# *Cloud and Big Data Technologies*

Vol. 9, Projects, Spring 2018

*Bloomington, Indiana*

http://cyberaide.org/papers/vonLaszewski-cloud-vol-9.pdf

Wednesday 9th May, 2018, 17:23

Editor:
Gregor von Laszewski
Department of Intelligent Systems
Engeneering
Indiana University
laszewski@gmail.com

# Contents

# Chapter 1

# List of Papers

| HID | Author | Title | Chapter | Status |
|---|---|---|---|---|
| hid-sp18-401 | Arra, Goutham | REST Service Framework for Singular Value Decomposition on MNIST Image Classification Problem | REST | 100 |
| hid-sp18-409 | Kadupitige, Kadupitiya | Location based crime data search and analysis with Spark UDF | REST | 100 |
| hid-sp18-411 | Kaveripakam, Venkatesh Aditya | Twitter Sentimental Analysis with Spark | Spark | 100 |
| hid-sp18-412 | Kotabagi, Karan | Benchmarking Hadoop and Spark on Mutiple Platforms | PaaS | 100 |
| hid-sp18-414 | Leite, John | Blockchain Implementation with Flask | Blockchain | 100 |
| hid-sp18-415 | Mudvari Khatiwada, Janaki | Linear Regression REST Service and API | REST | 100 |
| hid-sp18-416 | Ossen, Sabra | Big Data Reference Architecture using Python Celery | Workflow | 100 |
| hid-sp18-417 | Ray, Rashmi | Financial Analysis Service | Security | 100 |
| hid-sp18-419 | Sobolik, Bertolt | Benchmarking a Sentiment Analysis Algorithm Using Hadoop on Multiple Platforms | PaaS | 100 |
| hid-sp18-420 | Sowani, Swarnima | REST Abstract File System Service | REST | 100 |
| hid-sp18-503 | Arnav, Arnav | Swagger Service for Openstack Resources | REST | 100 |
| hid-sp18-505 | Cate, Jr, Averill | GraphQL Web APIs | Messaging | 100 |
| hid-sp18-507 | Giuliani, Stephen | Analyzing Twitter Activity to Identify Spam Accounts | Twitter | 100 |
| hid-sp18-510 | Kaul, Naveen | Edge Computing and Big Data Processing using Raspberry Pi | Edge Computing | 100 |
| hid-sp18-511 | Khandelwal, Sandeep | Automated Apache Spark Cluster Deployment on AWS EC2 using Ansible | Ansible | 100 |
| hid-sp18-513 | Kugan, Uma | Service for Managing the Public Key | Security | 100 |
| hid-sp18-516 | Pathan, Shagufta | Openstack deployment using Swagger and Libcloud | REST | 100 |

| | | | | |
|---|---|---|---|---|
| hid-sp18-518 | Robinson, Michael | Leveraging REST for cloud portability | REST | 100 |
| hid-sp18-521 | Steinbruegge, Scott | Interacting With Data Services on AWS | Amazon | 100 |
| hid-sp18-524 | Tian, Hao | Report: Big Data in Healthcare | Application | 100 |
| hid-sp18-526 | Whitson, Tim | CMENV: Deployable Cloudmesh Containers | REST, Containers | 100 |
| hid-sp18-702 | Dubey, Lokesh | Report: Big Blockchains | Blockchain | 100 |
| hid-sp18-703 | Rufael, Ribka | Deploying CouchDB Cluster | Ansible | 100 |
| hid-sp18-706 | Sylla, Hady | Diagnosis of Coronary Artery Disease Using Big Data Analysis | Application | 100 |
| hid-sp18-707 | Smith, Michael | Twitter sentiment analysis of the Affordable Care Act in 2018 | Twitter | 100 |

# REST Service Framework for Singular Value Decomposition on MNIST Image Classification Problem

Goutham Srivatsav Arra
Indiana University
Bloomington, IN 47408
garra@iu.edu

Priyadarshini Vijjigiri
Indiana University
Bloomington, IN 47408
pvijjigi@iu.edu

## ABSTRACT

For tasks such as image classification, Speech denoising etc.. Deep learning algorithms with complex neural networks are being used as they have proven more accurate and capable in the recent times. However, one of the downsides to running deep learning algorithms is they can be computationally heavy and require huge memory to execute. The solution to the above problem is performing Singular Value Decomposition on them. In this project we have built a dockerized swagger webservice that trains a fully connected neural network for MNIST image classification problem and applies singular value decomposition according to size of SVD and optimize the network again. Entire Model and singular value decomposition is done in python using Tensorflow.

## KEYWORDS

hid-sp18-401,hid-sp18-421, Volume: 9, Chapter: REST, Status: 100. Singular Value Decomposition, Swagger, Docker, Deep Learning, Neural Networks, Tensorflow

## 1 INTRODUCTION

### 1.1 MNIST Image Classification

MNIST stands for Mixed National Institute of Standards and Technology. MNIST image classification problem is one of the famous classification problems in the deep learning community. It is nothing but pattern recognition of images which have handwritten digits on them. The Aim of this problem is classification of images into ten categories of numbers from 0-9. Many say, MNIST classification problem is like 'Hello World!' type of problem in deep learning commnunity and yet it gives proper and perfect introduction to deep learning methods [8].

### 1.2 Need for Deep Learning

Compared to traditional machine learning techniques, applying Deep Learning method to this problem yields better results. This is done by building Neural Networks. Neural Network have come to become very powerful black boxes that are capable of imitating any non-linear function (of course linear too !) in the world. A typical Neural Network consists of several layers and each layer consists of several 'neurons'. Neurons are the fundamental building blocks of neural network. Each neuron-input path has an assiciated weight to it. It is similar to how important a certain neuron is in that layer. A neuron takes in as input a vector with a bias term added to that input vector and performs dot product of that combined input vector with weight vector and produces an intermediate output. Till now, there is no introduction of non-linearity, so the method by which neural networks approximate non-linear relationships is by activation functions. There is an activation function at the output of every neuron. This function takes in as input the above intermediate output of the dot product and depending on the function rules, gives out final output. More on this, is described later in detail.

### 1.3 What and Why Singular Value Decomposition?

Building neural networks for MNIST classification problem and using this network is a computationally heavy task. On a typical CPU, training the neural network and performing feed forward will take atleast 1 hour! And we are not even building Convolutional neural networks for analyzing the images, we are only building fully connected neural networks which are simpler to use. If only there were a way to compress this build neural network and use it without losing too much accuracy on predicting new images! Indeed there is a solution to it, it is Singular Value Decomposition. Without giving too much away here, as the name hints, singular value decomposition technique compresses our original neural network by coming up with different representation of the original weight matrices that are of the lower rank and therefore need fewer calculations to perform dot products. This implies our network is not so computationally heavy anymore, but the accuracy reduces a little bit, but not too much to question the whole method. We can even optimize our compressed network to further optomize the accuracy and it is one of the objectives of this project. There is a paremeter, D, called size of Singular matrices. The lower the value of D, the higher the compression of the neural network and therefore the lower the test accuracy of the network. So, there is a trade off we must be aware. We usually dont want too low value for D for the sake of simple network and have poor accuracy. Nor do we want too high value for D for sake of better accuracy but at the cost of complex network. We proposed some typical values for D, which are, [10,20,50,100,200] for which we will be performing singular value decomposition on the MNIST classification problem to compress the original network and also optimize the compressed network for further use. MNIST data is available on the internet in several formats. It can also be downloaded from the internet with direct tensorflow python commands in our program. It contains total 65000 data. Of the total data 55000 are train images on which we first train our neural network and the rest 10,000 are test images to test how well our model has fared on the test images.

Although the Deep Neural network modelled on MNIST classifies the images more accurately than usual Neural networks with less than 2 layers or other machine learning algorithms there are many problems that these networks will suffer like issues with speed, power consumption to update millions of parameters and these problems grows when the size of data set increases. Also there

are problems to train such a deep network like vanishing gradient, Overfitting and even after the model is trained, computational time taken by the model to predict the output is also high. One of the main method to reduce the complexity of Deep Neural Network is Singular Value Decompostion.

### 1.4 Rest Service with Swagger

We are implementing Rest Service Framework with Swagger Codegen for this project. This Rest service when implemented, will connect to the server on MNIST database website, download the image data, construct the original full sized neural network, then apply Singular Value decomposition on it, optimize the compressed network for different values of size of SVD (D), output the test accuracy of our down-sized network into a website.

## 2 DATASET DESCRIPTION FOR MNIST

The MNIST database is a large database of images that have hand written digits which are widely used for various image processing systems and in the field of machine learning and deep learning. The MNIST database consists of 60,000 training images and 10,000 testing images. The images in MNIST database come from two sources of NISTs databases, Speical Database 1 and Special Database 3. These were images of digits written by high school students and employees of United States Census Bureau. The MNIST database is hosted on Yann Lecunn's website, Director of AI research, Facebook. Each image in MNIST dataset is a 28x28 pixels, which is usually converted into 1 dimensional array of numbers to be used for model building. This results in a 784 dimensional vector per image [6].

Each image in MNIST data set is of 784 dimensions. It was a challenge to build a model on large number of dimensions. Many dimensional redcution techniques were used to reduce the dimensions of data. Building a model for this then was challenging with lots of preprocessing and feature learning. But with the discovery of using more layers with in the neural network this is achievable with an accuracy of 98 percent. So we have used Deep Learning techniques in this Paper to build a model as REST service.

Figure 1 and Figure 2 are the typical images of the MNIST dataset.

**Figure 1: Hand written numbers in MNIST Dataset [5]**

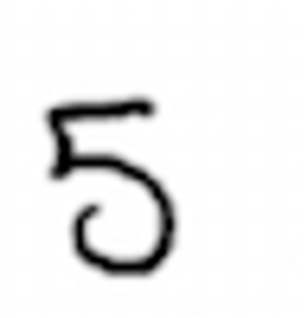

**Figure 2: Hand written number in MNIST Dataset [4]**

## 3 SINGULAR VALUE DECOMPOSITION

Singular Value Decomposition is factorization of a matrix into 3 matrices.

$$A = USV^T$$

Here U and V matrices are orthogonal matrices and S is singular matrix which is also diagonal. Singular Value Decompostion can be defined for any kind of matrix rectangular or square. Here the columns in matrix U are called left singular Vectors and columns in V are called right singular Vectors. These singular Vectors form an orthogonal set.

Singular Value Decomposition can be understood from a lucid example. Considering A (which is a matrix of shape n/d) as a matrix with n points in a d dimensional space. Singular Value Decompostion is to factorsize A into 3 matrices that is each point of n points in A is finding its respective point in k dimensional space that is finding a U matrix with shape n/k and then V matrix with shape k/n that again projects these n points in k dimensional space to n dimensional space. This fitting of points into lower dimensional space is done using best least squares fit. Least squares fit tries to minimize the distance between n dimensional space and k dimensional space.

Singular value Decomposition is applied to reduce the dimensions in many algorithms. One such traditional use is Principle Component analysis where any number of points in d dimensional space are reduced to k dimensional space where k is much less than d [12].

## 4 NEURAL NET MODEL FOR MNIST

Each image in MNIST data set is of 784 dimensions. It was a challenge to build a model on large number of dimensions. This problem is commonly called as 'The curse of dimensionality', because although many dimensions means more features about the image (which implies more data to build the model), it also means the network will become computationally heavy. Many dimensional redcution techniques were used to reduce the dimensions of data. Building a model for this then was challenging with lots of preprocessing and feature learning. But with the discovery of using more layers with in the neural network this is achievable with a typical test accuracy of around 96-98 percent.

In this project we used Tensor flow, a deep learning package in python language, for building our model to classify the image from MNIST data. A fully connected network is trained for the model.

### 4.1 Tensor-flow

Tensor flow is an open source software library in python which is used for programming a wide variety of tasks. It is a computational framework where we can build models at our preferred level of abstraction.It is a flexible architecture using a single API it allows to deploy the heavy computation on multiple CPUs or GPUs in a desktop or server. Tensor flow used data flow graphs to represent all the parameters, dependencies and operations. Program using Tensorflow starts with defining the graph, operations and then define the sessions to run this graph and update the parameters.

### 4.2 Hidden layers

Hidden layers are what makes neural networks look like biological neurons in human body. They take the information from the input and learn the features. Weight matrices connect the input layer to hidden layer, hidden layer to hidden layer and hidden layer to output layer. Using more than two Hidden layers output of a neural network can approximate any complex function.

- For the MNIST classification we used 5 hidden layers each hidden layer with 1024 neurons and also 6 weight matrices connecting all the layers.
- First weight matrix is of size 784/1024 that which connects the input layer and first hidden layer.
- Next four weight matrices are of size 1024/1024 which connect the hidden layers in between that have 1024 neurons each.
- The last weight matrix is of size 1024/10 which connects with the output layer.
- The output layer has 10 neurons.

Figure 3 is for reference only, and is not the exact architechture that applies for this project

For each image in the data set only one neuron in the output will be active and classification of the image is done based on which particular neuron is active. This brings us to the next section of activation functions that we used in this project.

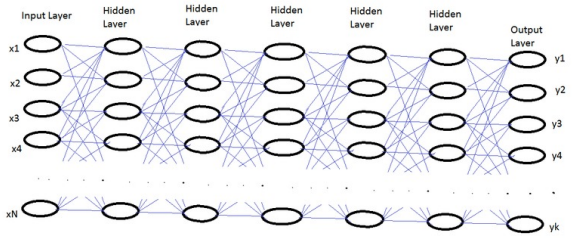


**Figure 3: Deep Neural Network [7]**

### 4.3 Activation Functions Used

Activation function are applied to each neuron, they decide whether the information neuron recieving should be activated or not. Without an activation function a neural network simply does a linear transformation so it is similar to linear regression model. Activation functions are important to neural network to develop non-linearity in the approximated function. A sufficiently large network using any of the common non linear activation function can approximate arbitrarily complex functions. Activaton function also have limitaions like they should be able to differentiable in order to propogate error back in backpropagation. There are many choices for activation function like Binary Step function, Linear function, Sigmoid function, Tanh function, ReLU function, Leaky ReLu function and Softmax function. We tried different choices for activation function like Sigmoid, Softmax, Leaky ReLU and ReLU, ReLU showed best results among all the 4 for the hidden layers, while the output activation must be sigmoid activation because we are outputting the probability.

*4.3.1 Relu.* ReLu function (Rectified Linear Function) gives output zero if input is negative and raw output otherwise.

$$f(x) = \max(x, 0)$$

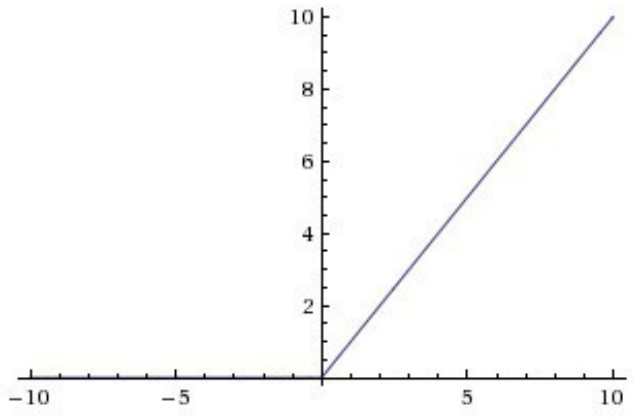


**Figure 4: ReLU Activation Function [9]**

Figure 4 shows ReLU Activation function.

3

Output of ReLU function is maximum of zero and itself. Gradient of this function is a constant value. So ReLU function overcomes the issue of Vanishing Gradient which is severe for Sigmoid function. ReLU outputs for the negative value is zero this makes the output more sparse where the sigmoid function always gives a value between -1 and +1.

*4.3.2 Sigmoid.* Sigmoid Function is specefic type of logistic function which has output range between 0 to 1. It is often used in classification problems, as we are assigning class to observation based on how likely it belongs to that class, in other words, probability. The mathematical formula for sigmoid function is below.

$$S(x) = \frac{e^x}{1 + e^x}$$

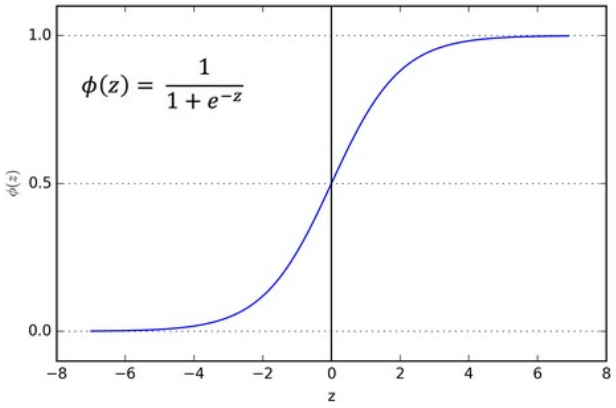


**Figure 5: Sigmoid Activation Function [10]**

Figure 5 explains Sigmoid Function.

Sigmoid function has a characteristic S-shaped curve or sigmoid curve. Some of the function characteristics are, it is bounded, differentiable, and it is real function defined for all real input values and its derivative is non- negative at each point.

## 4.4 Back Propagation

The fundamental nature of neural network or any machine learning model is that they can learn by themselves given some training. We now have constructed our network which has several layers, each layer has several neurons, and weights connecting between neurons of adjacent layers. Now, our neural network must be trained with train data, in our case, training images. More specefically, the neural net must learn the best weights suitable for this problem, because the knowledge of a neural network lies in its weight matrices. Neural networks train by a method called Back Propagation. Back Propagation is simply propagation of errors back from output layer till input layer. Errors are the difference between groud truth value and predicted value. In our case, value is the class of the image, which number (0-9) does a given image belongs to ? During back propagation, we do not literally propagate this error, instead we try to minimize this error. So naturally, we differentiate this error function, to get the slpe and direction of gradient descent of error function. We know that gradient of function takes a value zero at its minimum or maximum. We hope to achieve this minimum for the error function, sometimes called loss function, by updating our weights in the direction of minimum. It is recommended to multiply this derivative with a learning rate because all neural networks learn at different rates, and we dont want to take too big or small steps in updating the weights. This learning rate must be experimented with based on the problem. For MNIST classification we used a learning rate of 0.0001.

## 4.5 Optimization Techniques

There are several optimization techniques developed for training neural network that are more sophisticated than simply taking derivative of error function. Some of the typical ones are Adam Optimizer, RMSProp Optimizer, Gradient Descent Optimizer, Adagrad Optimizer. We have found Adam optimizer best suited for this training our MNIST network. Adam optimizer takes into consideration both the momentum of gradients and also maintains per-parameter learning rates that are adapted based on the average of recent magnitudes of gradients for weight.

## 4.6 Mini batching

As we know, MNIST dataset consists of 60,000 training images and 10,000 test images. During training of neural network, it is not advisable to send all of the 60,000 input vectors into our network and update weights all at a time. This will be computationally heavy, as so many dot products of matrices must be performed at a time. In theory, sending one image at a time is also not advisable, as there is a higher chance of neural network learning all the noise in the train data, istead of the underlying function. Instead, we can send the trainig images in batches of considerable size. This has added benefit, in that, weights gets updated quickly and our neural network learns fastly.

## 4.7 Summary of Training the Full Sized Network

- Learning rate: 0.001
- Minibatch size: 256
- Loss Function: Softmax Cross Entropy
- Optimizer: Adam Optimizer
- No of Epochs: 1000
- Test Accuracy: 97 percent

# 5 SVD ON MNIST

Neural Network trained on MNIST data is computationally heavy and complex with 5 hidden layers and 1024 neurons in each hidden layer. Not only the training the model but the classification on any test image also takes more time. This is because the weight matrices in our original full sized network are of the higher rank, and dot product of them with input vectors involves more number of multiplications and additions. In order to compress the network Singular Value Decomposition is applied on all the weight matrices. SVD reduces the rank of the resultant representation of the original weight matrices. Applying Singular Value Decompostion on the weight matrix with shape (n,m) gives 3 singular matrices they are left singular matrix (U), right singular matrix (V) and a Diagonal

matrix with a singular values that is Singular Value Matrix (S) with shapes (n,D), (D,D), (D,m), where D is the maximum of n and m. Values in Singular matrix are sorted in the order of their contribution to the approximation. So it means based on the approximation and complexity trade off, the least important singular values can be discarded.

$$W = U_{:,1:D} S_{1:D,1:D} V^T_{:,1:D}$$

D value can be choosen from between 0 to maximum of m and n. Discarding lesser significant singular values in each weight matrix of the Neural Network reduces the complexity of network to a great extent. Deep Neural Network trained with 5 layers have six weight matrices and first weight matrix is of shape (784,1024). When Singular value decomposition is applied and D value is choosen 20, then the weight matrix first factorizes into 3 matrices of size (784,1024), (1024,1024) and (1024,10) and then discarding the lesser significant values other than 100 reduces the sizes of each matrix to (784,20), (20,20) and (20,10). Initializing a new network where replacing weight matrices with the matrices obtained after SVD approximation where S and V are combined into one single matrix of size (20,10) gave the same test accuracy of 98 percent.

After we get our compressed network, the test accuracy on it is less than the accuracy of the full sized network. We can further optimize our compressed network by building a new network. New Network is initialized with reduced SVD applied weight matrices and trained again. This greatly reduced the memory occupied by the model.

# 6 REST SERVICE FRAMEWORK FOR MNIST

## 6.1 REST Service

REST stands for Representational State Transfer. It is an architechtural style that defines applications that are network based, like http for example. It is based on stateless, client-server, cacheable communications protocol. REST is not based on HTTP, although it is used in most cases. In such cases, RESTful applications use HTTP requests to post data therby create/update resources, read data or resources, delete data or resources. Methods such as GET, PUSH, POST and DELETE can be used to implement a REST service [11].

## 6.2 Swagger

'Swagger is the worlds largest framework of API developer tools for the OpenAPI Specefications (OAS), ending development across entire API lifecycle, from design and documentation, to test and deployement' [3].

Major swagger tools are

- Swagger Core
- Swagger Codegen
- Swagger UI
- Swagger Editor

We have used swagger codegen to generate server side code.

We implemented REST service with Swagger following Open API 2.0 specefications. This has been done in three steps:

## 6.3 Defining our REST Service

Here, our we are developing REST service for compressing original neural network in MNIST classification problem, and we are also optimizing the compressed network by connecting to MNIST database and gatherinng the necessary data first.

We are publishing our results on a web page as a JSON object. For this we are hosting the JSON output on localhost:4874 We have defined our REST service in svd.yaml file, which contains information about description of service, host address, basepath, input and output formats etc.

## 6.4 Server Side Stub Code Generation with Swagger

First we setup the Codegen environment by installing Swagger Codegen tool. Next, we generate the server stub code with the help of jave jar file, swagger- codegen-cli.jar. We find the generated python flask code in flaskConnexion directory, with python2 compatibility.

Next we find under the flaskConnexion directory, swagger_server directory, which has folders for models and controllers, which is where conroller code resides. Now, we define the exact service that we want to implement in the default_controller.py file. To make things more simple, we have defined my stub_code in a different python file, svd_stub.py and then we linked this file to the default_controller.py file.

So our entire implementation of this project is in svd_stub file, and the function svd_example () returns a string, which has entire information of the result for this project, different values of size of SVD considered (D), initial test accuracy of the MNIST network, final test accuracy of SVD compressed network for each value of D.

## 6.5 Install and Run the REST Service

First we make sure to install the latest pip, as we will be needing it to install other packages. Next we install requirements.txt file that was generated earlier in the folder flaskConnexion. Next we install server side code in setup.py file.

Next for this project we need some Machine learning and deep learning packages in python language. With the help of pip, we installed libraries like numpy, pandas, scipy, scikit-learn and tensorflow.

# 7 DOCKERIZING OUR WORK

We have made our entire project capable of running anywhere in cloud, on any platform by creating container image and dockerizing it.

'A Docker is the company driving the container movement and the only container platform to address every application across the hybrid cloud' [2].

'A container image is a lightweight, stand-alone, executable package of a piece of software that includes everything related to run it: code, runtime, system tools, system libraries, settings' [1].

We have used official docker image for Ubuntu for our docker. We have also installed necessary tools, packages required for the entire project, including tools required for Rest service with Swagger.

5

## 8 CONCLUSION

In summary, we had succesfully built a REST service framework that connects to MNIST database, downloads image data, builds a full sized Neural Net, thereafter, performs Singular Value Decomposition of the full sized network, for different values of size of SVD, then further more impproves the compressed network, by further training, then finally outputs all the test accuracies as a JSON object in a website hosted at localhost:8080/api/svd.This optimized and compressed neural network that we have built has a lot of image classification applications, especially mobile type, which require shorter computation time.

## 9 WORK BREAKDOWN

Priyadarshini Vijjigiri has worked on python code for building Deep Neural Network model and optimizing it for MNIST Data. Goutham Arra has worked on Applying Singular value Decomposition and reducing complexity of Network. Restful webservice with swagger and Dockerizing the swagger was equally contributed by Priyadarshini Vijjigiri and Goutham Arra. Project paper was equally contributed. by both the team members.


## ACKNOWLEDGMENTS

The authors would like to thank Dr. Gregor von Laszewski for his support and suggestions to write this paper.


## REFERENCES


[1] 2018. *Container.* Docker Inc. https://www.docker.com/what-container
[2] 2018. *Docker.* Docker Inc. https://www.docker.com/what-docker
[3] 2018. *Swagger Service.* SmartBear Software. https://swagger.io/
[4] Edwin D. de Jong. [n. d.]. *A Handwritten Digit.* https://edwin-de-jong.github.io/blog/mnist-sequence-data/
[5] Leif Johnson. 2015. *Classifying MNIST Digits.* Theanets. http://theanets.readthedocs.io/en/stable/examples/mnist-classifier.html
[6] Yann LeCun. [n. d.]. *The MINST Database.* http://yann.lecun.com/exdb/mnist/
[7] Abhishek Rao. 2015. *Deep Neural Networks.* Research Gate. https://www.researchgate.net/figure/Deep-Neural-Network-Has-multiple-hidden-layers_fig4_301597227
[8] tensorflow. [n. d.]. *MNIST for ML Beginners.* https://www.tensorflow.org/versions/r1.1/get_started/mnist/beginners
[9] Avinash Sharma V. 2017. *Understanding Activation functions in Neural Networks.* Theory of Everything. https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0
[10] Avinash Sharma V. 2017. *Understanding Activation functions in Neural Networks.* Theory of Everything. https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0
[11] wikipedia. 2008. *Rest Service.* Research Gate. https://en.wikipedia.org/wiki/Representational_state_transfer
[12] wikipedia. 2018. *Singular Value Decomposition.* https://www.wikiwand.com/en/Singular-value_decomposition

# Location based crime data search and analysis with Spark UDF

Kadupitiya Kadupitige
Indiana University
Smith Research Center
Bloomington, IN 47408, USA
jasakadu@iu.edu

Gregor von Laszewski
Indiana University
Smith Research Center
Bloomington, IN 47408, USA
laszewski@gmail.com

## ABSTRACT

Crimes are horrifying and cost the progress of mankind in many ways while influencing our society and day to day life. Because of that, higher priority is always given by the government and politicians to reduce the crime rate around the world. Crime solving and prevention get notoriously difficult due to complex causes such as poverty, parental neglect, low self-esteem, alcohol, drug abuse and etc. Crime data analysis could be helpful to understand the patterns hidden inside the crime data. One good way to prevent crimes is to raise the awareness of the crime prone areas and the patterns in which crime are committed. In this paper, location based crime data search and analysis framework is introduced and developed using a judicious combination of Spark user defined functions and Haversine method. The implemented framework allows users to analyze and see the crime patterns exist near a specific location defined with an address and a radius. Framework provides crime data analysis in geospatial, time series and statistical forms. Using this framework, we were able to clearly see seasonal effect in crime data near Chicago city area and we also noticed that crime rate is monotonically decreasing when we move away from the Chicago city area.

## KEYWORDS

hid-sp18-409, Volume: 9, Chapter: REST, Status: 100.
Crime data analysis, MapReduce, Google Maps, spark UDFs

## 1 INTRODUCTION

A crime is generally defined as an act punishable by law and is one of the dangerous factors for any country[2]. It is impossible to find a country with a crime- free society due to complex causes such as poverty, parental neglect, low self-esteem, alcohol, drug abuse and etc [4, 12]. Inspired by the big data revolution, the historical way of crime solving and prevention has greatly influenced and improved by the help of state-of-the-art data analytic tools and machine learning research. Law enforcement officers have started involving more data analytic expertises to speedup the crime solving process highlighting that, it is an interdisciplinary research area.

According to Chicago Police Department, crime prevention is important and much more difficult to handle than crime solving for law enforcement agencies [6]. Law enforcement officers could identify crime prone areas or suspicious activities even before a crime is committed with the aid of pattern recognition and big data analysis [11, 15]. According to the Los Angeles Police Department (LAPD) and Chicago Police Department (CPD), whenever a crime was committed somewhere, more crimes are likely to be occurred in a nearby area, and the patterns of those criminal activities could be identified through big data analysis [6, 13]. However, crime prevention could be so much effective if there is an easy way to check the crime prone areas so that people could be aware the of crime prone areas.

As inspired by the requirement to raise the awareness of crime prone areas, we implement a location based crime search website where people can find crimes happened near a particular geographical location (in a vicinity area with inserted radius) for a given address. For this project, we have selected a dataset which includes the crime information from 2001 to current date in Chicago city [9]. We further analyze this crime dataset using the crime type, incident location and timestamps to identify the trends of crime types for a particular geographical area. There are many applicational use for this kind of a project such as a user trying to buy a new house, may like to do a background search on criminal activities around the geographical area for safety concerns and a user who would like to visit some place would love to know the seasonal and time effect on the crime rate in that location before booking a hotel.

The organization of this project report is as follows. In section II, a literature review on existing crime data analyses and crime reporting web services are presented. Section III provides a methodology followed in this project while section IV elaborates the results and discussion. A summary about the project paper is provided in section V.

## 2 LITERATURE REVIEW

The recent research and development in crime data investigation area could be broadly categorized into two categories; analysis of criminology [2, 4, 11, 12, 15] and applicational development of crime data visualization platforms [7, 8, 14, 19]. Analysis of criminology data could be further categorized into crime control and crime suppression [2]. There exists many research focused on analyzing crime data based on different methodologies such as kmeans cluster analysis [2, 11, 15], time series analysis [20], variational data mining techniques [5], geographical analysis using using statistical techniques [17] and etc. This study experiments combinations of these techniques to build a crime data investigation platform and hence, they are discussed in this section.

