe historical data for the companies in the Dow index. However, some datasets are only available to premium users; data for both Walgreens Boots Alliance (WBA) and DowDuPont Inc (DWDP) are available for a small fee. The data descriptions for the 30 datasets is as follows **???**:

- Date - date in YYYY-MM-DD format
- Open - opening price of stock in dollars
- High - highest price for day in dollars
- Low - lowest price for day in dollars

- Close - closing price for day in dollars
- Volume - volume of traded stocks
- Dividend - unadjusted dividend on any ex-dividend date else 0.0
- Split - shows any split on a the given day else 1.0
- Adj_Open - adjusted (for splits and dividends) using the CRSP
- Adj_High - adjusted (for splits and dividends) using the CRSP
- Adj_Low - adjusted (for splits and dividends) using the CRSP
- Adj_Close - adjusted (for splits and dividends) using the CRSP
- Adj_Volume - adjusted (for splits and dividends) using the CRSP

The historical data for the DJIA needs to be downloaded manually from yahoo finance. The dataset description for the same is as follows:

- Date - date in YYYY-MM-DD format
- Open - opening price of stock in dollars
- High - highest price for day in dollars
- Low - lowest price for day in dollars
- Close - closing price for day in dollars
- Volume - volume of traded stocks
- Adj_Close - adjusted to include any influential corporate actions and distributions

The final dataset required is the given day's data about the stock. This can be available in one of two ways: parsing the Yahoo Finance website for each stock or by parsing the CNN Business website for a complete summary table for all the stocks. Both methods are implemented via python scripts and the pandas toolkit. To parse the Yahoo Finance website for data, the first step involved is to download static files of the html pages (????this would reduce overhead). Then it is a simple case of parsing the HTML documents. The similar process can be followed to download the summary statistics from the CNN website.

Note that it is also possible to download historical data from the U.S. Securities and Exchange Commission (SEC) website if quandl is not

preferred ???/.

27.3.2 Design

The data is obtained and compiled with python scripts. R code is then used to mine this data. The goal of this project is to figure out whether a stock is underperforming or outperforming. This being a classification problem, the support vector machine algorithm was used.

27.3.2.1 Data Visualization

Figure 134 shows the trend changes on the DOW index over the past three decades.

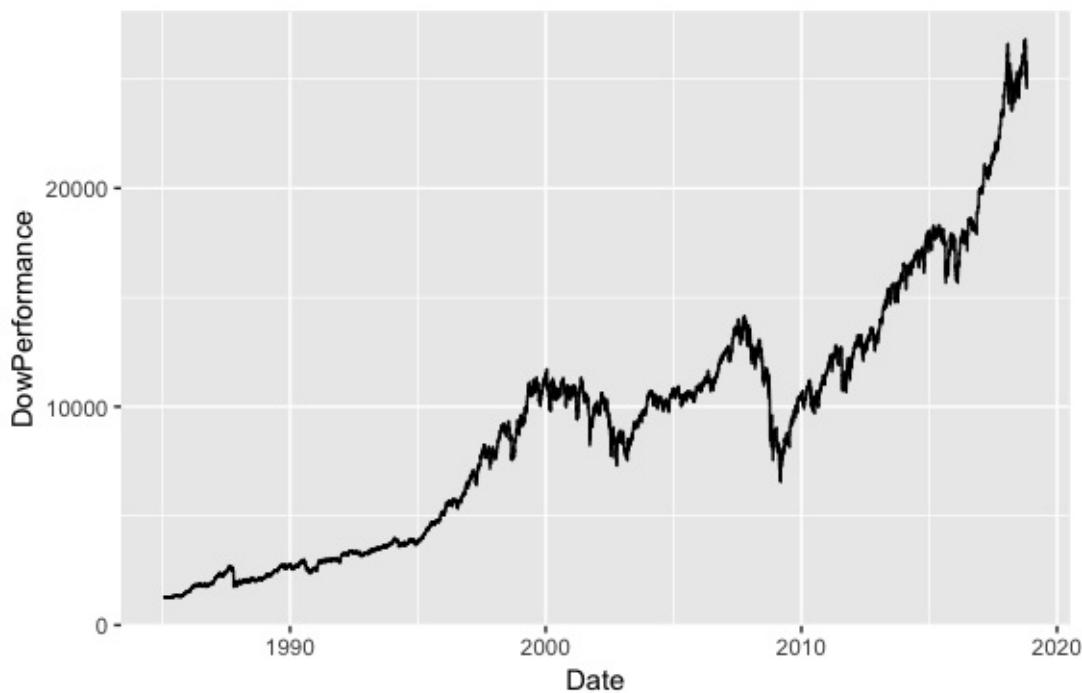


Figure 134: DJI Performance ???

Figure 135 shows the trend changes in the twenty-eight stocks over thirty years.

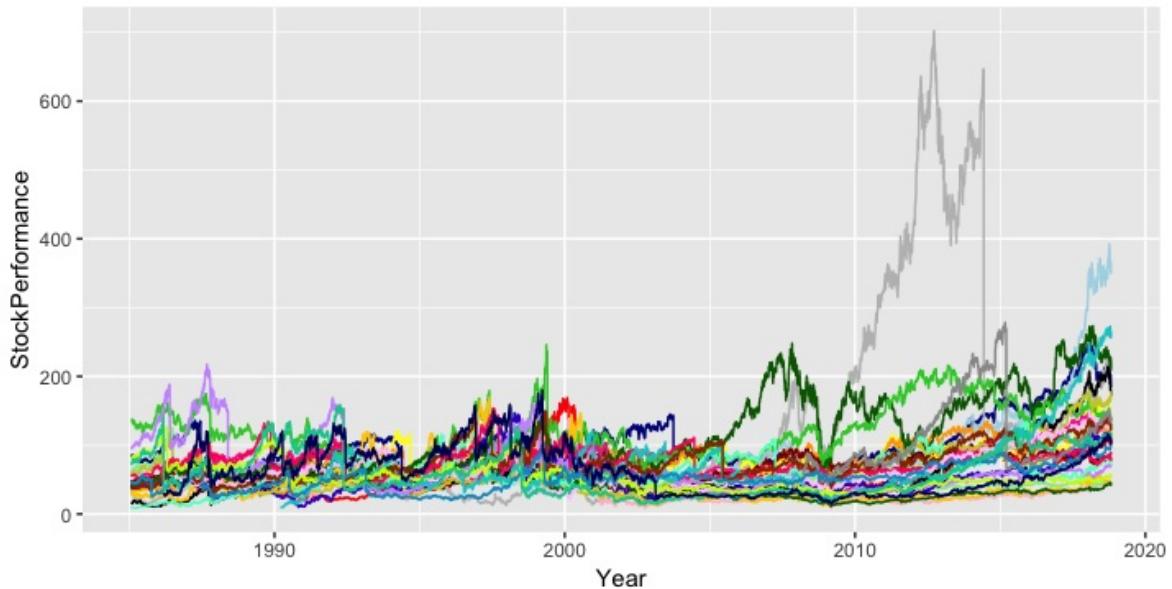


Figure 135: Stock Performance ???

The past century has shown a steep exponential increase in global capitals market. This in turn led to a boom in

Figure [136](#) shows the correlation between the open, low, high and close variables.

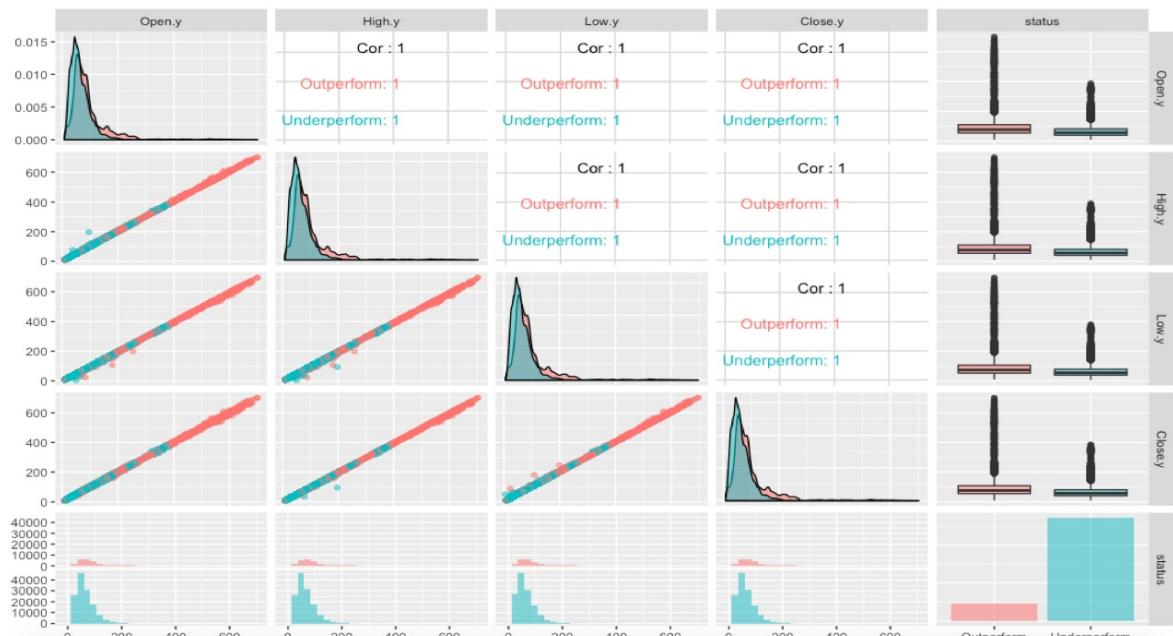


Figure 136: Data Correlation 1 ???

Figure [137](#) shows the correlation between the previous day close with

the day's open, low, high and close variables.

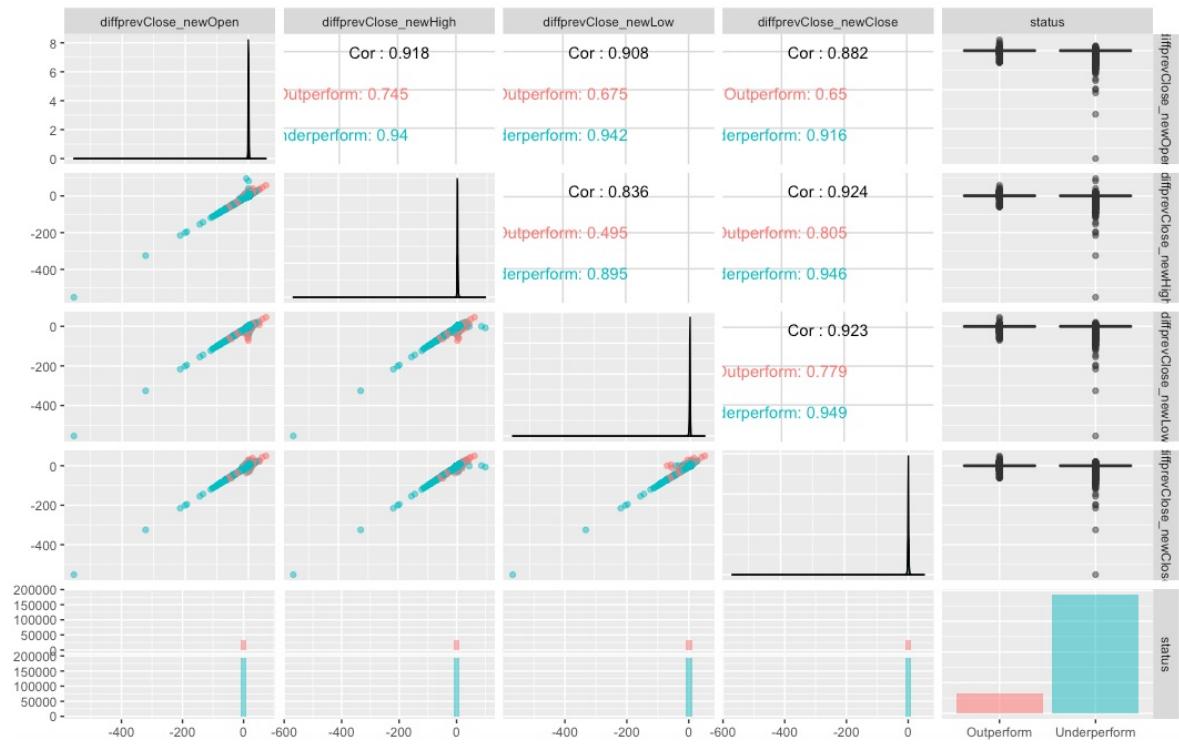


Figure 137: Data Correlation 2 ???

Figure 138 shows the correlation between the day's open and the low and high variables.

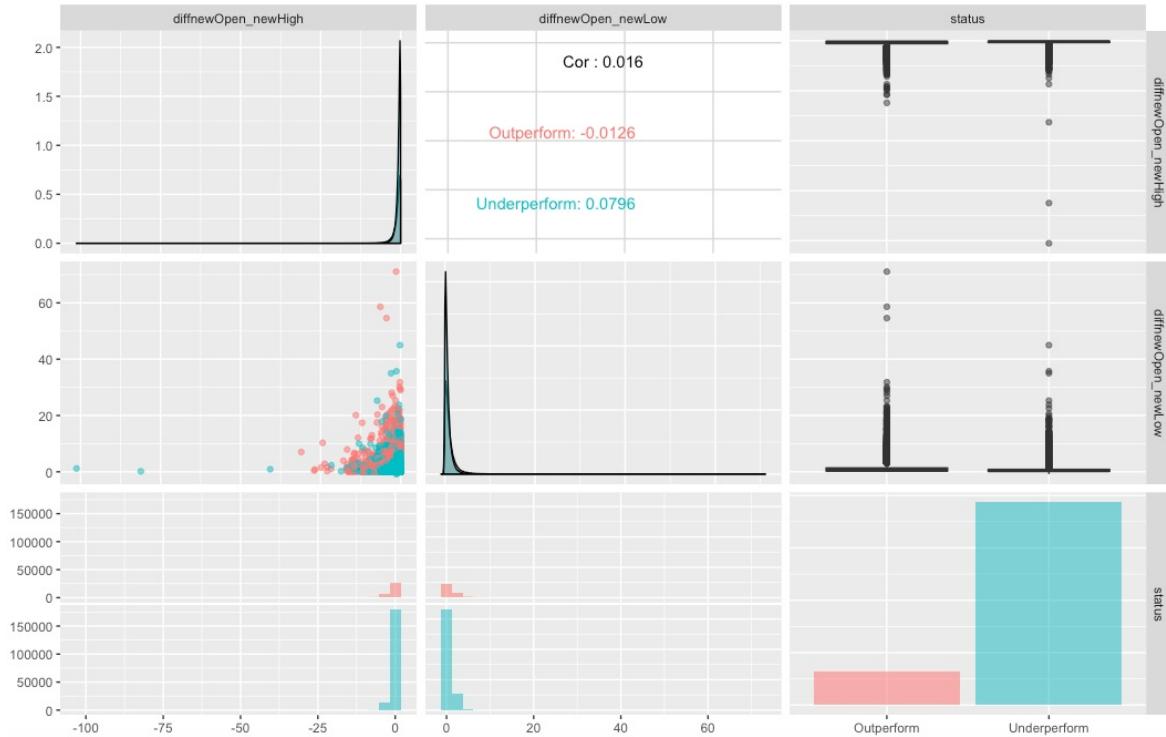


Figure 138: Data Correlation 3 ???

27.3.2.2 Support Vector Machine

"The SVM learning problem can be formulated as a convex optimization problem, in which efficient algorithms are available to find the > global minimum of the objective function. SVM performs capacity control by maximizing the margin of the decision boundary. Nevertheless, the user must still provide other parameters such as the type of kernel function to use and the cost function C for introducing each slack variable" ???.

27.3.2.2.1 Model Fitting

Figure [139](#) shows the summary after applying the svm model to the training dataset.

```

> summary(svm_model2)

Call:
svm.default(x = dataset1, y = dataset2, type = "C-classification", kernel = "radial",
            data = subdata_train)

Parameters:
  SVM-Type: C-classification
  SVM-Kernel: radial
    cost: 1
    gamma: 0.25

Number of Support Vectors: 44228

( 21556 22672 )

Number of Classes: 2

Levels:
  Outperform Underperform

```

Figure 139: SVM Model2 Summary ???

Figure [140](#) is the confusion matrix generated by the code for the svm model.

```

> table(pred2,dataset2)
           dataset2
pred2      Outperform Underperform
  Outperform        0          0
  Underperform     31        3469

```

Figure 140: SVM Model2 CM ???

Figure [141](#) shows the confusion matrix from applying the model to the test data.

```

> pred2 <- predict(svm_model2,dataset1)
> system.time(pred2 <- predict(svm_model2,dataset1))
  user  system elapsed
173.629   1.074 174.962
> table(pred2,dataset2)
  dataset2
pred2      Outperform Underperform
  Outperform        1446         309
  Underperform      21076      135045
> test_pred1 <- predict(svm_model2,type='response',newdata = dataset3)
> table(test_pred1,dataset4)
  dataset4
test_pred1      Outperform Underperform
  Outperform        619          140
  Underperform      9060       57842
>

```

Figure 141: SVM Model2 CM ???

27.3.2.2.2 Model Tuning

27.4 CONCLUSION

"It was not at all what the experts predicted. most of them did not foresee that an economic powerhouse could suffer so much damage in such a short period of time. They did not expect the fast-growing gross domestic product (GDP) to go so spectacularly into reverse, the real estate bubble to burst as violently as it did, and industrial production and capacity utilization to fall so steeply. Nor did they expect the stock market to plunge so dramatically from its all-time high-although it would recover some ground subsequently....But the real issue is not what has happened, but what happens net" ??? -out-of-the-great-recession.

The stock market is non-linear in nature, owing to which it is quite challenging to predict a trend with great accuracy ??? last tab opened. Nevertheless, it has attracted one too many investors and analysts in attempting to predict its performance. This report presents the study in using data mining algorithms (Support Vector Machines and

Logistic Regression) to predict whether a stock underperforms or outperforms the market. The imperfection in the results obtained via this project can be attributed to the change in political changes, current economical shifts and lastly the ever-changing investor prospects.

For future work, there are quite a number of improvements to be made. For example, kafka can be used to stream real-time data from websites. Also, it would be interesting to implement neural networks to improve the efficiency of the prediction. Text mining social media feeds could be implemented to bring aspects of changes in economy, politics and current trends to the model.

27.5 ACKNOWLEDGEMENTS

The author would like to thank professor Dr. Geoffrey C. Fox, Dr. Gregor von Laszewski for providing the opportunity to embark on this project. This project would not be possible without the guidance and inputs from the entire piazza team.

28 IMAGE CLASSIFICATION USING K-MEANS ON TENSORFLOW FA18-523-68

Selahattin Akkas
sakkas@iu.edu
Indiana University
hid: fa18-523-68
github: [cloud](#)
code: [cloud](#)

Keywords: Image classification, k-means, YFCC100M, TensorFlow

28.1 ABSTRACT

TBD

28.2 INTRODUCTION

The project goal is clustering the Yahoo Flickr Creative Commons 100 Million (YFCC 100 Million) using k-means on TensorFlow. Since data set is very large and some media has no tags, it will be hard to measure the accuracy. Yahoo also shares a subset of the dataset which has 10 tagged classes. In this work 10 class dataset will be used.

28.3 REQUIREMENTS

28.4 DESIGN

28.5 ARCHITECTURE

28.6 DATASET

Yahoo Flickr Creative Commons 100 Million (YFCC100m) dataset consists ~100 million photos(99.2 million) and videos(0.8 million). Medias in the dataset carry Creative Commons license [140]. Some medias have tags but in general the data is unlabeled. Therefore, total number of class is unknown and it will make the clustering harder. There is a subset of the dataset which have 10 classes. To see accuracy performance, this subset will be used in the project.

28.7 IMPLEMENTATION

28.7.1 TensorFlow

TensorFlow is a Machine Learning/Deep Learning framework developed by Google. It is continuation of DisBelief which is Google's internal use framework.

TensorFlow is widely used for Machine Learning/Deep Learning applications. It is easy to develop deep learning applications on TF. After training, applications can be easily deployed and used even on mobile phones [141].

28.8 BENCHMARK

28.9 CONCLUSION

28.10 ACKNOWLEDGEMENT

28.11 MILESTONES AND TIME PLAN

28.12 10/12 - 10/18:

- Clear the data if needed
- Extract the using VGG19 (It will be done one time and the extracted features will be used many times)

28.13 10/19 - 10/25:

- Tensorflow installation
- Run the built-in k-means on single node and decide the data size. (I need to get the results in reasonable time)
- Run the built-in k-means on 2-3 nodes.

28.14 10/26 - 11/08:

- Implement k-means to TensorFlow and run it on single node
- Run own implementation on multiple nodes.

28.15 11/09 - 11/26:

- Fix the problems
- Write the final paper

29 BIG DATA APPLICATION IN RECOMMENDER SYSTEMS

FA18-523-70

Sushmita Dash
sushdash@iu.edu
Indiana University
hid: fa18-523-70
github: [blue icon](#)

Keywords: recommender system, TV genome, KNN classification

29.1 INTRODUCTION

In today's world where people have a very busy lifestyle. They often do not have the time and patience to go through a very vast selection of options available to them. This is applicable in many aspects such as watching TV shows or getting a product online. Here is when our recommendation system comes into play. It plays a critical role in engaging the customers in the online service platforms. Earlier, in order to find a movie or a product that the user likes, they had to tediously browse through media catalogs or product catalogs. Amidst information overload, shorter attention span, and competing content, the only way to grab users' attention is personalization. This is where big data comes into play

There are many daily life examples where we can see the use of the recommender systems. Few examples are as follows:

1. Netflix uses this to show a personalized recommendation of the shows you may like based on the TV programs you have seen before
2. Various other tech giants also use this technology. For example, we often see friend suggestions in Facebook. It will be based on various criteria like how many mutual friends do

- we have. If we have been in any photo together or in the same school etc
3. We can see similar experience when we are browsing through a product catalog in Amazon or any other online shopping website. It will show us similar products based on our taste.

29.2 WHAT IS A RECOMMENDER SYSTEM?

A recommendation engine filters the data using different algorithms and recommends the most relevant items to users. It first captures the past behavior of a customer and based on that, recommends products which the users might be likely to buy or watch [142]. Below is a very simple illustration of how recommender systems work in the context of an e-commerce site.

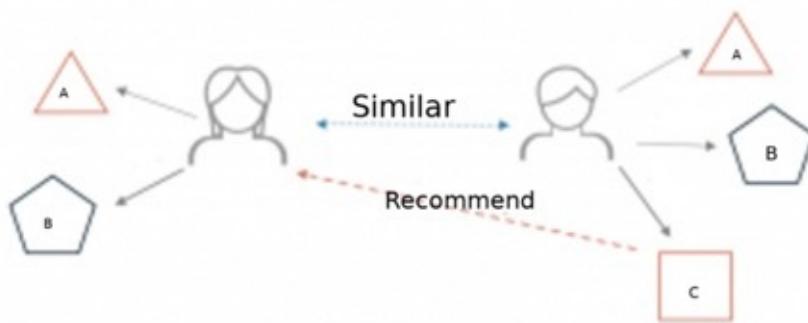


Figure 142: A simple recommender system analogy [142]

Two users buy the same items A and B from an ecommerce store. When this happens the similarity index of these two users is computed. Depending on the score the system can recommend item C to the other user because it detects that those two users are similar in terms of the items they purchase [143].

29.3 HOW DOES THE RECOMMENDER SYSTEM WORK?

A typical recommendation engine processes data through the following four phases namely collection, storing, analyzing and

filtering [144].

The phases are described below:



{#fig:Phasesofrecommendationengine]}

29.3.1 Collection of Data

The first step in creating a recommendation engine is gathering data. Data can be either explicit or implicit data. Explicit data would consist of data inputted by users such as ratings and comments on products. And implicit data would be the order history/return history, Cart events, Pageviews, Click thru and search log. This data set will be created for every user visiting the site.

Behavior data is easy to collect because you can keep a log of user activities on your site. Collecting this data is also straightforward because it doesn't need any extra action from the user; they're already using the application. The downside of this approach is that it's harder to analyze the data. For example, filtering the useful logs from the less useful ones can be difficult.

Since each user is bound to have different likes or dislikes about a product, their data sets will be distinct. Over time as you 'feed' the engine more data, it gets smarter and smarter with its recommendations so that your email subscribers and customers are more likely to engage, click and buy. Just like how the Amazon's recommendation engine works with the 'Frequently bought together' and 'Recommended for you' tab.

29.3.2 Storing the data:

The more training data is available for the algorithms, better the recommendations will be. This means that any recommendations project can quickly turn into a big data project. The storage of the

data depends on whether we are trying to capture user's input or behavior and on factors such as ease of implementation, size of the data, integration with other systems and portability. We can use noSQL database or a standard SQL database, etc. for data storage. When saving user ratings or comments, a scalable and managed database minimizes the number of tasks required and helps to focus on the recommendation. Cloud SQL fulfills both of these needs and also makes it easy to load the data directly from Spark.

29.3.3 Analyzing the data:

In order to find items with similar user engagement data, data is filtered using different analysis methods. Some of the ways in which we can analyze the data are:

- Real-time systems can process data as it's created. This type of system usually involves tools that can process and analyze streams of events. A real-time system would be required to give in-the-moment recommendations.
- Batch analysis demands you to process the data periodically. This approach implies that enough data needs to be created in order to make the analysis relevant, such as daily sales volume. A batch system might work fine to send an e-mail at a later date.
- Near-real-time analysis lets you gather data quickly so you can refresh the analytics every few minutes or seconds. A near-real-time system works best for providing recommendations during the same browsing session.

Algorithm:

for each item in product catalog, I_1

 for each customer C who purchased I_1

 for each item I_2 purchased by customer C

 Record that a customer purchased I_1 and I_2

for each item I_2

compute the similarity between I_1 and I_2

Algorithm Complexity: - Worst Case: $O(N^2M)$ - In practice: $O(NM)$, cause customers have fewer purchases

29.3.4 Filtering the data:

After collecting and storing the data, we have to filter it so as to extract the relevant information required to make the final recommendations. We need to filter the data to get the relevant data necessary to provide recommendations to the user. We have to choose an algorithm that would better suit the recommendation engine. For example

- **Content-based:** A popular, recommended product has similar characteristics to what a user views or likes.
- **Cluster:** Recommended products go well together, no matter what other users have done.
- **Collaborative:** Other users, who like the same products as another user views or likes, will also like a recommended product. Collaborative filtering enables you to make product attributes theoretical and make predictions based on user tastes. The output of this filtering is based on the assumption that two users who liked the same products in the past will probably like the same ones now or in the future.

Data about ratings or interactions can be represented as a set of matrices, with products and users as dimensions. Assume that the following two matrices are similar, but then we deduct the second from the first by replacing existing ratings with the number one and missing ratings by the number zero. The resulting matrix is a truth table where a number one represents an interaction by users with a product.

| Rating matrix | | Interaction matrix | | | | | | | | | |
|---------------|----------|--------------------|---|---|---|-------|----------|---|---|---|---|
| Users | Products | | | | | Users | Products | | | | |
| | 1 | 2 | 3 | 4 | 5 | | 1 | 0 | 1 | 1 | 0 |
| 1 | 2 | 5 | 3 | 1 | 4 | 2 | 0 | 1 | 0 | 1 | 0 |
| 2 | 1 | 3 | 2 | 4 | 5 | 3 | 1 | 0 | 1 | 1 | 1 |
| 3 | 4 | 2 | 1 | 5 | 3 | 5 | 0 | 1 | 1 | 0 | 1 |
| 4 | 5 | 4 | 3 | 2 | 1 | 4 | 1 | 0 | 1 | 1 | 1 |
| 5 | 3 | 5 | 1 | 4 | 2 | 3 | 0 | 1 | 1 | 0 | 1 |

Figure 143: Rating Matrix [142]

We use K-Nearest algorithm, Jaccard's coefficient, Dijkstra's algorithm, cosine similarity to better relate the data sets of people for recommending based on the rating or product.

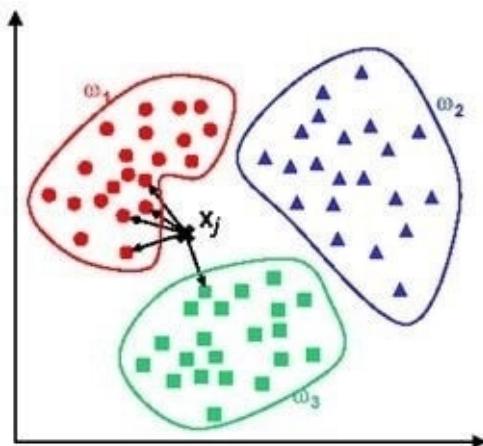


Figure 144: K-Nearest algorithm [142]

The above graph shows how a k-nearest algorithm's cluster filtering works. Then finally, the result obtained after filtering and using the algorithm, recommendations are given to the user based on the timeliness of the type of recommendation. Whether real time recommendation or sending an email later after some time.

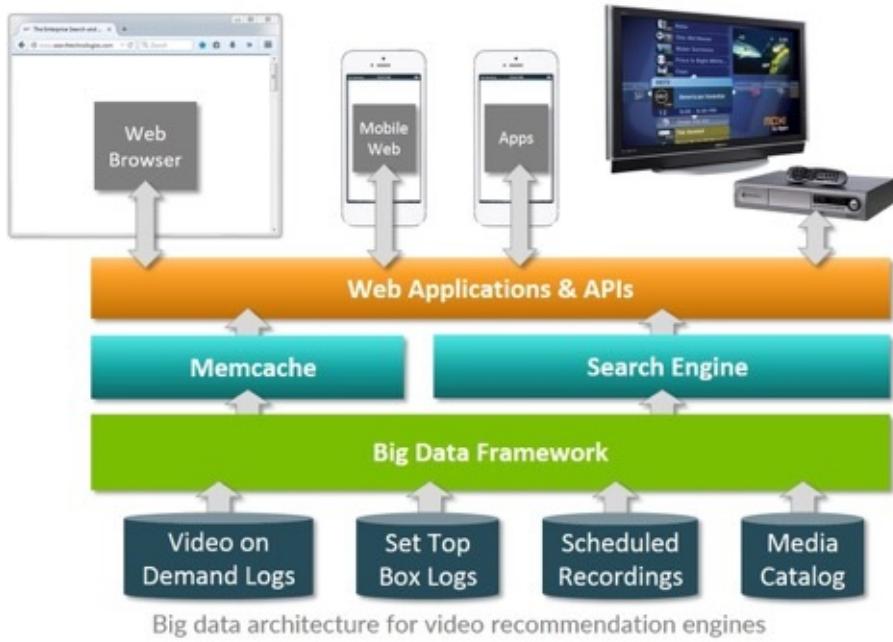


Figure 145: Big Data Architecture of Recommendation System [143]

29.4 TYPES OF RECOMMENDER SYSTEMS

Recommender systems are among the most popular applications of data science today [145]. They are used to predict the “rating” or “preference” that a user would give to an item. Almost every major tech company has applied them in some form or the other: Amazon uses it to suggest products to customers, YouTube uses it to decide which video to play next on auto play, and Facebook uses it to recommend pages to like and people to follow. What’s more, for some companies -think Netflix and Spotify-, the business model and its success revolves around the potency of their recommendations. In fact, Netflix even offered a million dollars in 2009 to anyone who could improve its system by 10%.

1. **Simple recommenders:** offer generalized recommendations to every user, based on movie popularity and/or genre. The basic idea behind this system is that movies that are more popular and critically acclaimed will have a higher probability of being liked by the average audience. IMDB Top 250 is an

example of this system.

2. **Content-based recommenders:** suggest similar items based on a particular item. This system uses item metadata, such as genre, director, description, actors, etc. for movies, to make these recommendations. The general idea behind these recommender systems is that if a person liked a particular item, he or she will also like an item that is similar to it.
3. **Collaborative filtering engines:** these systems try to predict the rating or preference that a user would give an item-based on past ratings and preferences of other users. Collaborative filters do not require item metadata like its content-based counterparts. Enables users to explore diverse contents, dissimilar to that viewed in the past.

29.5 ALGORITHMS

Content based methods are based on similarity of item attributes and collaborative methods calculate similarity from interactions [146]. Here we discuss few collaborative methods:

29.5.1 K-Nearest Neighbors

- Computes similarity of users
- Find k most similar users to user 'a'
- Recommends movies not seen by user 'a'

The simplest algorithm computes cosine or correlation similarity of rows (users) or columns (items) and recommends items that k—nearest neighbors enjoyed.

29.5.2 Association Rules

Association rules can also be used for recommendation. Items that are frequently consumed together are connected with an edge in the graph. You can see clusters of best sellers (densely connected items

that almost everybody interacted with) and small separated clusters of niche content [146].

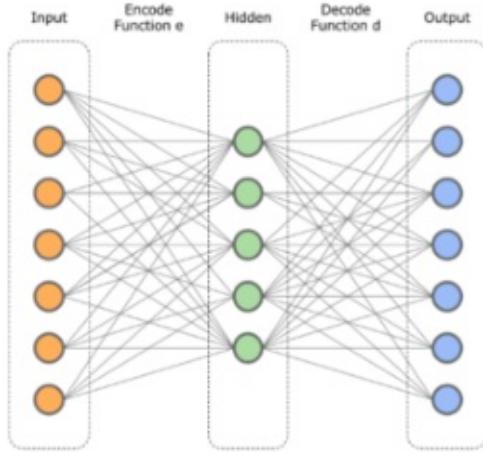
29.5.3 Matrix Factorization

"Matrix factorization models map both users and items to a joint latent factor space of dimensionality f , such that user-item interactions are modeled as inner products in that space. Accordingly, each item i is associated with a vector $q_i \in R^f$, and each user u is associated with a vector $p_u \in R^f$. For a given item i , the elements of q_i measure the extent to which the item possesses those factors, positive or negative. For a given user u , the elements of p_u measure the extent of interest the user has in items that are high on the corresponding factors, again, positive or negative. The resulting dot product, $q_i^T p_u$, captures the interaction between user u and item i —the user's overall interest in the item's characteristics. This approximates user u 's rating of item i , which is denoted by r_{ui} , leading to the estimate [147]."

Most popular training algorithm is a stochastic gradient descent (SGD) minimizing loss by gradient updates of both columns and rows of p a q matrices. SGD updates each parameter independently. Derive the loss function wrt each parameter.

29.5.4 Deep Neural Networks

Rating matrix can be also compressed by a neural network. So called autoencoder is very similar to the matrix factorization. Deep autoencoders, with multiple hidden layers and nonlinearities are more powerful but harder to train. Neural net can be also used to preprocess item attributes so we can combine content based and collaborative approaches.



{#fig: NeuralNetworks} [146]

$$\begin{aligned}
 \phi : X - > Z : x - > \phi(x) = \sigma(Wx + b) := z \\
 \Phi : Z - > Z : z - > \Phi(z) = \sigma(\bar{W}z + \bar{b}) := x' \\
 L(x, x') &= \sum_{i=1}^n \|x_i - x'_i\|^2 \\
 &= \sum_{i=1}^n \|x_i - \sigma(Wz_i + b)\|^2 \\
 &= \sum_{i=1}^n \|x_i - \sigma(W(\bar{W}x_i + \bar{b}) + \bar{b})\|^2
 \end{aligned}$$

Neural Network Equation

29.6 EVALUATION OF RECOMMENDER SYSTEMS

Few methods how the accuracy of a recommender system can be evaluated are as follows:

29.6.1 Validation of Recommender System

- Recommenders can be evaluated similarly as classical machine learning models on historical data
- Users are divided into:
 - Training set - This is fully submitted to the recommender system

- Testing set - This is submitted partially and used to evaluate the recommender

29.6.2 Root mean squared error

- When some observed data is provided, the recommender system is to predict the rating of an unknown user-item pair

"The root-mean-square deviation (RMSD) or root-mean-square error (RMSE) (or sometimes root-mean-squared error) is a frequently used measure of the differences between values (sample or population values) predicted by a model or an estimator and the values observed. The RMSD represents the square root of the second sample moment of the differences between predicted values and observed values or the quadratic mean of these differences[148]."

$$RMSE(model) = \sqrt{\frac{1}{|R_{test}|} \sum_{(u,i,r) \in R_{test}} (model(u, i) - r)^2}$$

RMSE equation

29.6.3 Top N Recommendations

"The explosive growth of the world-wide-web and the emergence of e-commerce has led to the development of recommender systems—a personalized information filtering technology used to identify a set of N items that will be of interest to a certain user. User-based Collaborative filtering is the most successful technology for building recommender systems to date, and is extensively used in many commercial recommender systems. Unfortunately, the computational complexity of these methods grows linearly with the number of customers that in typical commercial applications can grow to be several millions. To address these scalability

concerns item-based recommendation techniques have been developed that analyze the user-item matrix to identify relations between the different items, and use these relations to compute the list of recommendations. In this paper we present one such class of item-based recommendation algorithms that first determine the similarities between the various items and then used them to identify the set of items to be recommended. The key steps in this class of algorithms are (i) the method used to compute the similarity between the items, and (ii) the method used to combine these similarities in order to compute the similarity between a basket of items and a candidate recommender item. Our experimental evaluation on five different datasets show that the proposed item-based algorithms are up to 28 times faster than the traditional user-neighborhood based recommender systems and provide recommendations whose quality is up to 27% better[149]."

$$\text{Precision on Top - N} : \text{Precision}(u) = \frac{|\text{Recommended}(u) \cap \text{Testing}(u)|}{|\text{Recommended}(u)|}$$

$$\text{Recall on Top - N} : \text{Recall}(u) = \frac{|\text{Recommended}(u) \cap \text{Testing}(u)|}{|\text{Testing}(u)|}$$

$$\text{Serendipity, DCG} : \text{DCG} = \sum_{i=1}^p \frac{2^{rel_i} - 1}{\log_2(i + 1)}$$

Top N Recommendation equation

29.7 ACKNOWLEDGEMENT

I am thankful to Dr Gregor von Laszewski to help me complete the project and the paper for the Big Data Applications and Analytics course

30 CREDIT CARD DEFALTER ANALYSIS

FA18-523-59

FA18-523-71

Uma Kota, Jatinkumar Bhutka
umabkota@iu.edu, jdbhutka@iu.edu
Indiana University Bloomington
hid: fa18-523-71 fa18-523-59
github: [blue user icon](#)
code: Project report only

30.0.1 Keywords: TBD

In 2006, Taiwan saw credit card debt crisis, the banks of Taiwan over issued credit cards in pursuit of increasing their market share. Also, many cardholders consumed more than their capacity thereby accumulating heavy debts. Utilizing this data, we aim to classify the defaulters by performing different statistical analyses, to help predict an individual customer's credit risk. We believe this analysis will help us better understand the route cause behind defaulting and also provide a better way to classify an applicant given his data. We'll be working on different modelling techniques like k means, logistic regression, naive Bayes and artificial neural networks to get provide our solution.