Agarwal et al. [2] have investigated a comprehensive crime data analysis to identify the crime patterns and predict crimes based on spatial data distribution of existing data using kmeans clustering based approach and rapid miner tool. They have only focused on homicide crime type and plotted the number of reported crimes with respect to year for different clusters. This analysis had helped them to conclude that homicide is decreasing from 1990 to 2011 in in England and Wales [2]. Following Agarwal [2], Kiani et al. [12] have also studied the same dataset to improve the clustering approach by assigning weights to the features in the dataset and using Genetic Algorithm (GA) to optimize the Outlier Detection

using the same rapid miner tool. Gera et al. [11] have studied the possibility of profiling the crimes using cluster analysis. Gera et al. [11] have analyzed all type of crimes happened in Delhi police commission as well as through National Crime Records Bureau (NCRB), Delhi, India. They have created a crime database by interviewing concerned officers, through semi-structured interviews, group discussion, participant observation, documentation analysis and questionnaires in Delhi region to perform two types of crime profiling based on types of crimes and types of areas. Finally an association between two profiling methods were identified using weka data mining tool. Similarly, Nath et al. [15] also applied several clustering algorithms to identify the association in crime data taken from sherifffis office in the city of New Orleans. They have also used a semi-supervised learning technique for knowledge discovery from the crime records and to help increase the predictive accuracy in the crime data classification task.

Extensive time series analysis on Philadelphia crime data has been studied by Wei et al. [20]. They have compared stationary and non stationary models, nonseasonal and seasonal models, intervention and outlier models, transfer function models, regression time series models, vector time series models, and their applications using the crime data. They have further elaborated the procedures of time series analysis including parameter estimation, diagnostic checks, forecasting, and inference using autoregressive conditional heteroscedasticity model and generalized autoregressive conditional heteroscedasticity model [20].

Recent research done by DnzzL et al. [10], Abenassi et al. [1], Agsarthak et al. [3] and Opencity et al. [16] have focused on investigation of Chicago city crime dataset [9]. DnzzL et al. [10] have compared the crime prediction accuracies for logistic regression, multilayer perception and random forest algorithms and reported the highest accuracy 86.5% for multilayer perception based methodology. Similarly, Agsarthak et al. [3] have also used the same dataset for crime forecasting in Chicago city using machine learning Rest API provided by Microsoft Azure. They had also performed a statistical analysis and time series analysis on the crime data [3]. Abenassi et al. [1] have implemented a web based software to analyze the Chicago city crime data by primary type and the time. Similarly, Opencity et al. [16] have also implemented a web interface to analyze the crimes with year, wards, type and district. All these research and development had only focused on analyzing the dataset but have not implemented a user friendly web interface where users could search crimes near a given address and analyze the geographical location that they are actually interested.

There are few web based platforms which allow user to find the crime incidents near a particular address or GPS location such as Spotcrime [19], Crimereports [8], Mylocalcrime [14] and Crimemapping [7]. All these web based platforms only show geographical locations of the crimes and lacks crime data analysis with time and other categories. Those also lacks many important features such as variable radius search, address auto complete, clustering and highlighting crime prone areas and etc. However, these approaches could be enhanced up a great extend. Due to these facts, this project tries to implement a location based crime data search website which allows users to search crimes near a given address and analyze the crime data in one single web interface.

# 3 METHODOLOGY

As described in the literature review, most of the research on crime data investigation focused on analyzing crimes by type of the crimes, time series information and geographical location. These techniques could be extended to a great extent for our crime data framework. In this section, we describe the data acquisition, data preprocessing, technology usage and implementation details of individual modules based on python implementation.

## 3.1 Data Acquisition

We have selected a dataset which includes the crime information from 2001 to current date in Chicago city [9]. Dataset is created with the reported incidents of crime (with the exception of murders where data exists for each victim) that occurred in the City of Chicago from 2001 to present, without the most recent seven days. Data is extracted from the Chicago Police Department's Citizen Law Enforcement Analysis and Reporting system. In order to protect the privacy of crime victims, addresses are shown at the block level only and specific locations are not identified. Datasets size is 1.5 gigabytes with 22 columns and over 6 Million rows. Attribute information about the dataset are as follows:

**ID:** Unique identifier for the record.
**Case Number:** The Chicago Police Department record Number.
**Date:** Date when the incident occurred.
**Address:** The partially redacted address where the incident occurred.
**IUCR:** The Illinois Unifrom Crime Reporting code.
**Primary Type:** The primary description of the IUCR code.
**Description:** The secondary description of the IUCR code.
**Location Description:** Description of the location where the incident occurred.
**Arrest:** Indicates whether an arrest was made or not.
**Domestic:** Indicates whether the incident was domestic-related.
**Beat:** Indicates the beat where the incident occurred.
**District:** Indicates the police district where the incident occurred.
**Ward:** The ward where the incident occurred.
**Community Area:** Indicates the community area where the incident occurred.
**FBI Code:** Indicates the crime classification as outlined in the FBI.
**X Coordinate:** The x coordinate of the location where the incident occurred. This location is shifted from the actual location for partial redaction but falls on the same block.
**Y Coordinate:** The y coordinate of the location where the incident occurred. This location also is shifted from the actual location for partial redaction but falls on the same block.
**Year:** Year the incident occurred.
**Updated On:** Date and time the record was last updated.
**Latitude:** The latitude of the location where the incident occurred.
**Longitude:** The longitude of the location where the incident occurred.

[2]

**Location:** The location where the incident occurred in maps format.

## 3.2 preprocessing

When sweep over the rows in the dataset, we noted that many missing values in some of the rows and columns. Hence before using the dataset, it is imperative to treat the missing values. Our analysis highly depends on the column such as ID, Case Number, Date, Primary Type, Location Description, Arrest, Latitude and Longitude. So we have neglected the data rows where we found null values in any of the important columns. We used python programming environment and Pandas data processing library to achieve this task. These functionalities are built in to the data handling APIs in our framework and automatically triggered when update the data files using provided data source update swagger services.

## 3.3 Technology Usage

We used Python as the main programming language in this project and few web technologies such as HTML, CSS, JQuery and Google maps java script API are used for the front end development of the project. Full description of programming environments and library packages which used to implement the crime analyzing and visualization framework, are shown in Table 1.

**Table 1: Technologies used in the project**

| Technology | Version |
|---|---|
| Python programming environment | 3.6 |
| Flask | 0.12 |
| connexion | 1.1.15 |
| decorator | 4.2.1 |
| python-dateutil | 2.6.0 |
| setuptools | 21.0.0 |
| numpy | 1.14.0 |
| scipy | 0.18.1 |
| pandas | 0.20.3 |
| scikit-learn | 0.18 |
| pyspark | 2.1.1 |
| scikit-learn | 0.18 |
| java run time environment | 8.16.1 |
| Apache Spark standalone version | 2.3.0 |
| Swagger codegen | 2.1.2 |
| HTML | 5 |
| CSS | - |
| Java Script | - |
| JQuery | - |
| Google Maps Java script API | - |
| Dygraphs | - |
| Docker | 1.13.1 |

## 3.4 Overview of crime search framework

Figure 1 shows the suggested framework for crime data analysis and visualization. As shown in the Figure 1 there two main users in the system such as regular user and the admin user. Regular user is only accessing the Flask web application (see Section 3.7) to see the crime data trends in entire chichago city or in a specific area selected by the user with a desired address and a radius. This is achieved through the micro services defined in the swagger service explained in the Section 3.6. Admin user can trigger data source update and spark map reduce with basic authentication as elaborated in the Section 3.5. Screen captures of the Flask web application are shown in Figure 2. The complete framework is dockerized to run within a Docker container allowing easy installation.

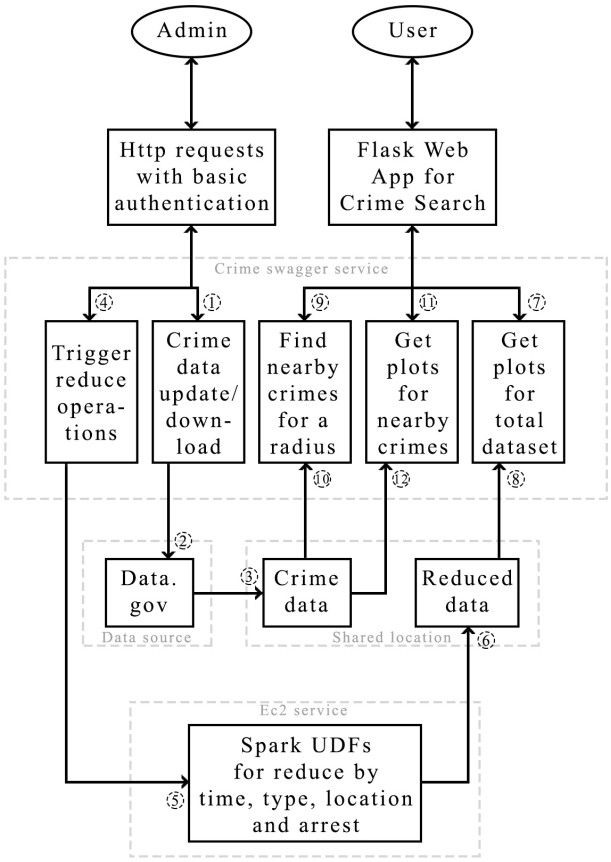


**Figure 1: Overview of suggested framework for crime data analysis**

## 3.5 Spark MapReduce module for crime analysis using time, type and location

Since the dataset described in Section 3.1 is so large to handle in single core computer, we have implemented Spark MapReduce as user defined functions (UDFs) for analyzing the dataset using time series information, basic crime category and reported location of the incident. These functionalities are only executed when the data source is updated by the admin user. Following pseudocode explains how we have implemented Spark MapReduce for the crime dataset.

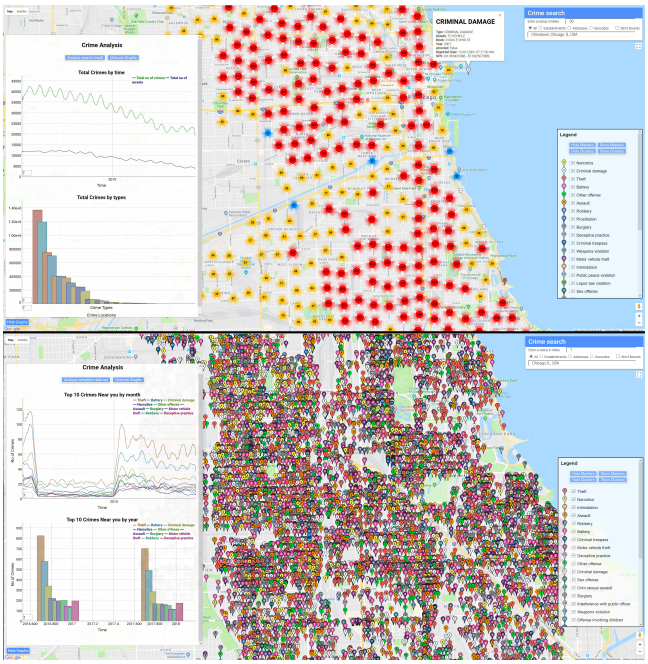

**Figure 2: Screen captures of the Flask web application**

```
Create_SparkSQL_session()
Load_data_with_pandas()
Process_data_to_get_desired_data_columns()
Create_Spark_context_with_data()
Create_RDD_of_Row_objects_using_groups()
Do_MapReduce_based_on_groups()
```

## 3.6 Swagger web service for location based Crime analysis

As shown in Figure 1, this swagger service contains nine micro services to serve the Flask web application and admin user. It also connect the admin user with Spark Mapreduce module and stores the reduced data. As the dataset contained GPS coordinate for every reported crime, we used famous Haversine formula to calculated the distance between the users GPS location and the data points using Equation 1 and 2 [18].

$$hav(\frac{d}{r}) = hav(\omega_2 - \omega_1) + \cos(\omega_1)\cos(\omega_2)hav(\lambda_2 - \lambda_1) \quad (1)$$

$$hav(\theta) = \sin^2(\frac{\theta}{2}) = \frac{1 - \cos(\theta)}{2} \quad (2)$$

Where:

- $d$ is the radius of the sphere,
- $\omega_1, \omega_2$ are latitude of point 1 and latitude of point 2, in radians,
- $\lambda_1, \lambda_1$ are longitude of point 1 and longitude of point 2, in radians.

Following list contains the micro services defined and implemented inside the swagger service.

**getCrime:** User can search for a particular crime with unique crime Id using this API.

**getCrimes:** The getCrimes API returns information about the crimes previously happened at a given location or nearby locations based on users GPS coordinates and radius. The response includes lists of crimes in the proper display order.

**getFilteredCrimes:** This API endpoint returns information about the crimes previously happened at a given location or nearby locations based on userfis GPS coordinates, radius and a primary type (Example- BATTERY).

**getPrimaryCrimeList:** User can get primary crime types as a list using this API.

**getCrimesByMonth:** User can get top x number of crimes by month using this API.

**getCrimesByYear:** User can get top x number of crimes by year using this API.

**downloadData:** This API downloads the data from the data source difined in the config.yml file.

**updateData:** This API updates the data from the data source difined in the config.yml file.

**triggerSparkFuctions:** This API calls the Spark UDFs to generate the reduced data files for total crime dataset analysis.

## 3.7 Flask web application for visualization

As illustrated in Figure 1, Flask web application is made for users to perform the crime data search based on their geographical location and to visualize the crime data analysis by utilizing the implemented swagger services described in Section 3.6. We have used Google maps javascript API to showcase the crime data in correct geogrphical location. Following sub sections describes the features of the developed web interface.

*3.7.1 Auto completed Addresses Search.* We have used Google maps javascript address auto complete API to provide a type-ahead search box for user to enter a address of any location such as Establishment, Addresses and Geocodes. Figure 3 shows the GUI widget which we have created for this to achieve the address search. The radio buttons allow users to filter the types of places that the autocomplete returns. The Strict Bounds option restricts the search to the area within the current viewport. If this option is not checked, the API biases the search to the current viewport, but does not restrict it. When a user search this address it automatically creates a marker on the Google map in the web interface. Inside this widget functionality, we obtain the GPS coordinate for that particular location and calls the "getCrimes" API described in Section 3.6 to get the crime data points relevant to the GPS coordinate and the radius entered inside the above mentioned widget.

*3.7.2 Show crimes data in markers mode.* We have created a custom markers for each of the primary crime types and kept those as templates to be used for creating markers for the search results from the "getCrimes" API call described in Section 3.7.1. Bottom image in Figure 2 shows how the markers are shown in map. A legend about each of the crime types are put on the right bottom pane as shown in Figure 2. User can turn on or off each of the crime types individually or group wise. When a user click on a marker,

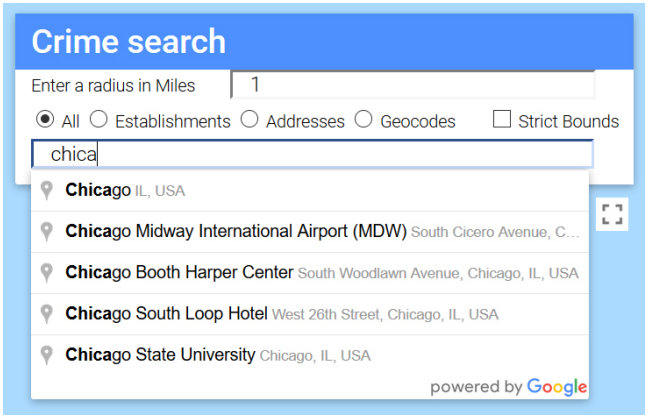


**Figure 3: Auto complete address search widget**

a pop up window will appears as shown in top right area of the Figure 2 and it contain important information about that particular crime which was clicked. We have also implemented some nice animation effects such as markers bouncing, dropping and toggling to enhance the user experience with the web interface. If needed user also can turn on or off all markers using the functionality provided in the legend pane.

*3.7.3 Show crimes data in cluster mode.* This cluster mode is created from the markers described in the Section 3.7.2 and there is a button placed on the legend pane to activated the cluster mode and it automatically deactivate the marker mode. This clustering mode is sensitive to the google maps zoom level and it automatically reclusters based on the map zoom level. We have achieved this using Google Marker Clusterer library which uses the grid-based clustering technique that divides the map into squares of a certain size (the size sensitive to the zoom level), and groups the markers into each square grid. It creates a cluster at a particular marker, and adds markers that are in its bounds to the cluster. It repeats this process until all markers are allocated to the closest grid-based marker clusters based on the map's zoom level. The number on a cluster indicates how many markers it contains.

*3.7.4 Plots for crime data analysis.* There are two types of graphs created in the project such as graphs which contains the information about entire crime dataset and graphs which only contains the information about crime data a user searched using a radius and an address as explained in Section 3.7.1. Both these types perform MapReduce operation on the data considering time, type and location information. The difference is that the graphs which requires information about entire dataset, are plotted using the reduced data from the Spark MapReduced module explained in the Section 3.5. These modes are enabled when user clicks on button called "Show Graphs" in the bottom left of the web interface. This button automatically changes to "Hide Graphs" and shows the left pane (where graphs are plotted) shown in Figure 2 as soon as user click on it. Immediate plots demonstrate the plots for total crime by time, type and location as shown in top left of the Figure 2. User can click on the "Analyze Search Result" button to see the top ten crime trends near the location which was searched. These graphs were created using the Dygraphs javascript library and contains many responsive features such as drag on axis, zoom and unzoom, move along axis to increase ability of the data visualization.

## 4 RESULTS AND DISCUSSION

This section provides the results we have achieved through the implemented crime data search and analyzing framework and provides a discussion about the results. It contains the benchmarking results, time series analysis results, geospatial analysis results and statistical analysis results.

### 4.1 Benchmarking

We measured and analyzed the performance of the crime search framework on three different hardware infrastructures as listed follows:

**Asuz Laptop:** 2 hardware cores, 2 threads per core, 8th Generation Intel Core i5-8300H processor (up to 3.9 GHz) with 8 GB DDR4 high-frequency 2666 MHz RAM, 1 TB SSHD hard drive and GeForce GTX 1050 1GB VGA.

**Dell Workstation:** 10 hardware cores, 2 threads per core, Intel Xeon Processor E5-2630 V4 Family (up to 3.1 GHz) with 32 GB DDR4 high-frequency 2133 MHz RAM, 500 GB SSD hard drive and NVIDIA Quadro M2000 4GB VGA.

**EC2 EMR (m4.xlarge instances):** 5 nodes (1 master and 4 worker), 18 hardware cores per node, 2 threads per core, Intel Xeon Processor E5-2686 v4 Family (up to 3.0 GHz) with 32 GB DDR4 high-frequency DDR4-2400 MHz RAM, with Spark 1.6 on Hadoop 2.6.0 YARN.

Since our frameworks takes an address and a radius value from the user as explained in Section 3.7.1, the data size for MapReduce operation increases with the radius. So we measured time it takes to perform the map reduce operations with different radius (different data sizes) values on all three computing platforms explained above. As our data set is from Chichago city area, we increase the radius from 0.1 miles to 25 miles. The total execution times (in minutes) for different data sizes (number of crimes) in three different computing platforms are shown in the Table 2. We were able achieve a maximum speed up of 32.5 with total 144 threads in EC2 m4.xlarge instances for the complete dataset. Figure 4 also shows the Performance comparison of Crime Search Framework using execution times in minutes for different data sizes in different computing platforms. Execution time breakdown in minutes for different components fo the system in different computing platforms for radius of 1.6 miles are shown in Table 3. It can be observed that complete system starts well in both EC2 EMR and workstation computing resources.

### 4.2 Time Series Analysis

Our crime search and analyzing framework allows user to analyze crime data under two modes such as total crime data analysis in the Chicago city and investigate the crime data relevant to the crime search performed by the user. Under the total crime data investigation mode, user can see the total number of crimes reported against the time stamp as shown in the Figure 5. This graph is dynamic and user can zoom the x axis of the graph up to month level to see the total number of crimes. Figure 5 allows us to infer a

5

**Table 2: Total execution times in minutes for different data sizes in different computing platforms**

| Radius | Data size | laptop | workstation | EC2 EMR |
|---|---|---|---|---|
| Processes | / threads | 4 | 20 | 144 |
| Time | in | Mins (m) | (m) | (m) |
| 0.1 | 52664 | 16 | 3 | 2 |
| 0.2 | 195334 | 23 | 5 | 2 |
| 0.4 | 444744 | 39 | 7 | 3 |
| 0.8 | 995512 | 68 | 12 | 3 |
| 1.6 | 1859480 | 102 | 19 | 3 |
| 3.2 | 3289296 | 172 | 32 | 5 |
| 6.4 | 5195988 | 307 | 58 | 10 |
| 12.8 | 6516889 | 364 | 72 | 12 |
| 25.6 | 6568036 | 391 | 75 | 12 |

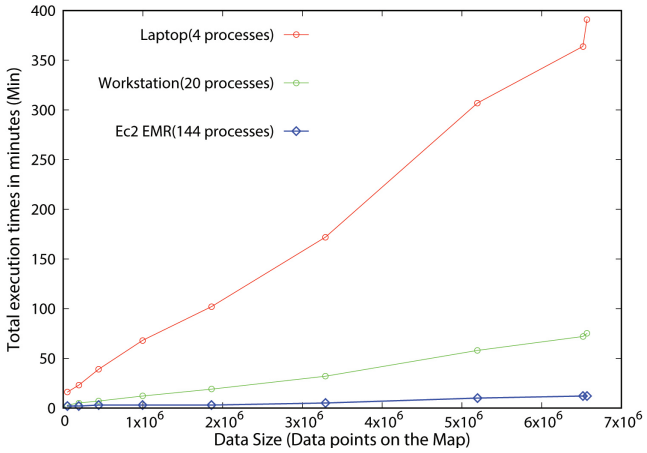


**Figure 4: Performance comparison of Crime Search Framework using execution times in minutes for different data sizes in different computing platforms**

noticeable seasonal effect in the total crimes reported in Chicago city. It can be observed that there is a trend where the total number of crimes always low from January to July and always high from August to December. Even though the total reported crimes has increasing and decreasing trend and seasonality, We also notice that total crime reported, has a downward trend from 2001 to 2018.

As our framework offers users to investigate the crime data based on the search, They can choose to observe 10 most frequently reported crimes near the searched location with respect month or year as shown in Figure 6 and Figure 7 respectively. User can learn a good insight about a particular location and see the most safest time periods to be in the searched place.

## 4.3 Geospatial Analysis

As explained in the Section 3.7.2 and 3.7.3, user can search for a location with a radius and get to know about 34 different types of crimes reported in that particular area using the implemented crime search framework. Results figures shown here are obtained

**Table 3: Execution time breakdown in minutes for different components in different computing platforms for radius of 1.6 miles**

| Component | laptop | workstation | EC2 EMR |
|---|---|---|---|
| Processes | 4 | 20 | 144 |
| Time in | Seconds (s) | (s) | (s) |
| Data preprocessing | 30.5 | 20.1 | 32.8 |
| Setup time of Swagger service | 15.8 | 15.6 | 15.9 |
| Setup time of Spark module | 120.1 | 95.4 | 90.8 |
| Setup time of Flask webApp | 14.8 | 12.6 | 13.5 |
| Time for full system to launch | 187.5 | 149.4 | 157.3 |

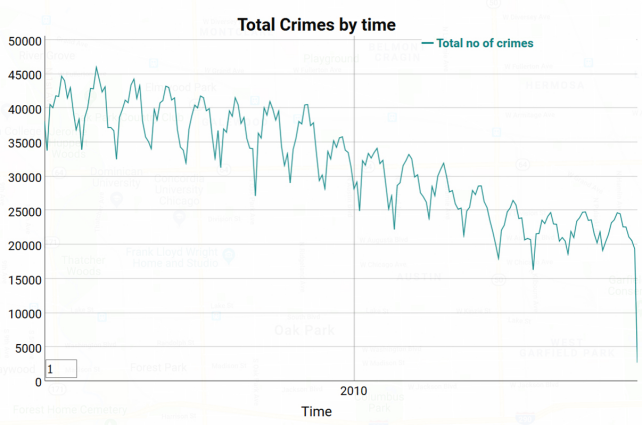


**Figure 5: Total crimes for each month in Chichago from 2001-01 to 2018-02**

by searching the address of Chichago Cook county with radius of 1.6 miles. Figure 8 and Figure 9 shows the search results in marker mode and cluster mode respectively. We can observe that the center of the Chichago downtown area has more reported crimes than rest of the area as shown by the cluster mode in Figure 9.

Figure 10 shows the bar-graph of total crimes grouped by the location in which the crime was committed. We can clearly see that the highest number of reported crimes were committed on a street. Figure 11 shows how total number of reported crimes varies when we move away from the center of the Chicago city and we can observe that the gradient of the graph is rapidly decreasing. So we can infer that the crime rate is high when you move towards the Chicago city area.

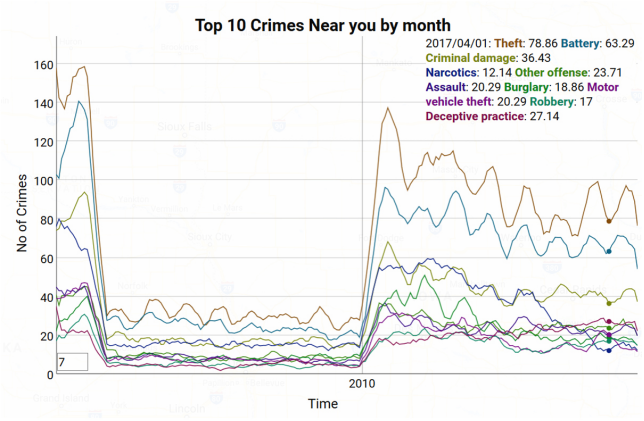


**Figure 6: Most frequent crimes near (1.6 miles) Chichago Cook county for each month from 2001-01 to 2018-02**

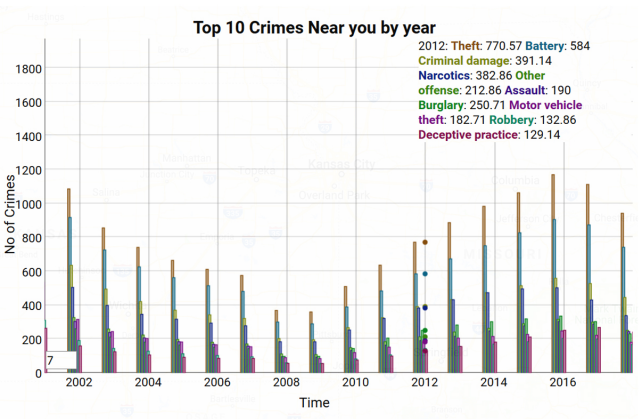


**Figure 7: Most frequent crimes near (1.6 miles) Chichago Cook county for each year from 2001 to 2018**

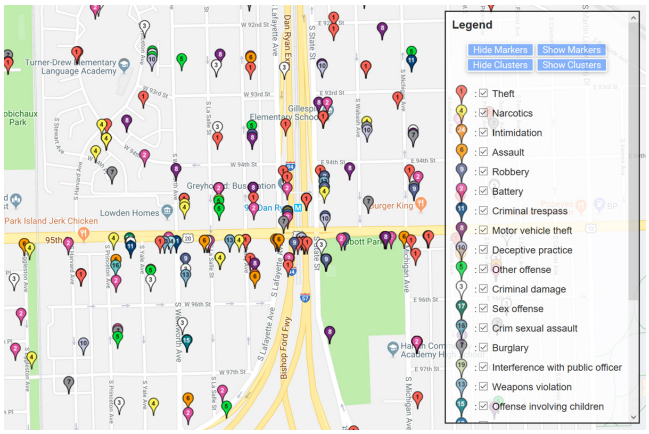


**Figure 8: Different types of crimes distribution on Google maps near (1.6 miles) Chichago Cook county**

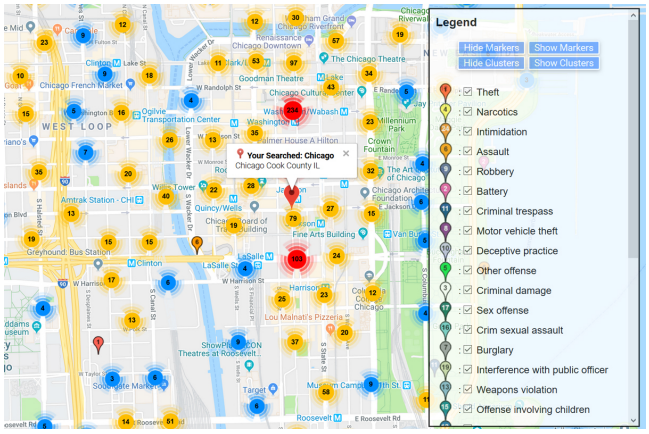


**Figure 9: Cluster mode for total crimes distribution on Google maps near (1.6 miles) Chichago Cook county**

## 4.4 Statistical Analysis

Our framework generates a time series plot that displays the comparison between the number of crime cases registered and the number of arrests with respect to time stamp as shown in Figure 12. User can zoom in to this graph if need to see the comparison in month level. We can clearly observe that total number of arrest are always really low compared to the reported crimes. Figure 13 shows a bar-graph of total crimes grouped by the crime type according to the Chicago police department. Theft is the highest number of committed crime in the Chicago city from 2001 to 2018. Figure 6 also proves that theft is the highest reported crime for a localized search near Chicago city.

## 5 CONCLUSION AND FUTURE WORKS

This paper presents a new framework for location based crime data search and analysis using Spark UDF. We have exposed the framework functionalities to users using a Flask web application where users can enter an address and a radius to see the spatial distribution of crime data by the crime type. It also enables time series analysis and statistical analysis for the searched crime data as well the full dataset. Currently, the framework is only powered by the Chicago city crime data set. As we have made the framework fairly generic, it should be easy to integrated other crime data sources. Using the implemented framework, we have reported the results under three sections such as time series, geospatial and statistical analyzes. We observed that there is a trend where the total number of crimes always low from January to July and always high from August to December. The total reported crimes has a downward trend from 2001 to 2018. We also observed that the highest number of reported crimes were committed on a street. By using a moving radius method, we were able to infer that the crime rate is high when you move towards the Chicago city area. We also saw that total number of arrest are always really low compared to the reported crimes.

One of the most important issues that should be addressed in the model presented in this paper to improve the MapReduce operations as they are taking so much time with larger radius values. As this

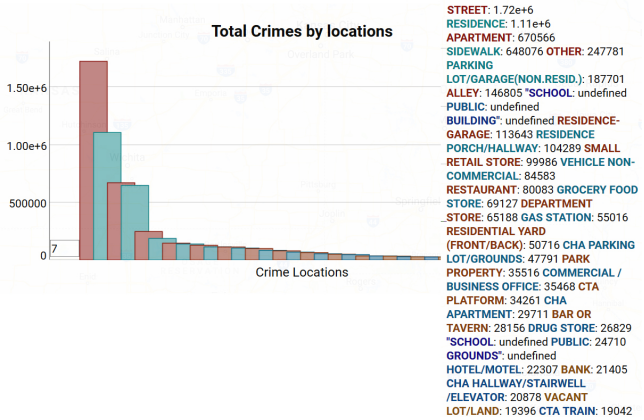


**Figure 10: Total Crimes by different locations in Chichago city from 2001 to 2018**

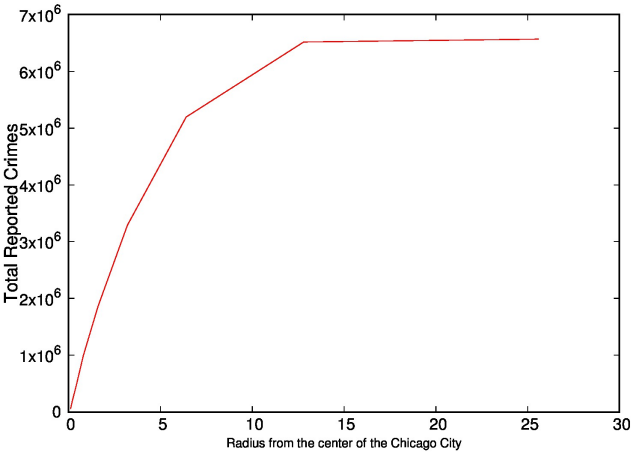


**Figure 11: Total Crimes by with increasing radius from the center of the Chichago city**

framework is currently only based on the crime data obtained from Chicago city, we could try to integrate with other available crime dataset for other cities to check the scalability of the framework and to enhance the experience of the users. Our Flask web application is not mobile friendly and we could make it more responsive to all sort of screens.

## ACKNOWLEDGMENTS

The authors would like to thank Dr. Gregor von Laszewski for his support and suggestions to write this extended abstract.

## REFERENCES

[1] Abenassi. 2015. Small visualization of crimes in Chicago by primary type and time. Web Page. (2015). https://github.com/abenassi/Chicago-Crime Accessed: 2018-04-11.

[2] Jyoti Agarwal, Renuka Nagpal, and Rajni Sehgal. 2013. Crime analysis using K-means clustering. *International Journal of Computer Applications* 83, 4 (2013).

[3] Agsarthak. 2017. Analyzing Chicago crime data set and applying Machine learning. Web Page. (2017). https://github.com/agsarthak/Chicago-Crime-Dataset-ML Accessed: 2018-04-11.

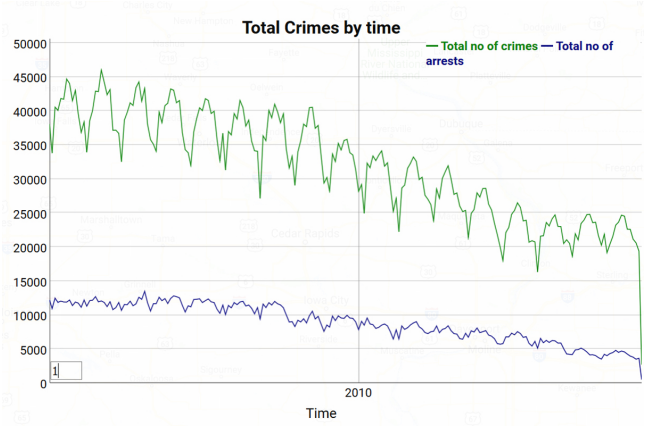


**Figure 12: Total crimes vss total arrests for each month in Chichago from 2001-01 to 2018-02**

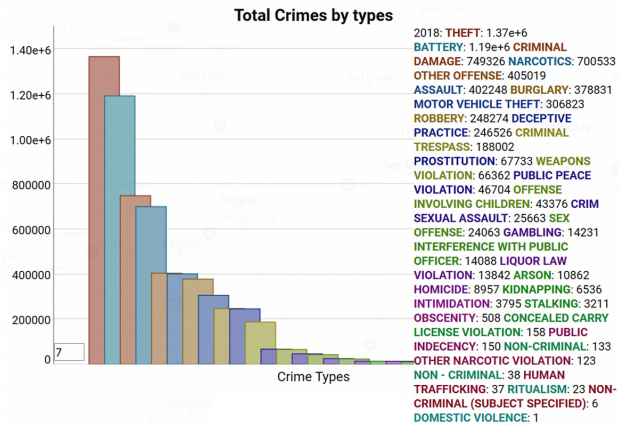


**Figure 13: Total Crimes by crime types in Chichago city from 2001 to 2018**

[4] A Shilpa Bharathi and R Shilpa. 2014. A Survey on Crime Data Analysis of Data Mining Using Clustering Techniques. *International Journal of Advance Research in Computer Science and Management Studies* 2, 8 (2014), 9–13.

[5] Hsinchun Chen, Wingyan Chung, Jennifer Jie Xu, Gang Wang, Yi Qin, and Michael Chau. 2004. Crime data mining: a general framework and some examples. *computer* 37, 4 (2004), 50–56.

[6] CPD. 2016. Crime data analysis: Chicago Police Department. Web Page. (2016). https://home.chicagopolice.org/online-services/crime-statistics/ Accessed: 2018-04-01.

[7] Crimemapping. 2015. Crimemapping crime search. Web Page. (2015). https://www.crimemapping.com/ Accessed: 2018-04-10.

[8] Crimereports. 2016. Crimereports crime search. Web Page. (2016). https://www.crimereports.com/ Accessed: 2018-04-10.

[9] DATA.GOV. 2018. Crimes - 2001 to present, City of Chicago. Web Page. (2018). https://catalog.data.gov/dataset/crimes-2001-to-present-398a4 Accessed: 2018-04-01.

[10] DnzzL. 2017. Data Analytics project - Dublin City University. Web Page. (2017). https://github.com/DnzzL/Chicago-Crimes-Dataset Accessed: 2018-04-11.

[11] Priyanka Gera and Rajan Vohra. 2014. City Crime Profiling Using Cluster Analysis. *Priyanka Gera et al,/(IJCSIT) International Journal of Computer Science and Information Technologies* 5, 4 (2014), 5145–5148.

[12] Rasoul Kiani, Siamak Mahdavi, and Amin Keshavarzi. 2015. Analysis and prediction of crimes by clustering and classification. *Analysis* 4, 8 (2015).

[13] LAPD. 2017. Crime data analysis: Los Angeles Police Department. Web Page. (2017). http://www.lapdonline.org/crime_mapping_and_compstat/content_basic_view/6363# Accessed: 2018-04-01.

[14] Mylocalcrime. 2017. Mylocalcrime crime search. Web Page. (2017). https://www.mylocalcrime.com/ Accessed: 2018-04-10.
[15] Shyam Varan Nath. 2006. Crime pattern detection using data mining. In *Web intelligence and intelligent agent technology workshops, 2006. wi-iat 2006 workshops. 2006 ieee/wic/acm international conference on*. IEEE, 41–44.
[16] Open-city. 2013. A data visualization that lets you explore crime trends in Chicago's 50 wards. Web Page. (2013). https://github.com/abenassi/Chicago-Crime Accessed: 2018-04-11.
[17] Rachel Boba Santos. 2016. *Crime analysis with crime mapping*. Sage publications.
[18] Andysah Putera Utama Siahaan. 2017. Haversine Method in Looking for the Nearest Masjid. (2017).
[19] SpotCrime. 2017. SpotCrime crime search. Web Page. (2017). https://spotcrime.com/ Accessed: 2018-04-10.
[20] William WS Wei. 2006. Time series analysis. In *The Oxford Handbook of Quantitative Methods in Psychology: Vol. 2*.

# Twitter Sentimental Analysis with Spark

Venkatesh Aditya Kaveripakam
Indiana University
Bloomington, IN 47408, USA
vekave@iu.edu

Surya Prakash Sekar
Indiana University
Bloomington, IN 47408, USA
sursekar@iu.edu

## ABSTRACT

In this project we have built a RESTful webservice that will accept a topic of interest as input from the user which is the twitter hashtag in this case and query the relevant hashtag data from the twitter API using tweepy package from python and making use of its cursors. All pre-processing and data cleaning involved with the tweets data will be handled in python using a variety of tweet pre-processing techniques and a consumable dataset for spark is created. A sentimental analysis model is created using python's textblob as well in order to label the tweets before in hand for the hashtag data that is selected by the user. SparkML library is then used to perform sentimental analysis on the tweets data and a summary is provided with respect to it. A graph displaying the distribution of the tweets based on the sentiment of the input hashtag is displayed. We have hosted the webservice that runs using the textblob model on digital ocean, which is a cloud platform with the help of a docker image built on Ubuntu. The dockerized package uses the textblob model for sentimental analysis classifier.

## KEYWORDS

hid-sp18-411,hid-sp18-418, Volume: 9, Chapter: Spark, Status: 100. Swagger, Spark, API, Cloud, Twitter

## 1 INTRODUCTION

A lot of research related to social media data would provide a relevant guide on how social media data can be used in research, by the inherent dynamism in society instantaneously reflected in social media demands a breakaway from the traditional process of academic publishing. This is true of many fields, but is especially relevant when we consider the immediacy and mutability of social media data. In order to truly capture the potential of social media, we need to explore methods of conducting and disseminating research that can keep up with the pace of modern life. Data in social media represents users behavior, attitudes, feelings, views and relationships are increasingly being re-used for research purposes in in both academia and industry. Even though social media has entered the deeper circle of our culture, we must not overlook the fact that social media is driven by the people of our society. So, it is very much likely that the posts can be simulated to get the desired results. Such kind of methods not only brute force their way through the algorithm, but also indirectly manipulate the human mind. Hence, we should be very careful when we are analyzing social media data for prediction and should take into consideration the above-mentioned issues while they are conducting the research.

With millions of users and thousands of tweets every second worldwide, enormous data is publicly available with Twitter that can be used for various research purposes. Twitter is one of the best places for breaking news because of itfis unique small-size story sharing and re-tweeting options. It is also well known that using certain words in speech can attract a lot of attention either positive or negative. Some amazing features of twitter are retweeting and favoriting that helps people support a personfis idea through showing their approval for it. We utilize some of these features from twitter to analyze what are the different words that have been used with respect to a hashtag that gained a lot of attention from public and led to a huge population participate in the trend. Our research in this project is mostly focused on understanding the different words around the tweets that have gained a lot of attention from public and also showing how the regular sentiment analysis might be very useful to understand peoplefis opinion on it. For example, lets consider that our favorite football team is playing a game, then using a relevant hashtag we would be able to visualize the emotions of the general public over the course of the game, and results would ideally be something like where if the team wins then the output would be a lot of positive tweets in contrast to negative and neutral tweets.

## 2 SENTIMENTAL ANALYSIS

Sentiment analysis is a contextual mining that extracts the sentiments hidden in the sentences. Sentiment analysis is extremely useful in social media research as it allows us to gain an overview of the wider public opinion behind certain topics. This technique is majorly used in the social media streams to understand how people are reacting to a situation or product or brand. This is very helpful in the marketing area where words that have more public approval can be identified and utilized in the marketing of a product or brand. Many sentiment analyses generally classify a sentence into one of different buckets like positive or negative of neutral. To make this classification, there needs to be a dictionary of words for each of these categories which are treated as positive or negative or neutral. To understand the sentiment in a tweet or a sentence, based on the number of positive words and number of words that belong to any particular category it contains, will be given a score. Based on the score, the sentence is then classified into positive or negative or neutral. There are some pre-defined libraries with words in these categories readily available to use for most of the situations. It is generally a good practice to manually tag some tweets as positive and negative and then calculate the frequency of different words that are occurring in these sentences can be identified as words belonging to that category. Once the word dictionaries are available, the sentiment of more tweets can be automatically identified using these word dictionaries. Some of the real world examples of sentimental analysis are,In order to understand consumer attitudes and restructure their operations, Expedia Canada took advantage of when they noticed that there was a steady increase in negative feedback to the music used in one of their television advertsiments and this helped them understand the negative impact the music was

causing to their advertsiments [11]. Some of the most growing and challenging directions of future sentiment analysis techniques on social media is argumentation. While sentiment analysis is about understanding users' opinions on some aspects, argumentation aims at identifying the the reasons of such opinions and the overall reasoning path in general. Also, problems like presence of sarcasm in tweets, fake user tweets, bot generated data still hurt the overall results of a sentimental analysis classifier.

## 3 LITERATURE SURVEY

We read some previous papers to understand how sentimental analysis is performed on social media data and some of the different techniques that can be applied to implement the sentimental analysis classifier.

The work from one of the paper utilizes the naive Bayes and fuzzy Classifier to classify Tweets into positive, negative or neural behavior of a person [9]. The combined proposed method which is a hybrid of naive Bayes and Fuzzy classifier, from this paper is more efficient in terms of Accuracy, Precision and Recall according to the evaluation in their dataset and classification results, this enables us to construct multiple classifiers to check the performance of the different methods considered in our project as well. They have also demonstrated about the common process in NLP that can help us derive the meaning or context of a given phrase, we have used some of these processes in our project to better understand the context of the tweets for the task at hand to correctly predict the sentiment behind each tweet [9].

The growing scale of social media data demands automatic data analysis techniques. A detailed survey on different techniques used in Sentiment Analysis is showcased and it was very useful for us to study across the different approaches as we can choose an efficient approach for our project. For the classification of sentiment applying a sentiment classifier trained on a tag specific data is not efficient because words that occur in the training data might not occur in test data, to overcome the feature mismatch a proposal to use a cross-domain sentiment classifier using an automatically extracted sentiment sensitive thesaurus was made. Also we read about some of the challenges faced in this paper like, detecting sarcasm from the expressions and finding out the correct context related sentiments, Grammatically Incorrect Words would be present and the results of sentiment analysis can be improved if these types of errors can be mapped to correct words and finally handling noisy data [10].

## 4 DATA DESCRIPTION

The tweets are extracted using the Twitter API through creating a developer account with Twitter. We would require the API secret key and consumer key for performing this task using the API. Since the amount of data that is available on twitter is really vast, we have used only a sample of tweets from the recent tweets in the April 2018 which has the user input hashtag in it. We utilized the tweepy cursor function in tweepy library to set the language of the tweets and also to search based on the hashtag provided by the user and also applying a limit on number of tweets being extracted to 10000. We were able to get fields like tweet text, date, favorites count, retweet count and username. But there is a lot of missing data in certain fields like demographics data or latitude and longitude. After the initial exploration into the data, we think that we can only use certain fields like date, text, username, favorites and retweets for our analysis. Below is the description of some of the fields from twitter. Some tweets would have URL data attached to them, emoticons and so on.

## 5 PRE-PROCESSING

Before the text can be used for sentiment analysis, it requires some preprocessing. Firstly, we removed all non-english tweets, tweets which have only images or URLs or videos because we are analyzing only text in the scope of this project. Nltk package in python provides many useful methods to perform text pre-processing like tokenizing, lower case conversion and stop words removal. The text in the tweets are tokenized used the fiword tokenizefi method in nltk library and then converted to lowercase to remove ambiguity in the same words with capitalized and non-capitalized words. Stop words could be misleading since their frequency in data is generally high, as the model might train on these words and could result in overfitting. Hence, they have been removed before fitting the model. Since, we are extracting the words that have gained a tweet more retweets or likes, URLs have been excluded from the tweets. We also performed lemmatization available in nltk Python which is very similar operation to stemming where a word is reduced to its most basic form but the major difference between these is, stemming can often create non-existent words, whereas lemmas are actual words which can be looked up in a dictionary.

### 5.1 NLTK

NLTK is a huge collection of quality libraries that handles most of the Natural Language Processing related tasks with respect to english in the python programming domain. "It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries, and an active discussion forum" [5]. NLTK data comprises of a huge number of corpora, grammars and models within it. NLTK also allows us to access these huge corpora in a memory efficient way. NLTK also comes bundled with many parser algorithms including the very modern and excellent Viterbi Parser, these can be used to parse the tweets data as well.

### 5.2 HTML characters

Data obtained from web usually contains a lot of html entities which gets embedded in the original data. It is thus necessary to get rid of these entities. We remove them directly by the use of specific regular expressions in our python code.

### 5.3 Stopwords

A scenario where the analysis needs to be data driven at the word level, the commonly occurring words which are known as the stop words should not be considered as a part of the analysis at hand and should be ignored from it. Stop words are natural language words which have very little meaning, such as the, a, an, and similar

words, they provide less value when we calculate the sentiment of a tweet [1]. There are two different approaches to it and they are, we can create a long list of stop words by hand or we can use predefined language specific libraries which consists of a huge dictionary of common english stop words.

### 5.4 Tokenization

"Tokenization is the act of breaking up a sequence of strings into pieces such as words, keywords, phrases, symbols and other elements called tokens" [13]. Tokens can be individual words, phrases or even whole sentences. In the process of tokenization, some of the special characters like punctuation marks are discarded [13]. The tokens become the input for another process like parsing and text mining usually for further analysis. Tokens themselves can also be separators. "For example, in most programming languages, identifiers can be placed together with arithmetic operators without white spaces " [13].

### 5.5 Null Values

We have found in our analysis, that after all of the pre-processing techniques have been applied, some of the tweets turn up to be empty. This is primarily because the tweets had just URL content or emoticons or just punctuations and so on. These values do not contribute that much towards sentimental analysis, hence they are removed and the null tweets are also removed from the dataset.

## 6 IMPLEMENTATION

We have built a RESTful webservice using flask that comprises of a sentimental analysis classifier built using Spark MLlib. We have used multinomial logistic regression model to classify the tweets data into positive, negatie and neutral tweets. Finally we create a visualization that displays the split of the tweets based on sentiments to portray the trend that exists currently over the given hashtag. Also, we have dockerized the entire project and hosted it on a cloud platform known as digital ocean.

### 6.1 Textblob

TextBlob is a Python library that handles processing of text data. Textblob consists of a simple API and it assists in handling some of the common natural language processing tasks like part-of-speech tagging, classification, noun phrase, extraction, sentiment analysis, and so on [7]. We used sentiment textblob in Python to classify the tweets into positive or negative or neutral. Below is a graph that shows these scores across these buckets.

The above plot shows 3 different bars, first bar is for frequency of positive tweets, second bar is for neutral tweets and third bar is for negative tweets. Neutral tweets are as frequent as positive tweets but negative tweets are very less compared to the other two categories. This is based on the sample of the data we collected from the MeToo movement which comprises of roughly around 10000 tweets. It has been in the news that as more and more victims came out sharing their experience through this movement, more guilty people were exposed and a positive sentiment is observed from majority of the people towards the movement lately.

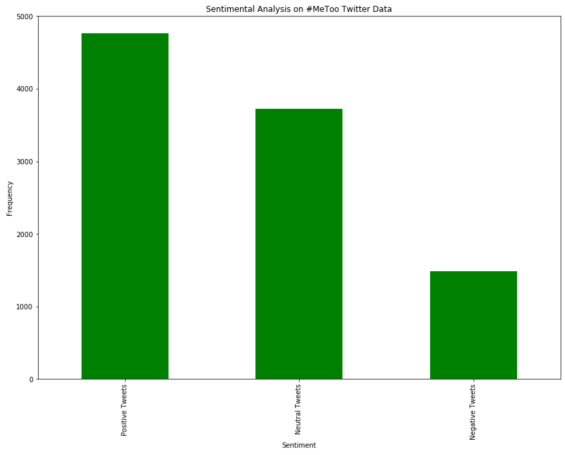


**Figure 1: Tweet Sentimental Classification**

### 6.2 Logistic Regression

Logistic regression falls under the category of supervised learning. It measures the relationship between the categorical dependent variable and one or more independent variables by estimating probabilities using a logistic or sigmoid function. In spite of its name this is not used for regression problem where the task is to predict the real-valued output. Logistic regression is a bit similar to the linear regression or we can see it as a generalized linear model. In linear regression, we predict a real-valued output y based on a weighted sum of input variables. Logistic regression makes predictions using probability. The name multinomial logistic regression refers to a case when the dependent variable has three or more unique values, such as Positive, Neutral, or Negative. Although the type of data used for the dependent variable is different from that of multiple regression, the practical use of the procedure is similar. Logistic regression is a powerful tool, allowing multiple explanatory variables being analyzed simultaneously, meanwhile reducing the effect of confounding factors. However, we must pay attention to model building, avoiding just feeding software with raw data and going forward to results [12]. Some difficult decisions on model building will depend entirely on the expertise in the field of sentimental analysis.

(1) **0 -** Indicates that the tweet is Neutral
(2) **1 -** Indicates that the tweet is Positive
(3) **2 -** Indicates that the tweet is Negative

### 6.3 Spark

"Apache Spark is a fast, in-memory data processing engine with elegant and expressive development APIs to allow data workers to efficiently execute streaming, machine learning or SQL workloads that require fast iterative access to datasets, with spark running on Apache Hadoop YARN, developers everywhere can now create applications to exploit Sparkfis power, derive insights, and enrich their data science workloads within a single, shared dataset in Hadoop" [6]. Apache Spark comprises of Spark Core and a repository of libraries. The core of spark is the distributed execution

engine and Python APIs offer a platform for distributed ETL application development. There are many libraries built on top of the core which support extensive machine learning operations.

### 6.4 Spark MLlib

MLlib is apache spark's inbuilt machine learning library, it is highly scalable. "MLlib fits into Spark's APIs and interoperates with NumPy in Python (as of Spark 0.9) and R libraries (as of Spark 1.5) " [4]. "We can use any Hadoop data source (HDFS, HBase, or local files), making it easy to plug into Hadoop workflows " [4]. MLlib has a huge repository of Machine Learning Algorithms including classification algorithms, Regression algorithms, Decision tree algorithms, Clustering algorithms, topic modeling algorithms and so on. In our project we have used the multinomial logistic regression classifier from the Spark MLlib. The MLlib is also capable of feature transformations, pipleline construction, parameter tuning, model evaluation and so on. "MLlib contains high-quality algorithms that leverage iteration, and can yield better results than the one-pass approximations sometimes used on MapReduce " [4]. Spark MLlib also has libraries that can do feature engineering related tasks like dimensionality reduction and so on, just in case the number of features that are being used to build the model turns out to be really high and we would need to scale it to construct an efficient model.

### 6.5 Spark ML Pipeline

Pipelining is a type of architectural technique used in data science related projects. It is the process of collecting the instructions from the processor through a well defined pipeline. It enables us to execute and store commands in an organized manner. This operation is also known as pipeline processing. Pipelining is a technique where multiple commands are stacked upon each other during execution. Pipeline is divided into numerous stages and all these stages are connected within themselves to form a pipe like structure. The commands enter from one end and exit from the other end. Pipelining the process increases the overall throughput of the task at hand.

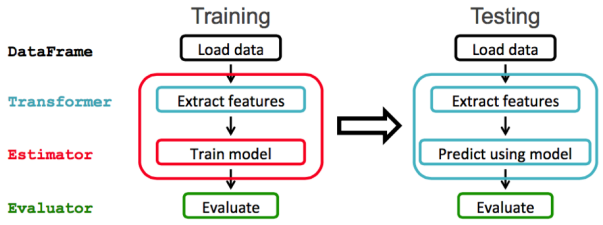


**Figure 2: SparkML Pipeline [8]**

The pipeline feature available at Spark ML Pipleline is a type of high level abstraction which helps in modeling of a entire data science project. The ML pipeline package available in the Spark ML models a typical data science oriented workflow and it provides overall abstractions like Transformer, Estimator, Pipeline and Parameters [6]. This is an overall general abstraction layer that makes the analysis part more productive.

### 6.6 Docker

Docker is the company driving the container movement. Docker provides a container platform to address every application across the hybrid cloud. "Todayfis businesses are under pressure to digitally transform but are constrained by existing applications and infrastructure while rationalizing an increasingly diverse portfolio of clouds, datacenters and application architectures, but Docker enables true independence between applications and infrastructure and developers and IT ops to unlock their potential and creates a model for better collaboration and innovation" [3]. A container platform allows solving multiple problems across a diverse range of applications. Container platforms deliver sustainable benefits throughout a community by providing enterprise application modules including security, governance, automation, support and certification over the entire application lifecycle [3].

### 6.7 Digital Ocean

Digital Ocean is a cloud infrastructure provider based out in the United States. The basic usage fee is 5 dollars which would enable us to use 3 CPU, 25 GB hard disk and finally a Bandwidth of 1TB provided with Docker and Ubuntu platform."DigitalOcean has optimized the configuration process to save time when running and scaling distributed applications, machine learning workloads, hosted services, client websites, or CD environments" [2]. Also, one of the biggest advantage of digital ocean is the community support forums that has well defined tutorials and developer forums which enables users to gain the knowledge required to work with the platform in a concise manner. We can also build and run applications with docker on the digital ocean platform. So we have dockerized our project and then pushed the image onto the dockerhub repository, from which we were able to pull the same onto our cloud environment and start a container instance to run the webservice. Our webservice is now live and accesible from anywhere.

## 7 RESULTS

We have successfully built a RESTful webservice with flask that comprises of a sentimental analysis classifier based on a multinomial logistic regression model which classifies the tweets data as positive or negatie or neutral tweets and also display the split of the tweets based on sentiments. we have dockerized the entire project and hosted it on a cloud platform known as digital ocean.

## 8 WORK BREAKDOWN

Surya Prakash Sekar has worked on the python code to pre-process and clean the entire dataset obtained, feature extraction, build the sentimental analysis classifier model and to visualize the results on graph using matplotlib. Venkatesh Aditya has worked on the feature engineering for the model, the spark code to build the multinomial logistic regression model that classifies the sentiment on the tweets. The RESTful webservice, Dockerization and hosting it on digital ocean was collectively done by both Venkatesh Aditya and Surya Prakash Sekar. The report work was collectively done by both Venkatesh Aditya and Surya Prakash Sekar as well.

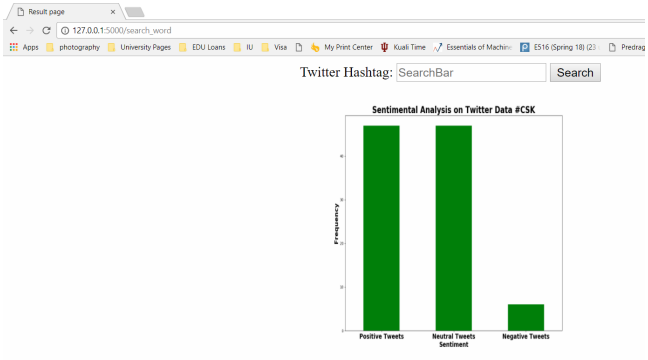


**Figure 3: Sentimental Analysis Webservice hosted on Digital Ocean**

## 9 CONCLUSION

We have successfully implemented our RESTful webservice and also succeeded in hosting the application on the cloud platform know as Digital Ocean. The application provides a visualization indicating the split of the latest 10000 twitter tweets with respect to a particular topic chosen by the user and we would be able to infer the trend of the tweet at that instant by inferring the sentiment scores from the latest tweets. The service would largely benefit people who are trying to figure out the latest scenario on twitter with respect to an issue, movement, campaign and so on. People can easily figure out the trend with respect to a particular topic this way and this application be applied across many domains like business and product analysis, marketing, movie performance, campaign performances, latest trends and so on, thus enabling people to take more data driven decisions in all aspects.

## ACKNOWLEDGMENTS

The authors would like to thank Dr. Gregor von Laszewski for his support throughout the project and suggestions to write this paper.

## REFERENCES

[1] Vangie Beal. [n. d.]. Stopwords. Webopedia website. ([n. d.]). https://www.webopedia.com/TERM/S/stop_words.html
[2] digital ocean. [n. d.]. deploying to scaling. digital ocean website. ([n. d.]). https://www.digitalocean.com/
[3] Docker. [n. d.]. what is docker. Docker website. ([n. d.]). https://www.docker.com/what-docker
[4] Apache Spark documentation. [n. d.]. MLlib. Apache Spark website. ([n. d.]). https://spark.apache.org/mllib/
[5] NLTK Documentation. [n. d.]. Natural Language Toolkit. NLTK website. ([n. d.]). https://www.nltk.org/
[6] Hortonworks. [n. d.]. Overview of Spark. Hortonworks website. ([n. d.]). https://hortonworks.com/apache/spark/
[7] Steven Loria. [n. d.]. Textblob. Textblob website. ([n. d.]). http://textblob.readthedocs.io/en/dev/
[8] Carol McDonald. [n. d.]. pipeline for predicting flight delays using Apache API. Dzone website. ([n. d.]). https://dzone.com/articles/fast-data-processing-pipeline-for-predicting-fligh
[9] Ruchi Mehra1, Mandeep Kaur Bedi, Gagandeep Singh, Raman Arora, Tannu Bala, and Sunny Saxena. [n. d.]. Sentimental Analysis Using Fuzzy and Naive Bayes. ieee explore website. ([n. d.]). https://ieeexplore.ieee.org/document/8282607/
[10] Harshali P. Patil and Dr. Mohammad Atique. [n. d.]. Sentiment Analysis for Social Media. ieee explore website. ([n. d.]). https://ieeexplore.ieee.org/document/7371033/
[11] Harshali P. Patil and Dr. Mohammad Atique. [n. d.]. understanding sentiment analysis. Brandwatch website. ([n. d.]). https://www.brandwatch.com/blog/understanding-sentiment-analysis/
[12] Sandro Sperandei. [n. d.]. Understanding Logistic Regression. NCBI website. ([n. d.]). https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3936971/
[13] Technopedia. [n. d.]. Tokenization. Technopedia website. ([n. d.]). https://www.techopedia.com/definition/13698/tokenization

# Benchmarking Hadoop and Spark on Mutiple Platforms

Gregor von Laszewski
Indiana University
Smith Research Center
Bloomington, IN 47408
laszewski@gmail.com

Karan Kamatgi
Indiana University
School of Informatics, Computing, and Engineering
Bloomington, IN 47404
krkamatg@iu.edu

Karan Kotabagi
Indiana University
School of Informatics, Computing, and Engineering
Bloomington, IN 47404
kkotabag@iu.edu

Manoj Joshi
Indiana University
School of Informatics, Computing, and Engineering
Bloomington, IN 47404
manjoshi@iu.edu

Ramyashree DG
Indiana University
School of Informatics, Computing, and Engineering
Bloomington, IN 47404
rdasego@iu.edu

## ABSTRACT

Big data and cloud computing embrace the potential benefits such as high volume, high velocity, and high variety. Whereas in distributed computing, a task is processed over two or more computers in the network within distributed system by breaking down the problem into many parallel smaller task. This method allows combining computational power of many computers to execute a program involving large data or multiple iterations in parallel. Apache Spark and Hadoop are two frameworks which are trending technologies in the field of distributed computing. In this project, we are comparing both frameworks by running a task implemented in different technology on a Raspberry pi distributed system setup with a single master and 4 workers. The final results are used to benchmark the performance in terms of efficiency and time elapsed. This analysis would help us realize scenarios and business use cases for each of these frameworks.