30.0.2 Data

The Default of credit card clients is multivariate Data Set from UCI machine learning Repository. It consists of 30,000 records with 23 features. The features includes amount of given credit, Sex, Education, Marital status, Age, History of past payments, amount of bill statements, amount paid.

30.0.3 Questions

- what is the best evaluation metric for the models and it's precision, recall or specificity and sensitivity.
- Which is the best model to predict the defaulters.
- Which features influence most to the default payments.
- How this can help to minimize bank's risk.
- We will analyze data to answer more questions.

30.0.4 Timeline

- Data Pre-processing – we will be using python for this
- Exploratory Data Analysis
- Data modelling
 - k-means
 - Logistic Regression
 - Naive Bayes
 - Artificial Neural Network
- Modelling Analysis
- Evaluation and Conclusion

30.0.5 References

[150]

31 BIG DATA IN EDUCATION

FA18-523-73, FA18-523-74

Tong Wang, Yeyi Ma
wangton@iu.edu, yeyima@umail.iu.edu
Indiana University Bloomington
hid: fa18-523-73, fa18-523-74
github: [blue icon](#)

Keyword: Education

31.1 ABSTRACT

The advancement of technology has affected education in a big way. In recent times, the educators and administrators mainly rely on the Student Information Systems (SISs), Learning Management Systems (LMSs), and Admissions Management Systems to streamline classroom, school and campus operations[151]. Therefore, as institutions continue to adopt such systems, user data is generated, which helps the education stakeholders including the parents to make more informed decisions on how the learning process can be improved. As a result[152], highlights that big data has fundamentally changed the way education is offered, and institutions that invest in the big data and successfully derive value from their data will have a distinct advantage over others. Moreover[153] anticipates that in the near future there will be a performance gap between the institutions with big data systems and those that have not implemented such systems. This is because as time goes by, more relevant data will be generated, and the emerging technologies and digital channels will offer better acquisition and delivery mechanisms. For instance, teachers can mine learning patterns to see how students master specific subjects and experimental designs. Therefore, this paper analyzes how big data is relevant in the education system.

Introduction

Today, many organizations are collecting, storing, and analyzing massive amounts of data. This data is commonly referred to as “big data” because of its volume, the velocity with which it arrives and the variety of forms it takes. Therefore, big data can be described as the process which is used to convert continuous, analog information into discrete, digital and machine reliable format. Examples of big data include web data, text data, time and location data, smart grid, and social network data. Such data can be utilized in educational institutions by using the data-driven approaches that make it possible to study learning in real-time and offer systematic feedback to students and teachers[152]. As a result, it is crucial to explore how the big data is used to mine learning information for insights regarding student performance and learning approaches.

31.2 FORMS OF BIG DATA IN EDUCATION INSTITUTION

31.2.1 Administrative Data

These type of data refers to the demographic, behavioral, and achievement data collected through schools, government agencies, and their contractors. It is obtained by sampling many participants, which can be longitudinally at prescribed, or regular intervals. In most cases, this type of data is made up by attendance records, test scores, transcripts, and surveys. Examples of the administrative data that can be taken as big data include international test scores, a national assessment of educational progress data, and the behavior data that is collected by the U.S. Department of Education’s Civil Rights Data Collection. ### Learning Process Data

They are continuous or near-continuous, fine-grained records, mostly in the form of digital interactions of students behaviors to illuminate learning processes. The learning process data is considered big data since it involves many participants and a large number of variables in regards to a single person. Examples of learning process include online assessments and courses such as the massive open online courses.

31.3 THE VALUE OF SYSTEMATIC, REAL-TIME DATA

The use of a computerized system in schools helps the instructors to assess the learning process in real-time. Therefore, by the use of data mining and analytic data software, it is possible to get the immediate feedback in regards to how the learning process is taking place. In the process, the analytic software analyzes the underlying patterns, which help advice on the best practices to undertake for improving or maintaining high performance. An example of the application of the real-time data is in Connected Chemistry that enables students to understand more in-depth the key concepts in molecular theory and gasses[153]. Therefore, the use of mining and analytic software allows the teachers to mine learning patterns to analyze how the students understand the various areas of chemistry.

Moreover, Joseph Beck and Jack Mostow are known individuals who have examined the benefits of the big data in education. In their experimentation, they applied the intelligent tutor software to study how the students comprehend a story. In the process, they analyzed the reading time, word knowledge, reading mistakes, and the help requests. Beck and Mostow concluded that when students re-read an old story, it assists them understands the words better than reading a new story.

The other example that showcases how big data relates to education is research carried out by James Theroux of the University of Massachusetts at Amherst. The study focused on students who did average courses, aiming to analyze their perceptions of the uses of online learning compared to the traditional approach. Theroux found out that majority of the students preferred learning via online platforms against the conventional methods. Their preference was mainly because it is a modern method that relates to them. Therefore, the big data technologies can help the institutional faculties to assess the degree to which the students understand the various concepts when learning via online as to other methods. Similarly, the same issue was raised by scholars Robert Perkins and Margaret McKnight of 139 who also concluded that most students

prefer using online platforms for learning as they enjoyed their collaborative and interactive nature[153].

Therefore, the big data is very applicable in schools. For instance, the data collected in real-time can help evaluate the performance of a learning process. The data-driven approaches make it possible to study learning in real-time and offer regular feedback to students and teachers[152]. One advantage of the real-time data is that it provides more accurate results than other methods since the data is not manipulated or distorted before it is analyzed. Thus, schools need to have learning analytics modes to scrutinize the learning process in real-time to identify areas of improvements.

31.4 LEARNING ANALYTICS

In recent times, learning analytics has become an area of research and application. However, unlike the traditional data mining, it draws on a broader array of academic disciplines, incorporating concepts and techniques from information science and sociology. Therefore, the approach primarily focuses on the analysis and reporting of the data. In other words, it aims to look at the application of known methods to answer the crucial questions that affect the performance in schools. Thus, learning analytics can monitor and predict student's learning performance and brings out the potential issues that the administration needs to address.

31.5 REQUIREMENTS

When establishing the requirement of the big data implementation, it is important to adopt a user-center design approach. It is an appropriate method rather than specifying the elements at the beginning for various reasons. According to Williamson[153], when requirements are collected at the beginning of the project, it may result in later problems in the system development. On the other hand, a user-centered approach enables the development team to establish the communication between users and developers and to

gather different requirements on each side. However, when determining the conditions for big data implementation, the procedures are guided by ISO/IEC/IEEE 42010:2011 “Systems and software engineering- Architecture description.” The following are various requirements that need to be established to implement the system successfully.

31.5.1 Prediction

The first requirement of big data in school is to develop a model that can help infer a single aspect of the data. For instance, it can assist detect students behaviors when they are involved in activities such as gaming, engaging in off-task behavior. Moreover, the predicting model is also applied to study the behavior of the students in an online learning environment by examining their participation in discussion forums and taking practice tests. Therefore, it is important to have a prediction model to forecast and understand student educational outcomes such as success on posttests after tutoring.

31.5.2 Clustering

The second requirement needs one to find data points that can assist in grouping the collected information. For instance, one can group the students based on their learning difficulties and interaction patterns. For example, in an online learning set-up, the analytics can examine the student’s cognitive interview and also how they post in the discussion board.

31.5.3 Relationship Mining

It involves establishing the relationship between variables in a dataset and then creating the rules to be used in later stages. For example, it is important to identify how students within a specific cluster relate. One of the methods used is the association rule mining, which assists in determining the behaviors students. The other method applied to discover relationship is sequential pattern mining which aids in capturing the connections between occurrences of subsequent

events. For instance, it is used in detecting students regressing to make errors in mechanics when they are writing with more complex and critical thinking techniques.

31.6 ARCHITECTURE

When designing big-data applications, one needs to have particular methods, tools, and technologies that can handle data efficiently from the initial stage of data collection to the final stage of the interpretation. As a result, a meaningful big data architecture should consider the characteristics of the data extracted. For this project, the architecture is divided into two main section: data management and data analytics. The data management process acquire, govern, integrate, secure and store data that is prepared to be used in data-analytics methods. The data management system is composed of five different categories: distributed file system, cluster management, data store, governance and security, and data ingestion. On the other hand, data analytics involves data modeling, analyzing and interpreting to transform raw data into valuable knowledge. It is divided into distributed data, processing and programming, visualization, data analysis, and pre-processing data components.

31.6.1 Data Management

31.6.1.1 Distributed File System (DFS)

As shown in the diagram above, DFS is the lowest level of the architecture. It is used for the storage and maintains large amounts of data across multiple nodes of commodity hardware. However, it is designed in such a way that it conforms to master and slave node architecture.

31.6.1.2 Cluster Management

The components help in deploying, scheduling and orchestrating the jobs across the other parts of the architecture to ensure there is a

readily available and highly scalable computing infrastructure.

31.6.1.3 Distributed Data Processing and Programming

It is used to process the big amount of batch data and millions of data in real time.

31.6.1.4 Data Store

It is used for the storage of the diverse and large number of data generated throughout the architecture. To increases the efficiency of the architecture, it is important to have a distributed, scalable, schema-free and fault-tolerant data stores.

31.6.1.5 Visualization

Visualization entails representing the massive amount of data in graphical formats for easy interpretation.

31.6.1.6 Data analysis

Data analysis is the process of deploying analytic data software to analyze the raw data and present useful, meaningful information to the end users. Therefore, an analytic model is used to infer knowledge by finding patterns and drawing conclusions with the aid of specialized tools and algorithms.

31.6.1.7 Data pre-processing

This step ensures that the data stored and used for interpretation is reliable. During this stage, the data is cleaned to eliminate noises, errors, or incomplete data to improve the overall success of data analysis.

31.6.1.8 Governance and Security

These entails the rules laid out to ensure the data is secure. Within the big data architecture, there exist several governance policies. For instance, Apache Atlas provides a scalable and extensible set of core foundational governance services. On the other hand, Apache Ranger gives the details of how to successfully implement a secure data framework to monitor and manage the data across the system.

31.6.1.9 Data Ingestion

It helps in transferring data from outside sources to the internal systems within the architecture. Since the big data involves a large volume of data and velocity characteristics of big data, data ingestion needs to be resilient and fault-tolerant.

31.6.1.10 Application

The application layer helps present the analyzed data to the end-users. An example of application is Nutch, which is a production-ready web crawler.

In conclusion, the architecture represents a key part of the big data. For that reason, it is crucial for an educational institution to try and create a unified information architecture, to ensure it leverages all types of data. Moreover, a well-designed architecture promotes a unified vision for information management and analytics. Therefore, it is important to get a better understanding of the role of the implementation of the big data, which would provide a better context for deriving requirements for architecture.

31.7 IMPLEMENTATION

31.7.1 Factors to Consider Before Implementing the Big Data

The big data has become a game changer in shaping the future of education. However, certain factors need to be considered before

applying the big data project.

31.7.1.1 Understanding Industry Point-of-View on Big Data

The education institutions needs first to assess the capabilities and trends to sidestep mistakes made in other sectors, particularly in the business world. For instance, it is advisable that before an institution or organization implement the big data to first study reports from well-regarded publications such as IDC and Forrester's.

31.7.1.2 Developing a Proof of Concept (PoC)

It is helpful first to create a business case that showcases all the values and the costs associated with the implementations of the big data.

31.7.1.3 Understanding the Current Tools and Technology

After the institution approves of continuing with the implementation of the big data, the next step entails analyzing the available tools and technology that can help meet architectural guidelines requirements.

31.7.1.4 Developing the Measures of Successful Big Data

It is important to have a target that will measure the benefits of the implementation of the big data system.

31.7.2 Steps of Successful implementation of the Big Data

31.7.2.1 Identification of the Goals

Both in business and education world requires one first to determine the purpose of the big data. For instance, in an education set-up, the institutional management needs to clarify whether the big data is to increase the efficiency of the learning process, improve institution's

marketing, or to improve the overall performance of the institution. Therefore, based on the purpose, then it is easy to choose a methodology, hire employees, and select the right sources of data.

31.7.2.2 Leverage a Proven Big Data Strategy

There exist four prove strategies, which are based on the objective of the big data as well as the availability of the data.

31.7.2.2.1 Performance Management

This approach involves applying the transactional data such as the client purchase history and their turnover. However, this approach is not suitable for an education set up.

31.7.2.2.2 Data Exploration

This strategy primarily relies on the use of data mining and research to obtain solutions to the various problems an organization might be facing. In the process, the analytic teams can identify new segments of data.

31.7.2.2.3 Social Analytics

Social analytics approach employs the non-transaction data on social media platforms such as Facebook, Twitter, and Instagram. It is an applicable method in an education setting since most of the students use such platforms as their modes of communication.

31.7.2.2.4 Decision Science

This approach relies on the analysis of non-transactional data such as online discussion forums. Therefore, it utilizes text and sentiment analysis to determine students/customer's opinions in regards to new services and schemes.

31.7.2.3 Identify infrastructural changes

To meet the requirements of the big data, the institution may need to change some of its infrastructures. For instance, the traditional storage mediums might not facilitate the running of complex algorithms and analysis. It may also involve the reconfiguration of the hardware platform to meet the new computation needs.

31.7.2.4 Training of the Staffs

It is crucial that all the staffs working within the institutions are conversant with the big data. Therefore, at the start, it is important to hire outside consulting team to guide on how best to utilize the big data. Moreover, it is essential to hire new strong system programmers who can work in a parallel processing environment and a network communications specialist.

31.8 BENCHMARKING

Big data benchmarks are used to evaluate and compare the performance of big data systems and architecture. A successful benchmarking entails five steps: planning, generating data, generate tests, execution, and analysis and evaluation.

The process provides a realistic and accurate measure of the big data systems and therefore, it can enable one to make efficient decision-making. For instance, a successful big data support education institutions to make decisions for planning system features, tuning system configuration and validate the deployment strategies. The following are the various steps of benchmarking:

31.8.1 Planning

One needs to make a plan on how the big data benchmarking will take place. Therefore, it is important to first come up with the goals of the benchmarking process. For instance, it should be able to generate application-specific workloads and tests such that it's easier to draw meaningful conclusions. Also, in the planning tests, it is imperative to

lay down the methodology to use in the benchmarking process.

31.8.2 Generating Data

There are four properties associated with big data: volume, velocity, variety, and veracity. Therefore, benchmarking should aim to cover those areas, and thus, it is crucial to apply real-world data or generates synthetic data for application-specific workloads. However, in most cases, most real-owners are not willing to share their real-data because of the security issues. Consequently, in big data benchmarks, the consensus is generated synthetic data as inputs of workloads by real data sets.

31.8.3 Generating Tests

In big-data benchmarks, one should first identify the typical workload behaviors for an application domain. Besides, it is important to take into considerations the diversity of workloads to cover different types of application domains, and at the same time to generate tests based on the workloads automatically. However, due to complexities involved in big data, it is challenging to develop big data benchmarks that reflect the real workload cases.

31.8.4 Execution

The execution stage needs to address various challenges. For instance, the big-data benchmarks need to adapt to different data formats. For example, the text data can be stored in the form of text files, web pages or pdf. The data also needs to be portable to represent different software stacks to provide a fully functional solution. Besides, the execution must reflect the user's experiences in using benchmarks.

31.9 CONCLUSION

From the discussion, it is a crucial time for the education institution to

implement big data. Massive quantities of educational data can now be stored, analyzed and shared. For instance, various big-data algorithms can help track individual students in school and as a result keep detailed information about their academic performance, behavior, and educational needs. Moreover, the big-data systems can help improve educational services by assisting the teachers to analyze what students know and what techniques are most useful for each pupil. Furthermore, technologies such as data mining and data analytics can provide fast feedback to students and teachers about their academic performance.

However, irrespective of the benefits associated with the big data, several challenges need to be addressed. For instance, the massive amounts of data generated by the big-data systems require much larger storage systems. To overcome such a problem, the data specialists need databases that do not use the traditional SQL based queries. The other challenge is in regards to the analysis as data is generated in different structure and size, and the study of the data may consume a lot of time and resources. Moreover, it is challenging reporting the analyzed information since when a large amount of data are involved, the traditional reports become difficult to interpret by human beings. Therefore, it requires useful tools and techniques that can address such challenges. An example includes the analytics software which can provide an in-depth analysis of some education patterns and extract valuable knowledge from them.

32 BIG DATA IN EDUCATION

FA18-523-73, FA18-523-74

Tong Wang, Yeyi Ma
wangton@iu.edu, yeyima@umail.iu.edu
Indiana University Bloomington
hid:fa18-523-73, fa18-523-74
github: [github](#)

Keyword: Education

32.1 ABSTRACT

The advancement of technology has affected education in a big way. In recent times, the educators and administrators mainly rely on the Student Information Systems (SISs), Learning Management Systems (LMSs), and Admissions Management Systems to streamline classroom, school and campus operations[151]. Therefore, as institutions continue to adopt such systems, user data is generated, which helps the education stakeholders including the parents to make more informed decisions on how the learning process can be improved. As a result[152], highlights that big data has fundamentally changed the way education is offered, and institutions that invest in the big data and successfully derive value from their data will have a distinct advantage over others. Moreover[153] anticipates that in the near future there will be a performance gap between the institutions with big data systems and those that have not implemented such systems. This is because as time goes by, more relevant data will be generated, and the emerging technologies and digital channels will offer better acquisition and delivery mechanisms. For instance, teachers can mine learning patterns to see how students master specific subjects and experimental designs. Therefore, this paper analyzes how big data is relevant in the education system.

32.2 INTRODUCTION

Today, many organizations are collecting, storing, and analyzing massive amounts of data. This data is commonly referred to as “big data” because of its volume, the velocity with which it arrives and the variety of forms it takes. Therefore, big data can be described as the process which is used to convert continuous, analog information into discrete, digital and machine reliable format. Examples of big data include web data, text data, time and location data, smart grid, and social network data. Such data can be utilized in educational institutions by using the data-driven approaches that make it possible to study learning in real-time and offer systematic feedback to students and teachers[152]. As a result, it is crucial to explore how the big data is used to mine learning information for insights regarding student performance and learning approaches.

32.3 FORMS OF BIG DATA IN EDUCATION INSTITUTION

32.3.1 Administrative Data

These type of data refers to the demographic, behavioral, and achievement data collected through schools, government agencies, and their contractors. It is obtained by sampling many participants, which can be longitudinally at prescribed, or regular intervals. In most cases, this type of data is made up by attendance records, test scores, transcripts, and surveys. Examples of the administrative data that can be taken as big data include international test scores, a national assessment of educational progress data, and the behavior data that is collected by the U.S. Department of Education's Civil Rights Data Collection.

32.3.2 Learning Process Data

They are continuous or near-continuous, fine-grained records, mostly in the form of digital interactions of students behaviors to illuminate learning processes. The learning process data is considered big data

since it involves many participants and a large number of variables in regards to a single person. Examples of learning process include online assessments and courses such as the massive open online courses.

32.4 THE VALUE OF SYSTEMATIC, REAL-TIME DATA

The use of a computerized system in schools helps the instructors to assess the learning process in real-time. Therefore, by the use of data mining and analytic data software, it is possible to get the immediate feedback in regards to how the learning process is taking place. In the process, the analytic software analyzes the underlying patterns, which help advice on the best practices to undertake for improving or maintaining high performance. An example of the application of the real-time data is in Connected Chemistry that enables students to understand more in-depth the key concepts in molecular theory and gasses[153]. Therefore, the use of mining and analytic software allows the teachers to mine learning patterns to analyze how the students understand the various areas of chemistry.

Moreover, Joseph Beck and Jack Mostow are known individuals who have examined the benefits of the big data in education. In their experimentation, they applied the intelligent tutor software to study how the students comprehend a story. In the process, they analyzed the reading time, word knowledge, reading mistakes, and the help requests. Beck and Mostow concluded that when students re-read an old story, it assists them understands the words better than reading a new story.

The other example that showcases how big data relates to education is research carried out by James Theroux of the University of Massachusetts at Amherst. The study focused on students who did average courses, aiming to analyze their perceptions of the uses of online learning compared to the traditional approach. Theroux found out that majority of the students preferred learning via online platforms against the conventional methods. Their preference was mainly because it is a modern method that relates to them.

Therefore, the big data technologies can help the institutional faculties to assess the degree to which the students understand the various concepts when learning via online as to other methods. Similarly, the same issue was raised by scholars Robert Perkins and Margaret McKnight of 139 who also concluded that most students prefer using online platforms for leaning as they enjoyed their collaborative and interactive nature[153].

Therefore, the big data is very applicable in schools. For instance, the data collected in real-time can help evaluate the performance of a learning process. The data-driven approaches make it possible to study learning in real-time and offer regular feedback to students and teachers[152]. One advantage of the real-time data is that it provides more accurate results than other methods since the data is not manipulated or distorted before it is analyzed. Thus, schools need to have learning analytics modes to scrutinize the learning process in real-time to identify areas of improvements.

32.5 LEARNING ANALYTICS

In recent times, learning analytics has become an area of research and application. However, unlike the traditional data mining, it draws on a broader array of academic disciplines, incorporating concepts and techniques from information science and sociology. Therefore, the approach primarily focuses on the analysis and reporting of the data. In other words, it aims to look at the application of known methods to answer the crucial questions that affect the performance in schools. Thus, learning analytics can monitor and predict student's learning performance and brings out the potential issues that the administration needs to address.

32.6 REQUIREMENTS

When establishing the requirement of the big data implementation, it is important to adopt a user-center design approach. It is an appropriate method rather than specifying the elements at the

beginning for various reasons. According to Williamson[153], when requirements are collected at the beginning of the project, it may result in later problems in the system development. On the other hand, a user-centered approach enables the development team to establish the communication between users and developers and to gather different requirements on each side. However, when determining the conditions for big data implementation, the procedures are guided by ISO/IEC/IEEE 42010:2011 “Systems and software engineering- Architecture description.” The following are various requirements that need to be established to implement the system successfully.

32.6.1 Prediction

The first requirement of big data in school is to develop a model that can help infer a single aspect of the data. For instance, it can assist detect students behaviors when they are involved in activities such as gaming, engaging in off-task behavior. Moreover, the predicting model is also applied to study the behavior of the students in an online learning environment by examining their participation in discussion forums and taking practice tests. Therefore, it is important to have a prediction model to forecast and understand student educational outcomes such as success on posttests after tutoring.

32.6.2 Clustering

The second requirement needs one to find data points that can assist in grouping the collected information. For instance, one can group the students based on their learning difficulties and interaction patterns. For example, in an online learning set-up, the analytics can examine the student’s cognitive interview and also how they post in the discussion board.

32.6.3 Relationship Mining

It involves establishing the relationship between variables in a dataset and then creating the rules to be used in later stages. For example, it

is important to identify how students within a specific cluster relate. One of the methods used is the association rule mining, which assists in determining the behaviors students. The other method applied to discover relationship is sequential pattern mining which aids in capturing the connections between occurrences of subsequent events. For instance, it is used in detecting students regressing to make errors in mechanics when they are writing with more complex and critical thinking techniques.

32.7 ARCHITECTURE

When designing big-data applications, one needs to have particular methods, tools, and technologies that can handle data efficiently from the initial stage of data collection to the final stage of the interpretation. As a result, a meaningful big data architecture should consider the characteristics of the data extracted. For this project, the architecture is divided into two main section: data management and data analytics. The data management process acquire, govern, integrate, secure and store data that is prepared to be used in data-analytics methods. The data management system is composed of five different categories: distributed file system, cluster management, data store, governance and security, and data ingestion. On the other hand, data analytics involves data modeling, analyzing and interpreting to transform raw data into valuable knowledge. It is divided into distributed data, processing and programming, visualization, data analysis, and pre-processing data components.

32.7.1 Data Management

32.7.1.1 Distributed File System (DFS)

As shown in the diagram above, DFS is the lowest level of the architecture. It is used for the storage and maintains large amounts of data across multiple nodes of commodity hardware. However, it is designed in such a way that it conforms to master and slave node architecture.

32.7.1.2 Cluster Management

The components help in deploying, scheduling and orchestrating the jobs across the other parts of the architecture to ensure there is a readily available and highly scalable computing infrastructure.

32.7.1.3 Distributed Data Processing and Programming

It is used to process the big amount of batch data and millions of data in real time.

32.7.1.4 Data Store

It is used for the storage of the diverse and large number of data generated throughout the architecture. To increases the efficiency of the architecture, it is important to have a distributed, scalable, schema-free and fault-tolerant data stores.

32.7.1.5 Visualization

Visualization entails representing the massive amount of data in graphical formats for easy interpretation.

32.7.1.6 Data analysis

Data analysis is the process of deploying analytic data software to analyze the raw data and present useful, meaningful information to the end users. Therefore, an analytic model is used to infer knowledge by finding patterns and drawing conclusions with the aid of specialized tools and algorithms.

32.7.1.7 Data pre-processing

This step ensures that the data stored and used for interpretation is reliable. During this stage, the data is cleaned to eliminate noises, errors, or incomplete data to improve the overall success of data

analysis.

32.7.1.8 Governance and Security

These entails the rules laid out to ensure the data is secure. Within the big data architecture, there exist several governance policies. For instance, Apache Atlas provides a scalable and extensible set of core foundational governance services. On the other hand, Apache Ranger gives the details of how to successfully implement a secure data framework to monitor and manage the data across the system.

32.7.1.9 Data Ingestion

It helps in transferring data from outside sources to the internal systems within the architecture. Since the big data involves a large volume of data and velocity characteristics of big data, data ingestion needs to be resilient and fault-tolerant.

32.7.1.10 Application

The application layer helps present the analyzed data to the end-users. An example of application is Nutch, which is a production-ready web crawler.

In conclusion, the architecture represents a key part of the big data. For that reason, it is crucial for an educational institution to try and create a unified information architecture, to ensure it leverages all types of data. Moreover, a well-designed architecture promotes a unified vision for information management and analytics. Therefore, it is important to get a better understanding of the role of the implementation of the big data, which would provide a better context for deriving requirements for architecture.

32.8 IMPLEMENTATION

32.8.1 Factors to Consider Before Implementing the Big

Data

The big data has become a game changer in shaping the future of education. However, certain factors need to be considered before applying the big data project.

32.8.1.1 Understanding Industry Point-of-View on Big Data

The education institutions needs first to assess the capabilities and trends to sidestep mistakes made in other sectors, particularly in the business world. For instance, it is advisable that before an institution or organization implement the big data to first study reports from well-regarded publications such as IDC and Forrester's.

32.8.1.2 Developing a Proof of Concept (PoC)

It is helpful first to create a business case that showcases all the values and the costs associated with the implementations of the big data.

32.8.1.3 Understanding the Current Tools and Technology

After the institution approves of continuing with the implementation of the big data, the next step entails analyzing the available tools and technology that can help meet architectural guidelines requirements.

32.8.1.4 Developing the Measures of Successful Big Data

It is important to have a target that will measure the benefits of the implementation of the big data system.

32.8.2 Steps of Successful implementation of the Big Data

32.8.2.1 Identification of the Goals

Both in business and education world requires one first to determine the purpose of the big data. For instance, in an education set-up, the institutional management needs to clarify whether the big data is to increase the efficiency of the learning process, improve institution's marketing, or to improve the overall performance of the institution. Therefore, based on the purpose, then it is easy to choose a methodology, hire employees, and select the right sources of data.

32.8.2.2 Leverage a Proven Big Data Strategy

There exist four prove strategies, which are based on the objective of the big data as well as the availability of the data.

32.8.2.2.1 Performance Management

This approach involves applying the transactional data such as the client purchase history and their turnover. However, this approach is not suitable for an education set up.

32.8.2.2.2 Data Exploration

This strategy primarily relies on the use of data mining and research to obtain solutions to the various problems an organization might be facing. In the process, the analytic teams can identify new segments of data.

32.8.2.2.3 Social Analytics

Social analytics approach employs the non-transaction data on social media platforms such as Facebook, Twitter, and Instagram. It is an applicable method in an education setting since most of the students use such platforms as their modes of communication.

32.8.2.2.4 Decision Science

This approach relies on the analysis of non-transactional data such as online discussion forums. Therefore, it utilizes text and sentiment

analysis to determine students/customer's opinions in regards to new services and schemes.

32.8.2.3 Identify infrastructural changes

To meet the requirements of the big data, the institution may need to change some of its infrastructures. For instance, the traditional storage mediums might not facilitate the running of complex algorithms and analysis. It may also involve the reconfiguration of the hardware platform to meet the new computation needs.

32.8.2.4 Training of the Staffs

It is crucial that all the staffs working within the institutions are conversant with the big data. Therefore, at the start, it is important to hire outside consulting team to guide on how best to utilize the big data. Moreover, it is essential to hire new strong system programmers who can work in a parallel processing environment and a network communications specialist.

32.9 BENCHMARKING

Big data benchmarks are used to evaluate and compare the performance of big data systems and architecture. A successful benchmarking entails five steps: planning, generating data, generate tests, execution, and analysis and evaluation.

The process provides a realistic and accurate measure of the big data systems and therefore, it can enable one to make efficient decision-making. For instance, a successful big data support education institutions to make decisions for planning system features, tuning system configuration and validate the deployment strategies. The following are the various steps of benchmarking:

32.9.1 Planning

One needs to make a plan on how the big data benchmarking will take place. Therefore, it is important to first come up with the goals of the benchmarking process. For instance, it should be able to generate application-specific workloads and tests such that it's easier to draw meaningful conclusions. Also, in the planning tests, it is imperative to lay down the methodology to use in the benchmarking process.

32.9.2 Generating Data

There are four properties associated with big data: volume, velocity, variety, and veracity. Therefore, benchmarking should aim to cover those areas, and thus, it is crucial to apply real-world data or generates synthetic data for application-specific workloads. However, in most cases, most real-owners are not willing to share their real-data because of the security issues. Consequently, in big data benchmarks, the consensus is generated synthetic data as inputs of workloads by real data sets.

32.9.3 Generating Tests

In big-data benchmarks, one should first identify the typical workload behaviors for an application domain. Besides, it is important to take into considerations the diversity of workloads to cover different types of application domains, and at the same time to generate tests based on the workloads automatically. However, due to complexities involved in big data, it is challenging to develop big data benchmarks that reflect the real workload cases.

32.9.4 Execution

The execution stage needs to address various challenges. For instance, the big-data benchmarks need to adapt to different data formats. For example, the text data can be stored in the form of text files, web pages or pdf. The data also needs to be portable to represent different software stacks to provide a fully functional solution. Besides, the execution must reflect the user's experiences in using benchmarks.

32.10 CONCLUSION

From the discussion, it is a crucial time for the education institution to implement big data. Massive quantities of educational data can now be stored, analyzed and shared. For instance, various big-data algorithms can help track individual students in school and as a result keep detailed information about their academic performance, behavior, and educational needs. Moreover, the big-data systems can help improve educational services by assisting the teachers to analyze what students know and what techniques are most useful for each pupil. Furthermore, technologies such as data mining and data analytics can provide fast feedback to students and teachers about their academic performance.

However, irrespective of the benefits associated with the big data, several challenges need to be addressed. For instance, the massive amounts of data generated by the big-data systems require much larger storage systems. To overcome such a problem, the data specialists need databases that do not use the traditional SQL based queries. The other challenge is in regards to the analysis as data is generated in different structure and size, and the study of the data may consume a lot of time and resources. Moreover, it is challenging reporting the analyzed information since when a large amount of data are involved, the traditional reports become difficult to interpret by human beings. Therefore, it requires useful tools and techniques that can address such challenges. An example includes the analytics software which can provide an in-depth analysis of some education patterns and extract valuable knowledge from them.

33 UTILIZING MACHINE LEARNING AND VISUALIZATION TOOLS TO ANALYZE BREAST TUMOR MEASUREMENTS

FA18-523-80

Evan Beall Indiana University Bloomington, IN 46408, USA
ebbeall@iu.edu

format not followed

github: [cloud](#)

33.1 ABSTRACT

Cancer is a life altering disease that affects people all over the world. In the United States alone it is estimated that throughout a person's lifetime they have around a 39.66% chance of developing cancer and a 22.03% chance of dying from cancer at all invasive sites **???**. This disease can be classified as a human problem that no one can really escape from. With the odds of developing this disease being over one in three, nearly every person knows someone that has been diagnosed or will no someone that is diagnosed at some point in their lives. The public has high visibility on this disease and therefore lots of funding is placed on research to find a cure.

Oncology research is an ever expanding field. While there have been huge advancements in how we treat cancer, a stop to the disease has yet to be found. The funding for research into a cure has been ever expanding. It is very hard to get an accurate estimate of how much is spent on cancer research each year due to the amount of players in the industry. This industry spans across all nations and is being researched in all corners of the global. Funding for cancer research comes from governments, foundations, pharmaceutical companies, private investors, and much more. The fact that there are so many different parties involved across the world make it impossible to come up with an actual figure of the cost of this research. In the

United States, there is branch known as the National Cancer Institute that helps to organize governmentally funded research into Cancer. The current budget that has been allocated to the NCI for the 2019 fiscal year is \$5.74 billion ????. This budget represents an increase of 79 million dollars from 2018.

In the past, cancer treatment and research has been mainly focused on a casting a wide net by attempting to treat all individuals with a generalized treatment regimen. The focus of most of the large pharmaceutical companies that are involved in the creation and research of the main cancer fighting drugs has been on a one drug fits all methodology. In recent years this mindset has been beginning to change. Many of the large pharmaceutical companies are moving to a more tailored approach in combating cancer. This involves utilizing data analysis in tailoring treatments and analysis of data to predict outcomes of future trials. To further explore this idea the present study explores fine needle aspirate data from breast cancer research performed at University of Wisconsin. This study will utilize machine learning techniques to analyze and create visual representations of this data set.

33.2 KEYWORDS

Breast Cancer, Python, big data, random forest

33.3 INTRODUCTION

Cancer is one of the most devastating conditions financially, emotionally, and physically to the person who is diagnosed and those around them. In the United States alone “the Agency for Healthcare research and Quality estimates that the direct medical costs (total of all health care costs) for cancer in the US in 2015 were \$80.2 billion.” This figure includes all health care costs that are associated with cancer. There are several aspects that go into the total cost of health care for cancer patients, but one of the main ones is the cost of cancer treatment.

Breast cancer has high visibility in the public eye and has some of the highest funding of any other types of cancer out there. Several large pharmaceutical companies and researchers get large grants to research new compounds and molecules to fight off these diseases. The oncology pharmaceutical industry is over a trillion dollar industry and each year there are thousands of clinical trials that are ongoing.

Oncology clinical trials function by exposing patient populations to new types of therapies and recording the results. Clinical trials are set up to collect data throughout the time that the patient is on trial. Some clinical trials are set up to run for decades at a time with hundreds of visits per patient. These trials can include tens of thousands of patients. All of this together means that there is an enormous amount of data collected during each of these trials. This data is stored in electronic data capture systems or EDCs. The amount of data that needs to be worked on and analyzed for each of these trials means that there is a need for better ways to accomplish this task.

This report will delve into the possible implementation of machine learning techniques to analyze one aspect of an Oncology clinical trial. The goal will be to determine how accurate the algorithm can predict tumor diagnosis in breast cancer patients. This report will also run further analysis of other tumor characteristics that are present in the dataset. If an algorithm like this can be proven to be accurate enough, its implementation in clinical trial research and oncology treatment in general could be life saving. It could also help to eliminate the human error that is seen when a physician alone is making a diagnosis.

33.3.1 Dataset

The dataset used in this analysis is the Breast Cancer Wisconsin Data set. This dataset was created by Dr. William H. Wolberg. Dr. Wolberg is a physician at the University of Wisconsin Hospital. The dataset utilized for this analysis is a collection of fine needle aspirate breast cancer tumor samples. Fine needle aspirate biopsies are biopsies that

are taken by inserting a fine needle into the affected site. The extracted tissue is then suspended into an aqueous solution and mounted on a slide. Dr. Wolberg then used a graphic program called Xcyt to analyze the features of each of these biopsies.

This dataset includes 569 unique patients with 12 attributes per patient. The attributes provided by this dataset includes de-identified patient identifiers, pathological diagnosis, and tumor characteristics. The specific characteristics included are clump thickness, uniformity of cell size, uniformity of cell shape, marginal adhesion, single epithelial cell size, bare nuclei, bland chromatin, normal nucleoli, mitosis, and class. Each of these characteristics are assigned a numerical value from 1 to 10 except for the "Class" characteristic. The Class characteristics contain the results for malignant vs benign, these are assigned values 2 or 4 respectively. The characteristics that are included in the data are of the cell nuclei that was present in the digitized images. The dataset is publicly available for analysis.