## KEYWORDS

hid-sp18-412, hid-sp18-406, hid-sp18-408, hid-sp18-410, Volume: 9, Chapter: PaaS, Status: 100.
Raspberry Pi, Hadoop, Spark Cluster

## 1 INTRODUCTION

Bigdata is ruling today's business and with technology boom and digitalization, large amount data As we all know big data has gained wide range of importance in various aspects. Big data, as a word refers to huge amount of data which are both organized and disorganized. However, the important thing here is not the amount of data, instead how this data is used by organizations. Big data, plays a vital role in having clear insights about the growth of business. Big data has determined its importance in various factors such as deducing the root cause of failed features, customizing coupon generation based on the customerfis buying habits, risk analysis of the business organization, determining irregular activities through data analysis which can affect the organizationfis growth. Big data has marked its importance in various organizations across practically every industry such as banking, education, government, health care, manufacturing and retail. To be specific Big data is used in various fields such as understanding and targeting customers, optimizing business process, personal qualification and performance optimization, improving health care and public health, Improvising business in sports, machine and device performance, improving security and law enforcement, financial trading and also in improving science and research. Thus, all these various applications of big data are not possible from the raw form of the data, it is required to process and analyze this data
[37].

## 2 INTRODUCTION TO HADOOP

Data is generated continuously and rigorously from each and every application in almost all the platforms such as social media, mobile platform and many more. The solutions to handle this data should be very quick and also should consider business cost required for this analysis. The problem with early storage tools such as RDBMS is that it is unable to process the semi-structured and unstructured data such as text files, videos, audios, clickstream data and it is suitable only for structured data such as banking transaction, location and information. These both forms of data are completely different in ways required for processing the data. The main flaw in RDBMS is that it is unable to scale vertically with a large number of CPU and other storages. It is the main problem if the main server is down, this created the need for a distributed system which can be robust and handle scalability. Hadoop is trusted for its effective management of large sized data which is both structured and unstructured in different forms such as XML, JSON, and also text at high fault-error tolerance. It is noticed that with clusters of many servers in the field of horizontal scalability, Hadoop has marked its significance by providing faster result rates from Big Data and also the unstructured data as Hadoop's architecture is based on the open source foundation. Other challenges faced with the big data where keeping the huge amount of data secure, analyzing the data without knowing the type of the data, dealing with data which has poor quality and is inconsistent and incomplete. Also, finding powerful algorithms to find patterns and insights was not an easy task. On the other hand, the organizations and enterprises realized that the amount of information which is used for analyzing is less and majority of the data is getting wasted. The main reason behind

this is that the organizations lack power and strong tools to analyze such huge amount of data. Due to these limitations of processing data, a huge amount of valuable data was termed as unwanted and discarded. It is useful and important to collect and keep all the data in a safe storage, which the business organizations realized [27].

Hadoop emerged as the strongest tool to deal with the above challenges of the big data. Now, almost all organizations use big data analysis to improve the functionalities in each and every business unit. The various kinds of business units include research, design, development, marketing, advertising, sales and customer handling. Sharing such huge amount of data across different platforms is also challenging, in such scenarios Hadoop is used to create a pond. Hadoop in this way represents a repository of various sources of data which may be from intrinsic or extrinsic sources. Hadoop is a set of open source programs and procedures which can be used for big data operations. It is an open source framework from Apache which is comprised of Java-based programming framework that can be used for data storage and processing of the data-sets clusters on commodity hardware. The basic idea behind Hadoop is that to make the computation of the data faster. Hadoop MapReduce is composed of shared and integrated foundation where developers can include additional tools and enhance the framework. Additionally, scalability is the core of the Hadoop system. The novel approach which emerged Hadoop is that by storing the all types of data available, focus on organizing and analyzing the data in new interesting ways [30].

The growth of Hadoop has marked its significance in changing the perception of handling Big Data, specifically for the unstructured data. Using Hadoop we can enable excess data streamlining for any distributed processing system across clusters. Hadoop system helps to scale up from single server to a large number of servers, and also assuring local computation and storage space on every distributed server. Hadoop does not depend on hardware to provide high-availability; instead, it makes its software efficient by building robust libraries. These libraries will handle the breakdown at the application layer and thus help the service to be efficient along with the cluster of computers [40].

Hadoop as a framework has established credibility for various factors. It allows the users to easily frame test cases on distributed systems. These test cases are efficiently deployed across various machines in turn and it also utilizes the parallelism of the CPU cores. Another interesting feature of Hadoop is that it does not rely on hardware to take care of the fault tolerance, instead, Hadoop is designed to detect the failures and handle them at the application layer itself. Additionally, servers configuration, addition, and removal are independent of the Hadoop operation and these servers can be added and detached to the clusters dynamically during the operations. One of the major advantage and main reason for Hadoop to be successful is that being an open source framework it is suitable for all the frameworks as it is Java based [35].

In simple words to state, Hadoop provides efficient ways of storing enormous data sets by distributing it to various clusters and then execute distributed analysis application in each cluster. The other main important feature about hadoop is that, it is a framework which is modular and is compatible to integrate with any of the required modules. Thus, it allows swapping of the components according to the need of software tool which results in flexible architecture, which makes Hadoop robust and efficient. To the business organizations, the flexibility feature of Hadoop is most advantageous, as it allows the business users to add or modify their data storage and analysis according to the business needs and suitable software.

## 2.1 Overview of Hadoop Modules

Apache's Hadoop framework composed of the four main modules:

- Hadoop Common: This holds the libraries and utilities required by the Hadoop modules.
- Hadoop Distributed File System: which manages to store data on commodity machines, which is helpful in providing efficient bandwidth across the cluster.
- Hadoop Yarn: this is basically a resource management platform which will be responsible managing compute resources in clusters and using them for scheduling of user's applications.
- Hadoop MapReduce: This is the programming module required for large-scale data processing. The main principle behind the above-mentioned modules of Hadoop is that the common scenarios of hardware failures are handled by software in the framework [28].

## 2.2 Importance of Yarn

Yet Another Resource Negotiator (YARN) is one of the most important components of Apache Hadoop as it takes care of managing resources and scheduling tasks and thus forming the clustering platform. YARN is responsible for setting up the global and application-specific components which are required for resource management. For any particular application, the required and suitable resources are allocated by the YARN. Initially, the application submission client will submit an application to the resource manager of YARN. Further, YARN takes the responsibility of scheduling application so that tasks are prioritized and big data analytics of the system can be managed. This results in the greater step of the system architecture for collecting the data and sorting them and further conducting requirement specific queries such as data retrieval. Such type of information retrieval has found very great business values because the organizations can use data analyzing platforms for supply chain maintenance, product documentation, and various service operations such as maintaining customer information and also for maintaining automated processes of business [39].

Being an essential part of core Hadoop projects, YARN is capable of allowing multiple data processing engines like interactive SQL, real-time streaming, data science and batch processing. Likewise, YARN is growing to be a mandatory foundation for new generation Hadoop tools and has become an important component which is required for realizing modern data architecture. Additionally, YARN is extending the capabilities of Hadoop by providing support for the new technologies which will be found in the data center. This helps in various factors such as cost-effective solutions, linear scalability for storing and processing. Further, YARN is still in progress for improvising factors specifically for the new engines which are emerging to interact with data storage and analysis. Thus, YARN is one amongst the reliable architectural center of Hadoop and definitely the most important foundational component [43].

### 2.3 Advantages of Hadoop

- Scalable : Hadoop is specifically designed to have a very flat scalability structure. Once After a Hadoop program is written and is functioning on ten nodes, very little work is required for that same program to run on a much larger setup. The underlying Hadoop platform will manage the data and hardware resources and provide dependable performance growth (but proportionate to the number of machines available). Also, it is a highly scalable storage platform, because it can store and distribute large data sets across hundreds of inexpensive servers that operate in parallel. Hadoop enables businesses to run applications on thousands of nodes involving thousands of terabytes of data [32].
- Advanced data analysis can be done in-house: With Hadoop environment its capable to work with large data sets and customize the outcome without having to outsource the task. Keeping operations in-house helps organizations be more agile, while also avoiding the ongoing operational expense of outsourcing.
- Organizations can fully leverage their data: With Hadoop systems, organizations can take full advantage of all their data fi?! structured and unstructured, real-time and historical.Earlier with traditional legacy systems, all the available data and inputs were not used to do the analysis to support business activity. Leveraging adds more value to the data itself and improves the return on investment (ROI) for the legacy systems.
- Flexible architecture: Some of the tasks that Hadoop is being used for today were formerly run by expensive computer systems. Hadoop commonly runs on commodity hardware. Because of big data standard, Hadoop is supported by a large and competitive solution provider community, which protects customers from vendor lock-in [26].
- Cost Effective: Hadoop systems provide a cost-effective storage solution for businesses. The problem with traditional systems is that it is extremely cost prohibitive to process such massive volumes of data. To reduce costs, many companies in the past would have had to downsample data and classify it based on certain assumptions as to which data was the most valuable. The raw data would be deleted, as it would be too cost-prohibitive to keep. While this approach may have worked in the short term providing a very cost-effective solution for expanding datasets. It is designed to scale-out architecture that can affordably store all company's data for use sometime later. This saves a lot of costs and improves the storage capability tremendously.
- Enhances Speed: Hadoop's unique storage method is based on a distributed file system that basically maps data wherever it is located on a cluster. The tools for data processing are often on the same servers where the data is located, resulting in much faster data processing. If youfire dealing with large volumes of unstructured data, it can efficiently process terabytes of data in just minutes [33].
- Great Data Reliability: Data reliability is one aspect that no organization wants to compromise on. Hadoop provides complete confidence and reliability; in a scenario where data loss happens on a regular basis, HDFS helps you solve the issue. It stores and delivers all data without compromising on any aspect, at the same time keeping costs down. Whether you are a start-up, a government organization, or an internet giant, Hadoop has proved its mettle when it comes to strong data reliability in a variety of production applications at full scale.
- Comprehensive Authentication and Security: All businesses are looking for software that makes their work safe, secure and authenticated. When it comes to authentication and security, Hadoop provides an advantage over other software. Its HBase [6] security, along with HDFS and MapReduce, allows only approved users to operate on secured data, thereby securing an entire system from unwanted or illegal access. Security is the top priority of every organization. Any unlawful access to data is sure to harm business dealings and operations [38].
- Flexible with Range of data sources: Hadoop enables businesses to easily access new data sources and tap into different types of data. The data collected from various sources will be of structured or unstructured form. This means organizations can use Hadoop to derive valuable business insights from data sources such as social media, clickstream data, or email conversations.A lot of time would need to be allotted to convert all the collected data into a single format. Hadoop saves this time as it can derive valuable data from any form of data. It also has a variety of functions such as data warehousing, fraud detection, and market campaign analysis. [29].
- Resilient to failure: A key advantage of using Hadoop is its fault tolerance. When data is sent to an individual node, that data is also replicated to other nodes in the cluster, which means that in the event of failure, there is another copy available for use [31].

## 3 INTRODUCTION TO APACHE SPARK

Apache Spark [11] is a cluster computing framework developed at the University of California, Berkeley. It is maintained as an open software by the Apache Software Foundation. It provides an interface for programming with data parallelization and also fault tolerance. Spark's architecture is based on Resilient Distributed Dataset (RDD) which is a read-only set of data items split over a cluster of machines. In short, instead of the code getting the data for computation, it goes to the location where the data is present [41].

### 3.1 RDD

RDD is a fundamental data structure of Spark. It is an immutable distributed collection of objects wherein each dataset is divided into logical partitions. Computation is performed on different nodes of the cluster. RDD's can have any type of objects like Java, Scala, Python objects or even user-defined class objects. An RDD is a read-only, collection of records that are partitioned. RDD's can be created through operations on data on storage or using other RDD's.

RDD's are a collection of elements which are fault-tolerant and operate parallely. RDD's can be created in two ways - parallelizing an existing collection in the driver program, or referencing a dataset in an external storage system [34].

Spark and its RDD's were developed to overcome the limitations of Map-Reduce in Hadoop which forces a linear dataflow on distributed programs. Map-Reduce programs read the data from the disk, map some function across the data, do the reduce operation on the results of the map and finally store the results from reduce onto a disk. On the other hand, Spark's RDD function by giving a working set for distributed programs and offer distributed shared memory in a restricted way.

Spark provides a platform for implementation of iterative algorithms and interactive data analysis. Iterative algorithms visit the data set multiple times and interactive data analysis involves repeated querying of the database. The latency of such applications can be reduced greatly when compared to a Map-Reduce implementation of the task.

Apache Spark requires a manager to handle the cluster load and also needs a distributed storage system. For cluster management, Spark supports standalone cluster, Hadoop Yarn and Apache Mesos. To provide distributed storage, Spark can be integrated with a variety of frameworks like Hadoop Distributed File System (HDFS), MapR File System (MapR-FS) [18], Cassandra [3], Kudu [9]. Also, a custom solution can be implemented and integrated into Spark [41].

### 3.2 Spark Core

Spark has a core component called the Spark Core that provides distributed task dispatching, Input-Output functionalities and scheduling exposed through an API centered on RDD abstraction. A driver program invokes parallel operations on an RDD by passing a function to Apache Spark which then schedules the function's execution in parallel on the whole cluster. Operations such as joins, take RDD's as an input and produce new RDD's. RDD's are immutable and fault-tolerance is achieved by keeping a track of the lineage of each RDD so that if there is a data loss, it can be reconstructed. RDD's support any type of Python, Scala or Java objects. Along with the RDD style of programming, Spark provides two restricted forms of shared variables - broadcast variables and accumulators. Broadcast variables reference read-only data and makes it available on all nodes. Accumulators can be used to program reduce part in an imperative style [41].

### 3.3 Spark SQL

Spark SQL is a component that is built on top of Spark Core that provides a data abstraction called DataFrame. DataFrame provides support for structured and semi-structured data. To support various languages like Scala, Java or Python, Spark SQL provides domain-specific language (DSL). Spark streaming makes use of Spark Core's fast scheduling capacity to perform streaming analytics. It takes in data in mini-batches and performs RDD transformations on those mini-batches of data. This enables the same set of application code written for batch analytics to be used for streaming analytics as well enabling easy implementation of lambda architecture. Spark Streaming has support to consume from Kafka [8], Twitter [22], Flume [4], Kinesis [1] and TCP/IP sockets [41].

### 3.4 Spark MLlib

Spark MLlib [21] (Machine Learning Library) is a distributed machine learning framework that is built on top of Spark Core. Many common machine learning and statistical algorithms have been implemented in MLlib which facilitates large scale machine learning pipelines. Some of the supervised and unsupervised Machine Learning algorithms implemented are Support Vector Machine, Logistic Regression, Linear Regression, Decision Trees, Naive Bayes, Latent Dirichlect Allocation [41].

### 3.5 Spark GraphX

GraphX is a distributed graph processing framework built on top of Apache Spark. Since RDD's are immutable, graphs are also immutable and hence GraphX is unsuitable for graphs that need to be updated. GraphX provides two seperate application programming interface for implementing parallel algorithms - a Pregel abstraction and a MapReduce style API [41].

## 4 APPLICATIONS OF SPARK

The applications of Apache Spark range from Streaming Data, Machine Learning, Interactive Analysis and Fog Computing [15].

### 4.1 Spark Streaming Data

Spark's key use case is its ability to process continuously streaming data. Spark Streaming unifies different types of data processing capabilities allowing developers to use just a single framework to handle the processing needs. The most common ways that Spark Streaming is used today are Streaming ETL, Data Enrichment, Trigger Event Detection and Complex Session Analysis. Streaming ETL - Traditional ETL tools used for batch processing, read the data, convert it to a database compatible format and finally write it to the database. In Streaming ETL, data is cleaned and aggregated continually before it is pushed into data stores. Data Enrichment - This streaming capability enriches live data by combining it with static data. This allows user to conduct more real-time data analysis. Trigger event detection - Spark Streaming allows the user to detect and respond quickly to unusual behaviors in real-time. Complex Session Analysis - Using the Spark Streaming feature, events with respect to live sessions such as user activity in a website or an application. The session information can be used continuously to update the machine learning models. Machine Learning: Apache Spark comes integrated with a framework for performing advanced analytics. It helps users run repeated queries on data sets which amounts to processing machine learning algorithms. Spark supports machine learning through its framework called MLlib which can perform clustering, classification among many other algorithms [36].

### 4.2 Spark Interactive Analysis

One of the most notable features of Spark is its ability to provide interactive analytics. Unlike Hadoop, Spark performs exploratory queries without sampling. By integrating Spark with visualization tools, complex data sets can be processed visualized in an interactive way. Fog Computing: Fog computing decentralizes data processing and storage instead of performing computation on the edge of the network . However, this creates a lot of complexities for processing decentralized data since it requires low latency along with massive

parallel processing for running machine learning algorithms. But with Spark Streaming, real-time querying tool (Shark), MLlib and GraphX, Spark qualifies as a fog computing solution [36].

## 5 RASPBERRY PI

Raspberry Pi is a series of mini single-board computers. It was developed by Raspberry Pi Foundation to promote the teaching of basic computer science. Till date, various Raspberry Pi's have been released with a system on chip and ARM compatible CPU along with on-chip GPU. The processor speed ranges from 700 MHz to 1.4 GHz and on-board memory ranges from 256 MB to 1 GB RAM. SD cards can be used to store the operating system and program memory. The boards have USB ports ranging from one to four ports. For video output, HDMI is supported and 3.5 mm phono jack supports audio output. The Raspberry Pi Foundation provides Raspbian, a Debian-based Linux distribution along with Ubuntu, Windows 10 IoT Core and specialized media center distributions. It supports Python and Scratch as the main programming language and also supports many other languages [42].

## 6 DIFFERENCE BETWEEN SPARK AND MAPREDUCE

Spark keeps data in memory whereas MapReduce keeps shuffling things. MapReduce takes a long time to write things to disk and also to read them back. This makes MapReduce slow. For SQL queries, a chain of MapReduce operations are usually required and since MapReduce keeps shuffling things, it required a lot of Input-Output activity. On the other hand, when SQL queries are run on Spark, it executes faster since it has data in memory and requires less Input-Output operations. Spark has a Map and a Reduce function like MapReduce, but it also has richer features such as Filter, Joins and Group-by. Hence development in Spark is easier and is flexible. Spark provides a lot of instructions at a higher level of abstraction than what MapReduce provides [25].

## 7 APACHE HADOOP ARCHITECTURE

Apache Hadoop [5] is distributed computation framework that provides the storage and capability of running the programs on multiple master slave architecture in order to enable the distributed processing. The very basic implementation of the Apache Hadoop started with the Googlefis MapReduce programming model, which now has grown to an enormous ecosystem with related technologies such as Apache Hive [7], Apache Spark and Apache HBase [6]. The architectural pattern of the Hadoop has made it so famous that it has been adopted by many of the companies such as Facebook, Yahoo, Adobe, Cisco and eBay [16].

### 7.1 HDFS Architecture

HDFS architecture was designed with the main goals of avoiding hardware failures, large datasets corresponding to around gigabytes and terabytes, portability and a simple coherent model [17].

The data is divided into the small units called the blocks in Hadoop cluster and subsequently, data is distributed across the cluster. The duplication of data in blocks is done twice, and totally with the three copies. The two of copies are called as replicas and are stored at a different place in a cluster. As a result of three copies, the data replication factor increases to three leading to more availability of data. When a certain copy is lost then the HDFS will again re-replicate the data in order to maintain the value of the replication factor to three [16].

There are mainly two models of the HDFS architecture which varies based on the characteristics and version of the Hadoop, they are as follows [16]:

- Vanilla HDFS
- High Availability HDFS

The main basic architecture of the HDFS follows a leader/follower pattern. A cluster would be specifically composed of a single Namenode, an optional secondary node and a random number of the data nodes [17].

The structuring of Vanilla Hadoop Deployment is shown in Figure 1.

*7.1.1 Namenode and DataNodes.* The HDFS cluster composes of a single Namenode, this node acts as the master server which handles filesystem namespace in order to control relevant access to files by the clients. There are a random number of datanodes that enable storage of the data in the nodes they are attached to. In order to effectively allow the user data to be stored in files the HDFS exposes a file system namespace which enables the storage of user data in the files. Subsequently, the file is further divided into one or more blocks which will be stored in the set of the datanodes. The execution of tasks with respect to the opening, closing, renaming files and directories and the other namespace functionalities are carried out by namenode. The name node also serves in order to decide the mapping of blocks to the data nodes. The read and write requests from the clients of the file system are fulfilled by datanodes [17].

The HDFS architecture is shown in Figure 2.

The general platform that the namenodes and the datanodes run are on GNU/Linux operating system. The main primary language used to build the HDFS is java. A typical machine that can support Java will be able to run namenode and the datanode. As system uses high portable java language HDFS can be easily deployed on a wide range of the machines. With the presence of the single namenode in the cluster there is simplification of HDFS architecture. The Namenode acts as an arbitrator and a repository for whole of the HDFS metadata. The system is designed by treating NameNode as black box and it never allows the user data to flow out of the Namenode [17].

*7.1.2 File System NameSpace.* The file organization in HDFS is similar to the traditional hierarchical file organization. The HDFS filesystem provides functionalities to the users to create, add, rename, remove and move the files to respective directories. The HDFS file system provides effective support that enables user quotas and grants permission to access the file system [17].

The namespace of the filesystem is managed by the Namenode. Any of the transactions with respect to the file system namespace or properties will be recorded by the NameNode. The replication factor can be controlled and changed by any of the respective client application. The data with respect to replication factor is also stored by NameNode [17].

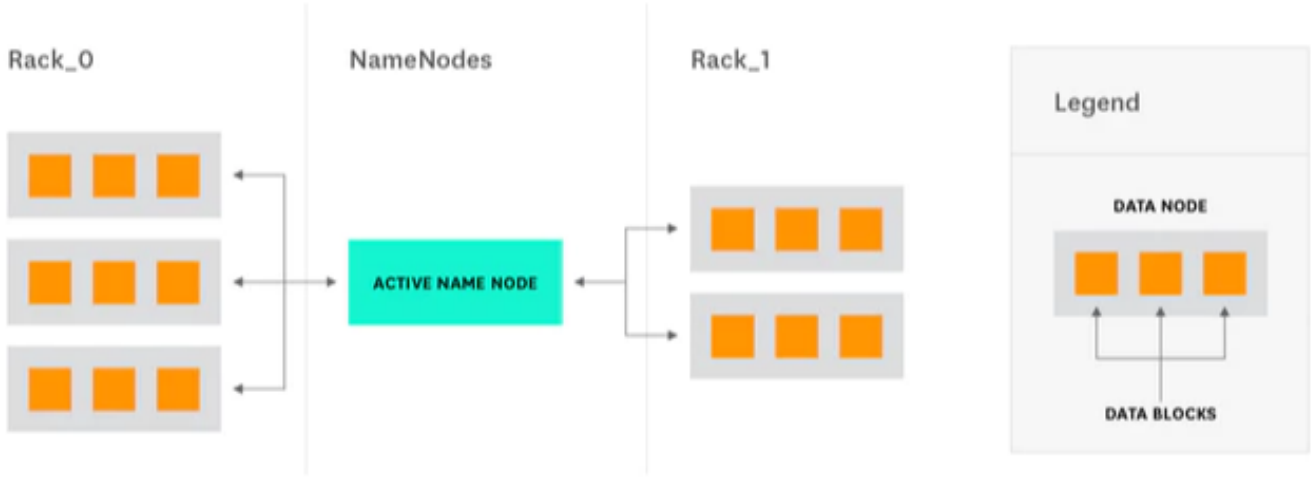


**Figure 1: Vanilla Hadoop Architecture [16]**

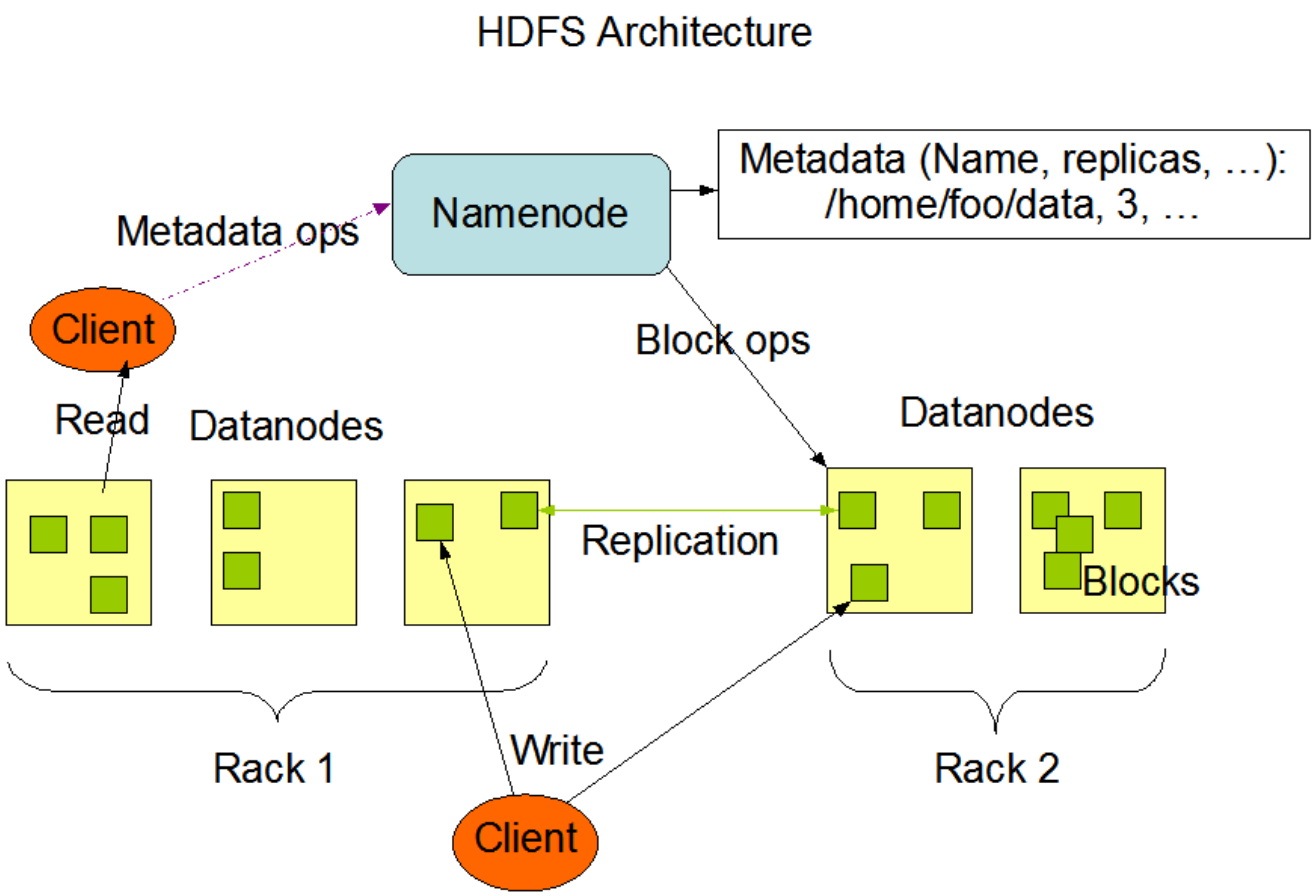


**Figure 2: HDFS Architecture [17]**

*7.1.3 fsimage and edit log.* File system metadata is stored in two of the different files by NameNode, those are fsimage and editlog. At the specific event of time the fsimage facilitates storage of file systems metadata. The few of changes that are going to be updated

6

such as renaming, appending to a file are maintained in an edit log to ensure durability. This avoids redundant creation of the new fsimage every time when a namespace is modified [16].

Structuring of the secondary NameNode is shown in Figure 3.

The Figure 3 shows the structuring of the secondary NameNode. Secondary NameNode independently validates and updates replicas of fsimage subsequently to the changes that are made in edit log. If active node fails in the system then NameNode needs to rebuild edit log on top of fsimage. Although, administrators of a cluster can easily retrieve an updated copy of fsImage from Secondary Namenode [16].

The `secondaryNameNode` also provides the advantage and benefit of the faster recovery in case of NameNode failure. Although, `SecondaryNameNode` will not provide automatic failover load balancing [16].

## 7.2 MapReduce

The Mapreduce is an effective framework that can be utilized to process large datasets in the distributed way. Basically, there are three operations in the MapReduce model, Map input dataset into the collective pairs, shuffle mapped data and handover the data to the reducers then subsequently reduce overall pairs with a same number of keys. The example of map-reduce job corresponds to WordCount example which is again illustrated in this paper that has been followed to analyze the performance of hadoop and spark on the different platforms. The high-level specification or unit in MapReduce framework is Job. The each of job composes of one or more map or the reduce tasks [16].

The Figure 4 shows the perfect example of the MapReduce job with respect to WordCount application.

*7.2.1 Related Frameworks.* There are other frameworks that perform and accomplish the MapReduce jobs in a similar way as hadoop. The most used frameworks other than Hadoop are Apache Spark and Apache Tez [13]. This paper also describes Apache spark and other relevant analysis that is performed with respect to Apache Spark with different computing platforms and different computing resources [16].

## 7.3 YARN

The major idea of the YARN is to divide the functionalities of resource management and scheduling/management into separate daemons. The very basic and the primary idea is to have the global resource manager (RM) and the per-application master. The application would be treated as a single job or Directed Acyclic graph of Jobs [23].

The Figure 5 describes the architecture of the YARN.

*7.3.1 Data Computation Framework.* The data computation framework is formed by the resource manager and node manager. The major authority is possessed by the resource manager to assign resources to all applications in a system. In contrast, Node manager is a framework that gets assigned to each machine and is responsible for monitoring of containers and resource allocation with respect to CPU, memory, disk and network [23].

*7.3.2 Resource Manager.* The Figure 6 describes architecture of the resource manager and displays high-level view of resource manager. The resource manager (RM) is assigned the responsibility of tracking the resources in a cluster and scheduling of jobs. The standby resource manager provides failover load balancing for active resource manager. The standby resource manager will be waiting to take over in case of any of disaster with respect to the active resource manager [23].

Figure 6 describes architecture of resource manager.

*7.3.3 Node Manager.* Node manager acts like a slave in a system infrastructure. When the node manager starts it indicates resource manager that it has gone live. At continuous periodic times, node manager sends respective signals to Resource Manager indicating that it is live. At next level of depth, a container is a fraction of node manager and it is in turn used by client to run an application [24].

Figure 7 displays the division of the Node Manager into containers

# 8 APACHE HADOOP ARCHITECTURE DEPLOYED ON THE RASPBERRY PI 3

The Figure 8 depicts architecture of Apache Hadoop on Raspberry Pi nodes. There are basically 5 nodes which have been divided into the master and workers. YARN is deployed on top of Apache Hadoop. There are five nodes and 4 of them are treated and configured as workers and one node is configured as master. Further, Java and Python WordCount applications are run in order to benchmark, evaluate and compare the performance of Hadoop and spark applications.

The Figure 8 describes the architecture of Apache Hadoop Deployed on Raspberry Pi 3.

# 9 DOCKER

The docker provides the containerization at operating systems level. The Docker virtualization system is developed by Docker. Inc. To effectively develop and maintain the containers easily on the single Linux Instance docker provides an effective framework to run different capability containers in a single machine [14]. The docker consists of three following components:

## 9.1 Software

The docker has the major Daemon process called as `dockerd` which will effectively manage the docker containers and in turn handle container objects. Docker client called as docker helps users to manage and provides a diverse range of options to interact with containers [14].

## 9.2 Objects

The various objects in docker are mainly images, containers, and services. Docker container acts as a black-box that runs applications virtually. These containers can be managed by the docker API service. The docker images are templates that can be used to build containers and acts as a base framework. The docker service is a process that will effectively allow numerous containers to be extended across different docker Daemons [14].

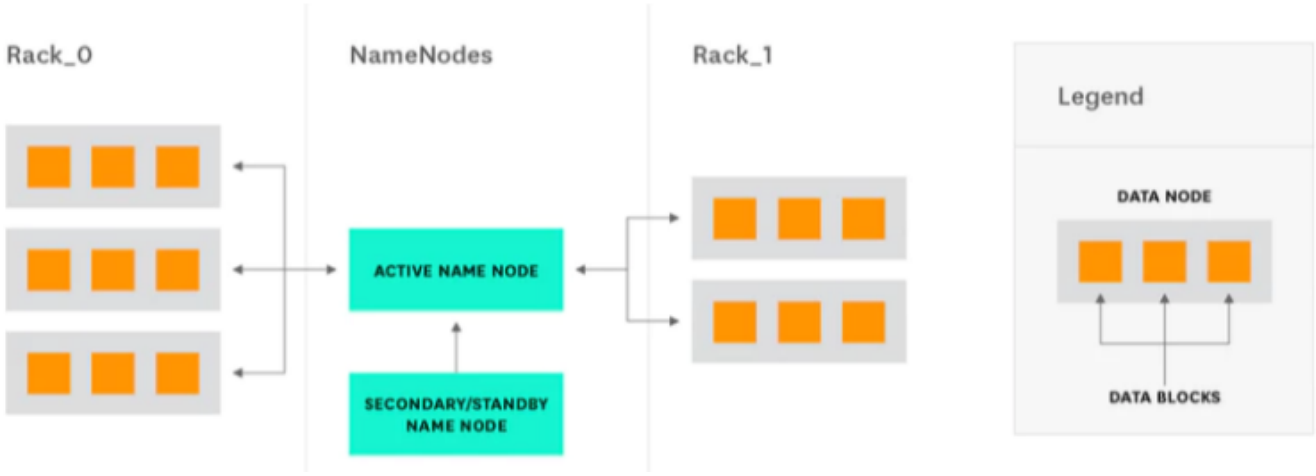


**Figure 3: Structuring of Secondary NameNode [16]**

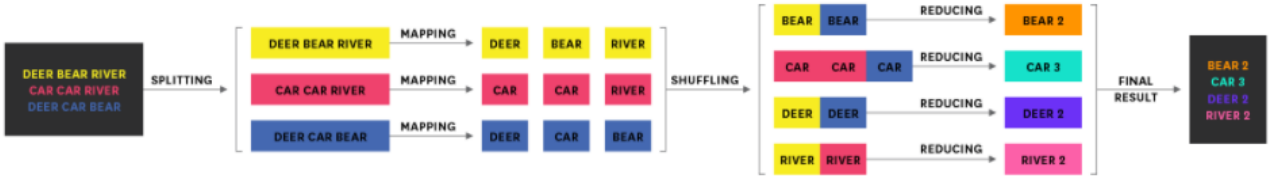


**Figure 4: Word Count MapReduce Job [16]**

### 9.3 Registries

Registries are of two types, public and private. At present, the Docker hub and the Docker Cloud are the two main registries. Docker Hub provides different images to be tagged and pulled [14].

### 9.4 Docker-Spark

This project is mainly targeted to benchmark and evaluate the performance of Apache Hadoop and standalone spark on different platforms. Docker was one of the platform that was selected in this project to evaluate the performance of an application with these frameworks. The dockerized WordCount program is run on the Ubuntu 17.10 platform.

### 9.5 Docker-Spark Architecture

The Figure 9 describes the implementation of the docker spark architecture that is deployed on Ubuntu 17.10.

## 10 SPARK ARCHITECTURE

Spark offers great features in terms of speed, simplicity and high level support for development of current environments and storage systems. They have wide range of developers that embrace the ongoing development for one of the Apache's largest and most active platform with around 500 contributors who are responsible for their code in the software code release [20].

### 10.1 Support for programming languages

Spark provides very easy interface and comprehensive support for the enhancement and development languages so that we could easily learn and apply accordingly into our existing applications with great flexibility and ease. Spark provides support for modern languages such as Java, Python, Scala, SQl and R. It has been a great concern for most people that Python performs sub-optimally when compared to its alternate language Java. This concern is less significant in a distributed environment mostly where Spark would be deployed. The slight loss in performance if introduced by the usage of Python language could be compensated elsewhere in the design and distributed operations of the cluster. One of the analysis carried out, informs us that familiarity of the user's chosen language is often times more important than the raw speed of code in that particular language.

Relational databases (RDBMS) which usually comprises of the structured query language (SQL) and the SQL queries are normally well understood by developers, Engineers, Data Scientists and Academia. The Apache Spark module Spark SQ offers native support for SQL. Spark provides excellent support and simplifies whole process of making query request to the data that is stored in the Spark's native RDD (Resilient Distributed Dataset) model along with data from other external sources like relational databases and data warehouses.

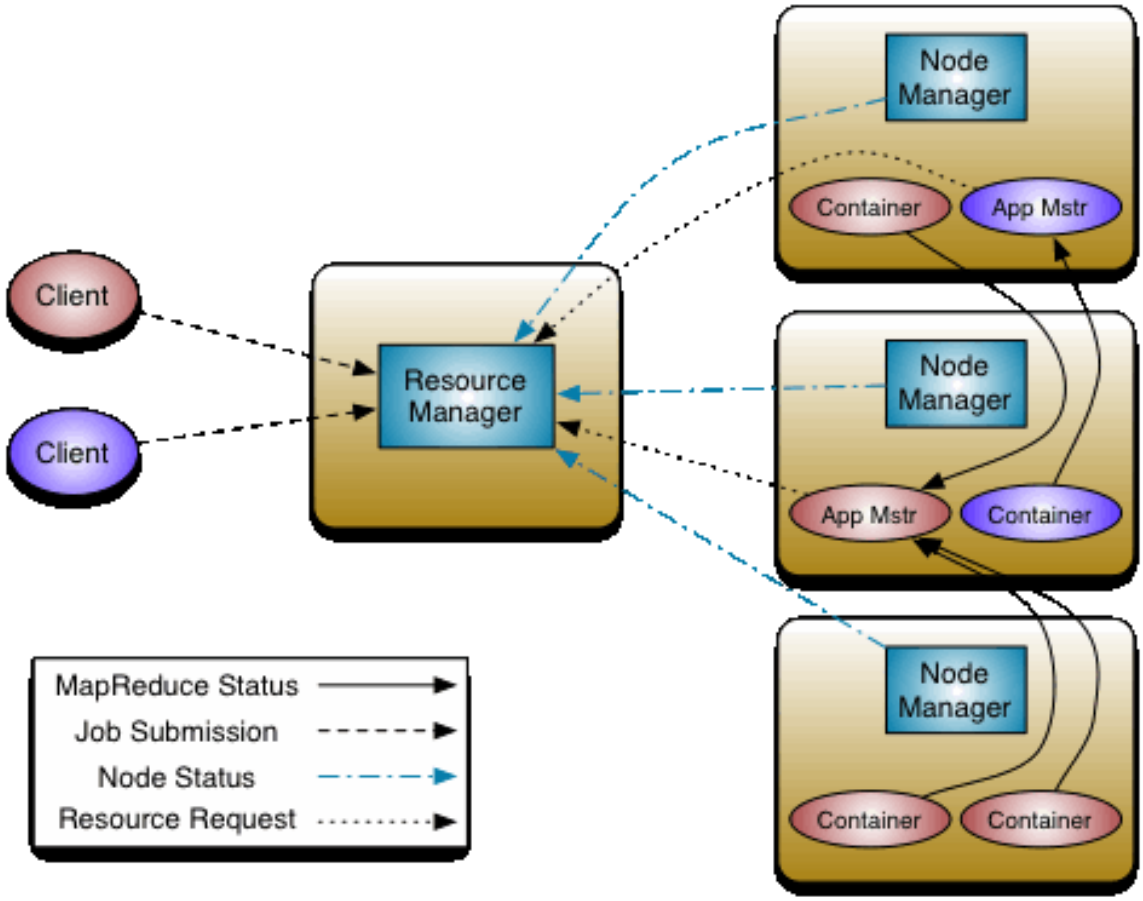


**Figure 5: YARN Architecture [23]**

Spark also provides extensive support for data science enriched language called R. It has a package named SparkR which first appeared in the release of 1.4 apache spark. Due to enormous popularity in the data science, this package proves to be one of the important features of Apache Spark [20].

## 10.2 Spark deployment options

Spark is very easy to download and install on a laptop or virtual machine. It is designed to be able to deploy on both standalone as well as part of a cluster.Most of the production environment workloads require computations to be carried at large scale and Spark would then be deployed on existing big data cluster. These clusters are used for running Hadoop jobs and Hadoop's YARN resource manager will manage the hadoop cluster. There are several options for running spark such as running spark on YARN (Yet another Resource Negotiator) which is a cluster management system. Yarn could be used to execute arbitrary applications on a hadoop cluster. There needs to be one application master node and several arbitrary number of containers acting as workers. Workers are responsible for the execution of application where as the master makes requests to the workers and monitors the progress and status of the work carried out by the respective workers [20].

YARN consists of two component types

- Resource Manager is a unique component for the whole of big data cluster. The main job of Resource Manager is to grant the requested resources and balancing the load of the entire cluster.It also starts the application master only in the initial phase and restarts the master in case of any failure.
- Each computing node (Worker) has one node manager which is executed when the master initiates the request. The node manager starts and monitors the containers assigned to the current node in the cluster as well monitors the usage of its resources such as CPU and memory consumption.

The other option for running spark where people prefer alternative resource managers, Spark could be run on other clusters controlled by Apache Mesos [10]. Spark also provides series of

[9]

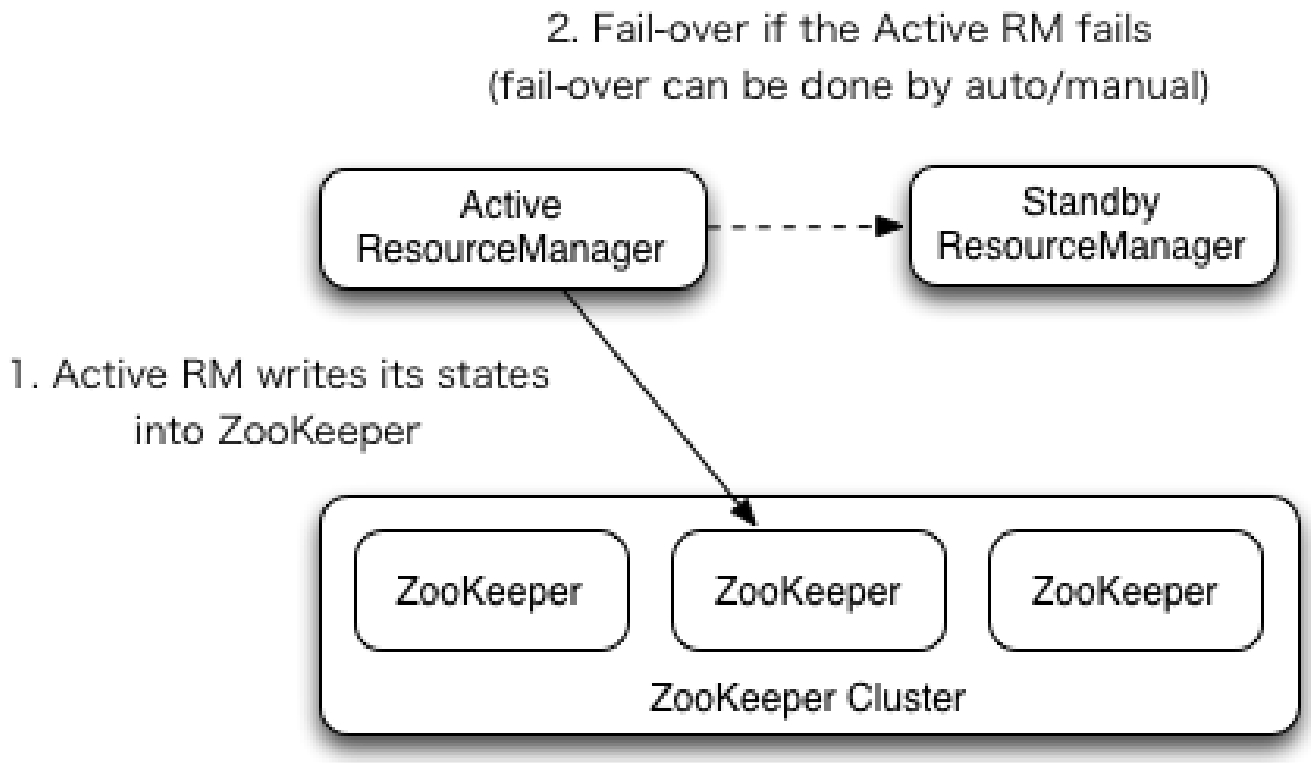


**Figure 6: Resource Manager Architecture [19]**

scripts bundled with current releases of Spark simplifies the process of launching spark on Amazon web services Elastic compute cloud EC2 [20].

### 10.3 Storage options for Spark

Spark can also easily integrate with myriad of commercial and open source 3rd party data storage systems such as MapR, Google Cloud, Amazon S3 [2] ,Apache Cassandra [3], Apache Hadoop, Apache Hbase [6] and Apache Hive [7] [20].

### 10.4 Spark stack

The spark project comprises of stack components mentioned in the Figure 10 and mainly consists of Spark core and 4 main libraries that were optimized to address the requirements of several different use cases. Any application would technically require Spark Core and at least one of these four libraries.The flexibility and main benefit of spark would become evident when applications that require combination of two or more libraries run efficiently on top of Spark core [20].

### 10.5 Spark Core

The heart of the spark lies at the Spark Core and is responsible for management functions like task scheduling, monitoring and other watchdog services. Spark implements and is based on a programming abstraction known as RDD (Resilient Distributed Dataset) [20].

### 10.6 RDD Design flow

RDD is designed to support in-memory data storage and distributed across a cluster in manner that is fault tolerant and efficient. Fault tolerance was achieved in part by tracking lineage of transformations that were applied to coarse grained data sets.Efficiency was achieved through process parallelization across multiple nodes in the cluster and minimization of data replication between respective nodes. Once we load the data into RDD, two basic types of operations are carried out [12].

- Transformations - this creates a new RDD by changing by actually changing the original by mechanisms such as filtering and mapping.
- Actions- some of the operations such as measuring the count which do not change the original data and the original RDD remains untouched throughout the process. Here the chain of transformations from RDD1 to RDDm would be logged and this process could be repeated in the scenarios such as data loss or failure of cluster node.

10

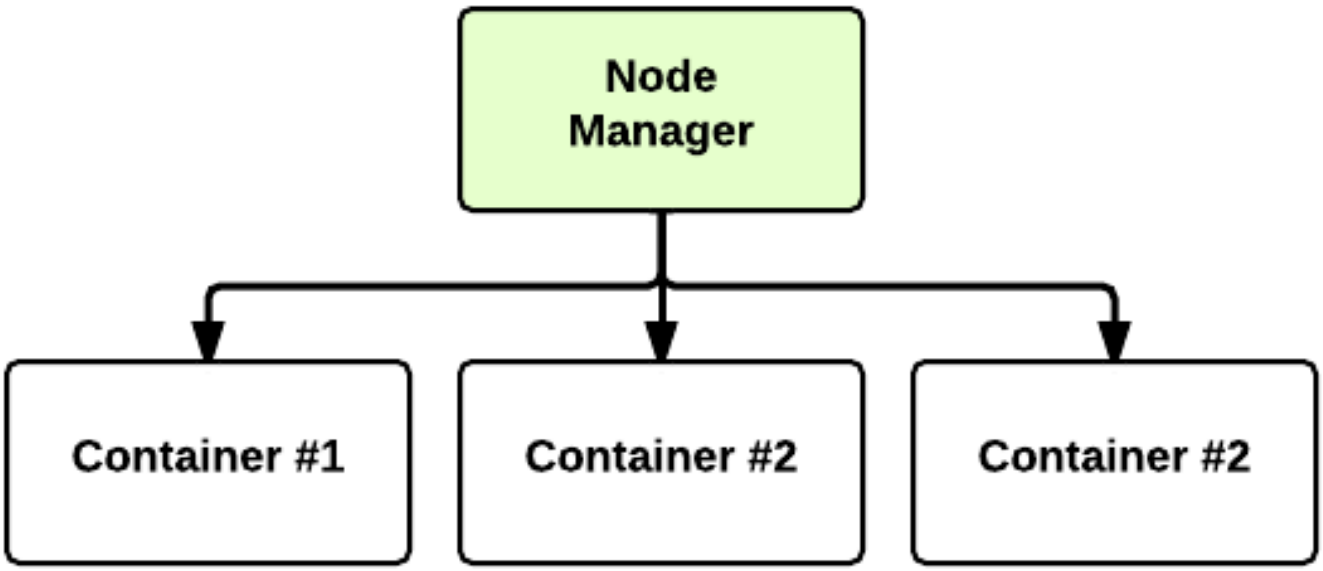


**Figure 7: Node Manager Division [24]**

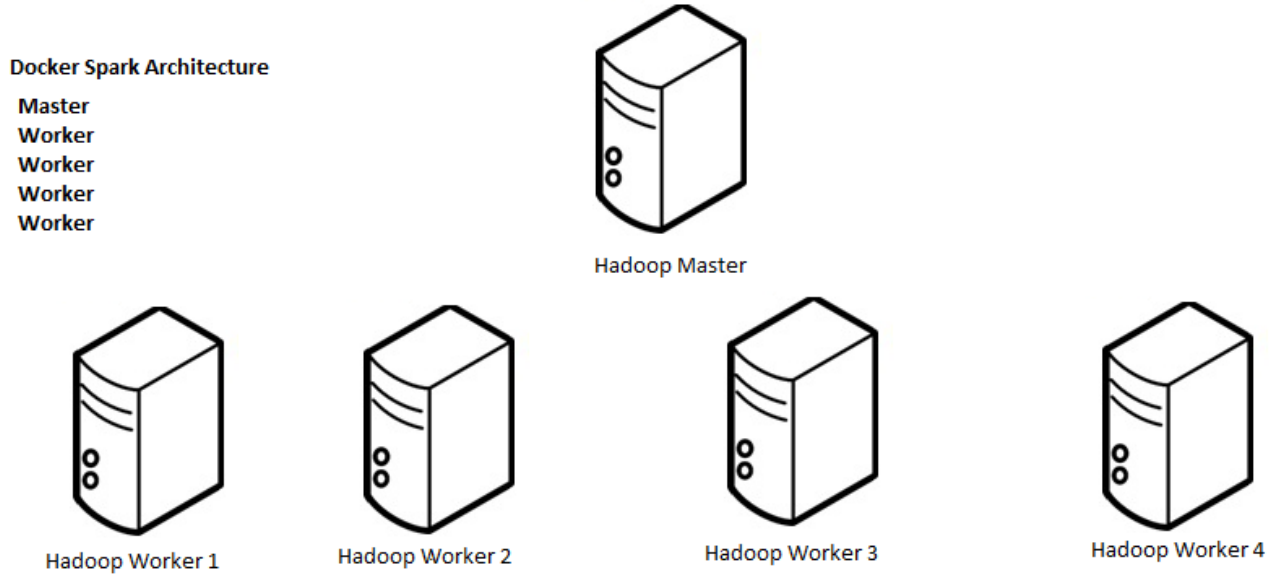


**Figure 8: Apache Hadoop Deployment Architecture on RaspberryPi 3**

Transformations tend to follow lazy evaluations in the sense that they would not be executed until a subsequent action/operation has an absolute need for result of previous action.This lazy evaluation helps in improving the performance as it could avoid unnecessary processing of the data. This would also introduce processing bottlenecks that could cause applications to stall while waiting for a processing action to conclude.

Spark tries to keep these RDDs in memory that greatly increase the performance of the cluster [12].

11

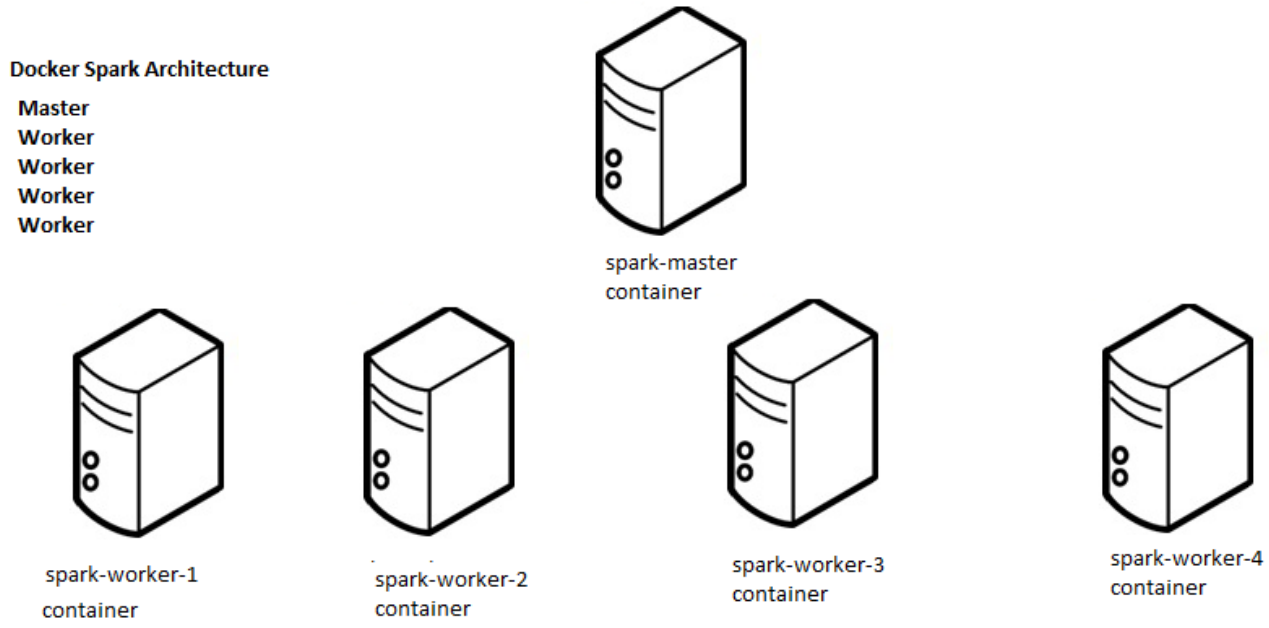


**Figure 9: Docker Spark**

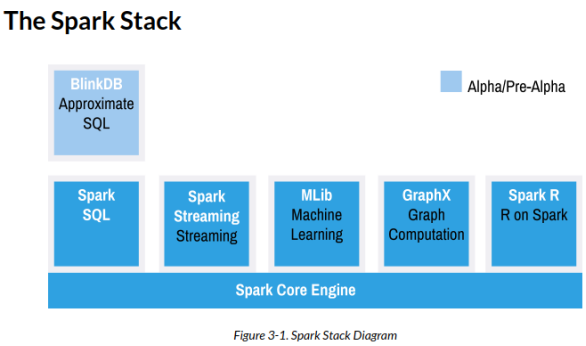


**Figure 10: Spark Stack [20]**

## 10.7 Reason for slowness in Data sharing on MapReduce

MapReduce is one of the most widely adopted technology for processing and generating large Datasets in parallel with a distributed algorithm on a cluster. Users were able to write parallel computations with help of high level operators and not worry about work distribution and fault tolerance. But in most of the current frameworks, the only way to reuse data between computations was write an external stable storage system such as HDFS. The fast growing user base needs interactive and interactive application that require data sharing at a faster pace. The sharing of data between jobs is slow in MapReduce due to replication, serialization and disk IO. Most of the hadoop applications tend to spend maximum time in reading and writing tasks in HDFS [12].

## 10.8 Iterative operations in MapReduce

The idea behind executing iterative tasks in MapReduce is to reuse intermediate results across multiple computations in multi storage applications. The Figure 11 clearly explains the working of the current framework in context of execution of interactive operations on Mapreduce. This clearly incurs substantial overheads due to I/O, replication of data and serialization that in turn degrades the system performance [12].

## 10.9 Interactive tasks in MapReduce

General users run ad-hoc queries on the same subset of data where each query would perform disk I/O on the stable storage that use most of the execution time. Figure 12 explains the current framework on execution of interactive queries on MapReduce [12].

12

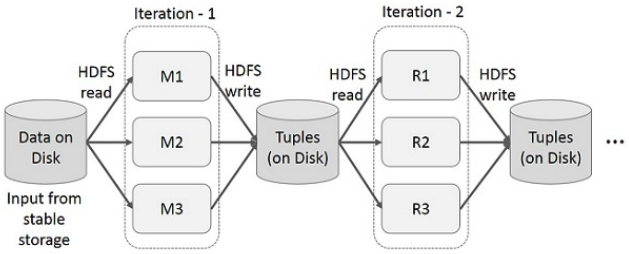


**Figure 11: Iterative Operations on MapReduce [12]**

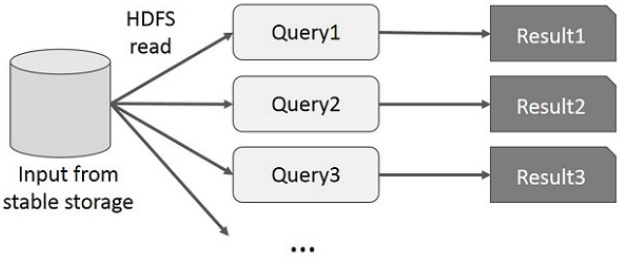


**Figure 12: Interactive Operations on MapReduce [12]**

## 10.10 Data sharing in Spark RDD

The key idea of spark is RDD which supports in-memory process computation.It stores the state of memory as object across the jobs and the object could be shared between respective jobs.Obviously data sharing is much faster than network and disk.

## 10.11 Iterative operations in Spark RDD

Illustration in the Figure 13 explains the iterative operations in spark RDD which stores the intermediate results in a distributed memory rather than stable storage such as HDFS and thus makes the system faster. Also if the distributed memory does not suffice to store the intermediate results then Spark stores those results on disk [12].

## 10.12 Interactive operations in Spark RDD

Figure 14 provides illustration of execution of interactive operation on Spark RDD where in if different queries are executed on same data set then that particular data is stored in memory for better execution times. Each transformed RDD might be re-computed each time we run an action against it.We could also persist an RDD in memory and spark would keep elements on the cluster for much faster access and increase the efficiency of whole process [12].

## 10.13 Spark Cluster Architecture

We have implemented spark cluster in two different ways

- Spark standalone cluster
- Running spark on Yarn on top of Hadoop

Figure 8 explains the architectural set up of both mentioned methods of implementing spark cluster.

For the first method, Apache Hadoop was not installed and hence does not consist of stable storage system (HDFS) . The basic cluster build configuration were followed that results to having a working cluster with 1 master and 4 workers and each name node is able to ssh to other name nodes in the cluster.

All the spark related configuration were mentioned in the file spark-env.sh and all 4 workers were listed in the slaves file. The second method uses Hadoop on YARN configuration where YARN provides all the information required for spark cluster to run successfully.

# 11 RESULTS

**Table 1: Stand Alone Spark Performance**

| Data Size | Technology | Execution time (seconds) |
|---|---|---|
| 3.6 MB | Java | 36 |
| 3.6 MB | Python | 23 |
| 1.5 MB | Java | 30 |
| 1.5 MB | Python | 18 |

**Table 2: Hadoop-Yarn-Spark Performance**

| Data Size | Technology | Execution time (seconds) |
|---|---|---|
| 3.6 MB | Java | 320 |
| 3.6 MB | Python | 450 |
| 1.5 MB | Java | 135 |
| 1.5 MB | Python | 189 |

The results in the Table 1 compare the performance of Standalone Spark on technology like Java and Python.

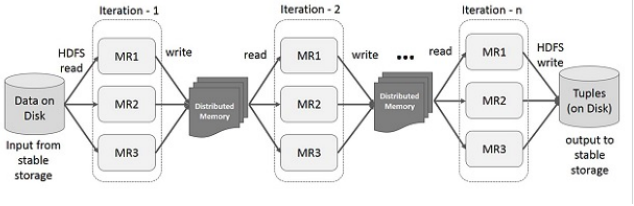


**Figure 13: Iterative Operations on Spark RDD [12]**

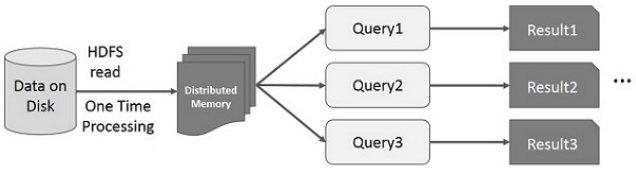


**Figure 14: Interactive Operations on Spark RDD [12]**

**Table 3: Stand Alone Docker Python Performance**

| Data Size | Iteration | Execution time (seconds) |
|---|---|---|
| 3.6 MB | One | 67 |
| 3.6 MB | Two | 24 |
| 3.6 MB | Three | 22 |
| 1.5 MB | One | 31 |
| 1.5 MB | Two | 12 |
| 1.5 MB | Three | 10 |

The Table 2 compares the performance of Hadoop Cluster on YARN with Java and Python.

The Table 3 makes a comparison of performance of standalone Docker running Python code on varies sized data sets.

Clearly, standalone Spark outperforms Hadoop (HDFS) because it persists the data in memory whereas the HDFS based Hadoop performs extra overhead operations such as shuffling of data and hence executes slower. Docker stores data in its own cache for subsequent runs, and hence the performace enhances for second and third run. The first run takes a long time on Docker since it has to register all the workers before execution.

At the outset it looks like Apache Spark seems to be performing better but at times HDFS Hadoop provides enough fault tolerance and stable storage which might help other applications. Hence both the frameworks have pros and cons and usage of these framework are greatly application specific.

## 12 CONCLUSION

Apache Hadoop and Apache Spark are widely used for parallel procesing of large amounts of data. Hadoop involves continous shuffling of data leading to an increased overhead of disk access. On the other hand, Spark takes the data in memory before doing the computation increasing the memory costs. The results show that for a smaller architecture such as a cluster Raspberry Pi's, Apache Spark outperforms Apache Hadoop. However, usage of either of the technologies depends entirely on the application, disk access and memory costs involved.