33.3.2 Xcyt

Xcyt is a program that has three parts. The initial portion of the program involves an operator selecting nuclei from FNA biopsy specimen by drawing rough lines around each nuclei in question. The program then applies a curve fitting program to determine the exact boundaries of each cell nuclei that was identified by the operator. This program then computes ten unique characteristics from the specimen nuclei. For each of these the mean, extreme value, and standard error are output. As a result, each patient that is analyzed by Xcyt has a 30-dimensional vector created. The data analyzed by his team included both this 30-dimensional vector and another dataset that included the overall nuclei characteristics. The second portion of this dataset is the data that is included within the Breast Cancer Wisconsin Data set that will be utilized for this analysis.

The next part of the Xcyt program is the diagnosis portion. Xcyt utilizes a linear programming-based diagnostic method to determine if the suspected malignancy contains benign or malignant cells based

off of the characteristic generated by the initial model. The system used by Xcyt is a variant of the multisurface method called the multisurface method-tree. This diagnostic method creates strict planes based on malignant or non-malignant based on selected characteristics from the initial part. In the initial analysis the most accurate output occurred when extreme area, extreme smoothness, and mean texture were plotted. These resulted in nearly 97.5% accuracy in identifying malignant vs benign tumors.

The final part of the Xcyt program involves prediction of recurrence of the tumor. This portion utilizes a kernel density estimation to compare the amount of benign vs malignant nuclei that are found within a sample. By comparing these two figure's densities a bayesian computation is applied to compare the height of malignant densities divided by the sum of the densities of malignant and benign nuclei.

33.4 IMPLEMENTATION

33.4.1 Requirements

- Python 3.6 or higher
 - This instance was performed in the Spyder IDE
- Python Packages
 - Python Standard Library
 - Pandas
 - Numpy
 - Matplotlib
- Dataset downloaded in .csv file format

While Dr. Wolberg and his team used the novel program of Xcyt to perform their machine learning algorithm on a more complex dataset, this report will instead use the popular k-mean clustering algorithm on a more high level dataset within the Spyder Python 3.6 environment. Once this environment has been downloaded and the dataset has been download in .csv format. The next step would be to import the pandas, numpy, and matplotlib packages will need to be imported into the code. This will be accomplished using the following

code:

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from pandas import DataFrame, read_csv
from matplotlib.backends.backend_pdf import PdfPages
```

The dataset that is in the working directory can now read using the `pandas.read_csv()` and placed in a dataframe. By utilizing this simplistic method it allows this program to be easily reproduceable and deployable by researchers that are not well versed in machine learning methodology. This would also allow for researchers to shape the analysis to their own data and create customizations to the base program. The 16 missing values in the dataset are also addressed by utilizing the `fillna()` method assigning the value A7 for all missing values via the following code:

```
df = pd.read_csv('breastcancer.csv', na_values = '?')
df = df.fillna(df['A7'].median())
```

This analysis will be split into several different parts. The first part will include a statistical output for each of the characteristics that have been derived by the dataset. The frequency of each of these characterisitcs will then be plotted on basic bar graphs. Further visualization is also derived to include characteristics that might be relevant to researchers. Some of the derived statistical features will include mean, median, and standard deviation of each of the characteristics. The following is a sample of code that shows how the visualizations were created:

```
with PdfPages('finalproject.pdf') as pp:
    for i in range (9):
        fig1 = plt.figure()
        sp1 = fig1.add_subplot(1, 1, 1)
        sp1.set_title(columns[counter])
        sp1.set_xlabel("Results")
        sp1.set_ylabel("Number of Cases")
        plt.xticks((1,2,3,4,5,6,7,8,9,10))
        sp1.hist(df[(column_header[counter])], bins=10, color="blue", edgecolor =
"black", alpha=0.5, label = [1,2,3,4,5,6,7,8,9,10])
        pp.savefig(fig1)
        counter = counter + 1
    pp.close()
```

In the second part of the analysis, a basic k means algorithm will be

created and implemented on the data set. The code will select two data points randomly using the `np.random()` method as the initial ‘means’ of the dataset these can also be seen as the initial cluster points. These clusters will be utilized for each datapoint to calculate the euclidian distance. This dataset will continue to iterate in this way by selecting randomly selected datapoints as new cluster points each time. Each time the final cluster assignment will be compared to the one before this. The condition for the system to stop this iteration will occur when the system identifies no change in cluster assignment or the system has iterated 1500 times. This is put in place to keep the cluster method from running continuously.

The final portion of this will compare how successful the k means algorithm we have implemented predicted the diagnosis condition. In this step we take the cluster assignment and compare it to the final column included in this dataset that contains the actual diagnosis of the patient. By doing this we are able to calculate the error rate for each cluster and the overall error rate for both clusters. This will give us a real world definition of how accurately the k mean algorithm is functioning.

33.5 IMPLEMENTATION

While Dr. Wolberg and his team used the novel program of Xyct to perform their machine learning algorithm, this report will instead use the popular k mean clustering algorithm that is used in many machine learning activities. To perform our analysis and visualization we will utilize a Python 3 environment. Within the python environment we will be utilizing numpy, pandas, and matplotlib libraries to load our dataset, perform analysis, and create visual representations of the data.

This dataset provides us with a large amount of data on different characteristics of each tumor. This dataset also provides us with a diagnosis of the individual with breast cancer. By using both the characteristics of the analyzed tumor along with the patients diagnosis we might be able to look for tumor characteristics that are

be indicative of a cancer diagnosis. This would be able to inform future clinical trial research in how to identify patients with metastatic versus benign tumor diagnosis. This could also allow for specialized treatment of specific type of breast cancer lesions. Depending on tumor density, radius, etc. might indicate that cancer was more likely to metastasize more quickly.

33.6 RESULTS

Applying the k-means algorithm to this data set happened in several different steps. The following analysis will be split into those different steps to breakdown how the final results were achieved.

33.6.1 Visualizaton

In the first section of the code the data is read from a .csv file using the pandas read_csv() method. This data was then inserted into a dataframe. Once here the mean, median, standard deviation, and variance were all calculated per characteristic. An example of these results can be seen below:

Each characteristic in this dataset is assigned a value of 1-10. Using these variable several visualizations were created to allow for the ressearch scientists and physicians to have an idea of how the cells in their research tumors look. This can be utilized in future studies to account for variances in tumor malignancies accross cancer diagnosis. An example of these visualizations can be seen below.

33.6.2 K-Means Clustering

The next part of this analysis included the creation and recalculation steps for the k-means clustering steps. During this step cluster points were chosen as random data points within the dataset. The eulicidan distance was then calculated from each of these clusters to determine a malignancy diagnosis. Upon either reaching 1500 iterations or the cluster assignment not changeing across iterations the clustering would complete. The system would then produce the

calculated k means for each characteristic in the study. The following table shows the calculated k-means for each characteristic depending on their malignancy diagnosis.

33.6.3 Clustering Assessment

The final portion of the study compares the results of the k-means clustering algorithm with that of the correct diagnosis that was provided in the initial dataset. This portion of the study calculates the error rate for both the malignant and benign clusters. This then goes on to calculate the total error rate across both clusters. The following table identifies the total amount of datapoints in each cluster and the error rates that were calculated.

33.6.4 Benchmarks

In Dr. Wolbergs experiments they were utilizing a 30-dimensional vector for each data point in this data set. The deployment used in this study only utilizes a 10-dimensional vector for each analysis. In this experiment it takes around 9.65 seconds for the analysis to import the data, read the data, sort the data into initial clusters, iterate over the dataset to ensure cluster assignments are correct and then run a comparison between estimated cluster assignments and the real cluster assignment. The only values that could be found from Dr. Wolberg's implementation of the Xcyt program were the length of time that the system would take to generate the characteristic map from the FNA imaging. This was estimated to take around five to ten minutes. This however is not a comparable time stamp as the system was estimating the characteristics and not running the diagnostic report.

33.7 LIMITATIONS

Limitations of this type of analysis are based around several factors. The first and biggest one is the necessity to utilize the initial portion of the Xcyt program to generate a characteristic map of FNA samples.

This program is currently accessible for free via Nick Street at University of Iowa. These packages have been outdated for many years, however, Broad Institute has maintained the packages for implementation in R. To utilize these datasets in the future some manipulations to port the datasets generated via R to python for machine learning would be required. Without this information, the machine learning algorithm implemented in this report will not run. The assumption is that the initial portion of the Xcyt program output could be utilized going forward and combined with the easy to implement and customize python algorithm that is displayed in the report above.

Another limitation of this type of algorithm is its ability to apply to a variation of cancer types. FNA biopsies are not able to be produced on cancers that are not solid tissue samples. This limits the effectiveness of this analysis as it only focuses on a type of biopsy only available to a selection of cancer tissues. While this is a limitation, implementing this algorithm for other solid tumor cancers could be extremely helpful for diagnostic purposes.

33.8 CONCLUSION

Dr. Wolberg team's implementation of the novel Xcyt program to diagnose malignancies in breast cancer patients is a revolutionary way that machine learning could be implemented in cancer research. This type of methodology was implemented on a very specific instance of cancer research. The machine learning algorithm was able to identify tumor nuclei characteristics that indicated malignant tumor cells at a rate of 97.5%. However, this type of machine learning program is extremely specific and not very reproduceable. This program utilized an analysis method that created a 30-dimensional vector for each patient. The team was then able to run analysis on each comparison point to find the highest rate of correct results. Finding that very specific combinations of characteristics would lead to higher rates of success. In the end they discovered that the specific combination of extreme area, extreme smoothness, and mean texture gave the highest success rate. This is a time consuming way to

run analysis as it took the algorithm nearly 5-10 minutes to create the 30-dimensional vector for each sample. On top of this, the researchers would need to then use trial and error to discover which combination of these vectors would give the highest success rate.

By cutting this down to only the high level data assessment with fewer characteristics to analyze the process on the front and back end can be sped up significantly. The Xcyt program implemented by Dr. Wolberg's team utilized very specific characteristics that may or may not apply across other cancer diagnosis. However, by utilizing the higher level dataset with only 10 points of comparison we could make this method more scalable and applicable across all oncological diagnosis. The k-means clustering algorithm that was implemented in this report showed around a 96% rate of success. This is only 1.5% lower than that seen by Dr. Wolberg's team.

By lowering the amount of complexity of assessment, we have increased the reproducibility across other cancer diagnosis and decreased the time investment for researchers to run analysis. By weighing the fact that the decrease in efficacy is only 1.5% we could allow researchers to make a judgement call as to if this decrease in efficacy is worth the increase in efficiency. This tool should not be used as the sole method for diagnosis, but should be used in combination with physician determination in combination with other diagnostic tools. That being said, making this type of analysis more easily reproduced and available to all researchers would more than make up the 1.5% loss in efficiency. Python tools are much more approachable and easier to deploy than the proprietary program Xcyt that was created by Dr. Wolberg's team. Since Python is an open sourced tool and each package that was used in this analysis is also available to everyone. This means this method is easily accessible to all researchers willing to take the time to set up analysis points for their teams.

34 ANALYSIS OF GLOBAL COMMODITY STATISTICS USING PYSPARK AND WATSON ANALYTICS FA18-523-81, FA18-523-79, FA18-523-82

format not followed

github: [◆](#)

34.1 ABSTRACT

Economists, governments, trading organizations and even general public may be curious to know about the international trading patterns between different nations across the globe on various commodities. For example, one may be curious to get answers to some interesting questions like which is the largest importer of steel in the world? Which country has the highest growth in sheep production? What is America's chocolate consumption this year? These kinds of questions can be answered by analyzing the huge volume of data that is collected from the trading transactions that happens globally between various countries. The idea is to do exploratory data analysis to find patterns and insights that may be helpful in answering such questions or raise more such new questions from observations.

34.2 KEYWORDS

fa18-523-81, fa18-523-79, fa18-523-82, Exploratory Data Analysis, Python, PySpark, HDFS, HBase, Watson Analytics

34.3 INTRODUCTION

This dataset contains import and export volumes for 5,000 commodities across most of the countries on the globe over the past 30 years. Trade across the globe plays a prominent role in the

economy of a nation. We are looking ahead to check for any peculiar patterns in the commodities traded by different countries in accordance with time in the past 30 years. We have imported the dataset from the UNData site published by the United Nations Statistics Division. The dataset is a 1.2 GB sized dataset and we aim to illustrate meaningful correlations among different volumes of commodities traded by different countries in different years.

34.4 DATA

The dataset was provided by United Nations Statistics division on the UNData website. It consists of data for over 30 years until 2016 covering all nations in the world and around 5000 plus commodities. It has over 3.5 million transactions/records with 10 columns. The total size of the dataset was 1.2 Giga bytes. The 10 variables names are: country, year, commodity code, commodity name/type, flow (import or export), trade in USD, weight in kg, quantity name, quantity and category. There are some missing values as well as outliers in the data. Hence, the data is dirty and needs good amount of cleaning and preprocessing.

34.5 RELATED WORK

We have found an earlier work on this dataset, that focused on specific goals about finding the trading patterns for a nation. It has taken only a sample of this dataset than whole data and was more focused on specific countries.

34.6 VISUALIZATIONS

We will look at different kinds of visualizations like boxplots, frequency distributions, histograms, bar plots, density plots, etc. of each variable/feature depending on if it is categorical or numerical variable. We will also look at heat maps on correlation matrix between the variables. We may also use geographical plots to visualize the trade volume of each country on map. We shall be using

python packages for visualization like Matplotlib, Seaborn, Plotly, Geographical plots majorly. We may also be using d3.js for interactive web-based visualizations.

34.7 TECHNOLOGIES TO BE USED

We will be programming in Python 3, which is an Object Oriented Programming language. It has got excellent tools and packages for Data analysis, visualization, machine learning, deep learning. We will be using Numpy, Pandas, Matplotlib, Seaborn, PySpark, SQLAlchemy etc. like packages that are required for our exploratory data analysis.

34.8 SUMMARY

34.9 FUTURE WORK

Though in this project we will be focusing more on big data storage and retrieval, data preprocessing and exploratory data analysis, there is scope to do more work on this by building predictive models using various machine learning, deep learning or forecasting models, depending on the problem statement or goal we set.

34.10 ACKNOWLEDGEMENT

We would like to thank the United Nations Statistics Division for kindly publishing this dataset on UNData site. We would also like to thank the professor Dr. Gregor von Laszewski for his valuable inputs in helping us to choose this dataset and also for all the guidance he has provided. We would like to appreciate the effort of the associate instructors for their timely help on Piazza.

35 CREDIT SCORING ALGORITHM AND ITS IMPLEMENTATION IN PRODUCTION ENVIRONMENT FA18-523-83

Nhi Tran
nytran@iu.edu
Indiana University
hid: fa18-523-83
github: [cloud](#)
code: [cloud](#)

35.1 ABSTRACT

A credit scoring algorithm is essential to help any bank determine whether to authorize a loan to consumers. Most of the decisions require fast results and high accuracy in order to improve the bank customer satisfaction and profit. Choosing a correct machine learning algorithm will result in high accuracy in the prediction and a reasonable implementation of the algorithm will result in the fast service a bank can provide to its customers.

35.2 KEYWORDS

fa18-523-83, machine learning, predict algorithm, classification, devops, optimization, API

35.3 INTRODUCTION

For every machine learning problem, there are normally two main areas that everyone focuses on: which machine learning algorithms to use and how to implement and integrate the machine learning code into existing production infrastructure.

Most of the time, machines make predictions by learning and observing the data patterns from previously existing data with known

results. Once the training is over, the machine learning code can then be applied to new data and predict the unknown results by applying the trained patterns and algorithms.

The next thing to do after prediction would be determining how to retrieve and apply the result of the prediction into a new or existing production application and how to ensure the continuous deployment into the production environment without resulting deployment code defects.

The business problem that will be focused on is how to determine in real-time whether or not a customer will be experiencing financial distress in the next two years. By predicting the business problem, banking companies can use the results as part of their business rules to decide whether to approve their products to the customers.

35.4 DESIGN

35.4.1 Dataset

We are utilizing an existing dataset from the 'Give Me Some Credit' competition on Kaggle to train and test algorithms and determine which algorithms would be the best to predict the probability of someone experiencing financial distress in the next two years [154].

The Kaggle competition contains a training set, a test set, and a data dictionary. The training set contains 150,000 records of previous customer data with an existing label indicating whether or not each customer had serious bank delinquency within two years. The test set contains about 100,000 records without any label data, which will not be part of the analysis but will be used as part of the benchmarking report.

Data descriptions [155]:

- **SeriousDlqin2yrs:** label data, contains 'Yes' or 'No' indicator
- **RevolvingUtilizationOfUnsecuredLines:** total balance of

- unsecured lines such as credit cards and personal lines
- **age:** bank customers' age
 - **NumberOfTime30-59DaysPastDueNotWorse:** number of times each customer has been 30-59 days past due but no worse in the last 2 years
 - **DebtRatio:** monthly debt payments divided by monthly gross income
 - **MonthlyIncome:** bank customers' monthly income
 - **NumberOfOpenCreditLinesAndLoans:** number of open loans and lines of credit
 - **NumberOfTimes90DaysLate:** number of times each customer has been 90 days or more past due
 - **NumberRealEstateLoansOrLines:** number of mortgage and real estate loans
 - **NumberOfTime60-89DaysPastDueNotWorse:** number of times each customer has been 60-89 days past due but no worse in the last 2 years
 - **NumberOfDependents:** number of dependents in family excluding themselves (spouse, children etc.)

35.4.2 Data Visualization

Using the `describe()` function in Python pandas package, the statistics for each attribute within training dataset is populated. The statistics include count, mean, standard deviation, minimum, quantiles, and maximum. The function helps with seeing outliers and identifies values or columns that need to be cleaned up. For example, in the training set, some areas that will need to be carefully examined are age with a minimum value of 0, MonthlyIncome and NumberOfDependents contains NaN value in their quantiles.

| | age | MonthlyIncome | NumberOfDependents |
|-------|----------|---------------|--------------------|
| count | 150000.0 | 120269.0 | 146076.0 |
| mean | 52.29520 | 6670.22123739 | 0.7572222678605657 |
| std | 14.77186 | 14384.6742152 | 1.1150860714872997 |

| | | | |
|-----|-------|-----------|------|
| min | 0.0 | 0.0 | 0.0 |
| 25% | 41.0 | NaN | NaN |
| 50% | 52.0 | NaN | NaN |
| 75% | 63.0 | NaN | NaN |
| max | 109.0 | 3008750.0 | 20.0 |

Data Visualization in Python can be done using graphing packages such as matplotlib, seaborn, etc.

Using matplotlib, Figure [146](#) shows that there is a small count of 0 value as outliers and the distribution without those outliers will be a right-skewed distribution. Therefore, it is better to replace those outliers with the median value of the distribution.

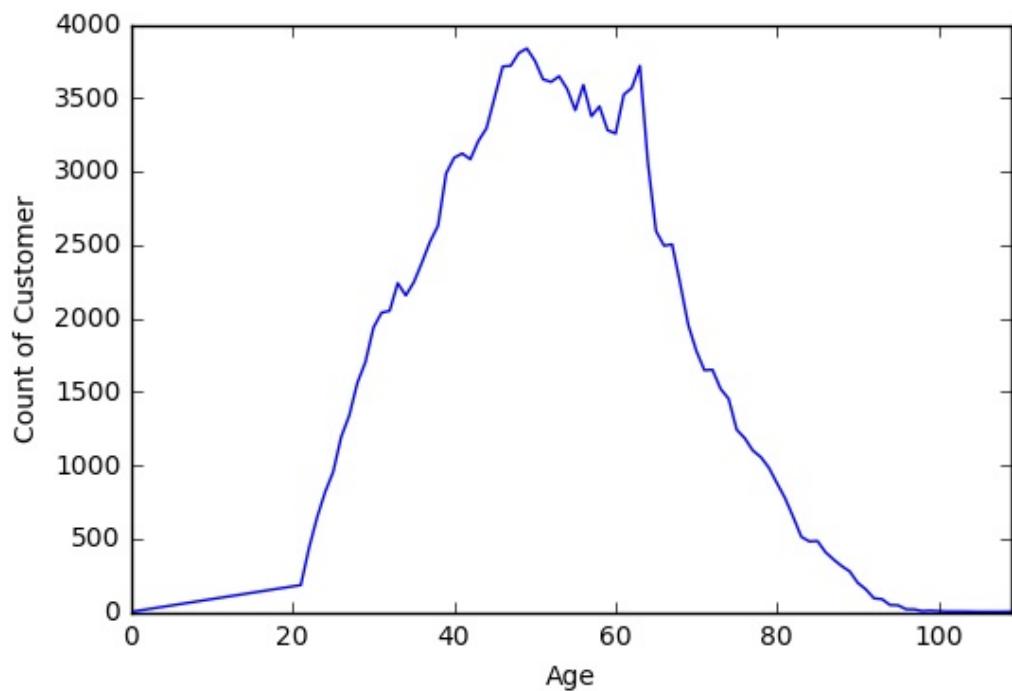


Figure 146: Count of Customer by Age

Figure [147](#) shows overall distribution of the count of customers that experienced past due, the count of customers that have open credit and real estate lines, and the count of customers that have

dependents other than themselves. Most of the distributions are right-skewed, the majority of them do not have any past due or dependents.

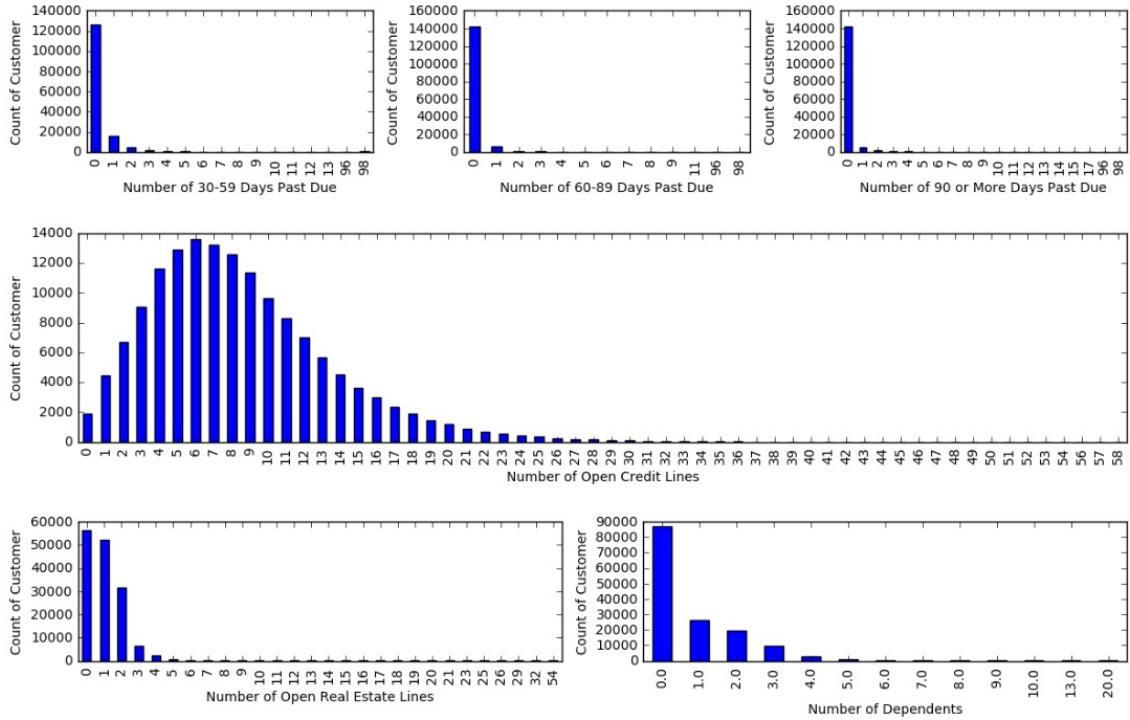


Figure 147: Count of Customer by Multiple Attributes

Figure 148 shows the distribution of the label that will be what the machines are trying to predict. If the label distribution is not even, the model might overfit and give a higher chance of predicting the label that has a higher population. In this case, the amount of customers that had no delinquency is 14 times more than the number of customers that had delinquency. This issue will need to be handled during the Data Cleaning or Model Training process to avoid overfitting.

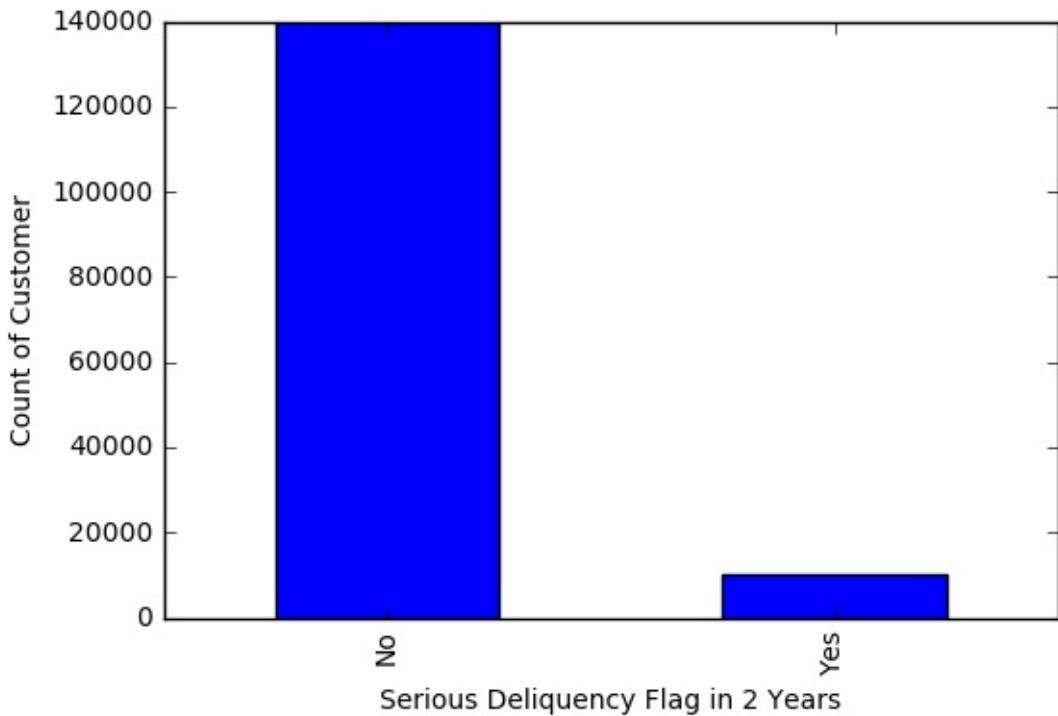


Figure 148: Label Distribution

35.4.3 Data Preparation

From the Data Visualization step, the first basic data preparation is replacing missing value with median value. Another way to handle this issue is to drop records that have missing data, however, due to low data volumes and mbalanced class data distribution, it is better to replace missing data instead.

Using seaborn, heatmap can be used to visualize the correlation of all attributes. Correlation is a measurement of the strength of association between two variables and the relationship's direction [156]. Correlation is an important indicator in the Feature Selection process to help determine which attributes should be used as part of the training set and which attributes should be irrelevant. There are multiple correlation methods to calculate the correlation coefficient, the method that is used for this training set is called Spearman. Figure [149](#) shows the first observation of the correlation between all variables.

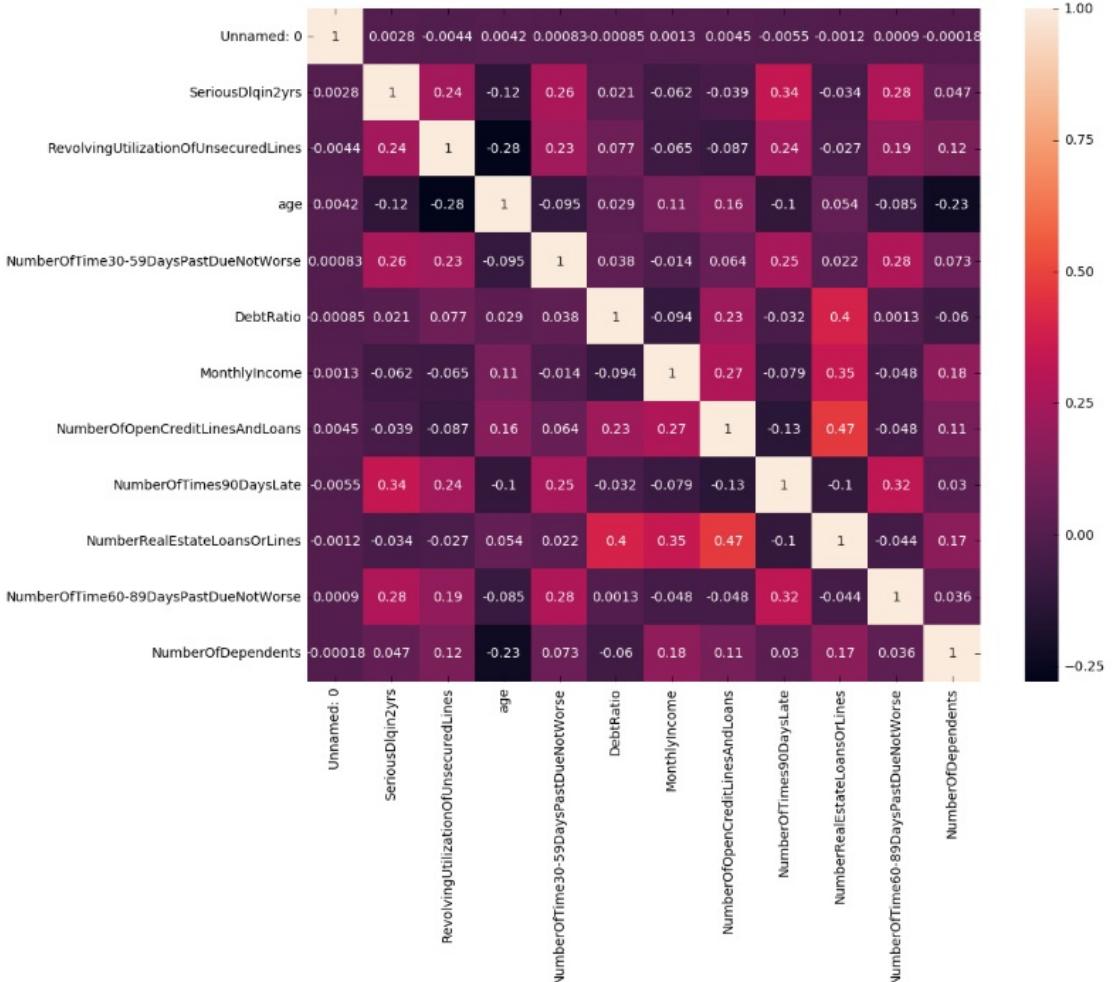


Figure 149: Correlation First Run

From Figure 149, `DebtRatio` has a very low correlation to the class label. It means that the debt ratio of a customer does not impact whether a customer will be likely to default in the next 2 years. At the same time, the `DebtRatio` variable is highly correlated with the number of open credit lines and real estate loans, which could potentially interrupt the training algorithms and result in a false prediction. Therefore, it is better to exclude `DebtRatio` out of the final dataset.

Another observation from +fig:correlation1 is that all number of past due variables have a high impact on the class label and between themselves at the same time. Since they are all past due type of data, adding them all into a new variable called `TotalNumberOfPastDue`

and dropping all individual past due variables could be a good idea to avoid conflict between those variables.

Similar to past due variables, NumberOfOpenCreditLinesAndLoans and NumberOfRealEstatesLoansOrLines are highly correlated to each other, therefore, adding them together into a new variable called TotalNumberOfOpenLines is a way to solve the problem.

Once all of the new variables are added and individual variables are removed, it is good to run the heatmap again to determine whether or not there is more cleaning to be done. Figure 150 is the second heatmap run with all prepared variables.



Figure 150: Correlation Second Run

The final observation is that MonthlyIncome and the new variables

TotalNumberOfOpenLines are also highly correlated to each other. Since MonthlyIncome has a higher correlation than TotalNumberOfOpenLines, the last step to have a final training dataset is to drop TotalNumberOfOpenLines.

35.4.4 Model Training

Because the training data has imbalanced classes, there are multiple ways to handle this issue:

Methods that will be used [157]:

- **Changing Performance Metric:** instead of using only Accuracy as the main performance metric to evaluate models, try to include other metrics such as Confusion Matrix, Precision, Recall, F1 Score, Kappa, ROC Curves.
- **Resample Dataset:** add copied of instances from low volume class as an over-sampling method, or delete instances from high volume class as an under-sampling method, or re-run the algorithm on shuffled data (k-fold cross-validation). Under-sampling and cross-validation are used as part of the training process for this problem and will be included as part of the result evaluation section.
- **Different Algorithms:** run data on multiple algorithms and evaluate and choose the best algorithms that fit the provided data.
- **Penalized Models:** Use penalized methods to cause the model to pay more attention to the minority class to evaluate.

Methods that will not be used [157]:

- **Increase Training Data:** gaining more data to provide more balance and insights in data.
- **Generate Synthetic Samples:** use popular algorithms such

as Synthetic Minority Over-sampling Technique to generate synthetic data.

The goal is to determine whether someone will experience financial distress in the next two years, therefore, there will only be two values in the label. With a binary classification problem on supervised data, it is best to use classification algorithms such as Random Forest, Logistic regression, XGBoost, Support Vector Machine, Neural Network, and Support Vector Machine.

- **Random Forest:** an ensemble of Decision Tree algorithm, builds and merge multiple decision trees together to get average results for prediction [158] . Using Python, the algorithm can be used from RandomForest() function in sklearn package.
- **Logistic Regression:** uses an equation with weights for coefficient values of input values to make a prediction. For Binary Logistic Regression, a threshold between 0 and 1 is needed to determine the category of the prediction [159]. This algorithm is LogisticRegression() function in sklearn package.
- **XGBoost:** xgboost is

“a scalable and accurate implementation of gradient boosting machines and it has proven to push the limits of computing power for boosted trees algorithms as it was built and developed for the sole purpose of model performance and computational speed” [160].

There is an XGBoost API that can be called via sklearn using function XGBClassifier.

- **Neural Network:** a neural network is a machine learning algorithm that is inspired by the human brain. Neural network identifies the pattern of input data through multiple layers of artificial neural layers [161]. The function

MLPClassifier from sklearn is one of the Neural Network algorithms for classification problem.

- **Support Vector Machine:** an algorithm that finds patterns and predicts output data by finding hyperplane(s) in an N-dimensional space fa18-523-83-www-svm]. The function LinearSVC from sklearn is one of the Support Vector Machine algorithms for classification problem.

35.4.5 Result Comparison

There are two sets of run that were evaluated in the provided dataset: normal run and balanced run. On a normal run, due to imbalanced data distribution, it is important to compare metrics for algorithms with defaulted parameters and algorithms with penalized configured parameters.

In order to evaluate and determine which algorithms to use, it is important to understand what each metric mean and what the expectation of each metric is. In the evaluation, there are five metrics that will be used to compare between models [162]:

- **Classification Accuracy:** the ratio between correct prediction over the total sample sizes. It only works well if we have a balanced class label distribution.
- **Area Under Curve (AUC):** often used for binary classification problem. AUC measures the area under the curve of False Positive Rate vs True Positive Rate plot. AUC ranges from 0 to 1 and a higher AUC indicates a better-performed model.
- **Precision:** the number of correct positive results divided by the number of positive results predicted. Precision indicates how many instances that were predicted correctly.
- **Recall:** the number of correct positive results divided by the number of records that should have been identified as positive. Recall indicates how much data was misclassified.

- **F1 Score:** the F1 score is the balance between precision and recall, a better model gives a higher F1 score.

The metrics for all models are being performed by `cross_validation()` function in sklearn with a parameter of `cv=10` to allow the data to be reshuffled and re-validated ten times. It is to ensure a high-quality result and optimize the data volume. The results are the average of all ten runs.

| | LR | LR PnLzd | RF | LSVC | LSVC PnLzd | XGBoost | XGE Pr |
|-----------------|-----------|---------------------|-----------|-------------|-----------------------|----------------|-------------------|
| fit_time | 0.64 | 0.43 | 2.50 | 32.10 | 31.87 | 5.34 | 5.10 |
| score_time | 0.03 | 0.02 | 0.26 | 0.02 | 0.02 | 0.35 | 0.30 |
| test_f1 | 0.00 | 0.00 | 0.11 | 0.01 | 0.01 | 0.03 | 0.00 |
| train_f1 | 0.00 | 0.00 | 0.86 | 0.01 | 0.01 | 0.03 | 0.00 |
| test_accuracy | 0.93 | 0.93 | 0.93 | 0.87 | 0.93 | 0.93 | 0.93 |
| train_accuracy | 0.93 | 0.93 | 0.98 | 0.87 | 0.93 | 0.93 | 0.93 |
| test_recall | 0.00 | 0.00 | 0.07 | 0.06 | 0.00 | 0.01 | 0.00 |
| train_recall | 0.00 | 0.00 | 0.77 | 0.06 | 0.00 | 0.02 | 0.00 |
| test_precision | 0.00 | 0.00 | 0.27 | 0.06 | 0.07 | 0.49 | 0.20 |
| train_precision | 0.00 | 0.00 | 0.99 | 0.04 | 0.05 | 0.60 | 0.25 |
| test_roc_auc | 0.64 | 0.58 | 0.68 | 0.59 | 0.58 | 0.80 | 0.79 |
| train_roc_auc | 0.64 | 0.58 | 1.00 | 0.58 | 0.58 | 0.80 | 0.80 |

Accuracy will not be a relevant metric to determine the performance of each model. It is included to emphasize a point that a model can provide a really high accuracy but can still perform poorly. In term of F1 Score, Random Forest seems to perform well comparing to the rest of the models. In term of AUC, Random Forest has a high AUC value in training dataset and a lower AUC value in test dataset whereas XGBoost performed consistently in both train and test run.

The same algorithms will be applied to the balanced dataset. Sklearn has a function that allows data to be shuffled with a specification of how many samples are needed for each label to ensure a result of the balanced-label dataset. This evaluation will not require penalized algorithms.

| | LR | Random Forest | XGBoost | MLPClassifier | Linear SVC |
|-----------------|-----------|----------------------|----------------|----------------------|-------------------|
| fit_time | 0.16 | 0.34 | 0.66 | 1.61 | 1.59 |
| score_time | 0.02 | 0.06 | 0.06 | 0.02 | 0.01 |
| test_f1 | 0.61 | 0.66 | 0.72 | 0.51 | 0.26 |
| train_f1 | 0.61 | 0.97 | 0.73 | 0.51 | 0.26 |
| test_accuracy | 0.60 | 0.68 | 0.73 | 0.55 | 0.50 |
| train_accuracy | 0.60 | 0.97 | 0.73 | 0.55 | 0.50 |
| test_recall | 0.61 | 0.63 | 0.72 | 0.58 | 0.36 |
| train_recall | 0.61 | 0.96 | 0.73 | 0.59 | 0.36 |
| test_precision | 0.60 | 0.70 | 0.73 | 0.59 | 0.40 |
| train_precision | 0.60 | 0.98 | 0.74 | 0.60 | 0.47 |
| test_roc_auc | 0.64 | 0.74 | 0.79 | 0.59 | 0.55 |
| train_roc_auc | 0.64 | 1.00 | 0.80 | 0.60 | 0.56 |

Accuracy is now an important factor to determine how well a model performed. In this case, XGBoost has the highest accuracy out of all models as well as a high and consistent AUC value. Random Forest also returns a great result but seems to be inconsistent due to its higher result in train data comparing to test data.

After both comparisons, XGBoost is the model that yields high and consistent performance. The XGBoost model on balanced dataset will be saved and used in the credit score application to predict the banking credit problem we are trying to solve.

35.5 IMPLEMENTATION

35.5.1 Technologies Used

35.5.1.1 Versions

35.5.1.2 Kaggle API

Kaggle API provides the ability to pull and push data from Kaggle website using command tool implemented in Python 3. The Kaggle API GitHub page provide detailed Installation, API credentials and commands instruction [163].

35.5.1.3 Python and Python Packages

- **click**: allows arguments to be passed to python code.
- **sklearn**: contains off-the-shelf machine learning algorithms packages with evaluation/report functions to cross-validation, shuffle data, and report metrics
- **xgboost**: contains xgboost algorithm
- **pandas**: allows easy-to-use abilities to read, process, slice, write and store data via dataframe and pickle
- **flask**: allows the ability to run and process APIs via web services via Python

35.5.1.4 Flask API

Flask API is a Python package that allows RESTful web service from Python.

35.5.1.5 Docker

Docker creates containers that are standalone images that bundle all dependent configuration files, packages, tools, libraries, and applications. A user can run any application through images without having to worry about the rest of the dependent components.

35.5.1.6 AWS EC2

AWS EC2 provides cloud services to host server, run and deploy the code.

35.5.2 Prerequisites

In order to run the code and reproduce the run, the following prerequisites need to be met:

- **Ubuntu 18.04 Bionic Beaver or 18.10 Cosmic Cuttlefish Server:** all codes were tested on Ubuntu 18.04 and 18.10 Server
- **Kaggle Account:** a Kaggle account is required to pull data from Kaggle.
- **Kaggle API Credentials File:** after the Kaggle account is created, go to the Account tab of user profile and select Create API Token to generate and download `kaggle.json` and save as `kaggle.json` at `~/.kaggle` directory and set the permission to 600 [163].
- **Make:** ensure that make is installed. If not, use the following command to install make:

```
sudo apt-get install make
```

This will allow the make command from Makefile to be run. The rest of the prerequisites packages and software can be run using make command.

- **Project's Git Command:** install git command by running the

following command:

```
sudo apt-get install git-core
```

- **Project's Github Repository Cloned:** ensure all project is cloned from GitHub using the following command:

```
sudo git clone https://github.com/cloudmesh-community/fa18-523-83.git
```

- **AWS Account :** an AWS account is required to be able to launch a cloud server instance for deployment and benchmarking results. The AWS account can be created via the AWS EC2 page [???](#). Credit card information is required during registration but the server instance can be launched for free.
- **Setting up EC2 Server Instance:** once the AWS is created, AWS EC2 instances can be created. Once logged in, the user can Launce Instance with the following configurations [???](#):

- Step 1: Choose an Amazon Machine Image (AMI) Cancel and Exit: select Ubuntu Server 18.04 LTS (HVM), SSD Volume Type
- Step 2: Choose an Instance Type: default to free tier option
- Step 3: Configure Instance Details: use all default options
- Step 4: Add Storage: use all default options
- Step 5: Add Tags: use all default options
- Step 6: Configure Security Group: allow SSH (Port 22) and HTTPS (Port 443) with Source 0.0.0.0/0 and ::/0
- Step 7: Review Instance Launch: after hitting launce, make sure to choose an existing or create a new key pair and download the key. This will be the only time user can download a key pair. Ensure the key permission is 400.

To connect to AWS EC2 server, run the following command:

```
ssh -i <key-file-and-directory> ubuntu@<AWS-public-DNS>
```

35.5.3 Project Code Structure and Components

The directory structure of the project are:

```
tree command
```

35.5.3.1 LICENSE

LICENSE file indicates which licensing the project and its code are under. In this case, it is MIT license.

35.5.3.2 README

README file provides a short description of the project and code components and instruction on how to run the code.

35.5.3.3 Makefile

Makefile contains a list of shortcut commands that can easily run via a make command when the user is in the directory where the Makefile is.

35.5.3.4 Dockerfile

This file contains the instruction and components to build the docker image. The file content is:

```
FROM python:3.6
WORKDIR /app
COPY . /app

RUN pip install --trusted-host pypi.python.org -r requirements.txt
EXPOSE 80
CMD python ./app.py
```

Dockerfile includes six major steps: FROM, WORKDIR, COPY, RUN,

EXPOSE, CMD. The steps instruct docker to know which programming language and version to use, temporary working directory, package dependencies, port to use, and the python main script to run.

35.5.3.5 requirements.txt

This file contains all the necessary packages to be part of building the docker image.

The four packages that are required for Flask API app are:

```
flask  
sklearn  
xgboost  
pandas
```

35.5.3.6 app.py

app.py is the main app that utilizes flask API to take JSON file data from API POST command and apply the machine learning model to output prediction into a JSON file.

35.5.3.7 Other python codes

- preprocessing.py: python code that reads raw data and cleans the data into a good form to feed into machine learning algorithms
- to_json.py: python code to convert file from dataframe format to JSON format
- train_data.py: python code to train and build XGBoost model and output the model for the credit flask API app
- evaluation.py: python code to evaluate multiple algorithms and output results into CSV files

35.5.4 Code Running Instruction

35.5.4.1 Environment and Files Preparation

After all the prerequisites are met and the Ubuntu server is up and running, the follow steps can be used to reproduce the environment and files preparation process starting at the directory that the Makefile is in:

Step 1: Environment preparation

```
make prepare-environment
```

Step 2: Download data files from Kaggle

```
make download-file
```

Step 3: Prepare Train and Test files:

```
make prepare-files
```

or run the one-step make command:

```
make prep-all
```

35.5.4.2 Analysis

```
make evaluation
```

35.5.4.3 Deployment

Step 1: Build a docker image

```
make docker-build-image
```

Step 2: Run docker image

```
make docker-run-image
```

Step 3: POST JSON test data to API Flask app

In a new terminal, run the following command:

```
make post-test-data
```

The result JSON file should be located in `data/processed/result.json`.

35.5.4.4 Clean Up

The following command to clean up files after the deployment:

```
make clean
```

35.6 RESULTS

35.6.1 Deployment Benchmarks

35.6.2 Application Benchmarks

35.7 LIMITATIONS

Data volume is limited due to the provided dataset is from a Kaggle competition; increasing in data volume would improve the prediction result. Furthermore, the project does not focus on security and authentication/authorization aspect of the code, most of the port and network are set to allow any IP access to the EC2 server. Prediction results could also be improved by running and evaluating more other classification algorithms. Also, parameters can be enhanced to provide more customization to all algorithms to observe this specific pattern of the training set.

35.8 CONCLUSION

35.9 ACKNOWLEDGEMENTS

The author would like to thank professor Lakowski and his class's TAs for helping with materials, markdown writing techniques, and review of this project.

35.10 REFERENCES

36 AN EXPLORATION OF THE INTERNET OF THINGS

FA18-523-84

Adam Hilgenkamp

ahilgenk@iu.edu

Indiana University - Bloomington

hid: fa18-523-84

github: [!\[\]\(6ebee2b02fca7a791084f227d9702a8c_img.jpg\)](#)

code: [!\[\]\(0229ad260decc107ae822d47a468a5a6_img.jpg\)](#)

Keywords: IoT, Nest, Thermostat

Working on Raspberry Pi cluster that will be integrated in the Pi book.

https://github.com/cloudmesh-community/fa18-523-84/blob/master/project-report/pi_cluster_setup.md

37 ANALYTICS IN CONSUMER BEHAVIORS IN BLACK FRIDAY WITH TENSORFLOW FA18-523-85

| Bo Li | bl15@iu.edu | Indiana University Bloomington | hid: fa18-523-85 | github: [cloud](#)

37.1 ABSTRACT

As internet developed, online shopping has become part of our daily life. Black Friday- a traditional deal day, has also transformed as a big day for online shopping. The internet retailers, such Amazon, also developed their specific strategy for the combat in online shopping, Amazon Prime Day. The developed internet techniques allow us collect and store the trading data, which could be the most valuable materials for researching customer behaviors. In this article, we get the dataset of trading data in Black Friday and use TensorFlow to analysis the data in different dimensions. The dataset has several features, some of them are valuable for the research and some are not, some meaningful insights are behind the dataset, by using TensorFlow, we are able to explore the dataset in details with models.

37.2 KEYWORDS

Behavior big data, human behavior, TensorFlow, data mining, deep learning

37.3 INTRODUCTION

Behavioral big data (BBD) refers to very large and rich multidimensional data sets on human and social behaviors, actions, and interactions, which have become available to companies, governments, and researchers. A growing number of researchers in social science and management fields acquire and analyze BBD for the purpose of extracting knowledge and scientific discoveries [1].

Besides, the online retailers also want to figure out the profound meanings behind the consumer's actions. So the behavioral big data comes across with research area and industry area, which results in different research methods. We use the methods in research area to analysis the dataset with specific models designed to the dataset from Kaggle.

Why we use big data method to conduct the analytics? The answer lies on the character of the real dataset generated in the real trading system. The system records the deal with some specific fields, which could be some information of price, time, discount, product information, product status, etc. The user behavior could also be recorded such as the action of adding to list, the time between order to payment, etc. If a user has the habit of adding many products to the list but only buy a few of them, you may observe the data and draw the conclusion that the user is rational and could hardly be affected by the advertisement. But if you have huge amount of data, maybe it is real time data, the only way to picture those consumers is establishing models and define the features, then use the big data methods to research them.

37.4 DATA DESCRIPTION

The dataset here is a sample of the transactions made in a retail store. The store wants to know better the customer purchase behavior against different products. Specifically, here the problem is a regression problem where we are trying to predict the dependent variable (the amount of purchase) with the help of the information contained in the other variables.

Classification problem can also be settled in this dataset since several variables are categorical, and some other approaches could be "Predicting the age of the consumer" or even "Predict the category of goods bought". This dataset is also particularly convenient for clustering and maybe find different clusters of consumers within it [2].

37.5 TECHNOLOGIES

TensorFlow is a framework to perform computation very efficiently, and it can tap into the GPU (Graphics Processor Unit) in order to speed it up even further. This will make a huge effect as we shall see shortly. TensorFlow can be controlled by a simple Python API, which we will be using in this Article [3].

Tensorflow is one of the widely used libraries for implementing Machine learning and other algorithms involving large number of mathematical operations. Tensorflow was developed by Google and it's one of the most popular Machine Learning libraries on GitHub. Google uses Tensorflow for implementing Machine learning in almost all applications. For example, if you are using Google photos or Google voice search then you are using Tensorflow models indirectly, they work on large clusters of Google hardware and are powerful in perceptual tasks [4].

37.6 ALGORITHMS AND METHODOLOGY

37.6.1 knn

In pattern recognition, the k-nearest neighbors algorithm (k-NN) is a non-parametric method used for classification and regression [5]. In both cases, the input consists of the k closest training examples in the feature space. The output depends on whether k-NN is used for classification or regression:

- * In k-NN classification, the output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor.
- * In k-NN regression, the output is the property value for the object. This value is the average of the values of its k nearest neighbors.

k-NN is a type of instance-based learning, or lazy learning, where the function is only approximated locally and all computation is deferred until classification. The k-NN algorithm is

among the simplest of all machine learning algorithms.

37.6.2 Naive Bayes Classifier

In machine learning, naive Bayes classifiers are a family of simple “probabilistic classifiers” based on applying Bayes’ theorem with strong (naive) independence assumptions between the features.

Naive Bayes classifiers are highly scalable, requiring a number of parameters linear in the number of variables (features/predictors) in a learning problem. Maximum-likelihood training can be done by evaluating a closed-form expression, which takes linear time, rather than by expensive iterative approximation as used for many other types of classifiers [6].

37.7 DATA ANALYTICS

37.7.1 Data Cleaning

37.7.2 Data Exploration and Processing

37.7.3 Data Analysis

37.7.4 Data Visualization

37.8 CONCLUSION

The conclusion drawn by the previous analytics, which could be the picture of owned consumers, the predict of the consumer’s preferences.

37.9 LIMITATIONS

The limitations in the analytics caused by the selection of research method, algorithms, models or the character of the dataset.

37.10 REFERENCE

will be supplemented and transferred to bibtex

- [1] Shmueli, G. (2017). Research dilemmas with behavioral big data. *Big Data*, 5(2), 98.
- [2] <https://www.kaggle.com/mehdidag/black-friday/home>
- [3] <https://hackernoon.com/introduction-of-tensorflow-with-python-f4a9624f2ab2>
- [4] <https://towardsdatascience.com/a-beginner-introduction-to-tensorflow-part-1-6d139e038278>
- [5] https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm
- [6] https://en.wikipedia.org/wiki/Naive_Bayes_classifier

Jeff Liu
liujeff@iu.edu
Indiana University
hid: fa18-523-86
github: [blue icon](#)

Keywords: Big Data

38.1 ABSTRACT

We're in the midst of a big data revolution. Around the world 2.5 quintillion bytes of data is produced daily. That's a mind-boggling amount. And how about this: approximately 90% of the world's data was generated in the last two years alone. For years, companies have been collecting and analyzing massive amounts of information – everything from structured data on production, marketing, sales, HR, finance, facilities and operations to transaction-level data on suppliers, customers and partners. The explosion of data created has led organizations into a period of necessary innovation. The imperative to transform this data into intelligent insights has never been greater and no industry or line of business is immune from the big data revolution. It therefore comes as no surprise that Procurement is turning to big data to drive digital change [164]. Ariba was founded to solve the problem of inefficient enterprise procurement through the network, in 2012, SAP US acquired Ariba for \$4.3 billion and was renamed SAP Ariba, right now, 2.9 million companies worldwide are connected using the SAP Ariba network platform, using Ariba's SaaS solution to manage expenses and business activities. SAP Ariba has built a complete corporate trading network.

38.2 INTRODUCTION

SAP Ariba is how companies connect to get business done. On the Ariba Network, buyers and suppliers from more than 3.3 million companies and 190 countries discover new opportunities, collaborate on transactions and grow their relationships. Buyers can manage the entire purchasing process, while controlling spending, finding new sources of savings and building a healthy supply chain. And suppliers can connect with profitable customers and efficiently scale existing relationships – simplifying sales cycles and improving cash control along the way. The result is a dynamic, digital marketplace, where more than \$1.6 trillion in commerce gets done every year[165]. Companies are increasingly relying on Ariba to enable them to connect and collaborate in today's digital economy which means demand for certified Ariba consultants is on the rise. SAP Ariba, a market leading cloud, network procurement offering is the fastest growing business for SAP. SAP Ariba is enabling friction-less commerce between companies, through the exchange of direct and indirect goods and services making SAP's vision of Run Simple, Run Real-Time and Run Networked a reality for customers, SAP Ariba's cloud-based solutions make it easier to collaborate and compete.

38.3 ARCHITECTURE

Perhaps more than any other business unit, procurement stands at the forefront of digital transformation. What began as snatching efficiency gains by digitizing and automating paper-, time-, and labor-intensive processes has evolved today into real-time analytics, enterprise-wide visibility and global supply chain optimizations. By harnessing the transformative power of digitalization, procurement's influence goes well beyond mere cost cutting and, today, profoundly impacts competitiveness and corporate strategy.

From sourcing and orders to invoice and payment, SAP Ariba's procurement solutions span the entire e-procurement process, and deliver serious bottom-line benefits:

- 60% lower operating costs
- 1% to 8% reduction in supply chain costs

- 50% to 75% faster transaction cycles
- 90% fully automated transaction processing
- 60% improvements in order accuracy
- 5% to 20% increases in revenue with new customers
- 30% or more higher share of wallet with current customers
- 15% improvement in retention rates among customers.

Automating key processes such as procurement, orders, invoicing and payment is a must if your business is to remain competitive. But to thrive, more must be done with the vast amount of data — internal and external — available to procurement. Mining, cultivating and analyzing transactional and operational data —together with data from customers, prospects, partners, and suppliers — empowers procurement executives to gain visibility into direct and indirect spending across global accounts. This visibility empowers procurement pros to identify opportunities for consolidation, cost reduction, and process improvement, and, in line with procurements' evolution to the role as trusted advisor, participate in more strategic and business-meaningful goals, such as discovering new business models, driving product and service enhancements, and identifying untapped revenue streams[fa18-523-86-www-Ariba-Procurement].

38.4 IMPLEMENTATION

With B2B ecommerce adoption hitting an inflection point, companies operating global supply chains are moving away from legacy systems and expensive EDI (electronic data interchange) toward ubiquitous and affordable online platforms and cloud-based solutions that reduce costs, streamline procurement and payment processes, and provide visibility across global supply chains. Leveraging advances in cloud, mobility, big-data analytics and IoT, today's digital supply chains are more data driven and operate faster, better, stronger, and leaner than ever before. According to research presented in an INFOGRAPHIC created by The Economist Intelligence Unit, the gains from digital supply chains are impressive:

- Today, 22% of manufacturing executives have "complete

visibility" into their supply chains, up from 9% in 2013.

- 44% of senior executives say their supply-chain function uses sophisticated tools that leverage big data to help with day-to-day decision making.

Cloud-based supply chain solutions and business networks present great options for strategically tackling this problem by providing a robust, unified platform that enables close collaboration. According to research by SAP Ariba, collaborative supply networks deliver quantifiable business value, including:

- 60% improvement in order accuracy
- 50% - 70% faster transaction cycles
- More than 90% "touchless" transaction processes[166].

Ariba can now integrate with solutions such as SAP ERP and S/4 HANA, and can use SAP HANA and SAP Leonardo technology for data analysis to provide customers with more insights. As SAP deepens the integration of Ariba with other software, users can more easily connect, share and query their own, partner ERP, CRM, SCM and e-commerce software. Procurement systems capture a vast amount of data, including sourcing information, weather reports, manufacturing and delivery data, supplier data, purchasing data, catalogue data. There are undoubtedly very valuable insights within that data, but the challenge for procurement is to understand and use it to make better, informed decisions. Big data, predictive analytics and prescriptive guidance can further scale with cognitive computing power to provide better information that will enhance situational awareness and speed-to-decision. The resulting business value will improve the way procurement roles and processes are executed, making both more effective, valuable and sustainable[164]. The Ariba Procurement Content Solution has a single user interface and is integrated with the Ariba network, the world's largest online trading community, to quickly implement product catalogs and their maintenance. With the right contract pricing, users simply search for the content that the supplier provides about the goods and services they need. Users will also enjoy it when they use it, because this

solution is designed to make users feel like they are shopping on their favorite websites. Once your e-procurement begins to include user-friendly content, you will be able to deliver on its promises of improved normative and bottom-line balances without any difficulty.

38.5 CONCLUSION

Today, most procurement organizations are faced with three core challenges when working with big data:

- * Digitising processes: The first challenge for procurement is to access unstructured data that resides in scanned documents, email inboxes and spreadsheets. The key to unlocking the data's potential is to ensure it is stored in a digital format that can be analyzed.
- * Driving insights: It is then vital to be able to analyses and utilize the digitized data. Human limitations, such as skills and time, as well as technical challenges can hinder find patterns, process the information intelligently and accurately, and derive insights. Customers are challenged to figure out how to tap into cognitive capabilities and build them into their processes.
- * Enabling talent and skills: Once this is in place organizations need to foster a change in culture and mindset that will help them embrace these disruptive technologies. Big data, predictive analytics and cognitive computing require new skills and new ways of working, such as self-service analytics.

Advances in technology and new concepts like intelligent computing fueled by SAP Leonardo provide ample opportunities to predict and respond more effectively to customer and market demands. This insight can guide buyers or requesters to create request for information or a contract template based on potentially unlimited amounts of information analyzed across multiple databases. Connecting people and information guided by "intelligent" procurement systems will fundamentally change how companies buy and sell[fa18-523-86-www-Ariba-BigData]. The digitalization of business is real, and it is here to stay. The combination of new technologies and skilled talent working with intelligent systems may provide the competitive edge that businesses need to stay ahead of competitors in the market. But it will also fundamentally elevate the

importance of procurement to providing guidance on innovations, mitigating risks and securing a sustainable supply chain. As with any revolution there will be winners and losers and those that make the most of big data will come out on top.

39 OCR EXTRACTION IMPLEMENTATION WITH TESSERACT

FA18-523-88

Joao Paulo Leite
jleite@iu.edu
Indiana University
hid: fa18-523-88
github: [cloud](#)
code: [cloud](#)

Keywords: OCR, Tesseract, Python

39.1 ABSTRACT

The main purpose of this project is to create a simple OCR extraction implementation which is able to extract key metadata from documents. To accomplish this, Google's Tesseract OCR Engine is leveraged to provide full-page OCR data. The goal is to have a configurable extraction engine that allows users to pin-point the meta-data to be extracted and output said meta-data.

39.2 INTRODUCTION

Optical Character Recognition (OCR) technology first appeared in the 1940's and grew alongside the rise of the digital computer. It was not until the late 1950's when OCR machines became commercially available and today this technology presents itself in both hardware devices as well as software offerings [167].OCR is the first step in enabling the extraction of actionable data by transforming print on an image(document) to machine encoded text. The analysis of the output provided by OCR engines allows for this key data to be used for downstream processes and reporting. Documents fall into three categories: structured documents, semi-structured documents and

unstructured documents.

Gartner, a leading technology analysis firm, has stated the following:

"...the amount of data stored in companies will increase by 800 percent by 2018, 80 percent of which would include unstructured data that are harder to tame and manage. The biggest challenges for companies will include: collecting, managing, storing, searching and archiving this content [168]."

As unstructured documents continues to grow, big data systems are being introduced as a solution to analyze and organize this data. As a precursor, an OCR extraction solution can extract actionable data from documents and provide structure to unstructured content.

39.3 OVERVIEW OF OPTICAL CHARACTER RECOGNITION

The main principle in Optical Character Recognition (OCR) is to automatically recognize character patterns. This is accomplished by teaching the system each class of pattern that can occur and providing a set of examples for each pattern. At the time of recognition, the system performs a comparison between the unknown character provided and the previously provided examples, assigned the appropriate class to the closest match[167]. This system is designed to solely transform text on a document into machine encoded text and additional systems must be built to further extract relevant information from the document, that is to say, the process of OCR is the first step in transforming structured, semi-structured and unstructured documents into valuable and relevant information.

39.4 CONTEXT BASED EXTRACTION ENGINE

This project utilizes Google's Open Source Tesseract OCR engine to provide HOCR output that is leveraged to begin the process of extracting information from unstructured data provided by Tesseract. The extraction engine's logic works in two distinct phases, the

identification of potential candidates(data which follows a specific format) and the scoring of each candidate based on context around said candidate. At the end of this process, the candidate which obtained the highest score will be selected.

For the extraction engine, there are 8 distinct phases:

1. Image Thresholding
2. OCR Process
3. Transform HOCR Data
4. Define Candidates
5. Set Context
6. Group Context
7. Score Context
8. Output Results

39.4.1 Image Thresholding

Before submitting the image into Tesseract, image clean up is performed to create a bitonal image and to remove any noise that may be present. This process consists of three steps; standardizing image DPI, smoothing the image and removing noise from the image[169].

Standarizing Image DPI to 300 DPI:

```
def set_dpi(path):  
    image = IMG.open(path)  
    len_x, wid_y = image.size  
    factor = max(1, int(1800 / len_x))  
    size = factor * len_x, factor * wid_y  
    image_resized = image.resize(size, IMG.ANTIALIAS)  
    temp_f = tempfile.NamedTemporaryFile()  
    temp_fn = temp_f.name  
    image_resized.save(temp_fn, dpi=(300, 300))
```

```
    return temp_fn
```

Converting to Bitonal Image via Adaptive Thresholding:

```
def remove_noise(name):  
  
    image = cv2.imread(name, 0)  
    filtered = cv2.adaptiveThreshold(image.astype(np.uint8), 255,  
    cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY, 41, 3)  
    core = np.ones((1, 1), np.uint8)  
    opening = cv2.morphologyEx(filtered, cv2.MORPH_OPEN, core)  
    closing = cv2.morphologyEx(opening, cv2.MORPH_CLOSE, core)  
    image = smooth(image)  
    original_image = cv2.bitwise_or(image, closing)  
  
    return original_image
```

Smoothing Image:

```
def smooth(image):  
  
    ret1, th1 = cv2.threshold(image, BINARY_THRESHOLD, 255, cv2.THRESH_BINARY)  
    ret2, th2 = cv2.threshold(th1, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)  
    blur = cv2.GaussianBlur(th2, (1, 1), 0)  
    ret3, th3 = cv2.threshold(blur, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)  
  
    return th3
```

39.4.2 OCR Process

Google's Tesseract OCR engine is an open source engine that has the ability to output HOCR. HOCR is an open standard of data representation for formatted text obtained from an OCR engine. This standard includes text, style, layout information, recognition confidence and other info in a XML structure.

Create HOCR data:

```
def Run(self):  
  
    DATA = pytesseract.image_to_pdf_or_hocr(image, lang=None, config='hocr', nice=0,  
extension='hocr')
```

39.4.3 Transform HOCR Data

Once the HOCR results are generated, we must transform the results into useable data for our extraction process. The first step is to target the ocrx_word data from the results and parser it into separate

words. After this initial parsing is complete, we separate each individual data point within a dictionary object with the values: value, confidence, left, top, right and bottom.

Parsing HOCR results with Beautiful Soup

```
def Run(self):
    soup = bs4.BeautifulSoup(DATA, 'html.parser')
    words = soup.find_all('span', class_='ocrx_word')
```

Creating word data structure:

```
def transform_hocr(self, words):
    for x in range(len(words)):
        word[int(words[x]['id'].split('_')[2])] = {}
        word[int(words[x]['id'].split('_')[2])]['Value'] = words[x].get_text()
        word[int(words[x]['id'].split('_')[2])]['Confidence'] = words[x]['title'].split(';')[1].split(' ')[2]
        word[int(words[x]['id'].split('_')[2])]['Left'] = words[x]['title'].split(';')[0].split(' ')[1]
        word[int(words[x]['id'].split('_')[2])]['Top'] = words[x]['title'].split(';')[0].split(' ')[2]
        word[int(words[x]['id'].split('_')[2])]['Right'] = words[x]['title'].split(';')[0].split(' ')[3]
        word[int(words[x]['id'].split('_')[2])]['Bottom'] = words[x]['title'].split(';')[0].split(' ')[4]
```

39.4.4 Define Candidates

After transforming the HOCR results, we use the generated word dictionary to find values that match the defined regular expression that was provided by the user. We store all candidates which match the regular expression are stored within the candidates dictionary object with the values: value, confidence, left, top, right and bottom.

Finding candidates:

```
def find_candidates(self, RE_ATT):
    y = 1
    for z in RE_ATT:
        for x in range(len(word)):
            m = re.match(r'' + z + '', word[x + 1]['Value'], )
            if m:
                candidates[y] = {}
                candidates[y]['Value'] = word[x + 1]['Value']
```

```

candidates[y]['Confidence'] = word[x + 1]['Confidence']
candidates[y]['Left'] = word[x + 1]['Left']
candidates[y]['Top'] = word[x + 1]['Top']
candidates[y]['Right'] = word[x + 1]['Right']
candidates[y]['Bottom'] = word[x + 1]['Bottom']
y = y + 1

```

39.4.5 Set Context

Using the location input define by the user, we will set the context of each candidate based on the proximity(top, bottom, left and right) in pixels. Each word which falls within the proper proximity is stored in the context dictionary with the values: value, candidate ,word number, confidence, left, top, right, bottom, line number and same line as candidate.

```

def set_context(self, candidates, word):
    line = 1
    z = 1

    for x in range(len(candidates)):
        for y in range(len(word)):

            if (int(word[y + 1]['Bottom']) > int(candidates[x + 1]['Bottom']) - 100) and
            (int(word[y + 1]['Bottom']) < int(candidates[x + 1]['Bottom']) + 20) and \
                (int(word[y + 1]['Right']) > int(candidates[x + 1]['Left']) - 700) and (int(word[y + 1]['Right']) < int(candidates[x + 1]['Left']) + 20):

                context[z] = {}
                context[z]['Value'] = word[y + 1]['Value']
                context[z]['Candidates'] = candidates[x + 1]['Value']
                context[z]['Word'] = str(y + 1)
                context[z]['Confidence'] = word[y + 1]['Confidence']
                context[z]['Left'] = word[y + 1]['Left']
                context[z]['Top'] = word[y + 1]['Top']
                context[z]['Right'] = word[y + 1]['Right']
                context[z]['Bottom'] = word[y + 1]['Bottom']

            if z == 1:
                context[z]['Line'] = line
            elif context[z - 1]['Bottom'] == word[y + 1]['Bottom']:
                context[z]['Line'] = line
            else:
                line = line + 1
                context[z]['Line'] = line

            if int(word[y + 1]['Bottom']) > int(candidates[x + 1]['Bottom']) - 15 and
            int(word[y + 1]['Bottom']) < int(candidates[x + 1]['Bottom']) + 15:
                context[z]['SameLine'] = "1"
            else:
                context[z]['SameLine'] = "0"

    z = z + 1

```

39.4.6 Group Context

Once the context for each candidate has been defined, we will group the context based on proximity. If multiple context words are in sequence, we will group those so that they are arranged as a phrase.

```
def define_groupcontext(self, context):
    z = 1
    for x in range(len(context)):

        if x == 0:

            groupcontext[z] = {}
            groupcontext[z]['Value'] = context[x + 1]['Value']
            groupcontext[z]['Word'] = context[x + 1]['Word']
            groupcontext[z]['Candidates'] = context[x + 1]['Candidates']
            groupcontext[z]['Weight'] = '0'
            groupcontext[z]['Confidence'] = context[x + 1]['Confidence']
            groupcontext[z]['Left'] = context[x + 1]['Left']
            groupcontext[z]['Top'] = context[x + 1]['Top']
            groupcontext[z]['Right'] = context[x + 1]['Right']
            groupcontext[z]['Bottom'] = context[x + 1]['Bottom']
            groupcontext[z]['SameLine'] = context[x + 1]['SameLine']

        elif int(groupcontext[z]['Word']) + 1 == int(context[x + 1]['Word']):

            groupcontext[z]['Value'] = groupcontext[z]['Value'] + ' ' + context[x + 1]['Value']
            groupcontext[z]['Word'] = context[x + 1]['Word']
            groupcontext[z]['Confidence'] = context[x + 1]['Confidence']
            groupcontext[z]['Top'] = context[x + 1]['Top']
            groupcontext[z]['Right'] = context[x + 1]['Right']
            groupcontext[z]['Bottom'] = context[x + 1]['Bottom']

        else:

            z = z + 1
            groupcontext[z] = {}
            groupcontext[z]['Value'] = context[x + 1]['Value']
            groupcontext[z]['Word'] = context[x + 1]['Word']
            groupcontext[z]['Candidates'] = context[x + 1]['Candidates']
            groupcontext[z]['Weight'] = '0'
            groupcontext[z]['Confidence'] = context[x + 1]['Confidence']
            groupcontext[z]['Left'] = context[x + 1]['Left']
            groupcontext[z]['Top'] = context[x + 1]['Top']
            groupcontext[z]['Right'] = context[x + 1]['Right']
            groupcontext[z]['Bottom'] = context[x + 1]['Bottom']
            groupcontext[z]['SameLine'] = context[x + 1]['SameLine']
```

39.4.7 Score Context

After grouping the context, using the context values provided by the user, we will score each grouping based on how strongly it matches the context values. We utilize a fuzzy algorithm that allows us to accommodate for any OCR errors or misspellings. The weight given to

each context word is also judged based on the weighted value provided by the user. This gives the ability for the user to define which context words should carry more weight in the scoring algorithm. For grouped context that fall within the same line as the candidate, the user can define a value to be added to the overall weight.

Score Context:

```
def weightcontext(self, KW_ATT):
    for z, value in KW_ATT.items():
        for x in range(len(groupcontext)):
            groupcontext[x + 1]['Weight'] = 0
            if int(groupcontext[x + 1]['Weight']) < fuzz.WRatio(groupcontext[x + 1]['Value'], z):
                groupcontext[x + 1]['Weight'] = int(fuzz.WRatio(groupcontext[x + 1]['Value'], z)) * int(value[0]) / 100
            if groupcontext[x + 1]['SameLine'] == '1':
                groupcontext[x + 1]['Weight'] = groupcontext[x + 1]['Weight'] + int(value[5])
```

39.4.8 Output Result

Outputting a resulting text file with the winning candidate as well as the entire results array. The text file name will be the same as the input image file.