## 13 WORK BREAKDOWN

Initial configuration of all the Raspberry Pi and network configuration required for the clusters were equally contributed by all the team members. Karan Kamatgi and Karan Kotabagi did the configurations and server setup for the Spark Standalone cluster. Ramyashree DG and Manoj Joshi did the configuration and server setup for Hadoop-Yarn-Spark cluster and Karan Kamatgi contributed for the Yarn configuration of this cluster. Python Docker setup was contributed by Karan Kotabagi. Spark-Java benchmarking program was contributed by Ramyashree DG, Spark-Python benchmarking program was contributed by Manoj Joshi. Hadoop-Java benchmarking program was done by Karan Kotabagi and the Hadoop-Python benchmarking program was contributed by Karan Kamatgi. Evaluation and benchmarking were carefully analysed by all the team members as a group. All the team members have equally contributed to the sections of project paper, configuration steps for the Raspberry Pi cluster and also have put equal team efforts to resolve the issues faced during the cluster configuration and evaluation process.

## ACKNOWLEDGMENTS

The authors would like to thank Dr. Gregor von Laszewski for his support and suggestions to write this paper.

## REFERENCES

[1] [n. d.]. Amazon Kinesis. Web Page. ([n. d.]). https://aws.amazon.com/kinesis/
[2] [n. d.]. Amazon S3. Web Page. ([n. d.]). https://aws.amazon.com/s3/
[3] [n. d.]. Apache Cassandra. Web Page. ([n. d.]). http://cassandra.apache.org/
[4] [n. d.]. Apache Flume. Web Page. ([n. d.]). https://flume.apache.org/
[5] [n. d.]. Apache Hadoop. Web Page. ([n. d.]). http://hadoop.apache.org/
[6] [n. d.]. Apache HBase. Web Page. ([n. d.]). https://hbase.apache.org/
[7] [n. d.]. Apache Hive. Web Page. ([n. d.]). https://hive.apache.org/
[8] [n. d.]. Apache Kafka. Web Page. ([n. d.]). https://kafka.apache.org/
[9] [n. d.]. Apache Kudu. Web Page. ([n. d.]). https://kudu.apache.org/
[10] [n. d.]. Apache Mesos. Web Page. ([n. d.]). http://mesos.apache.org/
[11] [n. d.]. Apache Spark. Web Page. ([n. d.]). https://spark.apache.org

[12] [n. d.]. Apache Spark RDD. Web Page. ([n. d.]). https://www.tutorialspoint.com/apache_spark/apache_spark_rdd.htm
[13] [n. d.]. Apache TEZ. Web Page. ([n. d.]). https://tez.apache.org/
[14] [n. d.]. Docker. Web Page. ([n. d.]). https://en.wikipedia.org/wiki/Docker_(software)
[15] [n. d.]. Fog Computing. Web Page. ([n. d.]). https://en.wikipedia.org/wiki/Fog_computing
[16] [n. d.]. Hadoop Architecture. Web Page. ([n. d.]). https://www.datadoghq.com/blog/hadoop-architecture-overview/
[17] [n. d.]. HDFS Architecture. Web Page. ([n. d.]). https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html
[18] [n. d.]. MapR-FS. Web Page. ([n. d.]). https://mapr.com/products/mapr-fs/
[19] [n. d.]. Resource Manager architecture. Web Page. ([n. d.]). https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/ResourceManagerHA.html
[20] [n. d.]. Spark Architecture Overview. Web Page. ([n. d.]). https://mapr.com/ebooks/spark/03-apache-spark-architecture-overview.html
[21] [n. d.]. Spark Mllib. Web Page. ([n. d.]). https://spark.apache.org/mllib/
[22] [n. d.]. Twitter. Web Page. ([n. d.]). https://twitter.com/
[23] [n. d.]. YARN architecture. Web Page. ([n. d.]). https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html
[24] [n. d.]. YARN Internal overview. Web Page. ([n. d.]). http://ercoppa.github.io/HadoopInternals/HadoopArchitectureOverview.html
[25] Nitin Bandugula. [n. d.]. The 5-Minute Guide to Understanding the Significance of Apache Spark. Web Page. ([n. d.]). https://mapr.com/blog/5-minute-guide-understanding-significance-apache-spark/
[26] bmc. 2016. *Advantages of using Hadoop*. bmc. http://www.bmcsoftware.in/guides/hadoop-benefits-business-case.html
[27] Infinity Consulting. 2015. *Benefits and Importance of Hadoop*. Infinity Consulting. http://infinitylimited.co.uk/benefits-and-importance-of-hadoop-big-data-platform/
[28] Haley. 2013. *Hadoop: What It Is And How It Works*. readwrite. https://readwrite.com/2013/05/23/hadoop-what-it-is-and-how-it-works/
[29] KnowledgeHut. 2016. *Top Pros and Cons of Hadoop*. knowledgehut. https://www.knowledgehut.com/blog/bigdata-hadoop/top-pros-and-cons-of-hadoop
[30] Davin Lewis. 2015. *Introduction to Hadoop*. Code club. http://www.abstract-thoughts.com/tech-talk/introduction-to-hadoop
[31] MindsMapped. 2016. *HADOOP ADVANTAGES AND DISADVANTAGES*. mindsmapped. http://blogs.mindsmapped.com/bigdatahadoop/hadoop-advantages-and-disadvantages/
[32] Michele Nemschoff. 2013. *Big data: 5 major advantages of Hadoop*. ITProPortal. https://www.itproportal.com/2013/12/20/big-data-5-major-advantages-of-hadoop/
[33] Ipsita Pattnaik. 2017. *Advantages of using Hadoop*. e-Zest. http://blog.e-zest.com/advantages-of-using-hadoop
[34] Tutorials Point. [n. d.]. Apache Spark - RDD. Web Page. ([n. d.]). https://www.tutorialspoint.com/apache_spark/apache_spark_rdd.htm
[35] Ritika Prasad. 2017. *Why is hadoop important*. Quora. https://www.quora.com/Why-is-Hadoop-important
[36] Qubole. [n. d.]. Top Apache Spark Use Cases. Web Page. ([n. d.]). https://www.qubole.com/blog/apache-spark-use-cases/
[37] Sas. 2015. *Big Data, What it is and why it matters*. Sas. https://www.sas.com/en_us/insights/big-data/what-is-big-data.html
[38] Technavio. 2016. *Top 5 Benefits of Using Hadoop*. technavio. https://www.technavio.com/blog/top-5-benefits-using-hadoop
[39] techopedia. 2017. *Hadoop Yarn*. techopedia. https://www.techopedia.com/definition/30154/hadoop-yarn
[40] Jagadish Thaker. 2016. *Big data Week*. BDW. http://blog.bigdataweek.com/2016/08/01/hadoop-important-handling-big-data/
[41] Wikipedia. [n. d.]. Apache Spark. Web Page. ([n. d.]). https://en.wikipedia.org/wiki/Apache_Spark
[42] Wikipedia. [n. d.]. Raspberry Pi. Web Page. ([n. d.]). https://en.wikipedia.org/wiki/Raspberry_Pi
[43] Horton Works. 2014. *Apache Hadoop Yarn*. Horton Works. https://hortonworks.com/apache/yarn/#section_1

# Blockchain Implementation with Flask

Joao Paulo Leite
Indiana University
Smith Research Center
Bloomington, IN 47408, USA
jleite@ui.com

## ABSTRACT

The main purpose of this paper is to create a simple Blockchain implementation and simulate a network of nodes interacting with the Blockchain. To accomplish this, all interaction with the Blockchain are brokered through an API deployed using Python Flask. The goal is to have multiple nodes running and interacting with the Blockchain to show how a distributed ledger system functions.

## KEYWORDS

hid-sp18-414, Volume: 9, Chapter: Blockchain, Status: 100.
Blockchain, Distributed Ledger, Flask, Python

## 1 INTRODUCTION

The concept of blockchain technology was first introduced by Satoshi Nakamoto in his whitepaper, "Bitcoin: A Peer-to-Peer Electronic Cash System" in early 2008 [2]. The concepts that were outlined from Satwik Kansal at IBM was the guidelines that were followed to build this simple blockchain example [4]. It is important to note that we have developed a basic example that is meant to be an introduction into how blockchains and distrubuted ledger technology functions. Although the technology has grown tremendously since 2008, this project will focus on the core technology behind Blockchain 1.0. Topics such as smart contracts (Blockchain 2.0) and Decentralized apps (Blockchain 3.0) are not part of this project but understanding the basic design principles behind Blockchain 1.0 gives a good foundation to build upon.

## 2 OVERVIEW OF BLOCKCHAIN TECHNOLOGY

Blockchain 1.0 is considered the first iteration of blockchain technology. With this iteration, the goal was to eliminate the centralization and to create an open and inclusive global currency. To accomplish this goal, the blockchain is reliant on a distributed ledger which is stored and shared across the network [1]. As records (blocks) are added to the ledger, they become a permanent and unalterable part of the blockchain. To protect against tampering and bad actors, Consensus algorithms are deployed to ensure that the network is always in agreement as to legitimacy of the chain. The first of these algorithms to be developed (and the one deployed for this project) was the Proof of Work Algorithm. This algorithm relies on extremely difficult cryptographic puzzles (work) which can only be resolved through brute force. Within the network, different members (miners) will attempt to solve this puzzle for a reward. Once they solve the puzzle, they announce the solution to the network. Before any node within the network adopts this new block, they will confirm that the solution provided did in fact solve the puzzle. Once it is confirmed,a new block is recorded on the chain and within this block, new transactions can be stored. From a functional point of view, once a transaction has been placed in the Blockchain, the transaction can be deemed complete. [8]

## 3 API

This project utilizes Python Flask to expose some standard operations that are common with any distributed ledger. The two most fundamental functions to any blockchain are the posting of transactions and the mining of new blocks. These functions go hand in hand, as new transactions can only be posted to a block when a new block is mined and added to the chain. Some secondary functions that provide stability and agreement between all the nodes in the network is the Consensus algorithm and the discovery of other nodes in the network through Node Registration. Between these four functions, the project is able to product a functioning Blockchain where multiple nodes can interact by posting transactions and mining new blocks, while staying in sync and maintaining the ledger's integrity.

### 3.1 Consensus

The Consensus command is implemented to provide a mechanism to maintain the Blockchain uniformly across all the nodes within the network. In this project, the consensus is resolved by analyzing which Blockchain across all the nodes is the largest.

**Consensus code which analyzes the current nodes length vs the other nodes in the network:**

```
def consensus(self):
    ...
    len_chain = len(self.chain)
    for node in others:
        strurl = f'http://{node}/chain'
        response = requests.get(strurl)
            if response.status_code == 200:
             chain = response.json()['chain']
                length = response.json()['len']

                if length > len_chain
```

When one node successfully mines a new block, they broadcast their solution so that the other nodes can begin the adoption process. The other nodes are then responsible to come to a consensus around the legitimacy of the new block. These nodes can quickly validate that the solution (proof) provided does maintain the integrity of the chain by using the solution to solve the cryptographic puzzle.

**Validation code that checks the proposed block's solution (block['proof']) as valid:**

```
def validation_chain(self, chain):
    ...
    if not self.validation_block(last_block,block,last_hash):
```

```
            return False

        last_block = block
        current_index += 1

        return True
```

As each node confirms the solution provided, the block is adopted by that node.

**Consensus code which adopts the new blockchain as it's own:**

```
def consensus(self):
    ...
if replacement_chain:
        self.chain = replacement_chain
        return True
    return False
```

If the node is a miner node, once the block has been adopted, all the work done to find the solution is lost and must be restarted with the information of the adopted block.

## 3.2 Transactions

The transaction command provides the ability for a node to issue a transaction which will be published on the blockchain. The transaction function is simplified in this project, consisting only of a sender address, a recipient address and an amount. In a production level scenario, transactions may include a transaction fee which can be set at the time the transaction is created. This is because each block has a max size which limits the number of transactions that can be posted within a block. Because of this fact, all transactions go to a pool (TX pool) and the transactions offering a higher fees are prioritized.

**Create Message code:**

```
def create_message(self, Timestamp, Sender, Message):
    ...
    self.trans.append({
        'Timestamp': Timestamp,
        'Sender': Sender,
        'Message': Message,
    })
```

## 3.3 Node Registration

The registration command provides the ability for a node to register other known nodes. In this project, a node is represented by URL (ex: `http://127.0.0.1:8888`). The nodes are stored in a set, to be used by the consensus command to access other node's chains.

**Node Registration code:**

```
def node_registration(self, address):

    url = urlparse(address)
    if url.netloc:
        self.nodes.add(url.netloc)
    elif url.path:
        self.nodes.add(url.path)
    else:
        print("ERROR")
```

**Consensus algorithm leveraging stored nodes:**

```
def consensus(self):
    ....
    others = self.nodes

    ....
    for node in others:
        strurl = f'http://{node}/chain'
        response = requests.get(strurl)
```

## 3.4 Mine

The mine command allows for new blocks to be mined for the chain. In this project, the difficulty threshold for the proof of work algorithm was set very low so that blocks can be created almost instantaneously.

**Proof of Work algorithm:**

```
def pow(self, oldBlock):
    'Proof of Work Algorithm'

    oldproof = oldBlock['proof']
    oldhash = self.hashing(oldBlock)
    newproof = 0

    while self.validation_block(oldproof,newproof,oldhash) is False:
        newproof += 1

    return newproof
```

Typically in a production scenario, the goal is to have the difficultly threshold set to a level which causes a block to be created every 10 minutes (on average). Because the mining pool's hash power can fluctuate, the difficulty threshold is adjusted to ensure it does not become to easy or difficult in relation to the hash power. [6]

**Validation Block code which is the main component of the PoW algorithm:**

```
def validation_block(oldproof, proof, oldhash):
    'Validates the solution to the PoW'

    encoding = (str(oldproof)+str(proof)+str(oldhash)).encode()
    attempt = hashlib.sha224(encoding).hexdigest()
    return attempt[:POW_Difficulty] == '0' * POW_Difficulty
```

To adjust the difficulty in this project, we can increase the number of consecutive zero's needed to be returned from the hash. Alternatively, if we decrease the number of consecutive zero's, the difficulty is reduced.

# 4 EXAMPLES

## 4.1 register

```
request:
curl -H "Content-Type: application/json"  \
        -X POST \
        -d '{"nodes": ["http://127.0.0.1:8888"]}' \
        http\://localhost\:9999/register

response:
{
  "message": "Nodes Added",
  "nodes": [
    "127.0.0.1:8888"
```

2

```
  ]
}
```

### 4.2 consensus

```
request:
curl -H "Content-Type: application/json" \
        -X GET \
        http\://localhost\:8888/consensus

response:
{
  "message": "Chain not Replaced",
  "replacement_chain": [
    {
      "TS": 1525148850.3083794,
      "index": 1,
      "oldhash": "1",
      "proof": 100,
      "trans": []
    }
  ]
}
```

### 4.3 newtransaction

```
request:
curl -H "Content-Type: application/json"  \
    -X POST \
    -d '{"Sender": "John", "Message": "Hello"}' \
    http\://localhost\:8888/newtransaction


response:
{
  "TS": 1525463137.4779193,
  "index": 2,
  "oldhash": "66ae3106045d3e00f8d62204a3e269381cd91deb6df574b067c125d9",
  "proof": 213
  "trans": [
    {
     "Message": "Hello",
     "Sender": "John",
     "Timestamp": 1525463137.0406077
     }
 }
```

### 4.4 chain

```
request:
curl -H "Content-Type: application/json" \
        -X GET \
        http\://localhost\:8888/chain

response:
{
  "chain": [
    {
      "TS": 1525148850.3083794,
      "index": 1,
      "oldhash": "1",
      "proof": 100,
      "trans": []
    }
  ],
  "len": 1
}
```

### 4.5 mine

```
request:
curl -H "Content-Type: application/json" \
        -X GET \
        http\://localhost\:9999/mine

response:
{
  "block number": 3,
  "message": "New Block Created",
  "previous hash": "5e67bb7aec15844ab65ebc890ff421eddd14eaa7aaca86009225d20c",
  "proof": 1758,
  "trans": [
    {
      "Message": "Hello",
      "Sender": "Luciana",
      "Timestamp": 1525463138.9406605
    },
    {
      "Message": "How are you",
      "Sender": "Luciana",
      "Timestamp": 1525463139.3841374
    }
  ]
}
```

## 5 TOOLS AND TECHNOLOGY

The tools and technology deployed for this project are going to be covered in this section.

### 5.1 Flask - Web Framework

Flask is a micro-framework for Python based on Werkzeug and Jinga 2. The reason it is called a micro framework is because it doesn't require any specific tools or libraries. For this project, Flask was used to build the REST API which allows us to interact with specific functions of the blockchain [**?** ].

**Code Example:**

```
from flask import request, Flask, jsonify
app = Flask(__name__)

@app.route('/mine', methods=['GET'])
def mine():
```

**Install:**

```
pip install flask
```

**Console Example:**

```
* Running on http://0.0.0.0:9999/ (Press CTRL+C to quit)
```

### 5.2 Hashlib

Hashlib is a Python module with a common interface to many of the more widely used hash and message digest algorithms.For this project, it was used to call the SHA256 hashing function [3].

**Code Example:**

```
def hashing(block):

return hashlib.sha256(...).hexdigest()
```

3

**Install:**

```
easy_install Hashlib
```

### 5.3 Python

Python is the high-level programming language that was used to develop this project.

### 5.4 Requests

Requests is a Python module that allows you to craft HTTP/1.1 requests without needing to include the query string or to format the POST data [5].

**Code Example:**

```
response = requests.get(f'http://{node}/chain')
```

**Install:**

```
pip install Requests
```

### 5.5 Universally unique identifiers

Universally unique identifiers (UUID) is a Python module that is used to create unique identifiers without the need for a centralized server/registrar. The values created by UUID are 128 bits long and work well with cryptographic hashing which provided useful when assigned node identifiers [7].

**Code Example:**

```
nodeid = str(uuid1()).replace('-', "")
```

**Install:**

```
pip install uuid
```

## 6 CONCLUSION

This project's aim was to provide a blockchain implementation with all the basic functionality needed to deploy it in a multi-node scenario. The project was able to demonstrate a blockchain that could add transactions, create new blocks to the chain and maintain the integrity of the chain across all the nodes on the network. This was accomplished using Python Flask as the backbone to create and deploying nodes.

## ACKNOWLEDGMENTS

The authors would like to thank Dr. Gregor von Laszewski for his support and suggestions to write this paper.

## REFERENCES

[1] Kyle Burgess and Joe Colangelo. 2016. The Promise of Bitcoin and the Blockchain. Web Page. (jan 2016). https://bravenewcoin.com/assets/Industry-Reports-2016/Bretton-Woods-2015-White-Paper-The-promise-of-Bitcoin-and-the-Blockchain.pdf Accessed: 2018-03-16.

[2] Nicola Dimitri. 2017. The Blockchain Technology: Some Theory and Applications. Web Page. (nov 2017). https://WebPage.msm.nl/resources/uploads/2017/10/Working-Paper-No.-2017-3.pdf Accessed: 2018-03-16.

[3] Hashlib [n. d.]. hashlib - secure hashes and message digests. Web Page. ([n. d.]). https://docs.python.org/3/library/hashlib.html Accessed: 2018-04-16.

[4] Satwik Kansal. [n. d.]. Develop a blockchain application from scratch in Python. Web Page. ([n. d.]). https://WebPage.ibm.com/developerworks/cloud/library/cl-develop-blockchain-app-in-python/index.html Accessed: 2018-04-17.

[5] Requests [n. d.]. Requests: HTTP for Humans - Requests. Web Page. ([n. d.]). http://docs.python-requests.org/en/master/ Accessed: 2018-04-16.

[6] Ameer Rosic. 2017. Proof of Work vs Proof of Stake: Basic Mining Guide. Web Page. (mar 2017). https://blockgeeks.com/guides/proof-of-work-vs-proof-of-stake/ Accessed: 2018-03-16.

[7] UUID [n. d.]. uuid - UUID objects. Web Page. ([n. d.]). https://docs.python.org/3/library/uuid.html Accessed: 2018-04-16.

[8] J Leon Zhao. 2016. Overview of business innovations and research opportunities in blockchain and introduction to the special issues. In *Financial Innovation*. Springer Open, 2:28. https://doi.org/10.1186/s40854-016-0049-2 Accessed: 2018-03-16.

# Linear Regression REST Service and API

Janaki Mudvari Khatiwada
Indiana University
Bloomington, IN 47408, USA
janu.khatiwada@gmail.com

## ABSTRACT

Python's flask application requires minimal application for application development. Flask make use of backend data as a resource to support an application running on the server. This project discusses development and deployment of REST api with python flask. This flask api gets current details info based on regression model created from car dataset. Once flask app runs successfully, it is then deployed to Heroku, a platform as a service cloud platform. For the purpose car data set from UCI machine learning data repository is being used. Before this, a regression model for the data has to be built to make sure that this model best describes the relationship between variables, in this case engine size and highway miles per gallon. Based on this model new data set can be tested to make prediction. Then regression results is used as the endpoint in flask api. This service is then tested by running dockerfile.

## KEYWORDS

hid-sp18-415, Volume: 9, Chapter: REST, Status: 100.
Flask, Cloud Computing, Regression Model

## 1 INTRODUCTION

Flask is a web application framework "based on the Werkzeug WSGI (Web Server Gateway Interface) toolkit and Jinja2 template engine" [**?** ]. Being an interface between web server and and web applications, "it implements requests and response objects" while jinja2 render web page by combining template and data source [10]. Flask api framework requires minimal framework for RESTful web services. Rest services is an architectural style web services which use HTTP protocol to establish communication between client and server during each service request. Being lightweight and scalable, flask is chosen as a good option for developing the RESTful apis. While flask does not have built–in support for database handling and validation support, these functionality can be added as extensions to the application [10]. The main purpose of this paper is to create a regression model for car data set and then create a web service with flask. The model is intended to predict highway miles per gallon based on engine size of the vehicle. Scikit-learn is used to run simple linear regression and build the prediction model. This model is going to be the resource in flask api. Using GET and POST methods to call endpoints. GET method fetch resource info while POST method creates a new resource. Resources and requests for API can be in different data format but json format is easy to understand and flask has in–built support for this data format. So our resource will be our json object stored in key–value pair.

Once flask api is succesfully run in localhost, the application is then deployed to heroku which is an open source platform as a service cloud platform. This also has a minimal deployment needs. It requires opening an account in heroku.com, then download heroku command line interface (CLI). Then run heroku login from CLI and and create an app. This flask appplication is about running machine learning learning algorithm in heroku. It requires python packages such as pandas and scikit–learn, we deploy our app by packaging all required packages in docker container and pushed to the platform. This process is discussed in more detail under *methods* section.

## 2 SIMPLE LINEAR REGRESSION

Just to provide a brief overview of why simple linear regression is chosen for the project, a brief introduction about it would be appropriate. Linear Regression is a statistical data analysis tool which describes the effect of one variable, also called independent variable, on dependent variable. In case of simple linear regression there is only one dependent variable with one independent variable. Our purpose of running this project is to create a line of best fit model to build a model that can be used for prediction. In this case prediction of mpg based on engine size of the vehicle. The line of best fit is described by the formula

$$y = a + bx$$

where a is coefficient or y intercept and b is slope for the line. In this case formula for prediction model or regression is given below, where `hwy\_mpg` is y variable and `engine\_size` is x variable.

$$hwy\_mpg = yintercept + slope.enginesize$$

## 3 METHODS

Data selection, cleaning and preprocessing, building a local environment, building a linear regression model using SciKit–learn, designing a REST api with Flask. The application is then tested by running locally and also deploying it into heroku are the main processes of this project. They are explained below in details.

Automobile Data Set, *https://archive.ics.uci.edu/ml/ machine–learning-databases/autos/imports–85.data*, from UCI Machine learning data repository is selected. It was created on 19 May, 1987 by Jefferey C. Schlimmer [6]. The data set is multivariate with 25 attributes. Since the project is focused on designing and deploying a simple linear regression api as a demonstration of implementing Flask service, only two attributes, 'engine size' and 'highway miles per gallon' are used. Rest of the attributes are dropped. The goal of the project is to build a best fit prediction model for highway mpg based on engine size. The data set have several missing instances as well which are removed as a clean up process. Cleaned data is stored in Dropox as a csv file so that it can be easily fetched into the application [2].

Ubuntu 16.04 is the operating system used for whole project. A virtual environment pyenv is created in Ubuntu 16.04 where python 3 environment is built. Besides, flask, this project requires different python packages to fetch and manipulate data and run the linear regression and then build a flask api. Using pip install package

name, packages Flask, pandas, numpy, scipy and SciKit–learn are installed. Besides these packages, python packages matplotlib and seaborn are also installed as they are required to create a regression plot and plot line of best fit.

Once environment is setup regression or prediction model needs to be build from the data set so that this model is used in Flask app. This is done by running python's machine learning application. There are two types of machine learning, supervised and unsupervised learning. Supervised Machine learning is learning properties of data set (training data set) and applying them to the test data set. This machine learning algorithm is also called supervised learning. Regression problem in Scikit–learn is supervised machine learning. In this case predicting highway mpg for the new data set based on engine size is the regression problem. Data is split into train set and test set in 2:8 ratio.

Regression analysis or building a best fit model for our data set is discussed in detail under section Analysis and Algorithm. Now the data set is split into two sets one used for defining a regression equation. This equation is going to be the prediction model for data. Once regression equation is defined that is value for slope and intercept is calculated. Source codes are located in github repository [8].

The final process is writing a flask api which can be run to call endpoints using 'GET' and 'POST' methods. In this project predict is the endpoint for POST method and and current details is the endpoint for GET method. Then this application is deployed into heroku cloud platform from CLI. Before deploying 'heroku create app–name' is run to create an app within heroku. Then run 'git init' to initiate a remote git repository, since app is deployed through git push. In order to deploy to heroku it needs 'Procfile' with the process it needs to run and requirements.txt or Pipfile that has lists of dependencies needed for the app. Since this api needs to run regression, all the needed packages numpy, scipy, pandas and scikit–learn are configured from Dockerile. For a minimal flask deployment following commands are run 'git remote', 'git add' to add files, 'git commit' and 'git push heroku master'. In this case app is pushed using docker container for simplification.

## 4 ANALYSIS AND ALGORITHMS

The cleaned data set has two attributes *engine size* and *highway miles per gallon* and 199 data points. Before developing a flask api, a linear regression machine learning model is created using scikit-learn. As discussed earlier this is going to be the prediction model and one of the endpoints for flask api.

As it is mentioned above data set is split into two, one is train set and other is test set using algorithm train_set_split. Train–set data is used to build the regression model. In order to train data set, we move label column to its own data frame called 'train_labels by pulling series"hwy_mpg" from train–set [8]. Then 'LinearRegression' class is created and fit method is called by

$$lin\_reg = LinearRegression()$$

$$lin\_reg.fit(train\_set, train\_labels)$$

. Now we calculate intercept and slope our model. These values are used to create model formula as discussed in linear regression section above. Then predict function is called by passing series test_set which returns an array of predicted values [8]. This model can be checked by calling print function for hwy_mpg_pred and test_set series and compare values for each series. Another method to check how significant the model is by calculating the value for r–square which in this case is lin_reg score by running following algorithm.

$$lin\_reg.score(test\_set, test\_set\_full['hwy\_mpg'])$$

The value of this score ranges between 0 to 1. R square is called coefficient of determination. R square means that all the data points of the model falls perfectly on regression line. If value is more towards 0 then the regression line far away from the data points and model is not a good fit for the data set. For the created model of this analysis r square value is 0.584835 can be rounded to 0.585.

After defining the model flask api is developed where just created model will be the resource object. The application file is located in github repository [7]. But before this, we should make sure the model do not change when new data points are added. For the purpose joblib is being used. handle arrays stored in the models. Joblib persists newly created scikit–learn model so that for future use the model does not need to be retrained [9]. Function joblib.load is being used in the application to load pickled regression model. This api calls one resource current–details and post predict value of hwy_mpg when value for engine size is entered.

## 5 REGRESSION RESULTS

R square value shows how likely that predicted highway mpg fits into the regression model. It is 58 percent likely that future prediction will fall on the best fit line. It can be concluded that the model is fairly fit. While 58 percent of variance in highwaympg can be explained by engine size, there is 42 percent unexplained variance in highway mpg. To compare real highway miles per gallon values with predicted values, print function is run for test_set and hwy_mpg_pred set.

Results can be compared between Table 2 and Table 1. The first Table 1 lists highway mpg with engine size from our test_set data. Just looking at the table we can see negative relationship between engine size and highway mpg.

while Table 2 shows an array of predicted highway mpg.

Looking at both tables the predicted values while not exactly same as observed values are quite similar to observed values from test dataset. If predicted output are exactly similar to that of observed then this would have been a perfectly fit model with all data points from test_set passing through the line of regression plot from model with r–square value equal to 1. The whole output can be found in github [5]. It shows that values are pretty close.

Another way to check how good or accurate the model is by plotting regression plot from from model over scatter plot from test set. Figure 1. The figure below shows a regression plot over scatterplot. The regression plot from original data shows the variability in observed values of highway mpg is explained by the independent variable.

Regression plot from model as seen in Figure 2 shows the regression line or line of best fit passes nearby most of the data points.

There are few outliers. The regression model can also be compared with the scatterplot for the whole data set in Figure 3 as shown below, shows fairly show how data points are aligned.

| engine_size | hwy_mpg |
|---|---|
| 82 | 32 |
| 15 | 22 |
| 111 | 25 |
| 177 | 34 |
| 76 | 30 |
| 163 | 30 |
| 68 | 18 |
| 67 | 25 |
| 120 | 30 |
| 173 | 24 |
| 176 | 46 |
| 148 | 32 |
| 65 | 25 |
| 30 | 38 |
| 86 | 37 |
| 85 | 30 |
| 55 | 23 |
| 60 | 42 |
| 90 | 37 |
| 159 | 29 |
| 16 | 20 |
| 124 | 25 |
| 96 | 34 |
| 172 | 24 |
| 66 | 25 |
| 189 | 28 |
| 147 | 37 |
| 9 | 29 |
| 18 | 43 |
| 128 | 28 |
| 190 | 28 |
| 45 | 19 |
| 192 | 22 |
| 164 | 30 |
| 101 | 25 |
| 69 | 18 |
| 126 | 28 |
| 123 | 25 |
| 75 | 38 |
| 78 | 32 |

**Table 1: Highway MPG for Each Engine Size From Test Dataset**

## 6 DEPLOYING APP TO HEROKU

This service is tested using Dockerfile and Makefile and is able to fetch output locally. Now, the application is tested in heroku cloud platform. Heroku is chosen as the platform is open source as long as the application needs few 'dyno', that is one or two. Which means as long as application has minimal requirements. Dynos are containers provided by heroku platform where we can push our application and instruct them to execute the application.

| Array of hwy_mpg_pred | | | | |
|---|---|---|---|---|
| 31.18833214 | 21.44002624 | 27.82684735 | 32.64497555 | 33.87751997 |
| 28.4991443 | 18.63878892 | 24.35331306 | 31.18833214 | 25.69790698 |
| 33.98956946 | 34.54981693 | 24.35331306 | 34.54981693 | 33.98956946 |
| 32.53292605 | 35.89441084 | 31.18833214 | 33.98956946 | 33.87751997 |
| 21.44002624 | 23.12076864 | 31.41243112 | 25.69790698 | 24.35331306 |
| 29.05939177 | 34.54981693 | 32.75702504 | 34.77391591 | 31.30038163 |
| 29.05939177 | 15.94960108 | 30.29193619 | 28.4991443 | 24.57741205 |
| 18.63878892 | 31.30038163 | 23.12076864 | 34.54981693 | 31.18833214 |

**Table 2: Array of Predicted Highway MPG from Model**

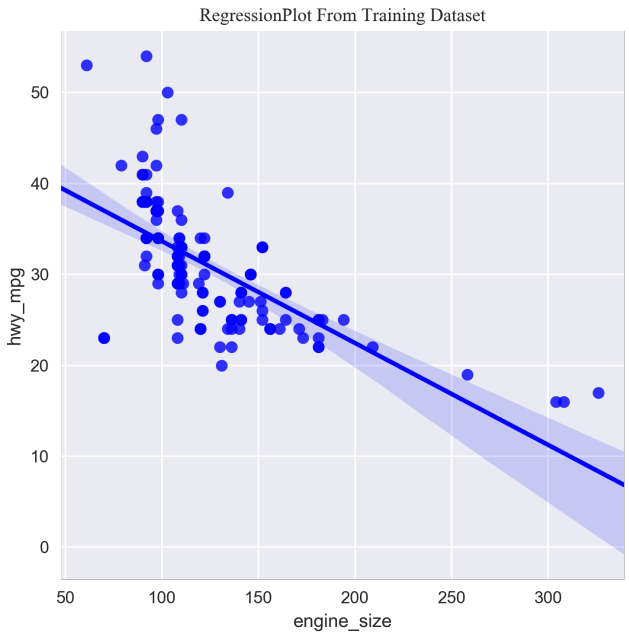


**Figure 1: Regresssion Plot Engine size and Highway MPG**

Another reason of choosing heroku is that we do not have to worry about containers or nodes to scale up app deployment. Heroku manages all of these needs. As an application developer, we just need to work on designing the intended app. For authentication an user account is created in heroku.com using heroku dashboard.Then heroku command line is downloaded in local terminal. Then run heroku login which in turn asks us to enter our email and password for authentication.

As this project requires python's data analysis and machine-learning packages it is then deployed using Dockerfile.

### 6.1 Dockerfile and Heroku Container Registry

Our app named *regressionAPI* needs several python packages. These packages can be added using heroku buildpacks. But for convenient deployment it is deployed using docker. Docker needs to be installed locally before starting the process. Then 'Dockerfile' is created so that all of our required packages are fetched from it. Docker image is built from miniconda which comes with python, pandas, numpy


**Figure 2: Prediction Plot Engine size and Highway MPG**


**Figure 3: Scatter Plot Engine size and Highway MPG**

and scipy. Using conda install scikit–learn, this package is also fetched into the container [3].

Now app is almost ready to be deployed to Heroku. The process needs following three files.

- A Dockerfile that helps with building a docker image and installing all the dependencies or requirements for running Flask api in Heroku. To deploy Flask api using Dockerile, container has to be registered into heroku by command.
- Requirements.txt file [4]
- Python files or application files. In this case regressionAPI.py
- wsgi.py file [4]
- A Procfile that tells heroku what process to run. Here is an example of Procfile `web: gunicorn appname:app`. In this example web is a process which tells heroku to run in the web and gunicorn handles requests and appname is name of our application which in our case is regressionAPI [7].

Now we need to run following commands from local terminal. `heroku container:login` [1]. This registers our docker container to heroku. Then `heroku create` creates app remotely into heroku and gives us app name from heroku Then to buildbuild image, push to Container Registry by command

```
heroku container:push web --app appname
```

and open the app in the browser herokuapp.com. With following command [1]:

```
heroku open --app appname
```

It launches app into heroku. By logging into heroku dashboard execution logs can be observed whcich helps debugging the whole process.

## 7 LIMITATIONS AND CHALLENGES

The project was intended also to do benchmarking by running rest services in locally developed raspberry pi cluster. Three node pi cluster is created for the purpose with raspbian images on all of them. Certain issues in packages installations are the challenges at the moment.

## 8 CONCLUSION

REST services with flask is simple to develop and deploy. This project build a simple regression model from car dataset. Regression plot and r square value has been used to explain variance in miles per gallon (our dependent variable). The model details is later used as a path for flask application. Also based on the model new data points can be used for future prediction. The application is successfully run in localhost with command python regressionAPI.py. Also deployed in heroku cloud platform using docker container. The whole project has been dockerized in a docker container using Dockerfile.

## ACKNOWLEDGMENTS

The authors would like to thank Dr. Gregor von Laszewski for his support and suggestions to write this paper and teaching assistants who helped me with all sorts of technical assistants.

## REFERENCES

[1] Heroku Dev Center. 2018. Container Registry and Runtime. Documentation. (2018). https://devcenter.heroku.com/articles/container-registry-and-runtime
[2] Cloudmesh Community hid sp18 415. 2018. CarData. Box. (April 2018). https://www.dropbox.com/s/986566hudfytl8h/cardata.csv?dl=0
[3] Cloudmesh Community hid sp18 415. 2018. Heroku Dockerfile. Github. (April 2018). https://github.com/cloudmesh-community/hid-sp18-415/blob/master/project-code/heroku/Dockerfile
[4] Cloudmesh Community hid sp18 415. 2018. Heroku Dockerfile. Github. (April 2018). https://github.com/cloudmesh-community/hid-sp18-415/blob/master/project-code/heroku/regapp
[5] Cloudmesh Community hid sp18 415. 2018. Regression Analysis. Github. (April 2018). https://github.com/cloudmesh-community/hid-sp18-415/blob/master/project-paper/regression_analysis.ipynb
[6] Dheeru Dua and Karra Taniskidou Efi. 2017. Automobile Data Set, Data Set Description. webpage. (August 2017). https://archive.ics.uci.edu/ml/machine-learning-databases/autos/

[7] Cloudmesh Community hid sp18 415. 2018. Project Code regressionAPI.py. Github. (April 2018). https://github.com/cloudmesh-community/hid-sp18-415/blob/master/project-code/regressionAPI.py
[8] Cloudmesh Community hid sp18 415. 2018. Project Code regression.py. Github. (April 2018). https://github.com/cloudmesh-community/hid-sp18-415/blob/master/project-code/regression.py
[9] Scikit learn Documentation. 2017. Model Persistence. webpage. (August 2017). http://scikit-learn.org/stable/modules/model_persistence.html
[10] Tutorialspoint. 2018. Flask Overview. Webpage. (2018). https://www.tutorialspoint.com/flask/flask_overview.html

# Big Data Reference Architecture using Python Celery

Sabra Ossen
Indiana University
Smith Research Center
Bloomington, IN 47408, USA
sossen@iu.edu

Gregor von Laszewski
Indiana University
Smith Research Center
Bloomington, IN 47408, USA
laszewski@gmail.com

## ABSTRACT

Big Data reference architectures are of great importance and can be built by various tools supporting distributed tasks. With the use of Python Celery this project focuses on building such a distributed Big Data reference architecture. The main goal of the project is to identify the key requirements for the domain and build the components of the system. K-means is a widely known clustering algorithm that is used on a huge amount of data sets. Therefore, the use case for this project is built around executing the K-means algorithm. The environment is also built on three environments; local, distributed cloud and multi-container Docker environments.

## KEYWORDS

hid-sp18-416, Volume: 9, Chapter: Workflow, Status: 100.
Python Celery, Swagger REST services, Node JS, Spark, Hadoop

## 1 INTRODUCTION

Today, big data is a highly available, crucial and necessary for every domain. Machine learning is the key methodology to analyze the big data obtained from various sources. With the high volume of big data, it becomes a necessity to have a distributed environment to handle the different components such as algorithm execution, storage, and summarization.

This project aims on building a reference architecture for executing machine learning in a distributed environment. The architecture will be built upon several components such as Python Celery [4], Swagger REST [20], Redis [14], Apache Hadoop [9], Node JS [13] and Spark [18]. The main components of the architecture are Python Celery and Swagger REST services. Python Celery allows user-responsive long-running tasks to run in the background using distributed task queues.

The first phase will focus on implementing the architecture in a local environment and the second phase will focus on implementing it in a cloud environment. Amazon Elastic Compute Cloud (EC2) instances [1] and Amazon Elastic File System (EFS) [3] will be used to build the distributed cloud environment. The third phase focuses on implementing a multi-container Docker environment for the distributed architecture using Docker [5] and Docker Compose [6]. Two 'Clustering' based datasets from the UCI Machine Learning repository [17] will be used for evaluation of the distributed architecture.

The project goals are listed as follows.

(1) Create a Node JS Server exposing the set of functions to upload to (input files) and download from (predictions and model files) in a Hadoop Distributed File System (HDFS) [12]
(2) Execute machine learning function (K-means example) using Apache Spark [18].
(3) Expose the tasks supported by the distributed Python celery [4] workers through a Swagger REST service [20]
(4) Build the reference architecture in the local, cloud and docker environments.
(5) Provide a thorough guide on building distributed Big Data analysis environments.
(6) Analyze the performance of the system in local and cloud environments in terms of uploading the input file, executing K-means and retrieving the model and prediction files.
(7) Analyze the performance of the system in local and cloud environments in terms of uploading various sized input files.

## 2 TECHNOLOGY USAGE

This section focuses on providing an overview of the many tools used in this project. The key tool supporting the distributed architecture is Python Celery [4] which is an asynchronous task queue based on distributed message passing. Node JS [13] and Swagger Codegen [19] have also been used to build REST services exposing the file system and celery workers. Further explanation on tools and technologies used for building different architectures on different platforms is also given.

### 2.1 Python Celery

Celery is an asynchronous task queue which is based upon distributed message passing and uses RabbitMQ or Redis as the communication system. The smallest unit of execution in Python Celery is a task, which can be used to execute either long running or quick tasks. Celery also provides the flexibility to execute tasks synchronously and asynchronously [4]. Celery can be understood as a tool that encompasses many communication systems, abstractions, scheduling and real time operation handling capabilities. Celery is a easy to use, highly configurable, flexible and fast tool that can be used to handle a very large amount of tasks of varying nature [21].

### 2.2 Node JS

Node JS is a widely used platform supporting server side JavaScript execution. Node JS greatly simplifies the development and maintenance of web applications and has an event driven architecture. This architecture design provides high scalability, high throughput and support for real time applications [23]. Node JS is most popular to create single page web and IOT applications, APIs and mobile backends [8].

## 2.3 Swagger Codegen

Swagger Codegen is an open source tool that provides ease of development for REST API developers by means of creating server stubs and client SDKs using the given OPENAPI specification. It can be categorized as a automatic API code generation tool. This allows developers to focus more on the API specification and not the service generation. Swagger Codegen also has the capability to generate client SDKs in many different languages including Python [19]. This is a key point to use Swagger Codegen generated REST services to expose the tasks of the Python Celery workers.

## 2.4 Hadoop

Apache Hadoop is an open source platform that supports processing very large data sets. The complete Hadoop eco system is targeted towards distributing the Big Data in the HDFS and then deploying the application or solver in each of the nodes in the cluster. Hadoop targets to achieve batch processing with the use of the MapReduce programming model and the HDFS [9]. The main goal of Hadoop is to provide scalability, fault tolerance, flexibility and performance for batch processing applications. Due to the support for structured and unstructured data Hadoop allows organizations to quickly leverage large datasets and perform analysis on them [10].

## 2.5 Spark

Apache Spark is an in-memory data processing engine that is based on a resilient distributed dataset (RDD) architecture. Spark requires cluster management and a distributed file system in order to function. For a distributed architecture, Spark is capable of interfacing with HDFS and Hadoop Yarn [22]. Spark contains a distributed core engine which is responsible for execution of tasks and libraries which provide many different user APIs from different languages and streaming support. Spark also has a machine learning library called 'MLlib' with a wide variety of machine learning algorithms used in the data science domain [11].

## 2.6 Amazon EC2

Amazons' Elastic Compute Cloud (EC2) is the most famous Infrastructure as a Service now. Amazon EC2 provides users the pay only for the computing resources they consume. This provides great flexibility and is economically attractive to the user. With autoscaling, EC2 also provides reliability for the user applications. Amongst other benefits are high end security and full integration for other Amazon services such as Amazon EFS [1].

## 2.7 Amazon EFS

Amazons' Elastic File System (EFS) is a scalable file system storage provided by Amazon. This easily integrates with existing Amazon EC2 instances and can be purchased only as much as it is used. EFS supports high availability and scalability in its shared file system. The main attraction points for Amazon EFS is its simplicity and integration with either on premise or cloud instances. There are several uses cases such as web service hosting, content management and Big Data analytics for Amazon EFS [3].

## 2.8 Docker Compose

Docker Compose is a tool that is used to create multi-container Docker applications. Docker compose provides the capability to define and run multiple Docker containers with ease. It also provides easy integration of volumes and networking between multiple docker containers. With YAML as the configuration file to define the multiple services Docker Compose only focuses on the build, start and stop of the multiple containers at the same time. Individual operations on each Docker image or container is irrelevant to Docker Compose and it only focuses on the orchestration of the environment as a whole [7].

## 2.9 Versions of the Technologies Used

Table 1 displays the versions used for each technology when creating the Reference Architecture. Please note that you have to use the Apache Spark version which is built using the Hadoop version in your Hadoop deployment. Also make sure to use the pyspark, py4j versions which are compatible with your Apache Spark version.

**Table 1: Versions of Technologies Used**

| Technology | Version |
|---|---|
| Python | 3.6 |
| Redis | 2.10.6 |
| Celery | 4.1.0 |
| hdfs (HDFS CLI) | 2.1.0 |
| py4j | 0.10.6 |
| numpy | 1.14.2 |
| scipy | 0.18.1 |
| pandas | 0.19.2 |
| pyspark | 2.3.0 |
| java | Oracle java 8 |
| Apache Spark | 2.3.0 |
| Hadoop Version used in Apache Spark build | 2.7 |
| Hadoop | 2.7 |
| Swagger codegen | 2.3.1 |
| Node JS | 9.11 |
| Docker | CE 18.03 |
| Docker Compose | 1.8.0 |

# 3 PROJECT PROCEDURE

This section focuses on providing guidance for building the distributed reference architecture for big data problems. The first subsection which is on building the environment is further divided into three sections each explaining the specific environments used, procedures and problems faced in that specific environment. The second subsection focuses on defining the format of the data that needs to be given to the service.

## 3.1 Building the Environment

This subsection focuses on describing the architecture and functionality of the system. The architecture is built on three different types of environments; local, Amazon EC2 (cloud) and multi-container

Docker environments. Further explanation on each of the environments will be given below.

*3.1.1 Local Environment.* Figure 1 shows the Big Data reference architecture built in the local environment having the following hardware specification.

**Sony Vaio Laptop** 4 hardware cores, 2 threads per core, Intel Core i7-2670QM processor (up to 2.2 GHz) with 8 GB DDR4 RAM, 500 GB SSD hard drive and Nvidia GeForce GT 540M 1GB VGA.

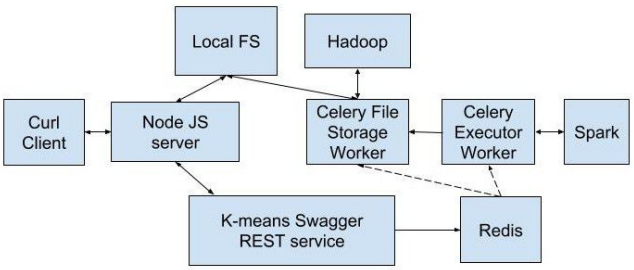


**Figure 1: Local Architecture Environment**

Following is the brief overview of the functionality and architecture of the system. The Node JS server will be used to interact with the local file system which will be used to store the input data, model and prediction data. Following are the APIs exposed by the Node JS server.

**Save input file** Given the input data file in a specific format which will be discussed in 3.2 save the file in the local file system.

**Execute KMeans** Execute K-means for the given input data set.

**Get Kmeans Model** After executing K-means retrieve the model or cluster centers.

**Get Predictions** After executing K-means retrieve the predictions for each data point.

A curl client can be used to invoke the functionalities which are exposed by the Node JS server. Next the Celery file storage and executor workers are both listening to the Redis Server to which the Kmeans Swagger REST service publishes messages after it is being invoked by the Node JS server. Both of the workers each listen to a specific queue and once a request (for a specific task) arrives in their queues they are listening to they execute the tasks. The file storage worker is responsible for moving the files to and from HDFS and the local file system. The executor worker is responsible for executing K-means using Spark.

*3.1.2 Distributed Cloud Environment.* Figure 2 shows the Big Data reference architecture built in the Amazon EC2 cluster with the use of the following different EC2 node types.

**t2.small** 2 nodes for Node JS server and Hadoop installation

**t2.medium** 1 node for the installation of K-means Swagger REST service, celery workers, redis and spark installations

The hardware specification for each of the instance types is shown below [2].

**t2.micro node** 1 vCPU, Intel Xeon Family Processor (up to 3.3 GHz) with 1 GB RAM, Intel AVX and Intel Turbo support

**t2.medium node** 2 vCPU, Intel Xeon Family Processor (up to 3.3 GHz) with 4 GB RAM, Intel AVX and Intel Turbo support

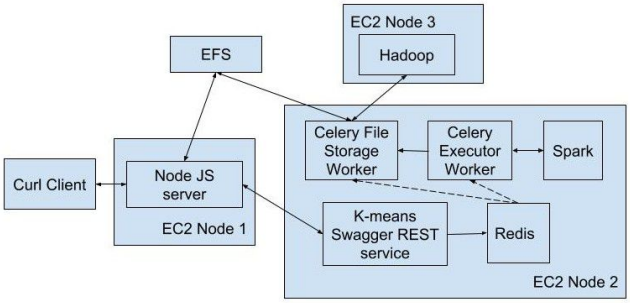


**Figure 2: Distributed Architecture in EC2 Cluster**

Figures 3 to 5 show how the communication between each of the components in the distributed architecture happens. It should be noted that requests are represented in black arrows and responses are displayed in red dashed arrows. The black dashed arrows represent that the Celery workers listen to the Redis servers. It should also be noted that the communication between the Node JS server and the K-means Swagger REST service and the communication between the K-means Swagger REST service and Celery Workers are asynchronous.

*Save Input Data File.* Figure 3 shows the order of request and response flow within the distributed architecture. When the request to save the input file is initiated, as the first step the Node JS server saves the file in the Elastic File System. Next the Node JS server sends a POST request to the K-means Swagger REST service to invoke the message passing to the Redis Server. Finally the Celery worker which listens to the 'file_storage' queue retrieves the file from the EFS and saves it in the HDFS node.

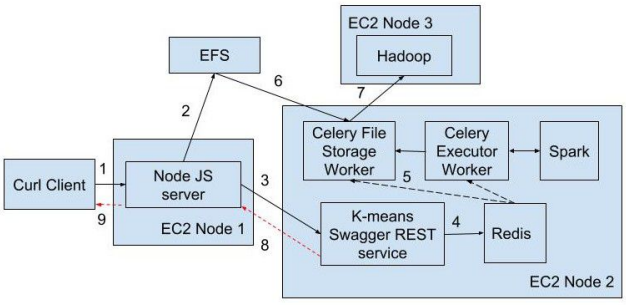


**Figure 3: Upload Input File Request/Response Flow in Distributed Architecture**

*Execute K-means Function.* Figure 4 shows the order of request and response flow within the distributed architecture for executing the K-means algorithm. When the request to execute K-means is initiated from the Curl client the request to execute the task is sent to the Redis Server. The Celery executor worker who listens to the 'executor' queue then passes a message to the file storage worker to retrieve the file from HDFS and executes K-means algorithm on the input data set. The model and prediction files are then written to the HDFS through the Celery file storage worker.

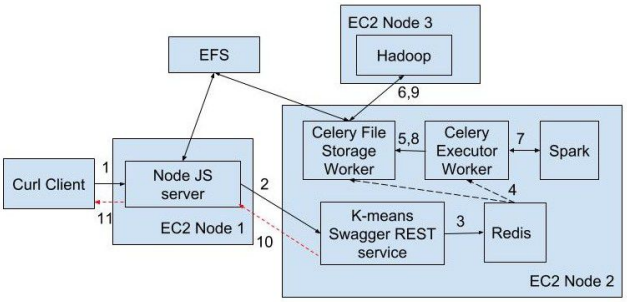


**Figure 4: Execute K-means Function Request/Response Flow in Distributed Architecture**

*Retrieve Model File.* Figure 5 shows the order of request and response flow within the distributed architecture for retrieving the K-means model. Once the request to retrieve the model file is initiated by the Curl client the message is passed through to the Redis Server to which the file storage worker listens. Then the file storage worker retrieves the file from HDFS and saves it in the elastic file system, which is given as the response to the client. The same flow can be seen when retrieving the predictions for the input data.

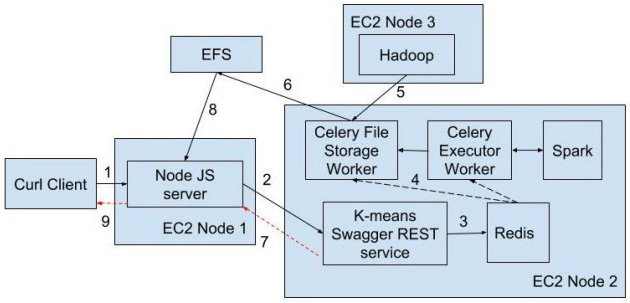


**Figure 5: Retrieve Model File Request/Response Flow in Distributed Architecture**

*3.1.3 Multi Docker Container Environment.* Figure 6 shows the components of the Big Data reference architecture in multiple docker containers. Each of the Hadoop, Node JS server, K-means Swagger REST service, Redis server, Celery file storage worker are in single containers while the Celery executor worker and Spark installation are in a single container. With the help of Docker Compose we can easily define the network through which each of the containers will communicate with each other and easily mount the same file system for the file storage worker and Node JS server.

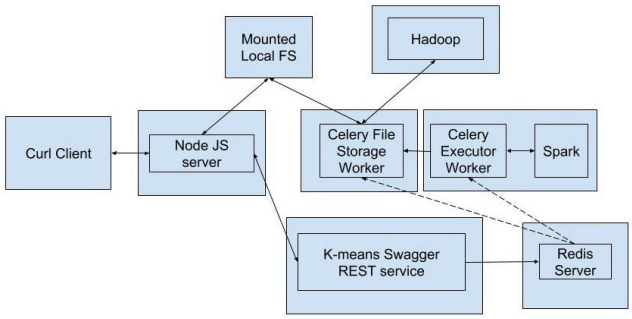


**Figure 6: Multi-Container Docker Architecture**

## 3.2 Data Format

K-means can only be executed on numeric data. Therefore a predefined format in which the data should be given to the system is described here. Table 2 shows the format of the data for the first three rows. The first column should contain the id by which we can identify the row. Consequently the other columns should contain the numeric feature data, make sure to only enter numeric data as the values will be cast to a 'double'. The column names should be given as the first row of the dataset. This input data file may contain 'na' values as the Celery executor worker performs removal of 'na' values from the dataset. Please note that the column names should only contain '_' as a separator.

**Table 2: Iris Data Set Representation Example [16]**

| id | sepal_length | sepal_width | petal_length | petal_width |
|---|---|---|---|---|
| row0 | 5.1 | 3.5 | 1.4 | 0.2 |
| row1 | 4.9 | 3 | 1.4 | 0.2 |
| row2 | 4.7 | 3.2 | 1.3 | 0.2 |

Figure 7 shows the classification for each of the flower types 'Iris Setosa', 'Iris Versicolor' and 'Iris Virginica' against the Sepal_Length and Sepal_Width.

## 3.3 Expected Results

The user can execute the K-means function with input data in the format described in Section 3.2. The following files will be generated as output from the system. The user has the freedom to provide the names of the model and predictions files which will be used to saved the files in HDFS. This approach allows ease when executing K-means for the same input data file for different number of clusters.

**Model File** containing the cluster centers will be saved in HDFS.

**Predictions File** for the input data is generated such that each input data point is assigned to the cluster to which it is nearest is also saved in HDFS.

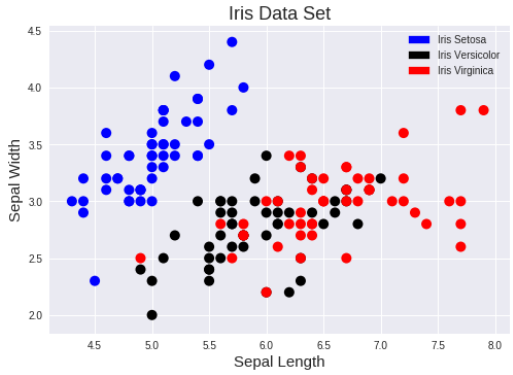


**Figure 7: Iris Input Data Set Classification**

Figure 8 shows the predictions obtained after executing the K-means function. When comparing the predictions obtained from the system after executing K-means and the original classifications it can be seen that the system produces accurate results.

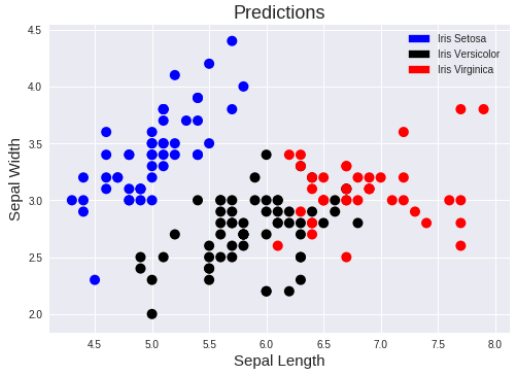


**Figure 8: Predictions for Iris Data Set**

## 4 RESULTS

This section presents the time breakdown results obtained when executing the function in local and cloud environments. The specification of the local and cloud environments can be found in Section 3.1.1 and 3.1.2.

### 4.1 Time Breakdown According to Task

Table 3 shows the time breakdown for executing the following tasks in the Big Data architecture.

- Upload Input File
- Execute K-means
- Retrieve Model File
- Retrieve Prediction File

In each task the time taken for processing by each worker is also provided. The time in Table 3 is provided for a dataset having a size 100MB [15]. The time breakdown provides insight about where most of the time is used and how much of the time is actually used for processing in the Celery workers. Uploading the input file consumes a significant amount of time compared to the retrieval of the model and prediction files in the distributed cloud environment. In terms of processing the K-means execution takes a significant amount of time compared to the file transfer processing regardless of the environment. The overhead in total time for the file transfer functions is due to having to move large files from client to cloud file system to HDFS nodes and vice versa and dependent on the file size and network.

**Table 3: Task Time Breakdown**

| Task | Time in Local Environment (s) | Time in EC2 Environment (s) |
|---|---|---|
| Upload Input File (Total) | 1.35 | 75 |
| Upload Input File (Request Reaching) | 0.95 | 74.05 |
| Upload Input File (Worker Processing) | 0.4 | 0.95 |
| Executing K-means (Total) | 12.56 | 11.44 |
| Executing K-means (Worker Processing) | 11.42 | 10.69 |
| Retrieve Model (Total) | 2.78 | 6.58 |
| Retrieve Model (Response Reaching) | 2.756 | 6.502 |
| Retrieve Model (Worker Processing) | 0.024 | 0.078 |
| Retrieve Predictions (Total) | 3.39 | 6.46 |
| Retrieve Predictions (Response Reaching) | 3.36 | 6.313 |
| Retrieve Predictions (Worker Processing) | 0.03 | 0.147 |

### 4.2 Execution Time Analysis Based on Number of Clusters

Table 4 shows the time breakdown for the K-means algorithm execution for different number of clusters. Users might have to execute K-means algorithm for a different number of clusters until the optimal number of clusters is found. The API for the Node JS server provides a flexibility such that users are able to execute K-means for the same input data set for different number of clusters (by providing different names for the model and prediction files).

It can be seen that regardless of the environment type and the number of clusters the K-means algorithm execution takes a constant amount of time of 11 seconds for the given input data set. Therefore, it can be stated that for a given input data set K-means execution takes a constant time in the 'executor' Celery Worker.

**Table 4: Execution Time Based on Number of Clusters**

| Number of Clusters | Execution Time in Local Environment (s) | Execution Time in EC2 Environment (s) |
|---|---|---|
| 10 | 12.56 | 11.44 |
| 20 | 11.87 | 11.55 |
| 30 | 11.22 | 10.77 |
| 40 | 11.28 | 11.17 |
| 50 | 11.89 | 10.90 |
| 100 | 11.56 | 10.68 |
| 125 | 12.07 | 10.59 |
| 150 | 12.51 | 11.44 |

### 4.3 Upload Time Analysis for Different Data Sizes

The input file upload time is highly dependent on the size of the data set. Therefore Table 5 shows the increase in time taken for the file to get uploaded to the cloud with the increase in data size. Network speed could be a factor that result in the increased time in upload as well. In the local environment a constant time for uploading the files can be seen as it is simply a transfer of files between the same file system.

**Table 5: Upload Time for Different Data Sizes**

| Data Size (MB) | Upload Time in Local Environment (s) | Upload Time in EC2 Environment (s) |
|---|---|---|
| 23 | 0.9 | 34 |
| 60.2 | 0.87 | 47 |
| 93 | 1.35 | 75 |
| 133.6 | 1.30 | 161 |
| 225.9 | 1.65 | 357 |

## 5 LIMITATIONS AND IMPROVEMENT SUGGESTIONS

Further analysis on the network requirements and alternatives to uploading larger input files to an FTP server can be investigated. Extensions to this project can be made by adding new Celery Workers with tasks or by adding the new tasks to existing Celery workers based on relevance. The newly added tasks need to be exposed via a REST service which can be invoked directly or via the Node JS server. Further improvement on the architecture can also be made to include an util worker or module and a MongoDB database to keep track of the progress of long running Celery tasks.

## 6 CONCLUSION

This project aims in building a Big Data reference architecture using Python Celery which is based on distributed message passing and REST services using Node JS and Swagger Codegen. A wide variety of tools have been used when building the reference architecture, in which most of them facilitate building the distributed nature for the architecture. The Celery workers are the key components of the system which execute tasks by listening to task requests which arrive in specific queues. The Celery workers provide modularization to the distributed environment. This project focuses on building the architecture for executing the K-means function and from the task breakdown analysis it is evident that when working with the Amazon Cloud environment most of the time is consumed for file transfer through the network. Due to connection time out issues with the EC2 environment it is also evident that further analysis into the network and minimizing of the transfer overhead is needed to make the architecture more robust. It is evident that the Python Celery workers takes minimal processing time for its tasks, therefore the architecture can be improved further by considering alternative components keeping the celery workers and the Swagger REST service component intact.

## ACKNOWLEDGMENTS

The authors would like to thank Dr. Gregor von Laszewski for his support and suggestions to write this paper.