```
def outputresults(self, groupcontext, fp):
    for x in range(len(groupcontext)):
        if groupcontext[x + 1]['Candidates'] in results:
            if int(results[groupcontext[x + 1]['Candidates']]) < int(groupcontext[x + 1]['Weight']):
                results[groupcontext[x + 1]['Candidates']] = groupcontext[x + 1]['Weight']
            else:
                results[groupcontext[x + 1]['Candidates']] = groupcontext[x + 1]['Weight']

        if(len(results.keys()) == 0):
            f = open(fp + '.txt', 'w')
            f.write("Could not find any valid candidates")
            f.close()
```

```
        else:  
  
            sorted_by_value = sorted(results.items(), key=lambda kv: kv[1], reverse=True)  
            f = open(fp +'.txt', 'w')  
            f.write("WINNING CANDIDATE (CANDIDATE , WEIGHT): " + str(sorted_by_value[0]) + "\n")  
            f.write("ALL CANDIDATES: " + str(sorted_by_value))  
            f.close()
```

39.5 EXAMPLE

39.6 TOOLS AND TECHNOLOGY

The tools and technology deployed for this project are going to be covered in this section.

39.6.1 Terresact

Python-tesseract is an optical character recognition (OCR) tool for python. That is, it will recognize and “read” the text embedded in images. Python-tesseract is a wrapper for Google’s Tesseract-OCR Engine[170].

Code Example:

```
import pytesseract  
  
hocr = pytesseract.image_to_pdf_or_hocr('test.png', extension='hocr')
```

Install:

```
pip install pytesseract
```

39.6.2 Beautiful Soup

Beautiful Soup is a library that makes it easy to scrape information from web pages. It sits atop an HTML or XML parser, providing Pythonic idioms for iterating, searching, and modifying the parse tree[171].

Code Example:

```
import bs4
```

```
soup = bs4.BeautifulSoup(DATA, 'html.parser')
words = soup.find_all('span', class_='ocrx_word')
```

Install:

```
pip install beautifulsoup4
```

39.6.3 FuzzyWuzzy

Fuzzy Wuzzy provides fuzzy string matching in an easy to use package. It uses Levenshtein Distance to calculate the differences between sequences in a simple-to-use package[172].

Code Example:

```
from fuzzywuzzy import fuzz
fuzz.ratio("fuzzy wuzzy was a bear", "wuzzy fuzzy was a bear")
Output:91
```

Install:

```
pip install fuzzywuzzy
```

39.6.4 Python

Python is the high-level programming language that was used to develop this project.

39.6.5 Numpy

NumPy is the fundamental package for scientific computing with Python[#hid-sp18-414-www-NumPy].

Code Example:

```
import numpy as np
core = np.ones((1, 1), np.uint8)
```

Install:

```
pip install Numpy
```

39.6.6 OpenCV

OpenCV (Open Source Computer Vision Library) is released under a BSD license and hence it's free for both academic and commercial use. It has C++, Python and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android. OpenCV was designed for computational efficiency and with a strong focus on real-time applications[173].

Code Example:

```
import cv2
img = cv2.imread('messi5.jpg',0)
```

Install:

```
pip install opencv-python
```

39.6.7 Python Imaging Library

The Python Imaging Library adds image processing capabilities to your Python interpreter. This library provides extensive file format support, an efficient internal representation, and fairly powerful image processing capabilities[174].

Code Example:

```
from PIL import Image as IMG
image = IMG.open(path)
```

Install:

```
pip install Pillow
```

39.6.8 Tkinter

Tkinter is Python's a standard GUI (Graphical User Interface) package. It is a thin object-oriented layer on top of Tcl/Tk[175].

Install:

```
pip install Tkinter
```

39.7 CONCLUSION

39.8 ACKNOWLEDGEMENT

The authors would like to thank the Big Data Applications and Analytics (I-523) course teaching staff, mainly professor Gregor von Laszewski for their support and guidance during this project.

40 AUTOMATION ON DRUG INTERACTIONS PROFILING

FA18-423-06 FA18-423-03 FA18-423-02 FA18-423-05

Yixing Hu, Kelvin Liuwie, Chandler Mick, Omkar Tamhankar
yixihu@gmail.com, kliuwie@gmail.com, chmick@iu.edu,
otamhank@iu.edu
Indiana University
hid: fa18-423-02 fa18-423-03 fa18-423-05 fa18-423-06
github: [cloud9](#)
code: [cloud9](#)

Learning Objectives

- Creating a program that can parse and query large data sets
- Exploring various “free” cloud servers to run our program
- Evaluating which cloud server provides the optimal working efficiency

40.1 INTRODUCTION

For our project, we created a Python program that took raw data from the United States Food and Drug Administration’s Adverse Event Reporting System (FAERS) and filtered out important components for a user to query the information simply. The main issue that we noticed was that the important information that Food and Drug Administration (FDA) was collecting was in large datasets that incredibly difficult to breakdown analyze for the average users of the information. Users of this information could be regulators, doctors, or even concerned consumers. Therefore, we believe that our program simplifies the analysis of these big data sets that are provided by the FDA and makes them fit for spreadsheets, such as Excel, for more in-depth review of the drug data.

One of the most important issues in the pharmaceutical industry

today is determining the quality of a drug on the market [176]. Currently a pharmaceutical company needs only prove in human clinical trials that a drug is safe when taken in isolation. However, humans are not basic research subjects and are often taking a number of medications. As the state of the healthcare system continues to fall victim to political cycles, it is always important to make sure that information on the safety and effects of drugs on the market need to be readily and easily available for review. We hope that our project represents an example of what could be implemented in the marketplace to present the information to those who need to be aware of the effects of the drugs that they are interacting with.

40.2 DATASET

The dataset is an Extensible Markup Language (XML) file from the FAERS on the FDA's website. FAERS is a collection of reports from doctors, nurses, and patients who have reported to the FDA side-effects of using certain commercial drugs [177]. This data is used by the FDA to formulate warnings and regulations against drugs to warn consumers of the potential side effects that come about from using pharmaceutical drugs.

The publicly available XML file is a 112 MB file with over 16 million rows of data. Therefore, the file cannot be easily evaluated and assessed without creating some sort of program to trim out the unnecessary data and store the valuable data for running the query. The data that we analyzed with our program covers the months of July 2018 to September 2018. However, a user would most likely want to analyze data from previous quarters, which is possible with our program.

The XML file contains 67 tags for each "adverse event report." An adverse event report is a negative side effect to a drug from anything including migraines to something as serious as death. Every tag includes information about the patient, the side effect, the drug, treatment duration, etc. For this project, we decided the most

important tags that would want to be accessed would be the reactionmeddrapt, medicinalproduct, and activesubstancename. These relate to the reactions from the drug, the name of the drug, and the substances in the drug, respectively.

40.3 IMPLEMENTATION

ProgramDesign

40.3.1 parser.py

Our parser.py file takes an XML file containing the initial FAERS raw data from the website and converts it to a dataframe format to be stored in a CSV file. This initial parse separates the three tags that we are looking to store for our program (reactionmeddrapt, medicinalproduct, and activesubstancename). The parser.py program flows into the project.py program where the dataframe format allows the data to be easily manipulated.

40.3.2 project.py

Our project.py file takes an XML file containing the initial FAERS raw data from the website and converts the selected data to a CSV file. Using the parsing.py function, the function takes our dataframe data and outputs the three tags that we used in our project. This program takes the original, large dataset and forms a smaller, compact dataset with the data that we deemed important.

40.3.3 query.py

Our query.py file takes the compiled CSV file and allows a user to return certain data based on inputted values. These inputted values can be certain drugs, substances within the drugs, or the reactions of the drugs. This user queried data can be used by the user to evaluate the specific results that they are looking for. This CSV file can be transferred to a spreadsheet for simple data analysis.

40.3.4 Amazon Web Services (AWS) EC2 Usage

Overview:

This document explains how to access AWS EC2 and how it was used to run the program in the cloud server [178].

Setup

Go to <https://aws.amazon.com/>. Click on “Sign In to the Console”. If you do not already have an account set up, create an account. In the “AWS services” search bar, search “EC2” and click on the EC2 option. Under “Create Instance” click “Launch Instance”. Click the “Select” button next to “Amazon Linux 2 AMI (HVM), SSD Volume Type”. Click on “Review and Launch” then “Launch”. In the drop-down menu select “Create a new key pair”, type in a key pair name, click “Download Key Pair”, select the check box below, and click “Launch Instance”. In the green box at the top of the screen click on Instance ID link, this will open the instance.

Accessing the instance

Open command prompt, if using Windows, or Terminal, if using Mac. Type in:

```
ssh -i ~/key-pair-location* ec2-user@*IPv4 Public IP*
```

This will connect you into the EC2 instance.

Uploading files into the instance

Download Cyberduck 2. Open the application. In the dropdown menu, select “SFTP (SSH File Transfer Protocol)”. In the Server field, enter the Public DNS. In the Username field enter “ec2-user” [179]. In the SSH Private Key field select “Choose” from the dropdown menu

and select your .pem key pair file. Click on “Connect”. Once connected through Cyberduck, you can simply drag and drop files into the instance.

Disclaimer

Our .xml files were greater than 1 GB in size. AWS EC2's free tier only offers up to 1 GB memory therefore we would receive memory errors when running our python files on the server.

40.3.5 Microsoft Azure Cloud Shell Usage

Overview:

This document explains how to access Microsoft Azure and how Azure is used to run programs in the cloud server [180].

Setup

Go to <https://portal.azure.com/> to register an Azure account. Once you get an account, you can go to Azure Portal to access all the tools you need. The search bar at the very top could help you access any resource you need in Azure.

Create a Python web app through Azure app services

First, clone the folder containing code files and data files from Github with Bash/Terminal in your local machine. Before further operation, run the Python files locally to check if the program runs correctly. Then, go back to the Azure portal which was accessed previously. There, you will open the Cloud Shell for the rest of your operations. The very first step in the Cloud Shell is to create a deployment user, if you do not already have one. The command should read:

```
az webapp deployment user set --user-name <username> --password <password>
```

where and are replaced with the username and password you are planning to use. After this command is ran successfully, there should be a JSON output with your password shown as null. The username and the password should be recorded for future use. After that, you have to create a resource group to make it easier to manage, by using the command:

```
az group create --name myResourceGroup --location "East US"
```

For this command, “East US” could be replaced by other regions where Azure is available, but for the connection stability, it would be best to use the closest region. Then, an Azure App Service plan should be created. The command should read:

```
az appservice plan create --name myAppServicePlan --resource-group myResourceGroup --sku B1 --is-linux
```

When those steps are done, it is the time to create the web app. Still working on the Azure Cloud Shell, type the following command lines:

```
az webapp create --resource-group myResourceGroup --plan myAppServicePlan --name <app_name> --runtime "PYTHON|3.7" --deployment-local-git
```

These command lines should return an output starting with:

```
Local git is configured with url of  
'https://<username>@<app_name>.scm.azurewebsites.net/<app_name>.git'
```

This URL of git should be kept as you will need it later for connection.

After those steps, go back to local bash for the following operations. The first step is to add an Azure remote to the local Git repository by using the command:

```
git remote add azure <deploymentLocalGitUrl-from-create-step>
```

And then use:

```
git push azure master
```

to push to Azure from Git repository. This command should take a bit longer to process. Once the processing is done, you can go back to the Azure Portal to access the App Services to find the one being created. Click on that service and go to the side bar to click on the SSH under Development Tools. Then, you can access all the files you have pushed onto the cloud server.

40.4 RESULTS

Our program works great on a local machine. If a local machine contains the Python programs that we created and the FAERS raw dataset, we are able to easily parse and query the proper results that we intended to find. However, the same cannot be said for our search for a proper Cloud Server to run this program via a virtual machine, as opposed to a local machine. Amazon Web Service's EC2 Server, Microsoft Azure's Cloud Shell, and IBM's VirtualBox all failed to run our program. However, through failure, we learned that two of the three servers' issues can be solved through a simple fix: money.

The services that were offered through Amazon Web Service and Microsoft Azure were "free trials," where we were limited to a certain number of hours or days that we were able to access. However, this also limits the amount of computing power we able to access through their programs. AWS only allows 1 GB of memory for their free trial version, a mark that was well over the datasets that we were working with. Microsoft Azure has a similar pricing structure where you actually have different tiers for computing power (RAM) and storage space [181]. Our issue with IBM's VirtualBox started at the source, where we were unable to create an initial virtual machine to load our program to the server.

The issues with Microsoft Azure and Amazon Web Services Cloud Servers proved to be issues that were brought about by having a dataset that was too large for the free trial versions of the software. We firmly believe that our program would be able to work with ease using our implementation procedures and spending the extra money to access a higher quality version of the same services that we tested.

40.5 CONCLUSION

At the beginning of this project, we had a goal to find a way to take the massive datasets provided by the FDA and turn them into files that can be easily queried by a user for further data analysis. At the same time, we wanted to be able to explore the possibility and usefulness of Cloud Servers to run our program so it does not have to be limited to being accessed via a local machine. While we can firmly

say that we have made a program that provides the services we set out to create, we can say to a degree of certainty that we have found that it can be ran on a cloud server. The two services that we found to be favorable were Amazon Web Service's EC2 and Microsoft Azure's Cloud Shell. In terms of usability and reliability, we believe that a paid subscription to AWS EC2 would be the better of the two.

40.6 WORK BREAKDOWN

- Yixing Hu: Microsoft Azure Server Testing
- Kelvin Liuwie: Parsing and Project Code Writer
- Chandler Mick: Report Writer, Oracle VirtualBox Testing
- Omkar Tamhankar: AWS Server Testing

41 SECCHI DISK VISIBILITY RECOGNITION

format not followed

use the sample and do not make it beautifu;; the sample will do formating correct

integrate refermnces

could we use master worker and not master slave

stream of data is not shown in design

arhitetur how to connect boat to cluster not shown

Yuli Zhao Indiana University Bloomington, Indiana yulizhao@iu.edu

github: [blue cloud icon](#)

41.1 ABSTRACT

Computer vision is designed to imitate human vision. In order to imitate human vision, enabling computer to see is not enough, computer needs to be able to analyze what it sees. What computer sees is essentially frames of images which means nothing to it, but computer is able to analyze the images and extract useful information out following specific instructions from the programmer. Due to the nature of computer, it is able to perform analyzes on a large scale with specific instructions. This paper will discuss how computer vision is used in determining water turbidity through recognizing the secchi disk and extracting measurements as useful information, and how computer vision is applied to a big data set.

41.2 INTRODUCTIONS

Secchi disk is an eight inch circular disk has alternating black and

white color on the surface. It is used to determine the turbidity or the transparency of the water. The way it works is that secchi disk is lowered into the water slowly using a tape that has measurements, and the researcher would record the measurement when the secchi disk is no longer visible. But one single measurement does not offer many information to be useful, useful information can only be extracted by researchers when there are many more measurements. But it would take researcher enormous amount of time to keep in track with each and every measurement. And computer vision is needed here to replace all that workload. Instead of having researcher record every measurement, a camera can be placed by the measurement that has visual on both the tape measurement and the secchi disk. The process of lowering secchi disk can be repeated in different locations by an automated boat, and each process will return result in the format of a video. Then computer will be able to extract the necessary information from the video and mitigate the workload for the researcher.

41.3 DATA COLLECTION

The data taken needs to be uniformed in the first place to avoid further data cleaning. For each video taken, the tape measurement needs to be placed on the right side of the video at a fixed position. In order to increase the identifiable feature of the secchi disk, four geometric shape of pentagons should be placed on the secchi disk, one on each sector. Pentagons on the white sectors are filled with color black, pentagons on the black sectors are filled with color white.



Original

41.4 DATA FLOW

One of the purposes of this project is to create a continuous data flow from the raw video input, to the output of whether if the secchi disk is visible and the measurements of the tape. First, the automated boat

needs to equip a hard drive disk to store recorded videos. At the end of each week, the hard drive needs to be picked up and the video storage needs to be uploaded into a cloud storage then cleared from the hard drive. To maximize efficiency, multiple machines are needed to implement a master/worker architecture. One machine will be the master, the rest of the machines are workers. The master machine retrieves raw video data from the cloud, and assigns the data to one of the worker machines. Each worker machine is programmed to break down the video into frames and perform analysis on each frame, then return two lists of results with secchi disk visibility and the measurements of the tape. When a worker machine is done with one analysis, it will tell the master machine that it is available, and the master machine will assign another raw video to the worker machine. This cycle will repeat to maximize the efficiency of analyzing the big data.



Original

41.5 WORKER MACHINE

The first step worker machine needs to do is to take apart the raw video into separate frames. The analysis need to be performed on each individual frames. Within each frame, the worker machine needs to run two functions. One function is to correctly identify the visibility of the secchi disk. The other function is to correctly identify the position of the tape measurement, and then run an Optical Character Recognition (OCR) on the position that reads the number of measurement. The worker machine will return a list of pair data matching each frame. At last, the worker machine will let the master machine know that it has finished all its work and available to work on the next one.

| Secchi Disk Visibility | Depth Measurement |
|-------------------------------|--------------------------|
|-------------------------------|--------------------------|

| | |
|------|------|
| True | 1'5" |
|------|------|

| | |
|-------|-------|
| True | 2'5" |
| True | 3'5" |
| False | 5'10" |
| False | 8'10" |

41.6 APPROACH: USE CNN TO DETERMINE THE VISIBILITY OF SECCHI DISK

The position of the secchi disk constantly changes in the process due to the water turbulence. Instead of trying to locate the exact position of the secchi disk, a convolutional neural network can be used to train a model that predicts the visibility of the secchi disk in each frame. Using a convolutional neural network approach requires many labeled data to begin with in order to get an accurate model. An example of data is a frame of a video, the label would be either “yes” or “no”, representing the secchi disk is visible or not in that certain frame. In order to train an accurate convolutional neural network model, a large number of data needed to be labeled, approximately 10,000 frames or more. To ensure the accuracy of the model, the training data needs to cover most scenarios. It would take a researcher enormous amount of time to label every frame. Instead, the researcher could design a survey where it is going display a frame and a “yes” or “no” option. The researcher can recruit 100 volunteers to do 100 of these secchi disk identification questions each. Each answer along with the frame will be stored in a CSV file where a frame is stored as a one-dimensional array.

41.7 TRAINING OF CONVOLUTIONAL NEURAL NETWORK MODEL

Before training the model, the label of the training data needs to be processed with one-hot encoding. To build a convolutional neural network model, Keras is a researcher friendly tool to create and test the model, the parameters should be chosen at the researcher’s

discretion. A sample code is provided in the project code section.

41.8 TAPE MEASUREMENT OPTICAL CHARACTER RECOGNITION(OCR)

The fixed position of the tape on every frame, allows the researcher to crop out the section of the frame, then run pytesseract OCR on the section. The instruction to install pytesseract can be found at www.github.com/tesseract-ocr/tesseract.

42 SCALABLE MICROSERVICES TO LABEL YELP IMAGES USING KUBERENETS SP18-616-02

Keerthi Naredla

knaredla@iu.edu

Indiana University Bloomington

hid: sp18-616-02

github: [!\[\]\(c355b5bf81721c7c67a4688d420678e1_img.jpg\)](#)

code: <https://github.com/cloudmesh-community/hid-sp18-602/tree/master/project-code>

Keywords: docker, vision, kubernetes, yelp, redis, pub/sub

42.1 INTRODUCTION

Can not be replicated

Missing Benchmark so a comparision is highlighted (that what a benchmark is)

All sections have to be revisited as sections wer done wrong possibly even in the original latex file. Take a look at the TOC.

The applications are growing complex, not just functionality wise but also the data generated and used are highly increasing. This makes it absolute necessary to break down such complex application, instead of a monolithic application so that product deployed is much more maintainable, scalable, reliable, independent of failure of other functional components of the application, portable, easy to deploy on different cloud platforms, and so on. Most of these suit of best practices called “Twelve-Factor Apps” can be bought into production of Software applications using technologies such Docker, Kubernetes and Microservice architecture [182]. In this project, an application is built using a combination of these three technologies. In the below

subsections, these concepts are briefly introduced to better understand the working of the application.

42.2 DOCKER

Modern applications are often built using different technologies with different versions based on the application requirement. Deploying many such application on a single Operating System could be a huge risk at times especially when there is a conflict in dependencies between applications. For instance, if you consider installing 2 versions of Nginx on the same OS, there would be a conflict with the namespace, network port making it logically cumbersome to install same software with 2 different versions. Although the concept of containerization is old, the ability to package applications with their dependencies into the container, making applications independent and isolated from other applications is huge progress to deploy applications, this is achieved with Docker container technology. Moreover, it is very easy to run multiple containers using Docker. The docker image which is a snapshot of the container is the basis to build a container. The Docker container image is a packaging format that contains all the dependencies necessary for the application along with required initial steps such as setting environment variables, or even running command to start the application. These images are hosted on public or private repository such as docker.hub, google storage. There are 2 ways to build a docker image, first is to build Dockerfile, and the second way is make changes on previous docker image. In most cases, a Dockerfile is built on a base image that is useful for the application and then all the required dependencies, commands are added. Once the docker image is ready it takes simple commands to build the container, push the final docker image to the storage repository, and pull the docker image whenever required. Thus, it is easy and robust to create, distribute and run applications using Docker Containers with docker images and docker command-line tools. There are less or no restrictions for docker usage, as containers can be built on any machine with Docker installed it is highly in use by DevOps [183].

42.3 KUBERNETES

Although Docker makes it easy to deploy and run applications using container technology when it comes to application configuration, service discovery, managing updates, secrets management and monitoring containers on the cluster, a better technology is required to leverage all of these tasks. Here comes Kubernetes, an open-source platform that provides a high level of abstraction and orchestration of containers deployed on one or more clusters, which in turn are treated as a single logical machine. Usually, a cluster has single Kubernetes master nodes that keep on running despite explicitly deleting, and zero or more worker nodes [184]. The master node is responsible for managing the cluster, whereas the worker nodes work like a VM, it consists of one or more pods, Volume, network ID and tools to handle container operations.

A pod is the smallest unit of Kubernetes and it consists of one or more containers. All the containers in the pod have shared the same filesystem and IP address, this makes the communication between containers in a pod easy. Each of these pods created based on the scheme which is usually in YAML or JSON file format. The scheme covers important aspects of spec which specifies the Pod behavior, container name, container ports. A pod without Services or Replication Controller cannot be accessed by the external client, neither scaling and distribution of the application are possible [185].

Services provide an external interface for one or more pods. The Service schema definition has 3 important parameters: kind, metadata, and spec. The kind is set to Service to indicate a Kubernetes Service, which is deployment, pod in case of Kubernetes deployment, pod definition files. The label app and the name constitute the metadata. The spec mapping includes a ports mapping for port 80 with name HTTP. The selector is the key mapping in the spec and specifies a mapping to be used for selecting the Pods to expose via the Service. Therefore, the service diverts the network traffic to all its pods with the same label as the label selector specified in the Service spec, in a round-robin manner. There are 3 different

types of Service: Load Balancer, Internal IP, Node port. If a Service type is ClusterIP, then the service is accessible only within the cluster via its internal port. Whereas if the service type is Node port then the service is accessible from outside the node port, which further routes the traffic to internal port Cluster IP of the service, that is automatically created. Similarly Load Balancer service type also automatically creates Node port and cluster IP. It gives access for the external user to ping the IP. In addition to this Load Balancer has the responsibility to balance the load between all the Pods in Service [186].

Another important aspect in scaling applications is Replication Controller, which manages the replication level of Pods by setting "replicas" in Replication Controller definition or on the command line with the -replicas parameter. This ensures that number of Pod replicas are running at any given time. If a replica fails or is stopped deliberately a new replica is started automatically. With these 2 crucial features scaling and replication factor, Kubernetes keep microservices up and running all the time. Hence, Kubernetes is production-ready, which provides dynamic container cluster orchestration in real time.

Kubernetes as a cluster manager provides the feasibility for deploying Microservices by breaking an application into smaller, manageable, scalable components that could be used by groups with different requirements; Fault-tolerant cluster in which if a single Pod replica fails (due to node failure, for example), another is started automatically; Horizontal scaling in which additional or fewer replicas of a Pod could be run by just modifying the replicas label in the Replication Controller or using the replicas parameter in the kubectl scale command;

42.4 GOOGLE CLOUD PLATFORM

Google Cloud platform gives the flexibility to scale quickly and handle intense data while having the luxury of not having to maintain the robust infrastructure, servers, networks etc and create business

solutions. It provides Cloud shell, which comes with a package of a command-line tool, temporary VM instance of GEC, and access to Google API with implicit authorization [187]. Also, it supports language such as Python, Java, Go, PHP, and Ruby. Moreover, the command-line tool exclusively supports Cloud SDK gcloud command line tool. The other alternative to Cloud Shell is to download Google Cloud SDK and enable Authorization through some keys.

42.5 GOOGLE API

Google API is a set of application programming interface which allows communication with google services and integration of other services. It is great tool for developers to perform operations and use its features easily, google map API, google Visualization API, good AJAX search are few examples.

42.5.1 Cloud Pub/Sub API

Cloud Pub/Sub API is a message passing product that is highly useful for communication between independent applications hosted on Google Cloud Platform. The concept of Cloud Pub/Sub has 2 endpoints sender and receiver and having one instance cloud pub/sub would allow interaction between many applications. The main advantage of Pub/Sub compared to other messaging tools like RabbitMQ is it is asynchronous and decouples publisher from Subscribers, that is any Client who subscribed as Sender or Publisher can send, Receive messages irrespective of the client on the other side. In this project psq: Cloud Pub/Sub, a powerful, scalable and reliable messaging tool, implemented using Python is used. It has features similar to rq, simpleq and celery. It forms the basis for communication between microservices, especially main application and frontend [188].

42.5.2 Cloud Vision API

Cloud Vision API is the most popular API that Google has till date. It is

very easy and efficient to analyze the content of the image, which has state-of-the art tools for Image detecting features like: face, text, label and document text, web detection. It is further made easy to use, through Cloud AutoML suite. Using Vision AutoML, it is just one click away to upload images and run pre-determined, custom machine learning models. It is built based on Google's powerful technology of learning-to learn, neural network architecture. In fact, building custom ML model is just few steps [189]. First, uploading training dataset with images labeled into google bucket or human-support to label images and the ML model is trained according to the provided dataset. And then test data is passed, and accuracy of prediction, classification of test data set is determined. However, this feature of Cloud AutoML is accessible to only limited customers, but the basic feature of labeling the images such as data in Google is quite possible through REST API and are available to use in different programming languages [190].

42.6 REDIS

Redis is open source in memory database and is useful as database, cache and message broker. It has different data structures, remote, persistent and scalable to address wide variety of problems [191]. The redis-cli command line tool allows to run command and get details of the clients link to redis-database, configuration and monitor command allows to check how server is responding to requests from client to read or write data. The redis-py package is a python implementation to interact with Redis storage. With popularity of Kubernetes, The Redis-Enterprise version is exclusively built to use with kubernetes. The documentation provides easy steps to create Redis cluster and link with one or more pods running on the kubernetes. All the features of kubernetes to replicate, scale and assign port are leveraged to Redis-cluster, this gives immense opportunity to work with multiple pods, connecting to one-single redis cluster, along with a private redis container on each of the pods. Another main reason for extending redis to kubernetes is the increase in usage of its docker image hosted in docker.hub public repository.

In this project StrictRedis class of Redis is used to connect with Redis backend storage. It gives access to simple functions like sadd,smemeber,sget to get and set key-value pair objects in-memory database of redis [191].

42.7 YELP DATASET

Yelp provides an open-source dataset for the challenge with students and university grads. The yelp data set is huge of nearly 2.66 gigabytes of dataset comprised of all the text details present in yelp, this may include all kinds of business and their details. Apart from this, yelp also holds photos dataset which is 7.50 gigabytes of compressed dataset, which are purely images of count nearly 200,000, for purposes like image analysis, apply machine learning and computer vision technologies. The text-based dataset is usually in JSON/ CSV, SQL format that can be downloaded from their website. In addition to this Yelp also gives access to their data through Yelp-Fusion. Yelp Fusion provides REST API to get access to search, business, metadata. This reduces lot of overload of downloading unnecessary data and gives the opportunity to choose selectively required parameters and leave the rest. This is advantageous not just in terms of space, but also the time taken using REST API is far more efficient and quick compared to the downloading such huge dataset and handling them again. Therefore, in order to make use of these REST API Authentication is required, which is recently modified to the Private Key authentication method, which is a simple 2 step process. Create an account, create manage app, fill in details and the private key is generated [192].

42.8 REST API

Representational state transfer (REST) is any interface between systems based on HTTP properties, web services to obtain data and send data. It is an architectural style that is as simple as function calls and it is adapted in various applications. REST API is of course central theme of this project. The main reason it is widely used across many

programming applications is it can be easily used in sync with the other functionalities of the application. On the other hand REST API doesn't need any software or particular usage of protocol, it simply works with HTTP and many other protocol. In order to use for a specific server, the package to authenticate that server would need to be installed.

There are 2 important aspects to the REST API is request and response object. The request sent to the API usually includes the http url that serves as an endpoint to client or server and the parameters, header details that has to be sent according to your need. Whereas the response object consists of data that is returned upon the request call, is usually in some format such as XML, JSON format. JSON is Javascript Object Notation is a data format that is extensively used to expose API. This can be easily imported via package json. There are many popular function that can convert any data into JSON and vice-versa. Two of the popular python based JSON encoder and decoder are JSON dump and JSON load. json.dump() function takes an json object and convert string-type data and json.load() converts data, from file-like objects and convert into JSON object. The response object is further stored and manipulated according to the requirements. The general requests are GET, POST, PUT, HEAD, DELETE, PATCH. In this project, mainly GET, POST request listed and explained briefly.

- **GET Request** GET Request is used to request data from a resources, and are stored in cache and so they have length restriction which makes it necessary to send limited data via GET Request. Hence a query string sent in the URL of a GET request has carefully chosen parameters that are required. The data received from the server with GET response is stored in response object which could be of JSON format.
- **POST Request** POST request is used to send data to the server, the data need not necessarily be the data that has to be stored in the server, rather it could be credential details, or simply acknowledgement. POST request are not cached and

hence no restrictions on data length. The data sent to the server with POST is stored in the request body of the HTTP request.

42.9 APPROACH

The main aim of the application is to label photos from Yelp dataset retrieved on passing location and search term such as food, dinner, using cloud vision API. The application is divided into 3 microservices frontend, backend, and mainapp. Each of these functionalities is explained below along with initial setup.

42.9.1 Initial Setup

As mentioned above, the application requires 2 important API Cloud Vision API and Pub/Sub API, which have to be enabled for the specific project id, the application can be started, in google cloud console. The best part for a software developer to test the working application is to launch directly using gcloud command-line tool, as it doesn't require authentication setup. For Cloud SDK installed on the local environment, setting up the authentication is crucial. For this, it is first required to create a service account and download service account key which is usually in JSON file format. Then set the environment variable `GOOGLE_APPLICATION_CREDENTIALS = [PATH]`, where `PATH` is the file path of the JSON file downloaded from Google Console Dashboard.

After setting the environment variable, it is important to activate gcloud command line tool, with the command "gcloud init". Then it is required to setup the cluster to run the project. The cluster is built on Google Cloud, for billing is activated on your Service Account. To setup the cluster, it is required to choose the compute zone. The command "gcloud config set compute/zone <zone-name>" sets up the particular zone you would want your cluster to locate at. This is important because it provides less latency when connecting to the cluster from gcloud, or via Cloud SDK. The zone-name would like east1-b, central1-a and so on, and it is better to choose according to location, although the features offered doesn't differ much. The

actual cluster can be created using the command “gcloud container cluster create <cluster-name> –num-nodes <num> –scope cloud-platform”. This command specifies cluster name and the number of nodes created for each zone of that cluster. After creating a cluster, the cluster should be authorized with the service account, in order to get access to all the api's. Hence the command " gcloud container clusters get-credentials <cluster-name>". If the cluster is perfectly created then you should be able to get the correct information for the cluster created by activating "kubectl cluster-info". Since initially, there are no pods, services, deployments created on running the command kubectl get pods you find no pods created. The execution section give more details on how to run the project.

42.9.2 Frontend Microservices

The frontend of the application plays a key role as the load balancer service for the entire application. It is basically a dynamic web-page, which allows the user to enter location, for example, San Francisco, CA and term like food, dinner. Based on these inputs, photos are fetched. The technologies used for the web-page along is python, html,javascript and css. The below paragraph explains breifly about each of these technologies and how they have been used in te frontend of this application.

- **Python** Python web-development framework Flask is used to make server calls to storage.py. The flask is an opensource tool which is an easy to use tool for server-side scripting. It can be installed using pip install flask command and imported in python via Flask package. Because it is a micro framework it can be implemented on top of any backend service with no restrictions like particular tools,libraries, or extensions are required. Moreover, Flask supports RESTful requests for dispatching GET,POST. Hence it is very useful in our project to easily make use of API to communicate and receive from backend.
- **Javascript** JavaScript is a core-technology mainly used to

build dynamic web-pages. There are different frameworks such vue.js, AngularJS, ReactJS, to design creative and interactive web pages. In this frontend javascript is extended via a material design file in storage.googleapis.com. This really reduces the time, effort and gives world-class view for the webpage.

- **HTML** HTML Hypertext Markup Language is useful to add content to the webpage. To make the webpages dynamic and add style to the page, Javascript, CSS are used. The HTML is the base of the webpage no matter any web-technologies out there, they must be added on top of HTML base page.
- **CSS**

Cascading Style Sheets (CSS) is used for styling the HTML webpage, with respect to display and it is added either internal with `<style></style>` tags or external using CSS file. The CSS for this project is taken from google style package, googleapis for fonts, material icons and for other design can be selectively picked from material designs stored at google storage.

In this frontend, the main.py imports flask and it is the main program that connects the frontend and server calls to the storage.py where StrictRedis of REdis is imported and labels, respective imageurl stored are retrieved. The received data from storage.py are then passed on the web page via `render_template()`, this will send the html page to view and also passes the data required. There are 2 views enabled in the webpage. The initial view consists of 2 input text box for user to enter location and category. This information is routed to `get_yelp_images()` function in mainapp via pubsub client. This enables the whole process of extracting photos of the top 10 business from the search results based on the location and term and then store their urls with annotations detected using vision api, in redis storage. After that process ends and the redis storage now has the relevant data, the 2nd view is activated with the labels and images as the storage.py functions in the frontend responds. Thus each of the services are very

much linked to the frontend service, it is like the start and endpoint of the project. Once the images with labels are loaded, how many ever times you refresh you don't see a change in the display because the data retrieved via storage.py doesn't change. As the frontend service is deployed as a load-balancer service an external IP is provided which enables user to access outside of the cluster through a web-browser.

42.9.3 Backend Microservice

The backend of the application is the storage service through Redis which is important for storage of images and their respective labels determined using Cloud Vision API [191]. The redis is accessible through redis image specified in backend.yaml file. The data is stored into redis instance via Mainapp and retrieved in frontend in order to populate the page with resulting images-label pair.

42.9.4 Mainapp Microservice

The mainapp provides the actual functionality of the application, starting from scraping the data to generating the desired output. The major functionalities involved in the service is divided into main.py, yelp_label.py, vision.py and storage.py. Each is further modularized with function, which are briefly explained in below paragraphs.

The `yelp_images.py` has 4 functions `query_api()`, `get_business()`, `search()` and `request()`; As the given location and term are passed to `query_api`, it sends the information to search function. The search function sends a GET request (`GET https://api.yelp.com/v3/businesses/search`) with parameters term, location and search limit equal to 10 in the json object. The JSON object response for the GET request is a dictionary type and is returned to calling function. The `query_api()` stores each business list into businesses dictionary. This business list consist of several other parameter like rating, price, id, categories, location details, reviewcount and so on. But yelp-fusion offers an exclusive option to get more details of each business via GET Request (`GET https://api.yelp.com/v3/businesses/id`) with business id. Therefore, for

each business, business id is extracted and passed to `get_business`. In this function, request is sent with business path `https://api.yelp.com/v3/businesses/` and business id is appended to form the appropriate url for the request. The response object for this request consists of details specific to that business only, which includes open hours, photos and reviews. For this project, we require only photos of each business, hence the urls of the photos for all the businesses are appended to a list. All the photos are returned to the calling function in `main.py`.

In `vision.py`, the authorization to access Vision api is set through Google Credentials python package. The authorization done at this step isn't specific to vision api rather it is common to access almost any Google API. Now, for image passed to the `detect_labels()`, a post request is sent to Vision API to annotate the image, by particular mentioning the label detection feature and the the image url for that image. It is important to specify the feature, or any further details of the image as it imporves Machine Learning model for the analysis of the photo by adding weightage to each of the parameters. Along with the label which is a description parameter of string type, response object consists of details of the image such as score, confidence, location, and so on. The label annotation is return to the calling function for the sent image url.

In `storage.py`, `StrictRedis` class is imported to instantiate redis object. The redis in memory storage is very useful because of it's ability to store objects in key-value pair. Taking advantage of this all labels are stored with labels as key and label name as the value. Also for each label as key the image url is stored as value . This makes very easy to retrieve data in the frontend service,by just simply looking for the associated imageurl for the label in the list of labels [193].The `add_labels` function stores all the annotated label for photos into a labels list and `add_image` fucntion stores label and photo as key-value pair.

To summarize, `main.py` brings together all the above functionalities, it retrieves the data from `yelp_images.py`, passes photos to `vision.py` to label each one of them and stores using `storage.py`. As pub/sub enqueues

the whole process in main.py, once the task is done, the frontend gets triggered.

42.9.5 Pods, Services, Deployments

There is one yaml file for each of the microservice, which includes schema for service as well as deployment. The kubectl create -f <file.yaml> command included in the Makefile creates the pods, services and deployments. As discussed above the important parameters in general are type, label selector, replica, container image and the port where they are exposed. In this application the frontend service is created as type load balancer. This exposes the service outside of the cluster through an external IP. Once each of the pods are deployed, the application is production ready. The scalability is maintained with replica factor, which ensures deletion or failure of one or 2 pods doesn't stop the application from running. Also the constant updates are made easy with rollback.

42.10 EXECUTION

The execution of the application is incorporated through Makefile. Just by running make all inside yelplabel directory, will spin the docker container images, creation of pods, services and deployments. By executing the command "kubectl get pods" list all the pods created and "kubectl get services" will show the services up and running with their given IP address. The external IP of the frontend service, is available to curl or run in the browser. This is also the load-balancer for the application, so on request the traffic is diverted to other services within the internal port and the entire application is up and running. In order to delete all changes you can run the command, where as in order to delete services, pods and deployments you can run the command "kubectl delete -all" pods, services, deployments.

42.11 CHALLENGES

As there are lot of advanced technology out which share same sort of features it is debatable to choose the right technology stack in order to develop an application, that atleast meet the requirements and development practices mentioned in 12app-factor. For instance, this application can be implemented with Google Bucket Storage instead of Redis as it also satisfies with key-value kind of storage system. The debug aspect is the challenge because of the layers of abstraction over the microservices, the deployments and everything works fine if there exists an error within one of the microservices, which gets hard to figure out at times. To an extent we can make use of kubectl logs and google cloud provide error report.

42.12 RESULTS

The results of the project is to display labeled images from Yelp photos dataset. And this achieved by populating the browser with the label and imageurl pair retrieved from redis. As the cloud-vision api is a pre-trained model on huge dataset of Google, the label detection is done, in less amount of time and with high-accuracy rate in terms of average score is 0.75. For this application, only photos of top 10 business for the search results are chosen and the image.annotate request is reiterated for 3 times to make sure best label suitable label is detected for the image, if not in a single request.

42.13 BENCHMARK

Deploying the application is made easy with the use of Docker and Kubernetes. The Makefile and Docker file included installs all the necessary dependencies to creating pods, services and deployments. This takes around 2-3 minutes to generate the external IP address. The runtime analysis of the application, depends on the dataset volume for the given input, as a result it takes few minutes for the label-detection and showcasing the results on the browser. The benchmark for this project extensively depends on Cloud Vision API and Redis Storage. As Redis operates as in-memory database with key-value storage it servers the applications purpose in

retrieving, storing labels, imageurl in efficient amount of time. Whereas the Cloud vision API although it has good reputation to show accurate results, the batch processing for the images is limited. Moreover, the challenge with yelp photos is to able to distinguish between different varieties of food category, for instance.

42.14 CONCLUSION

Thus application to scrape data from yelp-fusion API and detect label using Cloud Vision api, which is neatly displayed on a browser with the support of redis storage technology, follows MVC architecture workflow which is important factor in application deployment. With Kubernetes not just orchestration of docker components but the flexibility, scalability for the deployment of microservices is highly achieved.

42.15 FUTURE SCOPE

this is not an experience report. If you do a future scope we will keep the incomplete till you finished it. Otherwise the section is called limitations or something like that

I attempted to make use of Cloud AutoML, Vision API to label MNIST dataset as a part of this project, but unfortunately the cloud AutoML gives access to specific user to customize the pre-trained ML model based on Google Cloud Vision.

We do not understand this limitation

This is a huge dataset of NIST authorized handwritten dataset, highly used for testing the accuracy of ML models for Computer Vision and Image Analysis. Using the same or more technology stack, I would like to work on MNIST dataset to detect and label handwritten images as side-project, irrespective of my class schedule, during upcoming semester with the guidance of the professor Dr.Gregor von Laszewski.

certainly you can do this, you will have to take an indepoendent study. Its like the class with 100% focus on report.

42.16 ACKNOWLEDGEMENT

The authors would like to thank Dr.Gregor von Laszewski for his support and suggestions to write this paper.

43 CMD5 PLUGIN TO CREATE A DOCKER SWARM CLUSTER ON 3 RASPBERRY PIs

please update to markdown

see our example <https://github.com/cloudmesh-community/proceedings-fa18/tree/master/project-report>

the original latex had structural issues so please visit the section heading levels of all sections.

43.1 ABSTRACT

Information technologies are evolving from mainly one-host environments to more distributed environments. Docker Swarm makes it possible to avoid having a single point of failure and instead, have multiple nodes that can be properly balanced and contain replicas of the information. Currently, Docker must be individually downloaded, installed and configured on each physical computer in order for the desired computers to work in swarm mode. This paper details the development of a plug-in that would allow CloudMesh to deploy a Docker Swarm cluster. The creation of this plug-in would be the first step towards the development of a tool which would allow larger debian based networks to work as container oriented virtual environments with optimized usage of resources.

Author:

- Andres Castro Benavides
 - Uma M Kugan
-

Keywords

Raspberry Pi, Cloudmesh, CMD5, Big Data, i523, HID709, HID710

43.2 INTRODUCTION

43.2.1 Docker: Swarm mode, Current Use, Installation and Configuration

Docker is the technology used for containerization for software development. It is an open source tool which makes it easy to deploy applications. Applications are packaged in containers and then it is shipped to all the platforms that is supposed to work with. Applications are divided into manageable sizes and all the dependent functions are added and individually packaged. Both Linux and Windows are supported by Docker.

Docker Swarm is a clustering and scheduling tool for Docker containers. A swarm is nothing but multiple Docker hosts which run in swarm mode and act as managers to manage delegation and workers will run swarm services. A given Docker host can be a manager or a worker or it can perform both roles. If any of the worker node becomes unavailable, manager schedules that node's tasks on other nodes. A node is an instance of the docker engine participating in the swarm [194].

A swarm is made up of multiple nodes. We need to execute "docker swarm init" to enable swarm mode and to make current machine a swarm manager, run docker swarm join on other machines to add them to the swarm as workers and run docker node ls on the manager to view the nodes in this swarm.

Docker Swarms are used to orchestrate processes, optimizing the use of resources across clusters. In other words, the use of Docker Swarms allow individual computers to work as a cluster, sharing their RAM, processors, physical memory, among other features or abilities. The docker, when used in swarm mode, evaluates the assets across the network and manages tasks in real time. Each computer can contribute its assets to complete tasks in the most efficient way. It is dynamic and adapts based on the available resources and current

demands.

In order to set up a Docker Swarm, there needs to be direct access to each machine that will be used as a node which is how we will call an instance of Docker that will be part of the swarm. In order to set up the nodes, the docker must be independently installed and configured on each machine. Then, each machine must be added to the swarm, allowing it to communicate or interact with the other nodes.

This processes not only requires human resources like technicians working on installation and configuration, but also demands these actions be repeated manually on each individual node or manager. While this can be done virtually, it still requires individual attention in the setup of each machine. In order to optimize the setup of Docker Swarms, CloudMesh could be utilized to centralize installation and configuration of every node and manager.

43.2.1.1 Inside Docker

The four main internal components of docker are Docker Client and Server, Docker Images, Docker Registries, and Docker Containers.

43.2.1.2 Docker Client and Server

The docker server gets the request from the docker client and then process it accordingly. Docker server and docker client can either run on the same machine or a local docker client can be connected with a remote server running on another machine [195].

[Docker Architecture] [196]](<https://github.com/cloudmesh-community/hid-sp18-709/blob/master/project-markdown/images/High-level-overview-of-Docker-architecture.png>)

43.2.1.3 Docker Images

Base image are the Operating system images such as Ubuntu 14.04

LTS, or Fedora 20 which creates a container to run Operating system. The docker file contains a list of instructions to build an image. When using docker, we start with a base image, boot up, create changes and those changes are saved in layers forming another image [197].

43.2.1.4 Docker Registries

Docker images are placed in docker registries. It is same as source code repositories where images can be pushed or pulled from a single source.

43.2.1.5 Docker Containers

Docker image creates a docker container. Containers have everything for the application to run on its own.

43.2.2 Benefits of using Docker

43.2.2.1 Open Source Technology

The Docker containers are based on open standards which means that anyone can contribute to the Docker tool and at the same time customize it for their needs, if the features they are looking for is not already available.

43.2.2.2 Portability

Docker makes distributed applications to be dynamic and portable which can be run anywhere which makes it extremely popular among developers.

43.2.2.3 Sharing

Docker is integrated with a software sharing and distribution mechanism that allows for sharing and using container content which helps the tasks of both the developer and the operations team.

43.2.2.4 Elimination of Environmental Inconsistencies

Any changes made in one environment will be shared across other environments or all the applications can exist in the single environment.

43.2.2.5 Resource Isolation

Resource isolation adds to the security of running containers on a given host. Docker uses Namespaces technology to isolate work spaces called containers. Namespace is created when container is run and access is limited to that namespace only. Every container in Docker will have its own work space which makes it easier debug if there are issues with any particular container.

43.2.2.6 Easy Integration

Docker can be easily integrated into a variety of infrastructure tools like Amazon Web Services, Ansible, IBM Bluemix, Jenkins, Google Cloud Platform, Oracle Container Cloud Service, Microsoft Azure to name a few.

43.2.2.7 Better Security

Docker provides a interface for developers and IT teams to define and manage their security configurations for applications as it navigates from one stage to another.

43.2.2.8 Docker - Use Cases

The Docker platform is the only container platform to build, secure and manage the variety of applications from development to production both on premises and in the cloud. It also creates room for innovation, increases time to market, highly agile. Docker supports diverse set of applications and infrastructure for both developers and IT. It transforms IT without having to re-tool, re-code

or re-vamp existing applications, policies or staff [198].

43.2.2.9 DevOps

The main goal of DevOps is to eliminate the gap between the developers and IT operations team. Docker with DevOps get the developers and operations team to work together so that they both understand the challenges faced by each other, apply DevOps practices [198].

43.2.2.10 CI/CD

Continuous Integration and Continuous Deployment are the most common use cases of Docker. Continuous Integration testing and Continuous Deployment allows developers to build codes, test them in any environment. Docker integration with Jenkins and GitHub making it easier for developers to build codes, test them in GitHub and trigger a build in Jenkins and adding the image in Docker registries [198].

43.2.2.11 Docker Containers As A Service

Docker help any organization to modernize their application architecture. It can deploy scalable services securely on a wide variety of platforms, improving flexibility and maximizing capacity. Best use case for Docker installation is the US Government where they enhanced their applications and made their components and services of their system and easily transportable/shareable with other agencies within the government [198].

43.2.3 Docker - Services:

43.2.3.1 Docker Engine

Docker Engine is the foundation for the application platform which is used for creating and running Docker containers. It is supported on

Linux, Windows, Cloud and Mac OS. It is lightweight, open source and integrated with a work flow to build and containerize applications. User interface is very simple and it makes the environment easily portable from single container on single host to multiple applications on a many number of hosts [198].

43.2.3.2 Docker Enterprise

Docker Enterprise provides an integrated platform for both developers and IT operations team where container management and deployment services are together for end-to-end agile application portability. It is easy to manage, monitor and secure images both within the registry and those deployed across various clusters [198].

43.2.3.3 Docker Hub

Docker Hub functions as a hosted registry service that helps you store, manage, share and integrate images across various developer work flows. Integration testing is done each time when the image is shared [198].

43.2.3.4 Docker Compose

Docker Compose is a tool that developers deploy to define and run all multi-container Docker applications. Single host can be used to isolate multiple environments, even if they are of the same name. Data volume is copied automatically from old container whenever a new container is created. Compose uses the previous configuration to create the new container which reduces the time for replicating the same changes to the environment [198].

#@ CloudMesh

CloudMesh is an innovative tool that allows communication and interaction between cloud based solutions. Not all clouds are docker based and there are different types of virtual and cloud

environments. Through CloudMesh, data can be shared and utilized by cloud solutions that are not otherwise programmed to communicate with each other. Cloud mesh does not just manage a series of clouds, but centralizes and deploys them as one main system that manages the data resources.

Quote von Laszewski:

Cloudmesh is a project to easily manage virtual machines and bare metal provisioned operating systems in a multicloud environment. We are also providing the ability to deploy platforms.

43.3 CREATING CLOUDMESH PLUG-INS

##What it currently does and has the potential to do:

By creating CloudMesh plug-ins, it is possible to extend its potential from different kinds of cloud based environments interconnection to deployment of a container management system, in this case, Docker.

Utilizing CloudMesh to Centralize Docker Swarm Installation Cloud Mesh does not have a plug in that allows you to deploy container solutions on physical networks. Create a plug in that would allow Cloud Mesh to deploy container solutions, in this case, the Swarm mode of Docker, to a physical Debian based network, in this case, a series of raspberry pies. Could be used as a model to deploy other types of container oriented solutions. It is taking a simple network. Debian based network and allowing it to centralize resources and assigning tasks and optimizing different functions by installing a container management system, called Docker Swarm.

In order to simulate the deployment of a Docker Swarm cluster, this Cloudmesh project develops a Cloudmesh plug in, that deploys a Docker Swarm cluster on three Raspberry Pi, allowing them to be part of this multi cloud environment.

The cloud mesh allows you to use Methods to deploy the Docker

Swarms as container management tools, to the raspberry pi's.

43.4 RASPBERRY PI AS PLATFORM

The Raspberry Pi is a credit-card-sized computer with ARM processor that can run a Linux desktop operating system. Raspberry PI can plug into TV and a keyboard. It is a little computer which can be used for many of the things that desktop PC does, like spreadsheets, word processing, browsing the internet, playing games and also to play high-definition video. Raspberry is not intended to replace personal computer as its OS support, memory etc are limited when compared to Laptop [199].

43.4.1 Differences between Laptop and a Pi

Raspberry Pi uses an ARM based processor like ARM Cortex A7 or A53 depending upon the model while the traditional PC/Laptop uses a conventional x86 /x64 Processor from either Intel or AMD. Embedded systems had low cost and low power requirements and since ARM processor used in the Raspberry Pi is used in embedded systems, A raspberry pi consumes very much less power than a laptop. The processor is also much slower than most Intel/AMD processors used in PCs, so complex programs can not be executed. Pi does not have any wireless networking capability like WiFi, Bluetooth etc when compared to laptop. Pi comes with 1 GB ram for version 3 while most laptops have 2GB/4GB RAM that can be easily expanded to 16GB. Laptops can have secondary storage for about 1 TB. It also supports Flash based storage which tends to be more expensive per bit than traditional Magnetic Hard drives. Therefore the Raspberry Pi will have a smaller storage capacity than a traditional PC. A binary built for either system will not execute on the other. Images or binaries that was not created by you or from true source may pose a potential threat. Docker swarm cluster can be built easily on Raspberry Pi with just two basic commands: swarm init and swarm join [200].

43.5 DOCKER AND BIG DATA PLATFORM

It is always been a challenge to maintain or even to have a control deployment environment. It is very difficult to identify any issues without proper deployment environment. Most of the times, issue can be fixed as simple as disabling a service or just uninstalling a software or slightly tweaking the environment. This can be easily achieved only when we have complete control of the environment [201]. It is very difficult to manage a distributed environment whether in cloud or not. There are lot of manual effort whenever there is an installation across multiple nodes. Docker allows anyone to quickly create, launch and test Docker containers very easily. Container offer lightweight isolation and virtualization, yielding reduced overhead, faster deployments and restarts, and simplified migration. There are lot of frameworks like, Google's Kubernetes, CoreOS, Multi-Container orchestration, etc which comes in handy with Docker and Docker is very lightweight when compared to a Virtual Machine. Even though Docker comes very handy in addressing many of these issues, main selling point is building consistent environments which are very easy to replicate. Especially in big data environment, instead of installing every single component from the Hadoop ecosystem, required for their development or testing environment, we can just create it once and use it any number of times and everywhere. Docker allows usage of different versions on the same tool for different jobs without any conflict. Docker containers are a great way of deploying services at scale and giving isolation to services that run on the same host and improving utilization and we can even use Docker for scheduling batch analytical jobs.

43.6 DOCKER CRITIQUE

Docker was not designed to support the long-running containers that are needed to support production systems. While Docker gets a lot of visibility from the development and DevOps communities, its operational maturity still leaves a big void. There are no logs from containers and hence logging is difficult in a distributed Docker environment. Docker needs separate orchestration, provisioning and

automation [202]. Managing a huge amount of containers is challenging, especially when it comes to clustering containers. Running a container need root access and due to security and governance policy, many companies may not grant root privileges for everyone. In some companies, only software from official/trusted sources can be installed on their machines. Since Docker is not included in Red Hat Enterprise Linux 6, it needs to be installed from docker.com, which is an untrusted source [203].

43.7 METHODS: PROPOSED SOLUTION

The solution was created for a specific type of hardware and software, but is modular enough to be extended to different environments with similar features, such as basic architecture -which include but is not limited to ARM single boarded computers- and an operating system based on Debian, such as Debian, Raspbian, Ubuntu, etc.

43.7.1 Hardware

For the current proposed solution, the different pieces of hardware were chosen based on criteria such as Compatibility and Price.

The following is a list of the hardware that was used and below that list there is a description of each piece of hardware that was used.

- 3 Raspberry Pi
- 3 Micro SD Cards with a capacity of 64 GB
- 3 USB to Micro USB Cables for power supply to the Raspberry Pi
- 1 External monitor for the configuration step only.

43.7.2 Raspberry Pi

For this experiment, the 3 machines that were used were Raspberry Pi 3 Model B. Raspberry Pi are single boarded computers, that come in a small presentation. They have been developed with education

and extension in mind, making them very popular in the academic and entrepreneur communities. The specifications of the model that has been used for this experiment are the following:

- CPU: 1.2 GHZ quad-core ARM Cortex A53 ARMv8 Instruction Set.
- GPU: Broadcom VideoCore IV @ 400 MHz
- Memory: 1 GB LPDDR2-900 SDRAM
- USB ports: 4
- Network: 10/100 MBPS Ethernet, 802.11n Wireless LAN, Bluetooth 4.0

[204]

The Raspberry Pi are interacting with each other using a private wireless network, and they have been assigned static Internet Protocol Addresses. In this case 192.168.1.85, 192.168.1.86 and 192.168.1.87.

43.7.3 Micro SD Cards

Because of its architecture, Raspberry devices require the use of Micro SD Cards to contain the Operative system and other files. They emulate the Hard drive resource used on other kinds of computers. The reason that it is required to have at least 16 GB of memory, is because there will be several pieces of software installed in the devices, each one of them with different requirements:

Docker Memory Requirements [194]:

- 8GB of RAM for manager nodes or nodes running DTR.
- 4GB of RAM for worker nodes.
- 3GB of free disk space.

So at least 12 of the GB would be required for Docker and 4 GB used for the proper functioning of Raspbian. [205]

Taking these requirements in consideration, there should be a

minimum of 16GB of free space in the MicroSD in order to perform this experiment.

The Micro SD cards used were San Disc Memory Cards with a 64GB capacity.

43.7.4 Micro USB Cables

3 USB to Micro USB Cables for power supply to the Raspberry Pi Since these small computers don't use the regular power supply chords, they are equipped with MicroUSB ports to power the device. All of these devices are plugged to a main power outlet that allows to charge multiple devices at the same time. There are other options to power the devices include, such as attaching them to external batteries.

43.7.5 External monitor

Since the Raspberry Pi are headless machines, they require to be accessed directly for the initial set up and after that it is possible to continue the configuration and installation process using any kind of remote access, like SSH or RealVNC. For this initial connection, any kind of screen that is HDMI compatible is useful. In this case the initial setup of the Raspberry Pi was performed on a Toshiba 55 inch HDTV with HDMI port. After that they were accessed from a Laptop computer with Linux Ubuntu 17.10, using Remmina via ssh using XORG.

43.7.6 Initial input devices

In order to set up the devices. The Raspberry Pi will require a set of initial input devices attached to each computer. For this exercise, a USB enabled standard keyboard and a USB enabled standard mouse were used.

43.7.7 Software

43.7.7.1 Raspbian

Currently, the default way to deploy the operating system to the Raspberry Pi is by using an Operating System installation Manager called Noobs -which stands for “New Out Of Box Software”-. This manager can be downloaded directly from the Raspberry Pi website and it includes several Operating system options, among them:

- Raspbian
- Pidora
- LibreELEC
- OSMC
- RISC OS
- Arch Linux

Since Raspbian is the default Operating system and most commonly used, this experiment decided to use it. This is also helpful because there is material available in different websites with instructions on how to install Docker in Debian based Machines. Raspbian is Debian based. Another important reason is that Docker has as a requirement that the Linux kernel version on which it will be installed is 3.10 or higher. The Kernel version of the version of Raspbian that was used is 4.9.

The version of Raspbian that was used has the following specifications:

- Kernel version: 4.9
- Release date: 2017-11-16

43.7.8 Docker

There are several versions of Docker available. Each version with their own advantages and disadvantages. Because of the architecture used by Raspberry Pi -ARM instead of AMD-, the Docker version used is Docker for Debian ARM. With the following Specifications:

- Version 17.09.0-ce

- Release 2017-09-26

This version of Docker is Community Edition, which means that it is available for free and can be installed on bare metal or cloud infrastructure. This flexibility is good for the experiment, because it will be installed on Raspberry Pi, which are considered physical devices or bare metal Machines [194].

43.7.8.1 Prerequisites

There are several reasons to have the pre requisites that the user will find in this document. They will be explained in a separate section. Before using the proposed solution, the user's environment needs to meet the following requirements:

43.7.9 Raspbian Installed

Raspbian must be installed and configured on all Micro SD Cards. For this, the user may download Noobs from <https://www.raspberrypi.org/> and copy it to a formated Micro SD Card. Once the Raspberry Pi has the MicroSD loaded with noobs in place and has the input devices and display attached to it, the user may follow the OS installation guide found on: ><http://raspbian.org/>

It is advisable to be hooked up to the network where the user is planning on implement this solution before running Noobs for the first time. This will allow the user to download newer packages or Raspbian and avoid interruptions in the process.

This requirement exists because there is a function that is being explored to capture Raspberry Pi images to be deployed later on and avoid the present pre requisite, but it is not ready yet.

43.7.10 Update OS repositories

In order to ensure that the user is accessing the latest version available of the software, it is important to update the Raspbian

repositories. In this case, the user can access the Terminal and enter the following commands:

```
sudo apt-get update
```

to update the list of available repositories and then

```
sudo apt-get upgrade
```

to upgrade the available packages. The first time that the user runs one of these commands, the root password will have to be entered. This process might take a few minutes [206].

43.7.11 Remote access setup

Enable SSH on the Raspberry Pi. After Raspbian installation, enable SSH on all your Raspberry Pi machines.

To do this, the user has to add a line in the file `sshd_config` found in the directory `/etc/ssh/`. The line has to go at the end of in the `Authentication` section. It has to contain the following string: `>PermitRootLogin yes.` [207]

43.7.12 Changing hostnames

In order to keep the three Raspberry Pi organized it is highly advisable to assign an exclusive and distinctive hostname to each Raspberry Pi. The three Raspberry Pi have the following static IP addresses:

- 1. pi85 - 192.168.1.85
- 2. pi86 - 192.168.1.86
- 3. pi87 - 192.168.1.87

By default, all Raspberry Pi devices will have the same Host Name.

To change this feature on each machine, the user will have to modify the line that contains `127.0.1.1` and as hostname it includes the string `raspberrypi` in `/etc/hosts` file, in most of the cases it is the last line in the file. Then, the user may type the desired hostname instead of the

word raspberrypi and save the file and close it. This part can be done by using the text editor that comes by default with Raspbian, an editor called `nano`. It is not advisable for the users to modify the rest of the entries, at least as part of this project.

Once the file is modified, the user will have to initialize the hostname with the `hostname.sh` script this can be done using the following line in the Terminal:

```
sudo /etc/init.d/hostname.sh
```

To check if the modification has worked as expected, the user may check the hostname of the machine from the Terminal by running the command: `hostname -I`

43.7.13 Steps Followed

43.7.13.1 Testing shell commands prior to integrations with Cloudmesh

Since Raspberry pi is not currently listed under the supported operative systems for Docker or Cloudmesh, The process of deploying Docker and configuring the swarm Mode was successfully tested on the Raspberry Pi first using the commands that are intended for Debian. Once the Swarm was configured, the three Raspberry Pi devices were left on for over 24 hours and it was not observed any kind of abnormal behavior, like looping services in the OS or overheating.

43.7.13.2 Purchasing the hardware

The different hardware components were purchased via Amazon.com and took anywhere between 2 to 5 days to arrive. The different components can also be purchased through multiple on line sources or local electronics stores.

43.7.13.3 Installing the components via ssh into every node.

The following steps were followed on each device: Usig the TV as an external monitor, the USB input devices: keyboard and mouse, and the Raspberry Pi with Raspbian installed. An ssh key was generated and the device was accessed using Remmina via a XORG connection from a computer equipped with Linux Ubuntu 17.10 Artful Aardvark.

The components were installed in the following order:

Updated the Raspbian packages Installed Python 3.6.2 and Python 2.7.13 via PIP and also Installed Cloudmesh: following the instructions found in: <https://github.com/cloudmesh/> Installed Docker CE ARM via Terminal using the following command:

```
curl -sSL https://get.docker.com \| sh=\" as suggested in https://www.raspberrypi.org/
```

Started the swarm and assigned a master node

```
sudo docker swarm init --advertise-addr 192.168.x.x
```

Created the remaining two nodes, with

```
curl -sSL https://get.docker.com \| sh=\" as suggested in https://www.raspberrypi.org/
```

And then, using the docker swarm join command the token generated when the master node was created, they were added to the swarm.

43.8 INSTALLING AND CONFIGURING DOCKER SWARM

43.8.1 Manager

Since Docker requires at least one computer to be a Manager and Cloudmesh also requires at least one main configured piece of equipment, a Raspberry Pi was chosen to be the main device, in this case, the Raspberry Pi with the IP address 192.168.1.85. The following command was run on the Terminal or that device to set it as the manager:

```
sudo docker swarm init --advertise-addr 192.168.1.85
```

43.9 WORKERS

The other two Raspberry Pi devices. In this case, the Raspberry Pi with the IP address 192.168.1.86 and the one with 192.168.1.86, were defined as simple worker nodes.

To define the workers, the following command was used:

```
sudo usermod -a -G docker USER`
```

and to work as part of the swarm the command used was:

```
docker swarm join --token *** 198.168.1.85:2377`
```

As a last step, it was confirmed that all the nodes were added by using the following command:

```
sudo docker node ls
```

43.10 ADDITIONAL RESEARCH

43.10.1 Other functions considered

Initially, for this case, it was considered an option to developed a function called CaptureImage and a second function called Deploy Raspbian. As their names suggest, the first one intended to capture an image or backup of a Raspberry Pi. This first function would receive the IP address or hostname of the desired machine and the desired location to store the captured image, alongside the corresponding credentials and wrap a **dd** shell command similar to the following:

```
dd if=/dev/mmcblk0 bs=1M ` gzip -QUOTE \| dd of=imageDir\|
```

Among the challenges faced, this line was returning an invalid syntax, most likely because of the use of the variables. Since there was not a lot of time, the team decided to postpone this function.

The second function was called DeployRaspbian and would receive

the route and name where the image would be deployed, i.e. `./dev/bkp` and image name and route, i.e. `~/Desktop/raspbian.gz`. The shell command that would be wrapped would be:

```
gzip -dc diskNm PIPE sudo dd of=imageName bs=1m conv=noerror, sync
```

More information on this topic can be found in the section called **Backup www.raspberrypi.org**.

Among the challenges, there is no clarity on whether the image can be deployed over a lan connection and this point there is not enough time to run tests. Also, since this is a copy of a previously used Raspbian, there is a chance that there might be conflicts related to the IP addresses that might be stored in different files of the OS.

43.10.2 Final code

The final version of the code can be found on:

<https://github.com/cloudmesh-community/hid-sp18-709/tree/master/project-code>

43.11 OTHER OPTIONS CONSIDERED

Other options of coding were considered during the development of this solution. Since all of the deployment can successfully be done via terminal in Raspbian, two main options were considered:

Option 1. A bash script for every part of the deployment and wrap it in python. This option would have been less dynamic and wouldn't make the best use of the available resources, but at the same time it could have been easier to adapt to linux Operating systems other than Raspbian.

43.12 CONCLUSIONS

It is possible to create the plug in. Using the SH sub process included

in python 2.5-3.5. The team was able to try the steps one at a time at the level of py scripts, but encountered an error previously mentioned in this document when trying to implement it as part of cms. Also, as the professor suggested, this same system can be implemented as a different abstraction for deployments such as an abc class similar to the following:

```
class deployment

    def prepare
        prepares installation including downloads and other installs needed

    def deploy
        deploys the package or software

    def configure
        does some configurations

    def test(test)
        does a test wheer a name is passed of a test (you could have multiple)
        the name all woudl be running all tests
```

Since most of this work was working with bash commands tunneled through python scripts and implemented in CMS, Once this is fully functional, it is very possible that the same methodology can be followed to add more layers of complexity, i.e. Kubernetes.

It would be important to consider that the fact that the passwords would have to be either hard coded or transferred in plain text has to be seen as a vulnerability, that has to be addressed either by adding an encryption/decryption module or finding another way to safely access the root of the target device.

The authors would like to thank Dr. Gregor von Laszewski for his support and suggestions on this project.

43.13 WORK BREAKDOWN

- Uma Kugan 50% of the document and codeing.
- Andres Castro Benavides 50% of the document and testing and reviewing the code
- Gregor von Laszewski significant helo with markdown, suggestions to the report, structure, correction of some issue,

contribution of all the cm5 module on which this project relies, transition to markdown

44 NEW APPROACHES TO MANAGING METADATA AT SCALE IN RESEARCH LIBRARIES HID-SP18-705

Timothy A. Thompson
timathom@indiana.edu

Indiana University
hid: hid-sp18-705

github: [cloud](#)
code: [cloud](#)

as you can see from the epub your section headings are wrong
make sure they show up in the TOC correctly

urls in text must be replaced with bibtex entries

are last two figures self created? or do they need citations.

Keywords: i523, Research Libraries, Library Catalogs, Blockchain, BigchainDB

44.1 ABSTRACT

The analysis of big data often relies on distributed storage and computation; however, access to big data—and to the platforms capable of managing and processing it—continues to be largely centralized. Centralization is particularly evident in the case of the metadata produced, managed, and disseminated by academic and research libraries. Libraries typically create and share their catalog records by uploading them to a centrally managed database, which can then be searched by other libraries for records that can be copied and added to an institution's local catalog. This centralized approach, which operates on the basis of membership fees, has the advantage of scalability and availability, but it comes at the cost of a loss of autonomy. Although technical innovation is possible within the

current paradigm, the growing maturity of peer-to-peer protocols and decentralized solutions points toward an alternative approach, one that would allow libraries to share their data directly without having to pay an expensive intermediary.

44.2 INTRODUCTION

The problem of entity resolution (also known as record linkage or data matching [208]) is one that has a direct impact on the work of information professionals in research libraries. In library units responsible for catalog management, many workflows center on a procedure known as copy cataloging, which aims to expedite the processing of new acquisitions. Copy cataloging involves searching a shared database for records created by another cataloging agency, but that describe identical publications that have been acquired by one's local institution [209]. In the current environment, a single company, the Online Computer Library Center (OCLC-<http://www.oclc.org>), is the only viable platform for global cooperative cataloging [210].

OCLC provides data aggregation and warehousing services that allow libraries to effectively share their data, but its business model does not encourage peer-to-peer interaction and innovation among individual libraries. This vendor-driven paradigm entails the acceptance of a business model that, in effect, charges libraries for serving their own data back to them, with some added value through quality control and normalization. Once a library's data has been sent to OCLC, it also becomes subject to potential licensing restrictions, as well as the expectation that future dissemination of the data will include attribution of OCLC [211], [212].

44.3 NEW APPROACHES TO METADATA MANAGEMENT

Libraries have a tradition of experience with record matching and automation [213], but now stand to benefit from the increasingly mainstream availability of algorithms and routines developed within

the context of data science and machine learning. Sophisticated algorithms for string comparison and probabilistic record linkage have long been available, but are not widely used by libraries, with the exception of large-scale projects such as the Social Networks and Archival Context Project (SNAC) (<http://snaccooperative.org/>) and the Virtual International Authority File (VIAF) (<http://viaf.org/>), itself a project of OCLC. The former has employed methods based on Naive Bayes classification algorithms to aggregate and disambiguate data from across a wide range of libraries and archives (the reported accuracy of the approach fell with the range of 80-90 percent) [214]. More recent approaches to record matching have improved on probabilistic methods such as Naive Bayes by using Artificial Neural Networks, improving accuracy rates in some cases to 98 percent or more ???.

As machine learning tools and methods have become more accessible, however, large-scale, real-time access to library metadata has not necessarily followed suit. The catalog of a large academic library may contain around 10 million records [215]. By comparison, as of August 2018, the OCLC catalog database, WorldCat, contained 427,501,671 bibliographic records in 491 languages [216]. As long as service providers such as OCLC maintain centralized control over the aggregated metadata of research libraries, large-scale computational analysis—and the innovation it could produce—will remain proprietary and locked away.

The situation is further complicated by professional and cultural norms within libraries. Although decentralization may be appealing as an ideal, librarians who manage bibliographic metadata are also immersed in a discourse that centers on the idea of control: they use terms such as authority control, controlled vocabularies, and intellectual and physical control of collections [217]. The idea of control is closely related to the idea of trust: when workflows and systems are centralized, it becomes easier to enforce norms and standards, but it also becomes more likely that potential contributors may be excluded, especially when they are unable to afford the price of membership in a proprietary system.

New decentralized technologies such as blockchains could allow research libraries to form robust peer-to-peer networks that would enable data sharing on a larger scale. Public blockchains such as Ethereum and Bitcoin are limited in the amount of data that can feasibly be stored on chain, but alternative platforms that address this limitation have recently emerged. When discussing decentralization, there are a range of new technologies that should be considered, and blockchain may or may not be the most appropriate for a particular use case—or blockchain technology may need to be used in conjunction with other technologies in order to enable decentralized exchange. Several efforts are underway to develop systems for decentralized file storage using distributed hash tables, one of the most prominent being the Interplanetary File System (IPFS) (<https://ipfs.io>). In a way similar to the software versioning protocol Git, IPFS uses hash values to capture the state of a file at a particular point in time and then serves it on a peer-to-peer network. IPFS hashes might be referenced as links in blockchain transactions in order to decouple the storage layer from the accounting layer [218].

In this regard, the blockchain-based database service BigchainDB, implemented in Python, provides a robust storage data solution while preserving the benefits of blockchain features such as data immutability and an asset-based transactional model. By running a consortium blockchain network of BigchainDB nodes [219], libraries could be empowered to abandon centralized models and begin managing their data collectively.

44.4 BLOCKCHAINS FOR RESEARCH LIBRARIES

Some in the library profession have been skeptical of blockchain applications for their domain, arguing that they have been overhyped as a panacea, when in reality they are nothing more than slow, expensive, append-only databases [220]. Even core developers working to support the Bitcoin blockchain have argued that the constraints imposed by blockchain technology, such as immutability and decentralized consensus, make it appropriate for a very limited

set of applications—namely, currency and the exchange of value [221]. For individuals and organizations who are investigating blockchains as a technical solution, it is important from the outset to establish a framework for evaluating their applicability and appropriateness [222].

For example, a blockchain-based solution may be appropriate in a scenario in which there is a lack of trust among participants, or in which processes and collaboration would be more efficient if the need for trust were eliminated [222]. In the case of a shared catalog for research libraries, trust is an issue because not all participants can be trusted to provide data that conforms to expected levels of quality. A commercial, centralized solution mitigates these concerns by requiring participants to pay a membership fee. A blockchain solution addresses issues of trust by enforcing a decentralized consensus mechanism, which may take different forms, but which is designed to ensure that participants can trust the network to maintain a consistent state across all transactions [223].

The Proof-of-Stake consensus algorithm, employed by some blockchain networks as an alternative to Bitcoin's resource-intensive Proof-of-Work mechanism, is similar to the membership fee model in that validator nodes are elected based on their share of "stake" in the network, measured by their willingness to commit or stake an allocation of network tokens as a proof of honesty [224]. For research library applications, a variation of Proof-of-Stake known as Proof-of-Authority may be the most appropriate solution [219], [224]. In contrast to public blockchains such as Ethereum and Bitcoin, or fully private blockchains restricted to a single organization, so-called consortium blockchains may be the preferred approach, one in which consensus "is controlled by a pre-selected set of nodes" [219]. The model implemented by the BigchainDB project fits the parameters of a consortium blockchain that implements a Proof-of-Authority approach to consensus [225].

44.5 DESIGN REQUIREMENTS

A blockchain-based catalog for research libraries should support the creation of a decentralized marketplace for library metadata. Rather than paying a centralized exchange to distribute their catalog records, libraries could buy and sell records in a peer-to-peer exchange. Catalog records could thus become a source of revenue rather than a costly expenditure. Many blockchain systems support the creation of so-called smart assets, or the creation of tokens to represent real-world assets. A new token could be minted to facilitate the exchange of metadata objects, and payment and settlement channels could be created using smart contracts on a public blockchain such as Ethereum.

However, a public blockchain solution does not fully satisfy the requirements of decentralization for this use case. A data asset cannot be represented exclusively by a token—it also needs to be stored in a decentralized system optimized for read and write transactions. Public blockchains such as Ethereum have been designed for exchange, not storage. At the current price of the Ethereum blockchain's native token, Ether (ETH), at approximately \$200.00, storing 1 gigabyte of data on the blockchain would cost over \$7,000,000.00 [226]. A decentralized system for library metadata must be able to scale and store big data out of the box. BigchainDB is a production-ready solution that might meet the requirements for this use case: it supports the creation of assets and the direct storage of metadata objects on its blockchain [227].

44.6 Scope

Findings are presented from an initial exploration of BigchainDB as a blockchain database solution for a shared library catalog. An overview of the BigchainDB architecture and data model and is provided, and some of BigchainDB's features and functionality are probed. A preliminary analysis of library metadata standards and requirements is included, and the question of whether they can be accommodated using BigchainDB is examined.

44.7 BIGCHAINDB

44.7.1 Evolution

BigchainDB was created to address the scalability and storage limitations of traditional blockchains such as Bitcoin and Ethereum and to create a hybrid solution that builds a blockchain layer on top of an existing big data system [228]. Development of the BigchainDB framework initially focused on integration with the RethinkDB system, but now works exclusively with MongoDB [228], [229].

The early focus of BigchainDB development was to create an architecture that would allow existing big data databases to be “blockchainified” [230]. The original BigchainDB whitepaper, released in June 2016, focused on the scalability limitations of traditional blockchain networks such as Bitcoin and claimed that it should be possible to develop a blockchain-based distributed database that would enable “1 million writes per second throughput, storing petabytes of data, and sub-second latency”—in contrast to the storage restrictions and 7 transaction-per-second (tps) limit of the Bitcoin network [230]. The advantages of adding a blockchain layer to an existing distributed database would be to incorporate “decentralized control, immutability, and creation [and] movement of digital assets” [230].

The primary challenge in designing a decentralized system is how to defend against both arbitrary failure and malicious actors. In so-called Sybil attacks, an attacker attempts to generate false identities in order to gain majority control over a network [231]. To address Sybil attacks, BigchainDB proposes a governance model that would create a federation of trusted nodes. Because all participants are known, any attempt by one participant to gain control over the network would be obvious. A more pervasive vulnerability comes in the form of the so-called Byzantine Generals’ Problem [230]. Nodes in a distributed network must be able to reach consensus about the final order of transactions at each state of the system, even in the presence of node failure or malicious attempts to manipulate system

state in order to gain an unfair advantage—for example, in double-spending, in which a transaction is replayed so that the same asset can be used again (a particular problem in the case of financial transactions) [230], [232].

In its original design, BigchainDB relied on the consensus algorithm of its underlying database to manage benign node failure and incorporated additional constraints to verify the integrity of the voting process by which nodes in the network approved transactions—and the blocks containing them—as valid [230]. However, in its initial version, BigchainDB did not claim to be Byzantine Fault Tolerant (or BFT—the term used to indicate that a system can withstand unexpected node behavior, whether benign or malicious, up to a certain threshold [230]). In the original design, all nodes belonged to a single logical database. This made the system overly centralized and vulnerable to attack: a malicious actor who gained control over a single node would have been able to drop the entire database, which was shared among all nodes in the network [228], [229].

BigchainDB 2.0, released in June 2018, underwent a complete redesign and incorporated full Byzantine fault tolerance through integration with Tendermint, an application for managing consensus and state machine replication in blockchain systems [233], [234]. As a result of implementing Byzantine fault tolerance through Tendermint, BigchainDB’s original goal of supporting 1 million tps was no longer viable.

44.7.2 Benchmark

A recent benchmark of BigchainDB 2.0 throughput performed by the BigchainDB development team indicated that the system was able to process approximately 300 tps [235]. Benchmarking was carried out on a four-node network on Microsoft Azure-hosted virtual machines located in the same data center. Three separate experiments were run to test different options, and a full report of configurations and results is available for review [236]. The primary experiment tested how long it would take to commit 1 million transactions of 765 bytes

each to the BigchainDB blockchain under default settings. Results showed an average rate of 299.0 tps and a median rate of 309.0 tps. All 1 million transactions were finalized in 56 minutes with no failures [236].

44.7.3 Architecture

The architecture of a BigchainDB 2.0 network is shown in Figure [151](#), created by the BigchainDB development team. Each node in the network is self-contained and includes its own MongoDB database and Tendermint application server. Tendermint is used to manage consensus, communication, and state replication among nodes, whereas the software that is unique to BigchainDB is responsible for “registering and tracking the ownership of ‘assets’” [233]. In BigchainDB 2.0, as is the case in general with systems that are Byzantine Fault Tolerant, $3f + 1$ nodes are necessary to run a network, where f is the number of faulty nodes to be tolerated [237]. Therefore, at least four nodes are required in order to run a BigchainDB network: if one of the four nodes becomes unresponsive or attempts to approve an invalid transaction, the network will continue to function based on the majority consensus of the other three nodes [237].

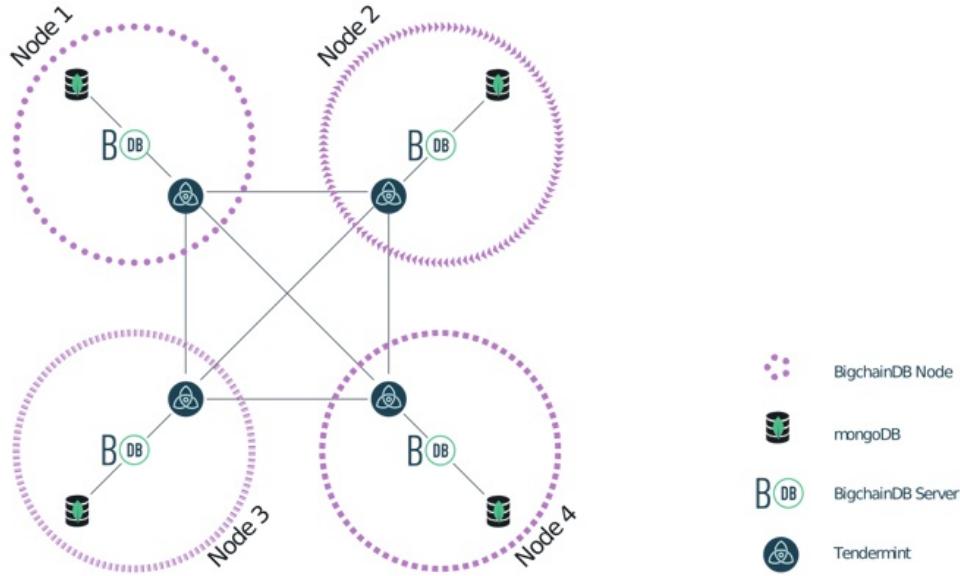


Figure 151: High-Level Architecture of BigchainDB 2.0 [238]

A BigchainDB client can potentially connect to any node in the network. Each MongoDB instance contains a full replication of the data stored in the network [237]. The BigchainDB project officially supports three client drivers to connect to a node server (in Python, Node.js, and Java) [239].

44.7.3.1 BigchainDB Server

The BigchainDB Server, written in Python, implements the logic to model, validate, and store transactions in the BigchainDB blockchain [233]. The server also incorporates a Python implementation of the Crypto-Conditions specification, which is a standard for enforcing complex boolean conditions for fulfillment (or transfer of assets) using cryptographic signatures [240].

All objects in BigchainDB are modeled as assets. Two transaction types are available for managing assets: CREATE and TRANSFER [241].

Each transaction must be cryptographically signed with the private key of its “owner” (the agent who created an asset through a CREATE transaction or to whom an asset was assigned through a TRANSFER transaction). Public/private keypairs are implemented using the Edwards-curve Digital Signature Algorithm Ed25519 [241]. A transaction is encoded using a dictionary or associative array that can be serialized as a JSON object. The BigchainDB Transactions Specification defines the structure and usage of a BigchainDB transaction object [241]. The key/value pairs that all valid BigchainDB transactions must include are listed here:

```
{  
  "id": ctnull,  
  "version": version,  
  "inputs": inputs,  
  "outputs": outputs,  
  "operation": operation,  
  "asset": asset,  
  "metadata": metadata  
}
```

Conditions for fulfillment and asset transfer are defined in the values of the “inputs” and “outputs” keys. An object representing the asset itself is stored as the value of the “asset” key and cannot be modified once an asset has been created and committed to a block in the BigchainDB blockchain. The “metadata” key is used to store an arbitrary object that records additional information about the asset or its state: in contrast to the asset object, the metadata object can be modified with each TRANSFER transaction [241].

44.7.3.2 Tendermint

Tendermint provides an application interface and BFT consensus algorithm for replicating application state across the nodes in a decentralized network [234]. Tendermint Core implements the consensus algorithm, which ensures that all nodes agree on a single order for transactions. Tendermint’s Application Blockchain Interface (ABCI) provides a language-agnostic interface for blockchain applications to use when validating and processing transactions [234].

Figure 152 is a sequence diagram, created by the BigchainDB development team, that illustrates the role of Tendermint in processing BigchainDB transactions. After a client prepares and signs a transaction, typically using a BigchainDB driver, the transaction is submitted to the BigchainDB server for initial validation. The server then sends the transaction to Tendermint, which includes it in a local memory pool. Tendermint returns its own validation request to the server and, upon confirmation, proposes a new block and begins a round of voting as part of its consensus algorithm. Each node in the network votes on the order and validity of transactions in the block, and if consensus is reached, the block is committed to the application's blockchain [228], [242]. BigchainDB stores a queryable copy of each block in MongoDB, while Tendermint appends each block to its canonical blockchain, which is stored in an internal LevelDB database and used for replicating transaction state to network peers [228], [234].

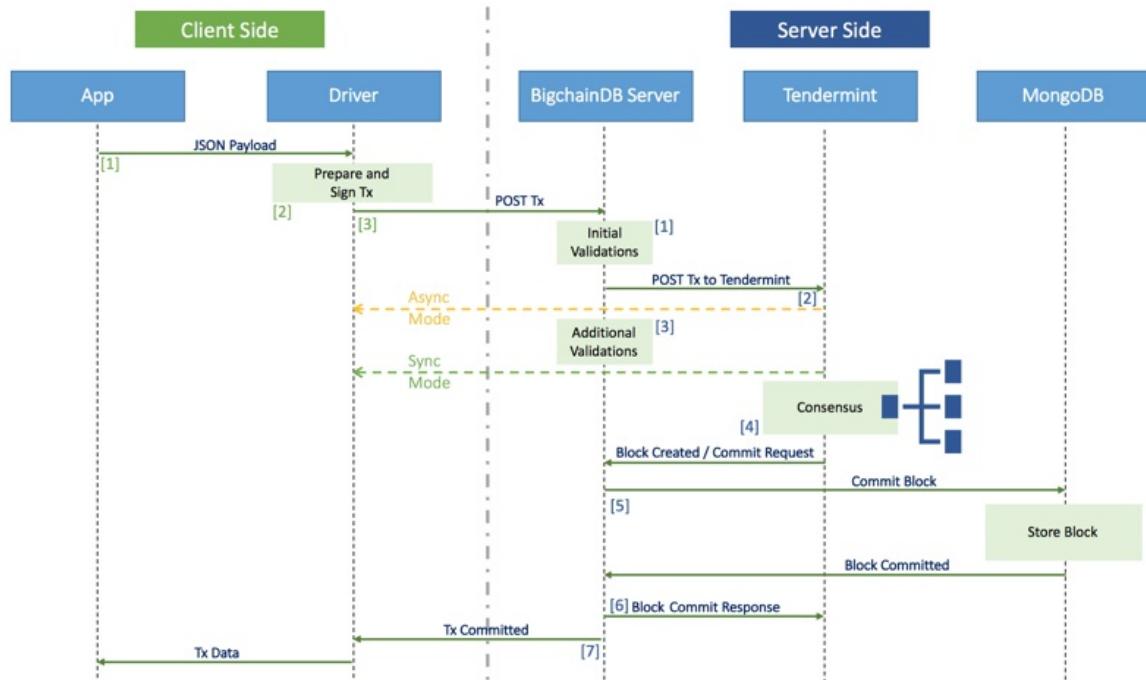


Figure 152: BigchainDB Sequence Diagram [242]

44.7.3.3 MongoDB

MongoDB is an enterprise-grade NoSQL database optimized for

storing JSON objects as documents. It supports both high availability (replication) and scalability (sharding) [243]. Early versions of BigchainDB used a single MongoDB replication set and were able to take advantage of these core MongoDB features. In BigchainDB 2.0, because each node maintains a separate MongoDB instance, replication and sharding are not supported out of the box [233]. In order to enforce practical immutability and decrease the likelihood of data tampering, the BigchainDB server limits access to MongoDB and does not expose any interfaces for deleting or making arbitrary modifications to database documents [228]. Although it is technically possible for a system administrator to modify the MongoDB database directly, each BigchainDB transaction is signed with a public/private cryptographic keypair—thus any tampering would result in a modified signature, which would be detectable by other nodes in the network and would violate its social contract [228].

BigchainDB does take advantage of MongoDB's query facility for read-only queries. Through its own HTTP API, it exposes a simple search interface for querying MongoDB, but it also allows node administrators to create custom indexes and leverage the full range of MongoDB query functionality [238].

44.8 DATASET

The dataset used is intentionally small and meant to test a potential use case for BigchainDB as a library catalog application. Currently, library catalog records are stored in a set of industry-specific formats maintained by the Library of Congress: the MAchine Readable Cataloging (MARC) formats for bibliographic and authority data (standardized as ISO 2709 and ANSI/NISO Z39.2) [244], [245]. In recent years, the Library of Congress has undertaken an effort to update library metadata standards and adopt standards and formats maintained by the World Wide Web Consortium (W3C)—specifically those related to linked data and the Semantic Web, such as the core data model known as the Resource Description Framework (RDF) [246], [247]. A new domain-specific data model and ontology for library metadata, expressed using the W3C's OWL standard for

semantic ontologies, is currently being developed and implemented ????. The data used here for testing with BigchainDB follows this model, known as the Bibliographic Framework Initiative (BIBFRAME) [246].

In the basic model proposed by BIBFRAME, descriptions of library resources are divided into three entity types or classes: Work (the abstract concept of the resource), Instance (the embodiment of a Work in a particular publication), and Item (a physical copy of an Instance) [246]. As an example, a catalog record from the Indiana University Library catalog was chosen. This record describes the Lilly Library's partial copy of the Gutenberg Bible. The data is divided into six files in the `project-data` directory:

```
ocm05084045.xml  
gutenberg-iul-item.rdf  
gutenberg-iul-instance.json  
gutenberg-iul-item.json  
gutenberg-iul-record.json  
gutenberg-work.json
```

The file `ocm05084045.xml` represents the original MARC-format record from the Indiana University Library catalog, encoded as XML [248]. The file `gutenberg-iul-item.rdf` provides an example of a partial conversion of the original MARC record to the BIBFRAME model using the RDF/XML serialization [249]. The remaining files represent the data used to create assets for storage in BigchainDB and are encoded in BIBFRAME using the JSON-LD serialization of RDF [249]. Several preprocessing steps of data conversion and cleanup were necessary. The original MARC/XML catalog record was converted to BIBFRAME RDF/XML using a suite of XSLT stylesheets provided by the Library of Congress [250]. The RDF/XML documents were then converted to JSON-LD using the Python RDFLib library (see the `convert_rdf.py` script in the `project-code` directory for a brief example). The JSON-LD produced by RDFLib was then broken into separate files to allow for the creation of individual assets in BigchainDB.

The JSON-LD output produced by RDFLib was further modified to support the inclusion of named graphs, a feature of RDF 1.1 that makes it possible to encode assertions about a collection of RDF

statements [247]. Using named graphs, it is possible to record administrative metadata about the creation or generation of the resource description itself (when it was created, by whom, using what standards, etc.). It should be noted that the usage of JSON-LD—which has some unconventional syntax features—with BigchainDB led to the discovery of a bug in the BigchainDB validation code that triggered a fatal Internal Server Error (HTTP 500). This bug was subsequently reported and has since been fixed by a BigchainDB core developer [251].

44.9 IMPLEMENTATION

44.9.1 Use Case

Currently, most large library catalogs are stored in enterprise relational databases such as Oracle. The catalog is one module in a suite of services known as an Integrated Library System (ILS), which also includes modules for circulation and ordering or acquisitions. The cataloging module in an ILS includes a public-facing interface for search and retrieval and a staff-facing interface for data entry and management. One advantage of using a distributed system such as BigchainDB for library cataloging functions would be to allow libraries to share their data more easily with peer institutions. BigchainDB's asset-based data model might also allow libraries to perform inventory and lending functions more efficiently. However, many functional components would need to be considered before determining whether a blockchain platform such as BigchainDB would be appropriate for the library catalog use case. One such component is focused on here: namely, the management of roles and permissions for data entry.

The Python component implements an extension to BigchainDB that adds support for Role-Based Access Control (RBAC) functionality [252]. The code is based on a Node.js example created by the BigchainDB development team to demonstrate the RBAC extension [253]. Support for RBAC is important for library cataloging because library personnel roles are typically divided between

professional librarians (catalogers) and paraprofessional technicians. Librarians are expected to create “original” descriptions of library resources, whereas paraprofessionals are responsible for copying existing data from a shared database such as OCLC WorldCat. Public blockchain systems do not usually impose write restrictions (allowing anyone to write to the database), so support for RBAC is an important consideration when evaluating BigchainDB.

44.9.2 Installation

BigchainDB was designed to be a federated network of distributed nodes. In an ideal setup, each node would be maintained in a different location by a different operator [228]. The official BigchainDB documentation provides instructions for those interested in setting up a production network [254]. However, for testing purposes, a full network is not necessary. A BigchainDB test network is currently available at <https://test.bigchaindb.com/>, but the testnet installation does not include the RBAC extension required here.

The official BigchainDB distribution includes a Docker Compose script that can be used for spinning up a single node. It also includes a shell script and a set of Ansible playbooks that can be used to provision a four-node network of virtual machines using Vagrant and VirtualBox. Both methods were tested, but frequent errors were encountered when trying to install and run a network of virtual machines. The Docker approach was chosen for simplicity and because it was possible to build the container from the RBAC branch of the BigchainDB Server codebase. The distribution includes a Makefile, and Docker containers for BigchainDB Server, Tendermint, and MongoDB can be easily run with a simple `make run` command.

44.9.3 Data Management

All BigchainDB CREATE transactions must include a JSON-serializable object to represent the asset being recorded on the blockchain. The `asset` field of a CREATE transaction takes an object with the required key `data`. The content of the `asset` field is treated as immutable—it

cannot be changed once a CREATE transaction has been committed, or when ownership of an asset is subsequently changed using a TRANSFER transaction. The following shows how a Work asset might be represented in BigchainDB:

```
{  
  "data": {  
    "@context": {  
      "rdfs": "http://www.w3.org/2000/01/rdf-schema#",  
      "schema": "http://schema.org/"  
    },  
    "@type": [  
      "http://id.loc.gov/ontologies/bibframe/Work",  
      "http://id.loc.gov/ontologies/bibframe/Text"  
    ],  
    "rdfs:label": "Bible. Latin. Vulgate. 1454."  
  }  
}
```

Because this data cannot be changed, it makes sense to represent it simply using its RDF type (in this case, it is a BIBFRAME Work with a subtype of Text), as well as a human-readable label. Any BigchainDB transaction may also include an optional `metadata` key that takes as its value an arbitrary JSON-serializable object. This flexible design makes it possible to effectively “update” data by using a TRANSFER transaction to indicate that the state of an asset has changed—and recording that change in the metadata object. The code in `rbac_demo.py` creates separate BigchainDB assets to represent the BIBFRAME types Work, Instance, and Item.

The data and code also illustrate how JSON-LD named graphs may be used to include both descriptive and administrative metadata about the same BigchainDB asset in a single transaction. The `rbac_demo.py` script inserts a second named graph into the `gutenberg-work.json` file so that it contains two named graphs: one representing the Work entity and one representing a separate Record entity (which is not part of the core BIBFRAME model). Within the named graph for the Record entity, there is an RDF property (`foaf:topic`) that links to the URI for the named graph representing the Work entity. Figure 153 illustrates this pattern, indicating how BigchainDB metadata objects may be used to create internal linkages among assets conforming to the BIBFRAME data model.

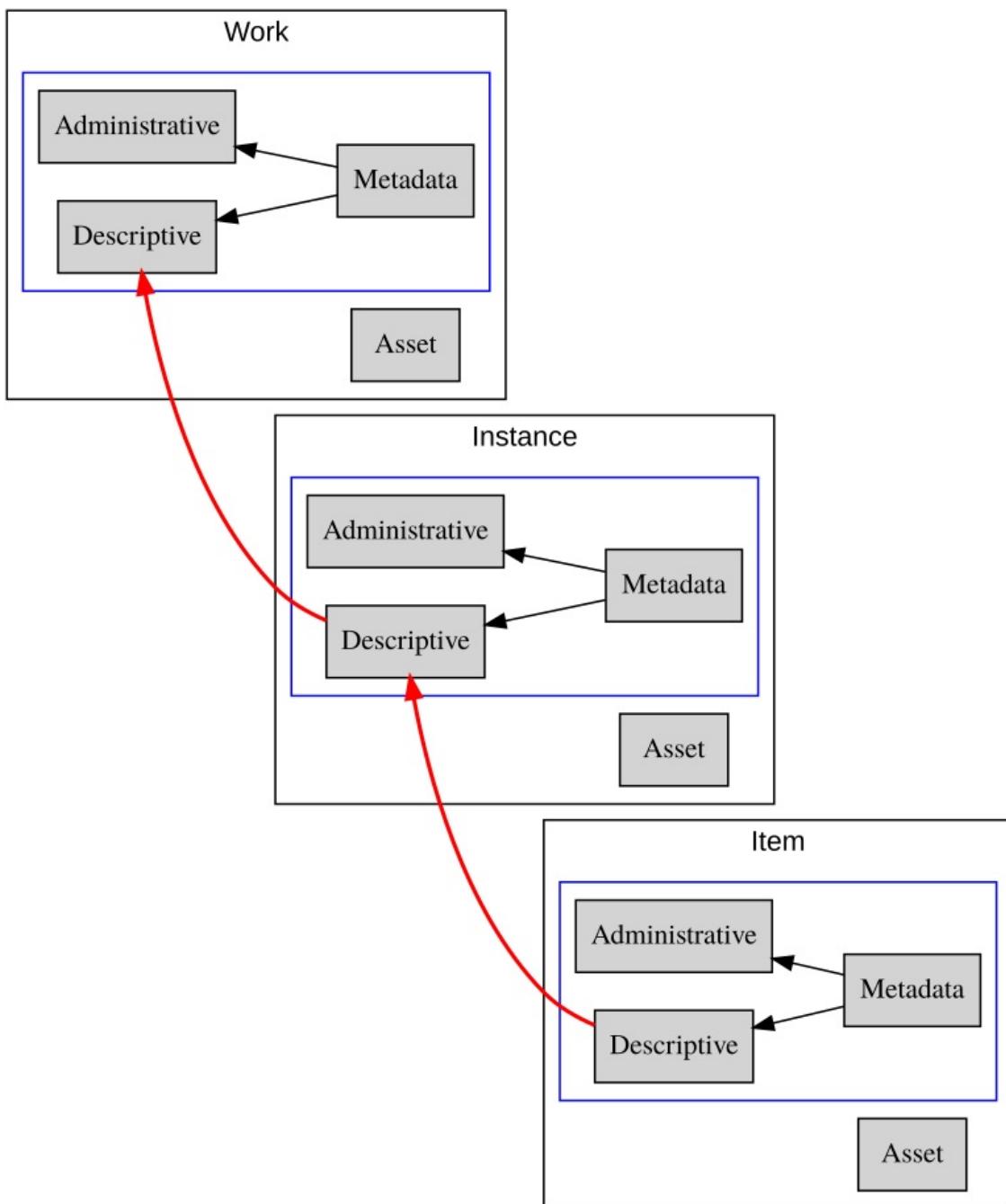


Figure 153: Graph of asset and metadata objects in BigchainDB

44.9.4 Role-Based Access Control in BigchainDB

The file `rbac.py` contains a single Python class, `BigchainRbac()`, that provides an interface to create new assets, users, types, and type instances for

Role-Based Access Control in BigchainDB. In the BigchainDB RBAC extension, roles and permissions, like everything else, are modeled as assets [252]. Two basic assets are necessary for bootstrapping: an asset to represent the application in which RBAC is being used and an asset to represent an admin group for admin users who can create other groups and users and assign permissions. BigchainDB RBAC employs two reserved keys, `link` and `can_link` that are used to create a dependency graph of users and asset types or groups. This graph is illustrated in Figure 154. Proper usage of these linkage keys is validated when a transaction is sent to the BigchainDB server, and a transaction is rejected if it violates the logic of the permissions scheme [252].

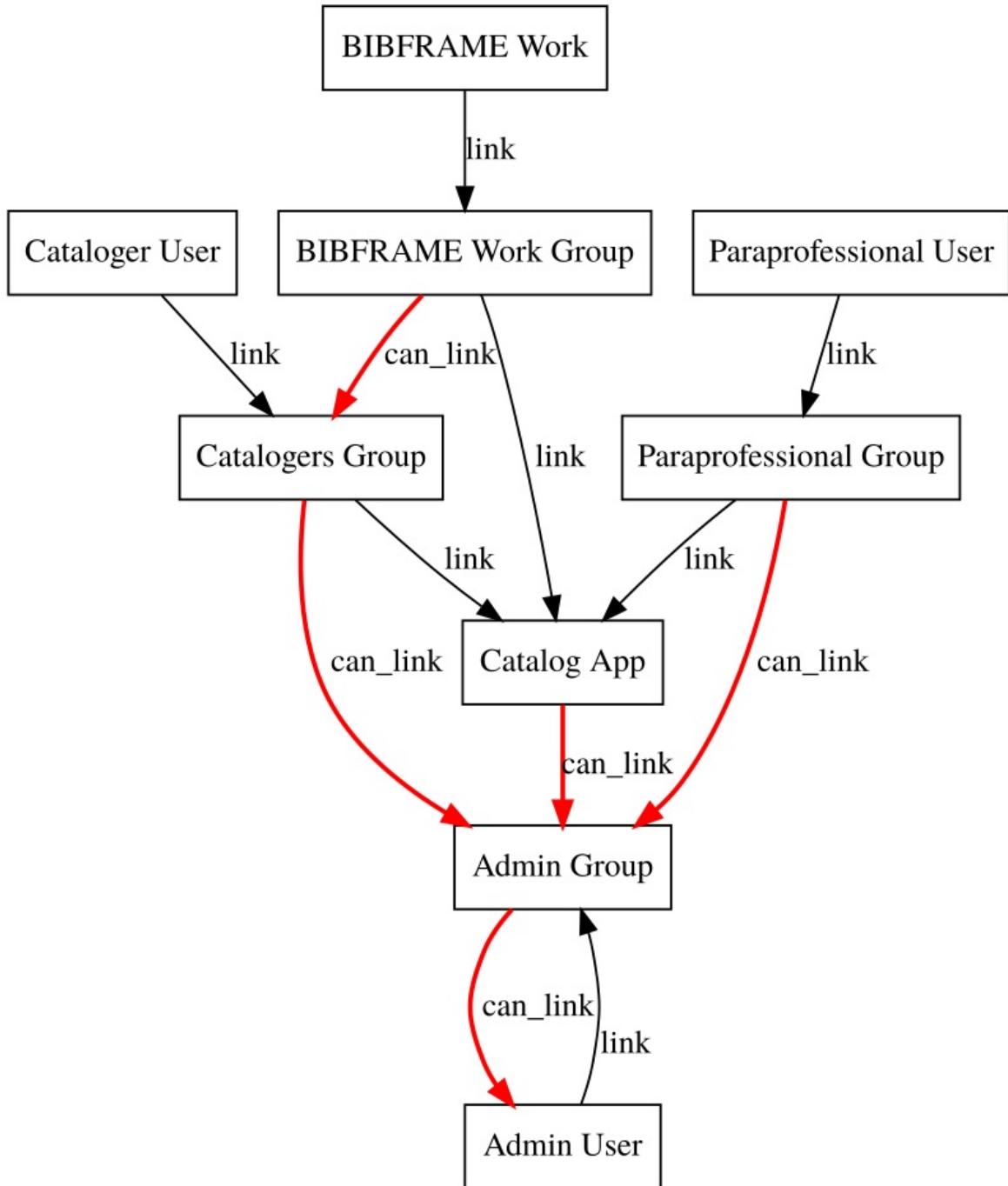


Figure 154: Graph of permissions in BigchainDB using Role-Based Access Control

The application logic proceeds according to the following steps:

1. User keypairs are generated for three different user types: admin users, catalogers, and paraprofessionals. In

BigchainDB, a user's identity is defined by a public/private keypair, and each transaction must be signed by one or more private keys, depending on the conditional logic that has been specified (especially in the case of TRANSFER transactions).

2. An admin user creates an asset to represent the admin user group and the application itself. The `can_link` key is included in the metadata object of the transactions. In the admin group asset, the value of `can_link` is a list of all admin user keypairs. In the app asset, it is the transaction ID for the CREATE transaction of the admin group asset—or, to put it more simply, the admin group ID. This linkage entails that only members of the admin group can link other groups to the app asset.
3. The admin user can then call the `create_type()` method of `BigchainRbac()` to instantiate new roles for the application. In `rbac_demo.py`, two roles have been created: one to represent “catalogers” and one to represent “paraprofessionals.” In the definition of each type asset, the `asset` object includes the `link` key, which takes as its value the app asset ID. This linkage sets the scope of permissions to the current application context. The admin user can then create new user assets and assign them to a group. The `can_link` key of the group asset points to the admin group ID because only the admin user can assign new users to this group. By contrast, the `link` key of the user asset points to the user’s respective group ID (such as that of the catalogers group). It is important to distinguish between a user—identified by a public key—and a user asset, which is the result of a BigchainDB CREATE transaction. In the RBAC scheme, when a user asset is created, it must then be assigned to the user’s public key through a TRANSFER transaction. This can be described as placing the asset in the user’s “wallet” [252].
4. The most important step occurs when an admin user creates a type to represent a resource group and then assigns

permissions to restrict which users can CREATE instances of that group. In `rbac_demo.py`, for example, an asset is created to represent BIBFRAME Work resources. Then, the `create_instance_type()` method is invoked in order to link a resource asset to its type. Figure 154 shows that individual BIBFRAME Work assets are linked to the BIBFRAME Work Group, which has a `can_link` relationship to the Catalogers Group. Therefore, only users associated with a cataloger user asset can CREATE BIBFRAME Work assets.

5. Finally, it is worth mentioning one concern that arises when working with BigchainDB as a data entry and data management system. The system's asset-centric approach requires that an asset have one or more owners. But in a shared scenario, how can one cataloger edit or update a record originally created by a different cataloger if they are not the owner of the asset? One solution would be for the original creator/owner to TRANSFER the asset to the person who wants to input new information or revise existing information. This may not be practical, however, since the original creator may no longer be present. Another solution might be to CREATE the asset with two original owners (that is, the CREATE transaction must be signed with two private keys): the cataloger as well as the admin user. The admin user could then TRANSFER the asset when needed. This is the approach taken by the demo developed here. However, in practice, it would have to be handled in an automated way, so that it would not be necessary for a single admin user to execute multiple TRANSFER transactions whenever the need arose. In general, best practices for managing public/private keypairs in the context of this use case is an area that calls for further examination.

When the `rbac_demo.py` script is executed, it populates a local BigchainDB instance with RBAC assets and tests whether catalogers and paraprofessionals can issue a CREATE transaction for a BIBFRAME Work asset. For successful transactions, the program simply outputs

an HTTP URL that can be used to request the result of each transaction. However, one transaction attempts to CREATE a Work resource with a paraprofessional user asset. This transaction should fail with a `Validation``Error`, as shown in the following:

```
BIBFRAME Work (IUL Paraprofessionals):  
(400, {'message': "Invalid transaction (Validation Error):  
Linking is not authorized for:  
6GcYiCCNFsDbBicna6YCVq8RmSjGyB7MGJw9CHjDjqwh", "status": 400}\n'...)
```

44.10 CONCLUSION

BigchainDB is a new blockchain-based solution for managing big data. A preliminary examination was undertaken to explore whether BigchainDB could be adapted for use as a library catalog database. The primary advantage of using a blockchain system instead of a conventional or distributed database is that it enables peer-to-peer interaction and exchange in a more fundamental way. Participants in a BigchainDB network are required to collaborate and coordinate their work very directly. Doing so would allow data scientists to access library metadata at scale and to more easily process, clean, and study it, potentially connecting it to other datasets. This could be particularly useful in large-scale text mining projects. In the current centralized scenario, library metadata remains locked away and can only be used for machine learning and complex analysis by the central service providers who control it.

A blockchain data model also allows for richer data modeling. Rather than a repository of records, a library catalog can be modeled as a collection of assets: actual books, journals, and multimedia resources, as well as the records that describe them. This could facilitate operations beyond cataloging, such as interlibrary lending, and make it easier to determine which library holds copies of which books, or subscribes to which journals. BigchainDB is flexible enough to accommodate the semantic data models, such as BIBFRAME, that are currently being developed by research libraries. BigchainDB itself is not a semantic data store, so additional solutions for semantically enriched information retrieval (such as graph databases) would also

need to be explored. However, BigchainDB's incorporation of MongoDB allows users to take advantage of the indexing and querying capabilities internal to MongoDB. Extensions to BigchainDB provide support for features such as Role-Based Access Control, which would be important for enabling library cataloging workflows. More systemic analysis would be needed before recommending BigchainDB as a solution for library catalogs, but preliminary results indicate that the possibility would merit further exploration.

44.11 ACKNOWLEDGMENT

The author would like to thank Dr. Gregor von Laszewski and the i523 and i516 teaching assistants for their support and suggestions in writing this report.

REFERNCES

- [1] G. von Laszewski, "Sample project." Report, Oct-2018 [Online]. Available: <https://github.com/cloudmesh-community/proceedings-fa18/blob/master/project-report/report.md>
- [2] M. Fowler, "Micro Services." Mar-2014 [Online]. Available: <https://martinfowler.com/articles/microservices.html>
- [3] SmartBear, "Using an API Gateway in Your Microservices Architecture." 2018 [Online]. Available: <https://smartbear.com/learn/api-design/api-gateways-in-microservices/>
- [4] SmartBear, "Monitoring with Prometheus and Grafana." Jan-2016 [Online]. Available: <https://github.com/hashicorp/faas-nomad/wiki/Monitoring-with-Prometheus-and-Grafana>
- [5] A. Innovations, "What is Serverless?" [Online]. Available: <https://serverless-stack.com/chapters/what-is-serverless.html>
- [6] OpenFaaS, "OpenFaaS: Introduction." [Online]. Available: <https://docs.openfaas.com/>
- [7] OpenFaaS, "Become your own Functions as a Service provider using OpenFaaS.".
- [8] Kaggle, "Dogs vs. Cats." [Online]. Available: <https://www.kaggle.com/c/dogs-vs-cats>
- [9] F. Chollet, "Building powerful image classification models using very little data.".
- [10] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. MIT Press, 2016 [Online]. Available: <http://www.deeplearningbook.org>
- [11] Christopher Olah, "Conv Nets: A Modular Perspective." Jul-2014

[Online]. Available: <http://colah.github.io/posts/2014-07-Conv-Nets-Modular/>

[12] M. Chang, "Applied Deep Learning 11/03 Convolutional Neural Networks." Oct-2016 [Online]. Available: <https://www.slideshare.net/ckmarkohchang/applied-deep-learning-1103-convolutional-neural-networks>

[13] Gerald Manipon, "HySDS Community Wiki." Web page [Online]. Available: <https://github.com/hysds/hysds-framework/wiki>

[14] ARIA Team, "Advanced Rapid Imaging and Analysis." Web page [Online]. Available: <https://aria.jpl.nasa.gov>

[15] J. Blumenfeld, "Getting Ready for NISAR—and for Managing Big Data using the Commercial Cloud." Web page [Online]. Available: <https://earthdata.nasa.gov/getting-ready-for-nisar>

[16] E. Fetzer, "A Multi-Sensor Water Vapor Climate Data Record Using Cloud Classification." Web page [Online]. Available: <https://earthdata.nasa.gov/community/community-data-system-programs/measures-projects/multi-sensor-water-vapor>

[17] Margaret Srinivasan, "NASA SWOT - Mission." Web page [Online]. Available: <https://swot.jpl.nasa.gov/mission.htm>

[18] Tony Greicius, "NASA-ISRO Synthetic Aperture Radar." Web page [Online]. Available: <https://www.jpl.nasa.gov/missions/nasa-isro-synthetic-aperture-radar-nisar/>

[19] Karen Yuen, "Orbiting Carbon Observatory-2." Web page [Online]. Available: <https://ocov2.jpl.nasa.gov/>

[20] European Space Agency, "Sentinel-1 Data Products." Web page [Online]. Available: <https://sentinel.esa.int/web/sentinel/missions/sentinel-1/data-products>

[21] US Geological Survey, "Digital Elevation Models." Web page

[Online]. Available: <https://ita.cr.usgs.gov/DEMs>

[22] IBM, “Hadoop.” website, 2018 [Online]. Available: <https://www.ibm.com/analytics/hadoop>

[23] V. Naresh Kumar and P. Shindgikar, “Configuring hadoop high availability clusters.” website, 2018 [Online]. Available: <https://www.i-programmer.info/projects/31-systems/11781-configuring-hadoop-high-availability-clusters.html>

[24] C. Wodehouse, “A guide to hadoop.” website, 2018 [Online]. Available: <https://www.upwork.com/hiring/data/a-guide-to-hadoop/>

[25] Hortonworks, “Apache hive.” website, 2018 [Online]. Available: <https://hortonworks.com/apache/hive/>

[26] C. Administrator, “Tutorial - apache hive - apache software foundation.” website, 2018 [Online]. Available: <https://cwiki.apache.org/confluence/display/Hive/Tutorial>

[27] Python community, “Pyhive project description.” website, 2018 [Online]. Available: <https://pypi.org/project/PyHive/>

[28] T. A. S. Foundation, “Apache derby.” website, 2018 [Online]. Available: <https://db.apache.org/derby/>

[29] multiple contributors, “Choosing a big data storage technology in azure.” website, 2018 [Online]. Available: <https://docs.microsoft.com/en-us/azure/architecture/data-guide/technology-choices/data-storage>

[30] “Cloudmesh nist services.” [Online]. Available: <https://github.com/cloudmesh-community/nist/tree/master/services>

[31] “Cloudmesh cm project.” [Online]. Available: <https://github.com/cloudmesh-community/cm/tree/master/cm4>

[32] J. Potter, “What is a lamp stack?” Web Page, Feb-2018 [Online]. Available: <https://www.liquidweb.com/kb/what-is-a-lamp-stack/>

- [33] Open API Initiative, “OpenAPI.” website, 2018 [Online]. Available: <https://www.openapis.org/about>
- [34] Amazon Web Services, “AWSEC2.” website, 2018 [Online]. Available: <https://aws.amazon.com/ec2/>
- [35] JupyterHub, “JupyterHub.” website, 2018 [Online]. Available: <http://jupyter.org/hub>
- [36] AWS, “AWS.” website, 2018 [Online]. Available: <https://console.aws.amazon.com/iam/home?region=us-east-2#/groups>
- [37] AWS, “AWS.” website, 2018 [Online]. Available: <https://us-east-2.console.aws.amazon.com/ec2/v2/home?region=us-east-2#SecurityGroups:sort=groupId>
- [38] AWS, “AWS.” website, 2018 [Online]. Available: <https://us-east-2.console.aws.amazon.com/ec2/v2/home?region=us-east-2#Instances:sort=instanceId>
- [39] E. Schadt, “The role of big data in medicine.” Web page, 2015 [Online]. Available: <https://www.mckinsey.com/industries/pharmaceuticals-and-medical-products/our-insights/the-role-of-big-data-in-medicine>
- [40] Y. Lee CH., “Medical big data: Promise and challenges.” Web page, 2017 [Online]. Available: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5331970/>
- [41] Y. Bhattacharya A., “Precision diagnosis of melanoma and other skin lesions from digital images.” Web page, 2017 [Online]. Available: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5543387/>
- [42] Anaconda, “What is anaconda?” Web page, 2018 [Online]. Available: <https://www.anaconda.com/what-is-anaconda/>
- [43] Jupyter, “Jupyter.” Web page, 2018 [Online]. Available: <http://jupyter.org/>

- [44] Jupyter, "The jupyter notebook." Web page, 2018 [Online]. Available: <http://jupyter.org/>
- [45] SciPy, "What is numpy?" Web page, 2018 [Online]. Available: <https://docs.scipy.org/doc/numpy-1.13.0/user/whatisnumpy.html>
- [46] pandas, "Pandas:Powerfule python data analysis toolkit." Web page, 2018 [Online]. Available: <https://pandas.pydata.org/pandas-docs/stable/>
- [47] matplotlib, "Home." Web page, 2018 [Online]. Available: <https://matplotlib.org/>
- [48] R. Tschandl P., "The ham10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions." Web page, 2018 [Online]. Available: <https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/DBW86T>
- [49] KNIME, "KNIME integrations." Web page, 2018 [Online]. Available: <https://www.knime.com/knime-software/knime-integrations>
- [50] A. Vidhya, "Building your first machine learning model using knime (no coding required!)." Web page, 2017 [Online]. Available: <https://www.analyticsvidhya.com/blog/2017/08/knime-machine-learning/>
- [51] KNIME, "JSON processing." Web page, 2018 [Online]. Available: <https://www.knime.com/whats-new-in-knime-211#JSON>
- [52] U. Sewwandi, "Guided analytics using knime analytics platform." Web page, 2018 [Online]. Available: <https://towardsdatascience.com/guided-analytics-using-knime-analytics-platform-b6543ebab7e2>
- [53] KNIME, "KNIME on amazon web services." Web page, 2018 [Online]. Available: <https://www.knime.com/knime-software/knime-aws>

- [54] KNIME, “KNIME on microsoft azure.” Web page, 2018 [Online]. Available: <https://www.knime.com/knime-software/knime-azure>
- [55] The Data Visualisation Catalogue, “Parallel coordinates plot.” Web page, 2018 [Online]. Available: https://datavizcatalogue.com/methods/parallel_coordinates.html
- [56] Statistics Solutions, “What is logistic regression?” Web page, 2018 [Online]. Available: <https://www.statisticssolutions.com/what-is-logistic-regression/>
- [57] American Cancer Society, “Skin cancer.” Web page, 2018 [Online]. Available: <https://www.cancer.org/cancer/skin-cancer.html>
- [58] Team Research Nest, “Real world applications of big data in healthcare.” Web page, 2018 [Online]. Available: <https://medium.com/the-research-nest/real-world-applications-of-big-data-in-healthcare-5c84696fd3d4>
- [59] Apache, “Kafka.” Web Page [Online]. Available: <https://kafka.apache.org/>
- [60] T. P. Neha Narkhede Gwen Shapira, Kafka: The definitive guide, First. O'REILLY, 2017 [Online]. Available: <https://www.confluent.io/wp-content/uploads/confluent-kafka-definitive-guide-complete.pdf>
- [61] Apache, “Apache kafka.” Web Page, 2018 [Online]. Available: <https://www.apache.org/dyn/closer.cgi?path=/kafka/2.1.0/kafka-2.1.0-src.tgz>
- [62] Apache, “Kafka.” Web Page [Online]. Available: <https://issues.apache.org/jira/browse/KAFKA-6855>
- [63] The Apache Software Foundation, “Apache nifi.” Web page, Oct-2018 [Online]. Available: <https://nifi.apache.org/>
- [64] S. Maarek, “Introduction to apache nifi (hortonworks dataflow - hdf 2.0).” Presentation [Online]. Available: <https://www.udemy.com/apache-nifi/>

- [65] A. Bridgwater, “NSA ‘nifi’ big data automation project out in the open.” Web Page, Jul-2015 [Online]. Available: <https://www.forbes.com/sites/adrianbridgwater/2015/07/21/nsa-nifi-big-data-automation-project-out-in-the-open/#68cdd7dc55d6>
- [66] A. DOKAEVA, “How to make etl simple and intuitive with nifi.” Web page, Mar-2018 [Online]. Available: <https://issart.com/blog/how-to-make-etl-simple-and-intuitive-with-nifi/>
- [67] Apache NiFi Team, “Apache nifi overview.” Web page, Oct-2018 [Online]. Available: <https://nifi.apache.org/docs.html>
- [68] hortonworks, “Analyze transit patterns with apache nifi.” Web page, Oct-2018 [Online]. Available: <https://hortonworks.com/tutorial/analyze-transit-patterns-with-apache-nifi/section/1/>
- [69] S. Gupta, “Creating custom processors and controllers in apache nifi.” Web page, May-2018 [Online]. Available: <https://medium.com/hashmapinc/creating-custom-processors-and-controllers-in-apache-nifi-e14148740ea>
- [70] Apache NiFi Team, “Apache nifi downloads.” Web page, Oct-2018 [Online]. Available: <http://nifi.apache.org/download.html>
- [71] Apache Software Foundation, “Apache kafka quickstart.” Web Page, Jan-2017 [Online]. Available: <https://kafka.apache.org/quickstart>
- [72] @humdata, “Asia pacific: Storm tracks 1956 to 2017.” Website [Online]. Available: <https://data.world/ocha-roap/b1068a22-ec52-459d-9541-0fb63906bb39>
- [73] stoltzman consulting llc, “EXPLORATORY data analysis of tropical storms in r.” Website [Online]. Available: <https://www.stoltzmanniac.com/exploratory-data-analysis-of-tropical-storms-in-r/>
- [74] Python, “What is python? Executive summary.” Website [Online].

Available: <https://www.python.org/doc/essays/blurb/>

[75] Matplotlib, “Documentation.” Website [Online]. Available: <https://matplotlib.org/>

[76] Seaborn, “Seaborn: Statistical data visualization.” Website [Online]. Available: <https://seaborn.pydata.org/>

[77] Altair, “Altair: Declarative visualization in python.” Website [Online]. Available: <https://altair-viz.github.io/>

[78] Folium, “Folium 0.7.0+8.g8adfb77 documentation.” Website [Online]. Available: <https://python-visualization.github.io/folium/>

[79] T. Voelker and R. McGlashan, “What is crowdfunding? Bringing the power of kickstarter to your entrepreneurship research and teaching activities,” in Small business institute journal, 2013, vol. 9.0, pp. 11–22 [Online]. Available: <https://sbij.org/index.php/SBIJ/article/view/175/124>

[80] R. Walters, “Getting started with python and mongodb.” Web Page, Apr-2017 [Online]. Available: <https://www.mongodb.com/blog/post/getting-started-with-python-and-mongodb>

[81] R. R. Nishad Tupe Izolda Fetko, “Analyzing france accidents data using mongo db,” Indiana University of Bloomington, technical report, Jul. 2018 [Online]. Available: <https://iu.instructure.com/courses/1718879/assignments/8129645/sul>

[82] K. Loughead, “Tableau and mongodb analytics: Why it’s a bad marriage.” Web Page, 2018 [Online]. Available: <https://blog.knowi.com/2018/03/tableau-and-mongodb-analytics.html>

[83] T. Matei, “MongoDB performance in the cloud,” Master’s thesis, San Jose State University, San Hose, California, United States, 2013 [Online]. Available: https://scholarworks.sjsu.edu/cgi/viewcontent.cgi?referer=https://www.google.com/&httpsredir=1&article=1300&context=sjsu_masters_theses

- [84] K. Bigbee, J. Chaves, and D. Seaman, "Benchmarking big data cloud-based infrastructures," Master's thesis, Worcester Polytechnic Institute, Worcester, MA, United States, 2017 [Online]. Available: https://web.wpi.edu/Pubs/E-project/Available/E-project-031617-185427/unrestricted/MQP_Final.pdf
- [85] S. Sverchkov, "Evaluating nosql performance: Which database is right for your data?" Web Page, Feb-2014 [Online]. Available: <https://jaxenter.com/evaluating-nosql-performance-which-database-is-right-for-your-data-107481.html>
- [86] Wikipedia, "Application programming interface." Web Page, 2018 [Online]. Available: https://en.wikipedia.org/wiki/Application_programming_interface
- [87] Kaggle, "How to use kaggle: Kaggle api." Web Page, 2018 [Online]. Available: <https://www.kaggle.com/docs/api>
- [88] tikivisik, "Kaggle-api." Web Page, Oct-2018 [Online]. Available: <https://github.com/Kaggle/kaggle-api>
- [89] M. Mouille, "Kicksarter projects." Web Page, Feb-2018 [Online]. Available: <https://www.kaggle.com/kemical/kickstarter-projects>
- [90] D. Research, "Correlation analysis - market research." Web Page, 2018 [Online]. Available: <https://www.djsresearch.co.uk/glossary/item/correlation-analysis-market-research>
- [91] S. Solutions, "Time series analysis." Web Page, 2018 [Online]. Available: <https://www.statisticssolutions.com/time-series-analysis/>
- [92] S. Solutions, "What is logistic regression?" Web Page, 2018 [Online]. Available: <https://www.statisticssolutions.com/what-is-logistic-regression/>
- [93] MongoDB, "PyMongo 3.7.2 documentation." Web Page, 2008 [Online]. Available: <https://api.mongodb.com/python/current/>

- [94] MongoDB, MongoDB manual 4.0. MongoDB Inc, 2018 [Online]. Available: <https://docs.mongodb.com/manual/replication/#edge-cases-2-primaries>
- [95] G. James, D. Witten, T. Hastie, and R. Tibshirani, An introduction to statistical learning, vol. 112. Springer, 2013.
- [96] "What is twitter and how does it work?" Web Page.
- [97] N. Zarczynski, "2 light bulbs and a 100 story building." Online, Mar-2011 [Online]. Available: <https://pointlessprogramming.wordpress.com/2011/03/11/2-light-bulbs-and-a-100-story-building/>
- [98] D. Maister, "Centralisation of inventories and the 'square root law,'" International Journal of Physical Distribution, vol. 6, no. 3, pp. 124–134, 1976 [Online]. Available: <https://doi.org/10.1108/eb014366>
- [99] O. Astrachan, "Bubble sort: An archaeological algorithmic analysis," SIGCSE Bull., vol. 35, no. 1, pp. 1–5, Jan. 2003 [Online]. Available: <http://doi.acm.org/10.1145/792548.611918>
- [100] R. Cole, "Parallel merge sort," SIAM Journal on Computing, vol. 17, no. 4, pp. 770–785, 1988 [Online]. Available: <https://doi.org/10.1137/0217049>
- [101] D. R. MUSSER, "Introspective sorting and selection algorithms," Software: Practice and Experience, vol. 27, no. 8, pp. 983–993 [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1002/%28SICI%291097-024X%28199708%2927%3A8%3C983%3A%3AAID-SPE117%3E3.0.CO%3B2-%23>
- [102] P. Madhusudan and X. Qiu, "Efficient decision procedures for heaps using strand," in Static analysis, 2011, pp. 43–59.
- [103] W3.CSS, "Python list sort() method." Online [Online]. Available: https://www.w3schools.com/python/ref_list_sort.asp

- [104] R. D. Dutton, "Weak-heap sort," BIT Numerical Mathematics, vol. 33, no. 3, pp. 372–381, Sep. 1993 [Online]. Available: <https://doi.org/10.1007/BF01990520>
- [105] sqlcourse.com, "What is sql?" Online [Online]. Available: <http://www.sqlcourse.com/intro.html>
- [106] MySQL, Limits on table size. MySQL [Online]. Available: <https://dev.mysql.com/doc/mysql-reslimits-excerpt/5.5/en/table-size-limit.html>
- [107] Microsoft, Excel specifications and limits. Microsoft [Online]. Available: <https://support.office.com/en-us/article/excel-specifications-and-limits-1672b34d-7043-467e-8e27-269d656771c3#top>
- [108] Python, Python 3.6.0. Python Software Foundation, 2016 [Online]. Available: <https://www.python.org/downloads/release/python-360/>
- [109] Spyder, Spyder: The scientific python development environment — documentation. [Online]. Available: <https://docs.spyder-ide.org/>
- [110] Python, Random - generate pseudo-random numbers. Python Software Foundation [Online]. Available: <https://docs.python.org/3/library/random.html>
- [111] Python, 15.3. Time — time access and conversions. Python Software Foundation [Online]. Available: <https://docs.python.org/2/library/time.html>
- [112] Python, The python standard library. Python Software Foundation [Online]. Available: <https://docs.python.org/3/library/>
- [113] Wikipedia, "Bubble sort." Web page [Online]. Available: https://en.wikipedia.org/wiki/Bubble_sort
- [114] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson, Introduction to algorithms, 2nd ed. McGraw-Hill Higher Education,

2001.

[115] Wikipedia, “Merge sort.” Web page [Online]. Available: https://en.wikipedia.org/wiki/Merge_sort

[116] D. E. Knuth, The art of computer programming, volume 1 (3rd ed.): Fundamental algorithms. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 1997.

[117] R. Sedgewick and K. Wayne, Algorithms, 4th ed. Addison-Wesley Professional, 2011.

[118] Wikipedia, “Insertion sort.” Web page [Online]. Available: https://en.wikipedia.org/wiki/Insertion_sort

[119] Wikipedia, “Shellsort.” Web page [Online]. Available: <https://en.wikipedia.org/wiki/Shellsort>

[120] Wikipedia, “Selection sort.” Web page [Online]. Available: https://en.wikipedia.org/wiki/Selection_sort

[121] A. López-Ortiz, U. C. Meyer, and R. Sedgewick, “Data structures and advanced models of computation on big data (dagstuhl seminar 14091),” Dagstuhl Reports, vol. 4, no. 2, pp. 129–149, 2014.

[122] Rosettacode, “Sorting algorithms/strand sort.” Web page, Jul-2018 [Online]. Available: https://rosettacode.org/wiki/Sorting_algorithms/Strand_sort#Python

[123] Wikipedia, “Strand sort.” Web page [Online]. Available: https://cs.wikipedia.org/wiki/Strand_sort

[124] T. Peters, “[Python-dev] sorting,” 2002 [Online]. Available: <https://mail.python.org/pipermail/python-dev/2002-July/026837.html>

[125] Wikipedia, “Heap sort.” Web page [Online]. Available: <https://en.wikipedia.org/wiki/Heapsort#Pseudocode>

[126] D. B. Thomas, W. Luk, P. H. Leong, and J. D. Villasenor, “Gaussian

random number generators," ACM Comput. Surv., vol. 39, no. 4, Nov. 2007 [Online]. Available: <http://doi.acm.org/10.1145/1287620.1287622>

[127] Interactive Python, "5.9. The insertion sort." Web page [Online]. Available:

<http://interactivepython.org/courselib/static/pythonds/SortSearch/The>

[128] W3Resource, "Python data structures and algorithms: Shell sort." Web page, Aug-2018.

[129] M. K. Shivi Aggarwal Abby Akku, "HeapSort." Web page [Online]. Available: <https://www.geeksforgeeks.org/heap-sort/>

[130] Interactive Python, "5.11. The merge sort." Web page [Online]. Available:

<http://interactivepython.org/courselib/static/pythonds/SortSearch/The>

[131] Interactive Python, "5.8. The selection sort." Web Page [Online]. Available:

<http://interactivepython.org/runestone/static/pythonds/SortSearch/The>

[132] "Yelp." Web Page [Online]. Available: <https://www.yelp.com/>

[133] Y. Li, Y. Liu, R. Chiou, and P. Kalipatnapu, "Prediction of useful reviews on yelp dataset." Paper [Online]. Available: https://bcourses.berkeley.edu/files/65096735/download?download_frd=1

[134] P. Hajas, L. Gutierrez, and M. S. Krishnamoorthy, "Analysis of yelp reviews." Web Page, Jul-2014 [Online]. Available: https://www.researchgate.net/publication/263736290_Analysis_of_Yelp_Reviews

[135] J. Koven, H. Siadati, and C.-Y. Lin, "Finding valuable yelp comments by personality, content, geo, and anomaly analysis." Paper [Online]. Available:

https://isis.poly.edu/~hossein/publications/yelp_finding_valuable_comments.pdf

[136] "How much data does the world generate every minute?" Web

Page [Online]. Available: <https://www.iflscience.com/technology/how-much-data-does-the-world-generate-every-minute/>

[137] “Big data analytics and the financial services industry.” Web Page [Online]. Available: <https://www.smartdatacollective.com/online-stock-trading-impacted-big-data/>

[138] J. Barrat, Our final invention : Artificial intelligence and the end of the human era. Thomas Dunne Books, 2013.

[139] “Stock prediction in python.” Web Page [Online]. Available: <https://towardsdatascience.com/stock-prediction-in-python-b66555171a2>

[140] S. Kalkowski, C. Schulze, A. Dengel, and D. Borth, “Real-time analysis and visualization of the yfcc100m dataset,” in Proceedings of the 2015 workshop on community-organized multimodal mining: Opportunities for novel solutions, 2015, pp. 25–30 [Online]. Available: <http://doi.acm.org/10.1145/2814815.2814820>

[141] TensorFlow, “TensorFlow lite.” Oct-2018 [Online]. Available: <https://www.tensorflow.org/lite>

[142] M. Techlabs, “How does a recommendation engine really work?” Oct-2017 [Online]. Available: <https://towardsdatascience.com/how-does-a-recommendation-engine-really-work-656bdf12a5fc>

[143] R. Banik, “Displaydetect.py.” github, Jan-2018 [Online]. Available: <https://www.datacamp.com/community/tutorials/recommender-systems-python>

[144] P. Sharma, “Comprehensive guide to build a recommendation engine from scratch (in python).” Jun-2018 [Online]. Available: <https://www.analyticsvidhya.com/blog/2018/06/comprehensive-guide-recommendation-engine-python/>

[145] Google Developers, “Using machine learning on compute engine to make product recommendations.” Web Page, Nov-2018

[Online]. Available:

[5.https://cloud.google.com/solutions/recommendations-using-machine-learning-on-compute-engine#storing_the_data](https://cloud.google.com/solutions/recommendations-using-machine-learning-on-compute-engine#storing_the_data)

[146] P. Kordik, "Machine learning for recommender systems." github, Jan-2018 [Online]. Available: <https://medium.com/recombee-blog/machine-learning-for-recommender-systems-part-1-algorithms-evaluation-and-cold-start-6f696683d0ed>

[147] R. B. Yehuda Koren and C. Volinsky, "MATRIX factorization techniques for recommender systems," Recommender-Systems-[Netflix], Aug. 2009.

[148] Wikipedia, "Machine learning for recommender systems." github, Jan-2018 [Online]. Available: https://en.wikipedia.org/wiki/Root-mean-square_deviation

[149] G. Karypis, "Evaluation of item-based top-n recommendation algorithms." github, Jan-2018 [Online]. Available: <https://www.semanticscholar.org/paper/Evaluation-of-Item-Based-Top-N-Recommendation-Karypis/0739fad62026ca36f101a36f29d53630207a5748>

[150] C.-h. L. Chebg Yeh, "The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients." paper, 2009 [Online]. Available: https://bradzzz.gitbooks.io/ga-dsi-seattle/content/dsi/dsi_05_classification_databases/2.1-lesson/assets/datasets/DefaultCreditCardClients_yeh_2009.pdf

[151] D. K. John Girard and K. Berg, Strategic data-based wisdom in the big data era. IGI Global, 2015 [Online]. Available: https://books.google.com/books/about/Strategic_Data_Based_Wisdom_id=yeqfBwAAQBAJ&printsec=frontcover&source=kp_read_button#v=o

[152] J. Vanthienen and K. D. Witte, Data analytics applications in education. CRC Press, 2017 [Online]. Available: <https://books.google.com/books?>

[id=c3JQDwAAQBAJ&printsec=frontcover&dq=Vanthienen,+J.,+%26+Wit+\(2018\).+Data+analytics+applications+in+education.+Boca+Raton:+CRC
4z7TeAhUHwYMKHVuYDUkQ6AEIMTAB#v=onepage&q&f=false](#)

[153] B. Williamson, Big data in education: The digital future of learning, policy and practice. SAGE, 2017 [Online]. Available: [https://books.google.com/books?id=yFAsDwAAQBAJ&pg=PP5&dq=Williamson,+B.+
\(2017\).+Big+Data+in+Education:+The+Digital+Future+of+Learning,+Po](https://books.google.com/books?id=yFAsDwAAQBAJ&pg=PP5&dq=Williamson,+B.+
(2017).+Big+Data+in+Education:+The+Digital+Future+of+Learning,+Po)

[154] Kaggle, "Give me some credit." Web page [Online]. Available: <https://www.kaggle.com/c/GiveMeSomeCredit>

[155] Kaggle, "Give me some credit." Web page [Online]. Available: <https://www.kaggle.com/c/GiveMeSomeCredit/data>

[156] Complete Dissertation by Statistics Solutions, "Correlation (pearson, kendall, spearman)." Web page [Online]. Available: <https://www.statisticssolutions.com/correlation-pearson-kendall-spearman/>

[157] J. Brownlee, "8 tactics to combat imbalanced classes in your machine learning dataset." Web page, Aug-2015 [Online]. Available: <https://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/>

[158] N. Donges, "The random forest algorithm." Web page, Feb-2018 [Online]. Available: <https://towardsdatascience.com/the-random-forest-algorithm-d457d499ffcd>

[159] S. Swaminathan, "Logistic regression—Detailed overview." Web page, Mar-2018 [Online]. Available: <https://towardsdatascience.com/logistic-regression-detailed-overview-46c4da4303bc>

[160] I. Reinstein, "XGBoost, a top machine learning method on kaggle, explained." Web page, Oct-2017 [Online]. Available: <https://www.kdnuggets.com/2017/10/xgboost-top-machine-learning->

[method-kaggle-explained.html](#)

- [161] Upwork, "Neural networks demystified." Web page [Online]. Available: <https://www.upwork.com/hiring/data/neural-networks-demystified/>
- [162] A. Mishra, "Metrics to evaluate your machine learning algorithm." Web page, Feb-24AD [Online]. Available: <https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234>
- [163] Github, "Official kaggle api." Github [Online]. Available: <https://github.com/Kaggle/kaggle-api>
- [164] Marcell Vollmer, "Big data – what's the big deal for procurement?" Web page, Sep-2017 [Online]. Available: <https://www.cbronline.com/big-data/big-data-whats-big-deal-procurement/>
- [165] Ariba Inc., "SAP ariba live: The journey continues." Web page, Mar-2018 [Online]. Available: <https://www.ariba.com/about/news-and-press/sap-ariba-live-the-journey-continues>
- [166] Baskar Radhakrishnan, "SAP ariba — procurement and supply chains for the digital world." Web page, Mar-2017 [Online]. Available: <https://us.nttdata.com/en/blog/2017/march/sap-ariba-procurement-and-supply-chains-for-the-digital-world>
- [167] L. Eikvil, "OCR - optical character recognition." 1993 [Online]. Available: <https://pdfs.semanticscholar.org/9484/96f9d73cab9c7b4fd5c3b656d1>
- [168] S. Viala, "ECM and big data, better together." 2014 [Online]. Available: <http://www.revasolutions.com/ecm-and-big-data-better-together/>
- [169] A. Rosebrock, "Using tesseract ocr with python." www, Nov-2018 [Online]. Available:

<https://www.pyimagesearch.com/2017/07/10/using-tesseract-ocr-python/>

[170] M. Lee, "Pytesseract - product description." www, Nov-2018 [Online]. Available: <https://pypi.org/project/pytesseract/>

[171] L. Richardson, "Beautiful soup - product description." www, Nov-2018 [Online]. Available: <https://pypi.org/project/beautifulsoup4/>

[172] J. Gonzalez, "Fuzzywuzzy." www, Nov-2018 [Online]. Available: <https://github.com/seatgeek/fuzzywuzzy>

[173] O. team, "OpenCV library." www, Nov-2018 [Online]. Available: <https://opencv.org/>

[174] A. Clark, "Overview - python imaging library." www, Nov-2018 [Online]. Available: <https://pillow.readthedocs.io/en/5.3.x/handbook/overview.html>

[175] A. Salnikov, "TkInter - python wiki." www, Nov-2018 [Online]. Available: <https://wiki.python.org/moin/TkInter>

[176] J. Wechsler, "Drug pricing and quality are top issues for 2018," PharmTech, vol. 42, no. 1, pp. 20–21, Jan. 2018 [Online]. Available: <http://www.pharmtech.com/drug-pricing-and-quality-are-top-issues-2018-0>

[177] U. D. of Health and H. Services, "FDA adverse event reporting system (faers) quarterly data extract files." Online, Sep-2018 [Online]. Available: <https://fis.fda.gov/extensions/FPD-QDE-FAERS/FPD-QDE-FAERS.html>

[178] Amazon, "Amazon free tier details." Online [Online]. Available: <https://aws.amazon.com/ec2/?ft=n>

[179] H. Chaplin, "Accessing an ec2 instance on aws using cyberduck," ionic melon, Dec. 2014.

[180] Microsoft, "Microsoft azure." [Online]. Available:

<https://azure.microsoft.com/en-us/>

[181] Microsoft, “Linux virtual machines pricing.” Online [Online]. Available: <https://azure.microsoft.com/en-us/pricing/details/virtual-machines/linux/>

[182] 12factor, “12-factor.” [Online]. Available: <https://12factor.net/>

[183] Docker, “Docker.” [Online]. Available: <https://docs.docker.com/get-started/>

[184] Google, “Kubernetes.” [Online]. Available: <https://kubernetes.io>

[185] Kubernetes, “Pods.” [Online]. Available: <https://kubernetes.io/docs/concepts/workloads/pods/>

[186] Google, “Kubernetes services.” [Online]. Available: <https://kubernetes.io/docs/concepts/services-networking/service/#publishing-services--service-types>

[187] G. Cloud, “Cloud shell.” [Online]. Available: <https://cloud.google.com/shell/docs/>

[188] G. API, “Cloud pub/sub api.” [Online]. Available: <https://cloud.google.com/pubsub/docs/overview>

[189] G. C. AI, “Cloud automl-main.” [Online]. Available: <https://cloud.google.com/automl/>

[190] Google, “Cloud vision.” [Online]. Available: <https://cloud.google.com/vision/docs/reference/rest/v1/images/annot>

[191] Redis, “Redis.” [Online]. Available: <https://redislabs.com/blog/getting-started-with-kubernetes-and-redis-using-redis-enterprise/>

[192] yelp, “Yelp.” [Online]. Available: <https://www.yelp.com/fusion>

[193] Redis, “Redis.” [Online]. Available: <https://redis.io/topics/rediscli>

[194] Docker, “Docker ce release notes,” Docker Documentation. San Francisco, CA, Dec-2017 [Online]. Available: <https://docs.docker.com/release-notes/docker-ce/>

[195] J. Turnbull, The docker book: Containerization is the new virtualization. New York, USA: James Turnbull, 2014.

[196] F. Paraiso, S. Challita, Y. Al-Dhuraibi, and P. Merle, “Model-driven management of docker containers,” in 9th ieee international conference on cloud computing (cloud), 2016, pp. 718–725 [Online]. Available: https://www.researchgate.net/figure/High-level-overview-of-Docker-architecture_fig1_308050257

[197] B. B. Rad, H. J. Bhatti, and M. Ahmadi, “An introduction to docker and analysis of its performance,” International Journal of Computer Science and Network Security (IJCSNS), vol. 17, no. 3, p. 228, 2017 [Online]. Available: http://paper.ijcsns.org/07_book/201703/20170327.pdf

[198] Hackernoon, “Docker-the popular containerization technology for an effective software development.” 2017 [Online]. Available: <https://hackernoon.com/docker-the-popular-containerization-technology-for-an-effective-software-development-4e2cddc5a329>

[199] R. Pi and R. Pi-Teach, “FAQS,” Raspberry Pi. Cambridge, UK [Online]. Available: <https://www.raspberrypi.org/help/faqs/>

[200] A. Ellis, “5 things about docker on raspberry pi.” Sep-2016 [Online]. Available: <https://blog.alexellis.io/5-things-docker-rpi/>

[201] V. Murugesan, “Why we chose docker to build our data processing platform.” 2015 [Online]. Available: <http://bigdata-madesimple.com/why-we-chose-docker-to-build-our-data-processing-platform/>

[202] S. Charrington, “Running hadoop on docker, in production and at scale.” 2015 [Online]. Available: <https://thenewstack.io/running-hadoop-docker-production-scale/>

- [203] P. Hauer, "Discussing docker. Pros and cons," Philipp Hauer's Blog. Oct-2015 [Online]. Available: <https://blog.philippauer.de/discussing-docker-pros-and-cons/>
- [204] B. Benchoff, "Introducing the raspberry pi 3," Hackaday. Feb-2016 [Online]. Available: <https://hackaday.com/2016/02/28/introducing-the-raspberry-pi-3/>
- [205] R. Pi and R. Pi-Teach, "SD cards," SD cards - Raspberry Pi Documentation. Cambridge, UK [Online]. Available: <https://www.raspberrypi.org/documentation/installation/sd-cards.md>
- [206] Debian, "DebianPackageManagement," DebianPackageManagement - Debian Wiki. 2017 [Online]. Available: <https://wiki.debian.org/DebianPackageManagement>
- [207] L. Bailey, L. Novich, T. Hildred, and D. Jorm, "Red hat customer portal," 5.2.2. Enable root login over SSH. 2012 [Online]. Available: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/v2v_guide/preparation_before_the_enable_root_login_over_ssh
- [208] P. Christen, Data Matching. Berlin, Germany: Springer, 2012.
- [209] C. Doran and C. Martin, "Measuring Success in Outsourced Cataloging: A Data-Driven Investigation," Cataloging & Classification Quarterly, vol. 55, no. 5, pp. 307–317, 2017.
- [210] A. H. Turner, "OCLC WorldCat as a Cooperative Catalog," Cataloging & Classification Quarterly, vol. 48, nos. 2-3, pp. 271–278, 2010.
- [211] OCLC, "WorldCat Data Licensing." Web Page, 2012 [Online]. Available: https://www.oclc.org/content/dam/oclc/worldcat/documents/worldcat_data-licensing.pdf
- [212] OCLC, "WorldCat Rights and Responsibilities." Web Page, Jun-

2010 [Online]. Available:
<https://www.oclc.org/en/worldcat/cooperative-quality/policy.html>

[213] J. McQueen, "Record Matching: Computers Cannot See That Which Is Obvious to 'Any Idiot' ... and Vice Versa," *Information Today*, vol. 9, no. 11, pp. 41–44, Dec. 1992.

[214] R. R. Larson and K. Janakiraman, "Connecting Archival Collections: The Social Networks and Archival Context Project," in *Research and advanced technology for digital libraries, tpdl 2011*, 2011, pp. 3-14.

[215] Yale University Library, "YUL quicksearch search results." RSS Feed, Nov-2018 [Online]. Available:
http://search.library.yale.edu/catalog?commit=Search&format=atom&q=&search_field=all_fields

[216] OCLC, "Inside WorldCat." 2018 [Online]. Available:
<https://www.oclc.org/en/worldcat/inside-worldcat.html>

[217] H. A. Olson, "The Power to Name: Representation in Library Catalogs," *Signs*, vol. 26, no. 3, pp. 639–668, 2001 [Online]. Available:
<http://www.jstor.org/stable/3175535>

[218] Coral Health, "Learn to Securely Share Files on the Blockchain with IPFS!" Blog, Feb-2018 [Online]. Available:
<https://medium.com/@mycoralhealth/learn-to-securely-share-files-on-the-blockchain-with-ipfs-219ee47df54c>

[219] V. Buterin, "On Public and Private Blockchains." Blog, Aug-2015 [Online]. Available: <https://blog.ethereum.org/2015/08/07/on-public-and-private-blockchains/>

[220] S. Alman, "Blockchain: Apps and Ideas." Web Page, Jul-2018 [Online]. Available:
<https://ischoolblogs.sjsu.edu/blockchains/blockchain-apps-and-ideas/>

[221] J. Song, "Why Blockchain Is Hard." Blog Post, May-2018 [Online].

Available: <https://medium.com/@jimmysong/why-blockchain-is-hard-60416ea4c5c>

[222] B. A. Scriber, "A Framework for Determining Blockchain Applicability," IEEE Software, vol. 35, no. 4, pp. 70–77, Jul. 2018 [Online]. Available: <https://www.computer.org/csdl/mags/so/2018/04/ms02018040070.html>

[223] E. Buchman, J. Kwon, and Z. Milosevic, "The Latest Gossip on BFT Consensus." arXiv.org, Sep-2018 [Online]. Available: <https://arxiv.org/abs/1807.04938v2>

[224] G. Marin, "Understanding the Value Proposition of Cosmos." Blog, Apr-2018 [Online]. Available: <https://blog.cosmos.network/understanding-the-value-proposition-of-cosmos-ecaef63350d>

[225] T. McConaghy, "[Reply in BigchainDB Gitter Chat]." Public Chat, Jun-2018 [Online]. Available: <https://gitter.im/bigchaindb/bigchaindb?at=5b16ac9599fa7f4c0648cc13>

[226] T. Hess, "[Reply on Ethereum Stack Exchange]." Ethereum Stack Exchange, Feb-2016 [Online]. Available: <https://ethereum.stackexchange.com/questions/872/what-is-the-cost-to-store-1kb-10kb-100kb-worth-of-data-into-the-ethereum-block>

[227] BigchainDB Contributors, "Key Concepts of BigchainDB." Web Page, 2018 [Online]. Available: <https://www.bigchaindb.com/developers/guide/key-concepts-of-bigchaindb/>

[228] BigchainDB GmbH, "BigchainDB 2.0: The Blockchain Database," BigchainDB GmbH, Berlin, Germany, May 2018 [Online]. Available: <https://www.bigchaindb.com/whitepaper/bigchaindb-whitepaper.pdf>

[229] killerstorm, "BigchainDB: A Prime Example of Blockchain Bullshit." Reddit, May-2016 [Online]. Available: https://www.reddit.com/r/Bitcoin/comments/4j7wjf/bigchaindb_a_prir/

- [230] T. McConaghy et al., "BigchainDB: A scalable blockchain database." Code Repository, p. 64, Jun-2016 [Online]. Available: <https://github.com/bigchaindb/whitepaper>
- [231] J. R. Douceur, "The Sybil Attack," in Revised papers from the first international workshop on peer-to-peer systems, 2002, pp. 251–260 [Online]. Available: <http://dl.acm.org/citation.cfm?id=646334.687813>
- [232] A. M. Antonopoulos, Mastering Bitcoin: Programming the Open Blockchain, 2nd ed. Sebastopol, CA, United States: O'Reilly, 2017.
- [233] T. McConaghy, "BigchainDB 2.0 is Byzantine Fault Tolerant." Blog, May-2018 [Online]. Available: <https://blog.bigchaindb.com/bigchaindb-2-0-is-byzantine-fault-tolerant-5ffdac96bc44>
- [234] Tendermint Contributors, "Tendermint." Web Page, 2018 [Online]. Available: <https://tendermint.com/docs/>
- [235] T. McConaghy, "And We're Off to the Races!" Blog, Sep-2018 [Online]. Available: <https://blog.bigchaindb.com/and-were-off-to-the-races-1aff2b66567c>
- [236] A. Granzotto, T. McConaghy, and M. Khan, "Performance Study: Analysis of Transaction Throughput in a BigchainDB Network." Code Repository, Sep-2018 [Online]. Available: <https://github.com/bigchaindb/BEPs/tree/master/23>
- [237] T. McConaghy, "[Reply in BigchainDB Gitter Chat]." Public Chat, May-2018 [Online]. Available: <https://gitter.im/bigchaindb/bigchaindb?at=5b055eaf9ed336150ea41180>
- [238] BigchainDB Contributors, "Querying BigchainDB." Web Page, 2018 [Online]. Available: <https://docs.bigchaindb.com/en/latest/query.html>
- [239] BigchainDB Contributors, "Drivers & Tools." Web Page, 2018

[Online]. Available:
<http://docs.bigchaindb.com/projects/server/en/latest/drivers-clients/index.html>

[240] S. Thomas, R. Reginelli, and A. Hope-Bailie, "Crypto-Conditions," IETF, Jan. 2017 [Online]. Available: <https://tools.ietf.org/html/draft-thomas-crypto-conditions-02>

[241] T. McConaghy, "BigchainDB Transactions Spec v2." Code Repository, Sep-2018 [Online]. Available: <https://github.com/bigchaindb/BEPs/blob/master/13/README.md>

[242] G. Dhameja, "Lifecycle of a BigchainDB Transaction." Blog, Aug-2018 [Online]. Available: <https://blog.bigchaindb.com/lifecycle-of-a-bigchaindb-transaction-c1e34331cbba>

[243] MongoDB Contributors, "Introduction to MongoDB." Web Page, 2018 [Online]. Available: <https://docs.mongodb.com/manual/introduction/>

[244] K. M. Ford, "LC's Bibliographic Framework Initiative and the Attractiveness of Linked Data," ISQ: Information Standards Quarterly, vol. 24, no. 2/3, pp. 46–50, 2012 [Online]. Available: <https://groups.niso.org/publications/isq/2012/v24no2-3/ford/>

[245] Network Development and MARC Standards Office, Library of Congress, "The Library of Congress Network Development and MARC Standards Office." Web Page, Oct-2013 [Online]. Available: <https://www.loc.gov/marc/ndmso.html>

[246] Library of Congress, "Bibliographic Framework Initiative." Web Page, 2018 [Online]. Available: <https://www.loc.gov/bibframe/>

[247] R. Cyganiak, D. Wood, and M. Lanthaler, "RDF 1.1 Concepts and Abstract Syntax," World Wide Web Consortium, Feb. 2014 [Online]. Available: <https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/>

- [248] Yale University Library, “[IUCAT record 8251671].” Library Catalog, Nov-2018 [Online]. Available: <https://iucat.iu.edu/catalog/8251671>
- [249] E. L. Morgan, “RDF Serializations.” Web Page, 2013 [Online]. Available: <https://sites.tufts.edu/liam/2014/01/31/serializations/>
- [250] Network Development and MARC Standards Office, Library of Congress, “marc2bibframe2.” Code Repository, Oct-2018 [Online]. Available: <https://github.com/lcnetdev/marc2bibframe2>
- [251] T. Thompson, “Invalid json key in metadata causes bigchaindb to crash.” Code Repository, Nov-2018 [Online]. Available: <https://github.com/bigchaindb/bigchaindb/issues/2605>
- [252] G. Dhameja, “Role Based Access Control for BigchainDB Assets.” Blog, Sep-2017 [Online]. Available: <https://blog.bigchaindb.com/role-based-access-control-for-bigchaindb-assets-b7cada491997>
- [253] BigchainDB Contributors, “BigchainDB RBAC Sample.” Code Repository, Sep-2018 [Online]. Available: <https://github.com/bigchaindb/project-jannowitz/tree/master/rbac>
- [254] BigchainDB Contributors, “How to Set Up a BigchainDB Network.” Web Page, 2018 [Online]. Available: <http://docs.bigchaindb.com/projects/server/en/latest/simple-deployment-template/network-setup.html>