## REFERENCES

[1] Amazon. 2018. Amazon EC2. Web Page. (2018). https://aws.amazon.com/ec2/ Accessed: 2018-04-19.
[2] Amazon. 2018. Amazon EC2 Instance Types. Web Page. (2018). https://aws.amazon.com/ec2/instance-types/ Accessed: 2018-04-19.
[3] Amazon. 2018. Amazon EFS. Web Page. (2018). https://aws.amazon.com/efs/ Accessed: 2018-04-19.
[4] Python Celery. 2018. Celery: Distributed Task Queue. Web Page. (2018). http://www.celeryproject.org/ Accessed: 2018-04-01.
[5] Docker. 2018. Docker. Web Page. (2018). https://www.docker.com/ Accessed: 2018-04-19.
[6] Docker. 2018. Docker Compose. Web Page. (2018). https://docs.docker.com/compose/ Accessed: 2018-04-19.
[7] Docker. 2018. Docker Compose. Blog. (2018). https://blog.docker.com/2016/02/docker-compose-networking/ Accessed: 2018-04-19.
[8] Node.js Foundation. 2018. How Companies Benefit from Node.js and the Node.js Foundation. Blog. (2018). https://medium.com/@nodejs/how-companies-benefit-from-node-js-and-the-node-js-foundation-d9a0dc0ce442 Accessed: 2018-04-19.
[9] Apache Hadoop. 2018. Hadoop. Web Page. (2018). http://hadoop.apache.org/ Accessed: 2018-04-19.
[10] HORTONWORKS. 2018. Apache Hadoop. Blog. (2018). https://hortonworks.com/apache/hadoop/ Accessed: 2018-04-19.
[11] HORTONWORKS. 2018. Apache Spark. Blog. (2018). https://hortonworks.com/apache/spark/ Accessed: 2018-04-19.
[12] IBM. 2018. HDFS. Web Page. (2018). https://www.ibm.com/analytics/hadoop/hdfs Accessed: 2018-04-19.
[13] Node JS. 2018. Node JS. Web Page. (2018). https://nodejs.org/en/ Accessed: 2018-04-19.
[14] Redis. 2018. Redis. Web Page. (2018). https://redis.io/ Accessed: 2018-04-01.
[15] UCI Machine Learning Repository. 2018. detection of IoT botnet attacks N BaIoT Data Set. Web Page. (2018). https://archive.ics.uci.edu/ml/datasets/detection_of_IoT_botnet_attacks_N_BaIoT Accessed: 2018-04-19.
[16] UCI Machine Learning Repository. 2018. Iris Data Set. Web Page. (2018). https://archive.ics.uci.edu/ml/datasets/iris Accessed: 2018-04-19.
[17] UCI Machine Learning Repository. 2018. UCI Machine Learning Repository. Web Page. (2018). http://archive.ics.uci.edu/ml/index.php Accessed: 2018-04-01.
[18] Apache Spark. 2018. Apache Spark. Web Page. (2018). https://spark.apache.org/ Accessed: 2018-04-01.
[19] SWAGGER. 2018. SWAGGER CODEGEN. Web Page. (2018). https://swagger.io/swagger-codegen/ Accessed: 2018-04-19.
[20] Swagger. 2018. THE WORLD'S MOST POPULAR API TOOLING. Web Page. (2018). https://swagger.io/ Accessed: 2018-04-01.
[21] VINTA. 2018. Celery: an overview of the architecture and how it works. Blog. (2018). https://www.vinta.com.br/blog/2017/celery-overview-archtecture-and-how-it-works/ Accessed: 2018-04-19.

[22] Wikipedia. 2018. Apache Spark. Web Page. (2018). https://en.wikipedia.org/wiki/Apache_Spark Accessed: 2018-04-19.
[23] Wikipedia. 2018. Node JS. Web Page. (2018). https://en.wikipedia.org/wiki/Node.js Accessed: 2018-04-19.

# Financial Analysis Service

Rashmi Ray
Indiana University
107 S. Indiana Avenue
Bloomington, Indiana 43017-6221
rashray@iu.edu

## ABSTRACT

The project analyses how the Kubernetes deployment metadata affects performance of a containerized application. The experiment platform will be Google Kubernetes Engine. Kubernetes cluster will be setup in a Chameleon node to ensure isolated setup. The project will also involve developing a Flask REST API for processing stocks data in Python that will provide real-time analysis of the stock market data. The performance of the API will be compared in both GoogleCloud and local. The stock API will be using Python, pyGal, Quandl. The benchmarking process will be registered in the project report.

The project will also discuss the setup process of Google Kubernetes Engine.

## KEYWORDS

hid-sp18-417, Volume: 9, Chapter: Security, Status: 100.
Kubernetes, deployment, performance, GKE, containers, replication manager

## 1 INTRODUCTION

The area of research for this paper is Kubernetes. In today's age of containers, it is important for an organization to adopt a container orchestration and management system, that will also enable seamless integration and upgrades. Kubernetes is one of the widely popular open source platform provided for this specific purpose by technology giant Google Inc. Looking at the high popularity of the tool, it is becoming a buzz word in the world of cloud computing and containers. The project looks into the ease of deployment, configuration and performance in Google Kubernetes System. Before going into the detail of the project outcome, it is important to introduce several key components of the Kubernetes and GoogleCloud.

## 2 KUBERNETES

Kubernetes is an open-source system developed and maintained by Google Inc. The system provides a container orchestration platform for ease of deployment, management and scaling of services and resources. The system places containers automatically based on it resource requirement not compromising availability. Similarly, it provides seamless management of data volumes and storage system. It allows updating configurations of the deployment without rebuilding the image. The system can facilitate public, private or hybrid container management system based on an organization's requirement. Kubernetes resource monitoring system oversees automatic rollout and rollback of the resources in case of fallbacks.

Kubernetes is also known as K8 or kube. This is originally developed based on Google's container management system. Kubernetes can be deployed in a single host, but it is designed to be benefitted efficiently when used for a cluster of connected hosts. Use of a cluster of multiple hosts facilitates high availability.

The primary components of the system are:

A master node controls the other nodes. The master monitors all the events in the clusters and execute appropriate rules for the flawless execution of a cluster. Nodes are the other hosts in the cluster. The containerized applications are run in the nodes. Pod is a group of one or more containers deployed in a node. The Replication Controller monitors and controls the replicas of the services. A Kubernetes cluster serves as a single unit that has a collection of rules/configuration for effective deployment of containerized applications. Various metadata such as replicas, environment, track, port, can be defined in a service configuration. Kubectl is the command line configuration tool provided by Kubernetes.

The Figure 1 explains the Kubernetes architecture. The master is responsible for managing the deployments, scheduling, exposing containerized applications. In order to maintain high availability a microservice deployment can define number replicas. Each of the replicas run in a separate pod. The set of a replicas for a deployment is called a replica- set. The replication manager monitors the replicas for fall-back handling.

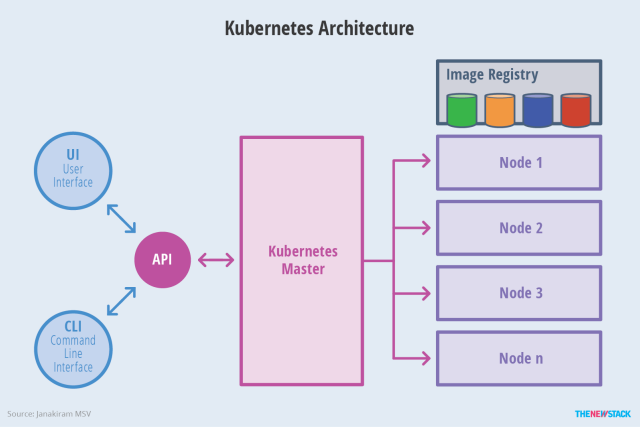


**Figure 1: Kubernetes Architecture [3]**

## 3 GOOGLE KUBERNETES ENGINE

Google Kubernetes Engine[GKE] is the tool developed by Google Inc to simplify the management and orchestration of Kubernetes systems, in Google's cloud services. This will help an organization focus more into their own product development than worrying about Kubernetes networking, upgrades and maintenance. As the

main focus of the paper was Kubernetes, it was vital for the author to choose GKE as the primary focus of the project.

### 3.1 Benefits

Here are some benefits: With use of GKE, user need not worry about Kubernetes master. The system ensures that master is always up and running. User need not worry about underlying networking on what to user e.g. weave, flannel etc. It makes access and identity management easier with Google's Identity and Container Management System[IAM] [2]. Auto scaling is easier with just simple commands. With Kubernetes' frequent releases it is important for the system to stay updated always. The upgrade is easier with gcloud with just a single command rather than the manual overhead of porting the system step by step.

### 3.2 Initial Setup

The project leveraged the free tier offer from Google to bring in the setup and configuration experience for GKE. As of Apr 2018, Google is offering a 300 dollar credit to be used with in 12 months. The free trial has certain resource usage limitations applied. Please refer to Google's documentation for details. But the bottom-line was that the limitation was well within the scope of the project to explore a cluster's performance. So the author decided to go ahead with the research.

Here are the key steps involved to get you started with GKE:

**Account Setup** Before you can use GKE, you need to register to google cloud. You can use an existing google account. It is required to provide credit card information. Google assures that the card won't be charged without your permission.

**Shell Setup** GKE provides an option of using google cloud shell [from the browser - no installation needed] or local shell [gcloud installation]. The project kept the local setup in the scope. To set up the environment a user must follow the steps listed in Figure 5. It includes the import of the public key, and installing Google SDK. Aletrnatively, one can use the Makefile provided in the project repo to execute the same.

**Install Kubernetes** Now install Kubernetes:

```
sudo apt-get install kubectl
```

**Initiate GoogleCloud** Now GKE is ready to be initiated locally. Before initiating it is important to know that you will setup the project and compute zone during the setup. You can setup the compute zone later but it is advised to set your preference at the beginning to ensure that your requests are processed in the nearest processing center. [e.g. central-zone-1]

Now ready to initiate gcloud:

```
gcloud init
```

**Create Cluster** A cluster consists of atleast one master multiple worker machines. gcloud creates a cluster of three nodes by default. This takes arounf two minutes. Once done it is important to get the credential, so that containers can be deployed in the cluster. During the project this process consistently recorded to be more than two minutes [7].

```
gcloud container clusters create [CLUSTER_NAME]
```

**Deployment** A containerized docker image can be deployed to the cluster using the following:

```
kubectl run [SERVICE_NAME] --image [IMAGE_NAME]
        --port [port number]
```

**Exposing Service** Exposing a service to a port will enable external access: [A loadbalancer exposes the service externally.]

```
kubectl expose deployment [SERVICE_NAME]
      --type 'LoadBalancer'
```

Once the command is run it may take around a minute for the service to get exposed and to avail the external IP. The external IP can be fetched using the following:

```
kubectl get service [SERVICE_NAME]
```

**External Access** Once the IP is shown, the exposed contained can be accessed in any web browser.

```
http://[EXTERNAL_IP]:[EXPORTED_PORT]
```

**Cleanup** Finally, here are some useful clean up commands:

```
kubectl delete pod [POD_NAME]
kubectl delete service [SEVICE_NAME]
kubectl delete deployment [DEPLOYMENT_NAME]
gcloud container clusters delete [CLUSTER_NAME]
```

Please keep in mind that if a pod is generated through a deployment then to remove it the deployment has to be deleted. Deleting the pod will regenerate the pod because the replication manager is monitoring the failure and handling them. A Makefile is provided in the project-code repo for the gcloud setup.

This Figure 3 shows the GKE dashboard in the current state. Some of the vital sections are useful to discussed. The current balance is for the account is shown shown in the top-left corner. In the left menu option will provide the current state of the account.Identify the active projects through the project name and ID in the top left block in the content area. The resources block displays the details of the resources used in the project. The middle column displays current state of the compute engine and request/response flow for the containerized APIs. The Google Browser Shell can be used as an alternative to the local shell.

## 4 API IN USE

Before the project is discussed, its important to discuss the following tools and technologies used in the project. The following subsections also discuss the necessary installation and code pertaining to the project.

### 4.1 Python - API

Python is a high-level object-oriented programming language. The language is popular because of its simple readable syntax, minimal required setup and high availability of large collection of free libraries. The project uses Python 2.7 that is available in the native version of Ubuntu 16.04.

```
# Environment variable setup
export CLOUD_SDK_REPO='cloud-sdk-$(lsb_release -c -s)'

# Add the Cloud SDK distribution URI as a package source
echo 'deb http://packages.cloud.google.com/apt $CLOUD_SDK_REPO main' | sudo tee -a /etc/apt/sources.list.d/google-cloud-sdk.list

# Import the Google Cloud Platform public key
curl https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -

# Update the package list and install the Cloud SDK
sudo apt-get update && sudo apt-get install google-cloud-sdk
```

**Figure 2: Setup of gcloud SDK**

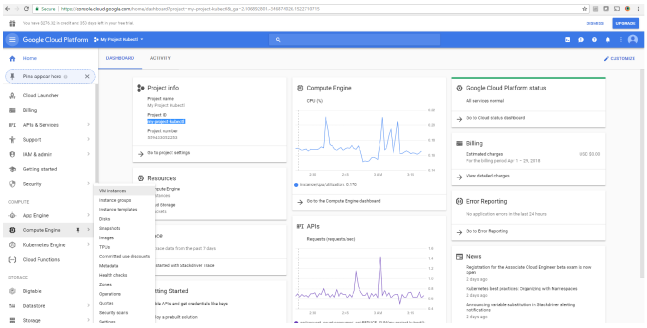

**Figure 3: gcloud Dashboard**

## 4.2 Flask - Rest Service

Flask is a micro framework developed for web-based applicatios. The framework is widely popular because of its light weight and has minimal dependencies. Flask was originally written in Python. Some of the key features of Flask are integrated rest service, secure cookies and unit testing support. The framework is also compatible with Google App Engine [1]

Flask-RESTful is an extension of Flask that supports quick and easy API deployments with minimal setup. In order to use the rest service in Python the Flask module needs to imported. As this project uses Python 2.7, the Flask compatible version used is 0.10.1 . Here is the basic minimun information needed to get started with the REST service.

main.py code:

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def init_func():
      code here
```

Just install Flask for the installed version of Python.

```
sudo apt-get install python-flask
```

Here is how to execute the code:

```
python main.py
```

Here is the sample console output when the Flask server spins up The DockerFile is provided in the prihectrepo to containerize the application.

```
* Running on http://localhost:5000/
```

A point worth noting here is that Flask server spins up at 5000 port by default. While exploring deploying the web service on gcloud, the following command was found crucial.

```
if __name__ == '__main__':
  app.run(host = '0.0.0.0', debug = False,
          port=int(os.getenv('PORT', '5000')))
```

## 4.3 Quandl - Realtime Data provider

Quandl [5] is a useful Python module that brings in millions of datasets pertaining economics and finance data.

Installation:

```
easy_install quandl
```

The data is available free, but Quandl requires you to register with them in order to avail critical data. There are still some data available in public that doesnfit need any registration code. The code can be formatted to receive data is one of the available formats i.e. json, excel, csv. To retrieve data from quandl two vital parameters are needed: source code and the ticker code.

The project is using 'WIKI/PRICES' source and ticker is expected to be a trading code for a desired organization.

```
completeData = qd.get_table('WIKI/PRICES', ticker =
 [companycode])
```

In the project ticker is set to be query param for the flask REST api.

```
http://localhost:5000/home?code={code}
```

## 4.4 Pygal - Graph Module

Pygal [4] is a Python module for interactive plotting. There other Python modules are available in Python, but Pygal specializes in SVG [11] graphics. The graph considers the last 2000 days record fpr the selected company to analyse its stocks high, low and close value.

Installation:

```
easy_install pygal
```

These graphics being scalable and light weight, they are preferred for websites. The project chose Pygal for the very specific reason. It collects real-time data taking in the user entered code in the query prams and renders the specific a line chart using the respective data collected from Quadl.

```
completeData = qd.get_table('WIKI/PRICES', ticker =
[companycode])line_chart = pg.Line()
line_chart.title = 'Stocks Analysis for ' + companycode
 line_chart.add('high', mydata['high'])
```

```
line_chart.add('low',   mydata['low'])
line_chart.add('close', mydata['close'])
chart = line_chart.render_data_uri()
```

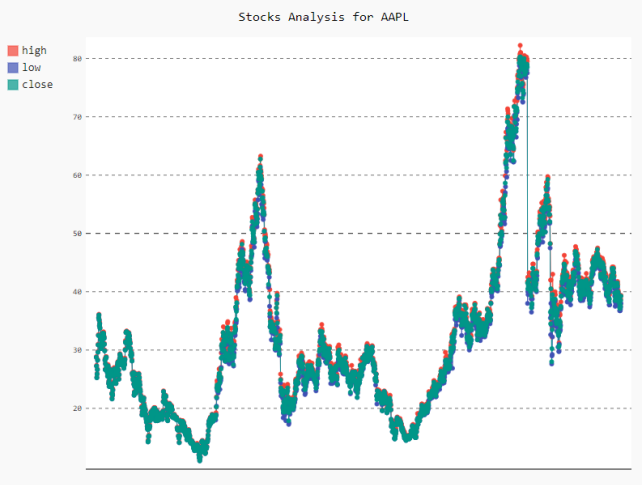


**Figure 4: Project Graph**

## 4.5 Datetime

The project uses the Python datetime module to record the data download time and processing time.

```
Data Load Time: 0:00:03.251368

Data Processing Time: 0:00:01.612638
```

## 4.6 Docker - Container

Docker is an opensource tool that enables virtualization at the Operating System level. The process is also known as containerization. Docker was primarily developed for Linux system to containerize an application and its dependencies so that the application can run in any Linux system.

Installation: For the project the stocks microservice is containerized to be deployed in Kubectl using a Dockerfile [9].

## 4.7 Docker-compose - Container Management

Docker Compose is a tool used for managing metadata for multi-container applications. Docker-compose.yml [8] was not mandatory for this project, but was incorporated keeping future improvisations in mind.

Installation:

## 4.8 Makefile

Makefile is a file that contains instructions to get certain task done. The project has provided a Makefile that takes care of building a cleaning up the Docker container. To use this file ensure build-essentials is installed.

Provisioning of Makefile makes the automation process simpler. The project repo provides two Makefiles. One for gcloud installation and setup, the other one is for the deployment of the python-flask microservice.

## 4.9 YAML - Kubernetes Configuration

It is a good practice to manage Kubernetes deployment through yml files. The step is not mandatory but desirable for ease of maintenance and Convenience. Yaml is a markup language that is popularly used for specifying configurations. If multiple structure are defined in one single file then its separated by — . Here is an extract of the yml file used for the project. Few metadata worth noting is apiversion, kind, replicas, containerPort. Figure 7 provides a sample configuration file used for the project.

## 4.10 Chameleon

Once the required knowledgebase is ready, project was set up in Chameleon cloud node to ensure that all the required installation and configurations are registered appropriately for the report.

## 4.11 Setup

Once the Python API was developed it was tested to retrieve the times recorded. Apple Inc was used for this project. This is because, it has a good number of records. The Flask server was run locally to check the download and processing time. The Figure 8 depicts the final webpage.

Now, a Chameleon node was spawn to setup Google Kubernetes Engine. The google SDK and CLI was installed and configured.

The Docker image was created using the Makefile:

```
make docker-build
```

A kubectl cluster was created and it was ensured that the nodes are up and running.

kubectl get nodes

The web service was deployed.

```
docker-compose up -d

kubectl apply -f kubernetes/flask-web-ui.yml
```

Once deployed, it takes around a minute for the external IP to be available. With the available external IP, User can access the deployed microservice from anywhere. The default flask-web-ui.yml in the git repo defines 3 replicas.

A short video was recorded to present the overview of the project [10].

Finally once all the required data was collected the deployment and cluster was cleaned up.

A demonstration of the replication manager and the cleanup process is recorded [6] during the project.

## 4.12 Performance Analysis

Now that Good Kubernetes Engine is explored for ease of deployment and management, it was important to log an analysis. Timestamp was added in the Makefile to check on the cluster creation time. It was found that both cluster creation and deletion take on an average 2 minutes. Similar approach was taken on checking the time taken to expose a web-service to a specific port. It was concluded that it takes on an average one minute to avail the external IP.

```
sudo apt-get update
sudo apt-get install apt-transport-https ca-certificates curl software-properties-common
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
sudo apt-key fingerprint 0EBFCD88
sudo add-apt-repository 'deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs)  stable'
sudo apt-get update
sudo apt-get install docker-ce
sudo apt-get update
```

**Figure 5: Docker Installation**

```
sudo curl -L https://github.com/docker/compose/releases/download/1.18.0/docker-compose-`uname -s`-`uname -m` -o /usr/local/bi
sudo chmod +x /usr/local/bin/docker-compose
```

**Figure 6: Docker-compose Installation**

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: web-ui
spec:
  replicas: 3
  template:
    metadata:
      labels:
        role: web-ui
    spec:
      containers:
      - name: web-ui
        image: rashray/stocks
        ports:
        - containerPort: 5000
```

**Figure 7: Deployment Configuration**

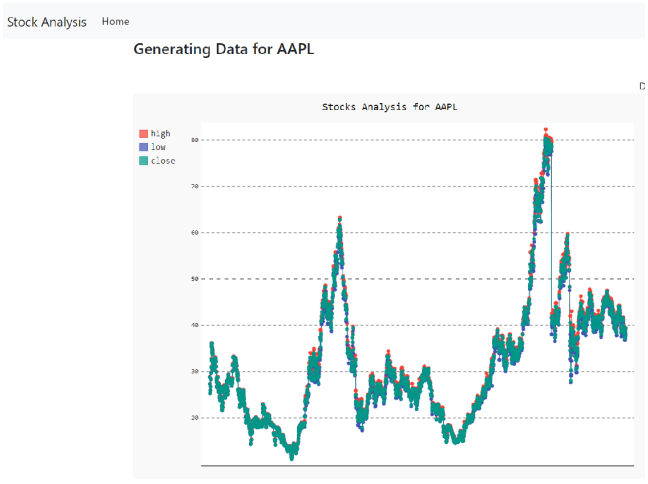


**Figure 8: Localhost - REST service**

Depending of the web traffic estimated the number of replication of webservice is defined so that the fall backs can appropriately addressed. Though Kubernetes replication manager monitors the resources and pods to ensure that replications mentioned for a deployment is maintained, it take few seconds for the restoration to take place in case of fallbacks. The presence of replications balances the loads and manages fall-backs. Behind the scenes all the user requests are directed to the master, which in turn directs traffic appropriately to ensure even distribution. As a part of the benchmarking process, it was analyzed if the number of service replicas affect performance.

The Python-Flask microservice developed for the stock analysis has two components one is fetching the stocks data from WIKI and the other is to process the data in order to render the pygal graph. Python script was added to log both the download and processing time. Now this timing was compared to check if a gcloud deployment replicas affect the processing time. The times were logged initially by running the service locally and then later in the gcloud cluster. The replicas were changed by changing the respective metdata in kubectl yml file in API codebase. This is how processing time was logged for: local-flask server run, containerized microservice with 3, 2 and 1 replica in a gcloud kubernetes cluster. Time in case was recorded five times and the mean of each set was taken in consideration.

The Figure 9 depicts the confiuration change to deploy the microservice with single replica.

Here is the outcome of the excercise in Figure 1:

**Table 1: Downaload and Processing time in different platforms**

| Environment | Loading Time | Processing time |
|---|---|---|
| NA | sec: MS | sec: MS |
| Local hosting | 4:365 | 2.173 |
| GKE deployment: 3 replicas | 3:834 | 1:525 |
| GKE deployment: 2 replicas | 3:246 | 1:671 |
| GKE deployment: 1 replicas | 3:315 | 1:730 |

A visual reprsentation is also provided in Figure 10 With above data the improvement in performance is considerable in the containerized Kubernetes cluster.

## 5 CONCLUSION

In the presented work, the developed Flask-Python microservice was successfully deployed in a GKE cluster. The setup and configuration process for Google Kubernetes cluster was explained. It was

```
  cc@kubenode:~/hid-sp18-417/project-code/API/flask$ cat kubernetes/flask-web-ui.yml
---
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: web-ui
spec:
  replicas: 1
  template:
    metadata:
      labels:
        role: web-ui
    spec:
      containers:
      - name: web-ui
        image: rashray/stocks
        ports:
        - containerPort: 5000

---
apiVersion: v1
kind: Service
metadata:
  name: web-ui
  labels:
    role: web-ui-svc
spec:
  ports:
  - port: 5000
    targetPort: 5000
  selector:
    role: web-ui
  type: LoadBalancer
---
cc@kubenode:~/hid-sp18-417/project-code/API/flask$ docker-compose up -d
Creating flask_web-ui_1 ... done
cc@kubenode:~/hid-sp18-417/project-code/API/flask$ kubectl apply -f kubernetes/flask-web-ui.yml
deployment.extensions "web-ui" created
service "web-ui" created
cc@kubenode:~/hid-sp18-417/project-code/API/flask$ kubectl get services
NAME         TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
kubernetes   ClusterIP      10.35.240.1     <none>        443/TCP          26m
web-ui       LoadBalancer   10.35.247.125   <pending>     5000:32594/TCP   44s
cc@kubenode:~/hid-sp18-417/project-code/API/flask$ kubectl get services
NAME         TYPE           CLUSTER-IP      EXTERNAL-IP     PORT(S)          AGE
kubernetes   ClusterIP      10.35.240.1     <none>          443/TCP          26m
web-ui       LoadBalancer   10.35.247.125   35.232.229.66   5000:32594/TCP   50s
cc@kubenode:~/hid-sp18-417/project-code/API/flask$ kubectl get pods
NAME                      READY     STATUS    RESTARTS   AGE
web-ui-78496b97dc-mgh2w   1/1       Running   0          13m
cc@kubenode:~/hid-sp18-417/project-code/API/flask$
```

**Figure 9: gcloud Deployment - 2 replicas**

felt that the time duration for creating and deleting the cluster can be improvised.

It was demonstrated how a replication manager automatically assure that the deployment configurations are managed in case of fallback. In one of the videos demonstrated how the replication manager monitors and handles fallbacks.

It was also analyzed if the number of replications does affect the performance of a containerized application. It was found that there is considerable improvement when locally run service is containerized and deployed in Kubernetes cluster, but the number of replicas does not affect the performance much. The automatic monitoring and management of the Kubernetes master makes the system very efficient.

## ACKNOWLEDGMENTS

The author would like to thank Dr. Gregor von Laszewski for his support and suggestions in writing this paper.

## REFERENCES

[1] Google Inc. 2018. GAE. (2018). https://cloud.google.com/appengine/
[2] Google Inc. 2018. IAM. (2018). https://cloud.google.com/kubernetes-engine/docs/how-to/iam-integration
[3] Janakiram MSV. 2016. Kubernetes Architecture. (2016). https://thenewstack.io/kubernetes-an-overview/
[4] Pygal. 2018. Pygal. (2018). http://pygal.org/en/stable/
[5] Quandl. 2018. Quandl. (2018). https://www.quandl.com/tools/python
[6] Rashmi Ray. 2018. gcloud Cluster Cleanup. (Apr 2018). https://youtu.be/fiuwjaxBqLc
[7] Rashmi Ray. 2018. gcloud Cluster Create. (Apr 2018). https://youtu.be/FBGw3F2sjjI
[8] Rashmi Ray. 2018. Project Docker Compose. (Apr 2018). https://github.com/cloudmesh-community/hid-sp18-417/blob/master/project-code/API/flask/docker-compose.yml

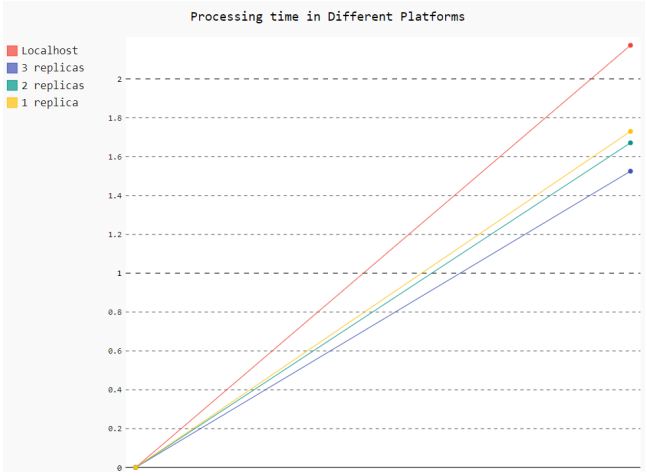


**Figure 10: Graph - Benchmarking Performance Data**

[9] Rashmi Ray. 2018. Project Dockerfile. (Apr 2018). https://github.com/cloudmesh-community/hid-sp18-417/blob/master/project-code/API/flask/Dockerfile

[10] Rashmi Ray. 2018. Project Overview. (Apr 2018). https://www.youtube.com/watch?v=ilQ54AXBCEM&feature=youtu.be

[11] wikipedia. 2018. Pygal. (2018). https://en.wikipedia.org/wiki/Scalable_Vector_Graphics

# Benchmarking a Sentiment Analysis Algorithm Using Hadoop on Multiple Platforms

Min Chen
Indiana University
School of Informatics, Computing, and Engineering
Bloomington, IN 47408
mc43@iu.edu

Gregor von Laszewski
Indiana University
Smith Research Center
Bloomington, IN 47408
laszewski@gmail.com

Bertolt Sobolik
Indiana University
School of Informatics, Computing, and Engineering
Bloomington, IN 47408
bsobolik@iu.edu

## ABSTRACT

A sentiment analysis algorithm is run on several Hadoop configurations: in pseudo-distributed mode on an Ubuntu 16.04 virtual machine on a MacBook Pro with and without YARN, in pseudo-distributed mode in a Docker container on various personal computers and on FutureSystems Echo, in a cluster created using Docker Compose on various personal computers and on FutureSystems Echo, and in a cluster created using Docker Swarm on FutureSystems Echo. Performance tests are natural language processing based sentiment analysis on movie reviews (Polarity 2.0) implemented under MapReduce framework using Hadoop Streaming. Configurations are described in detail and steps to recreate them are outlined. In an appendix, steps toward creating a Hadoop cluster on five networked Raspberry Pi 3 model B computers in a repeatable and scalable fashion, automating as much of the setup process as possible are detailed, and next steps are discussed.

## KEYWORDS

hid-sp18-419, hid-sp18-405, Volume: 9, Chapter: PaaS, Status: 100. Hadoop, Docker, FutureSystems

## 1 INTRODUCTION

Various configurations of personal computers are benchmarked on their performance of a sentiment analysis algorithim written in Python leveraging the Hadoop framework. Software and hardware technologies used in this study are described: Hadoop, Docker Compose, Docker Swarm, VirtualBox, four personal computer configuratons and one cloud platform. The data set being used for the benchmarking is described. The sentiment analysis algorithm used for this study used for is explained in both its original form and in the modified version that was developed to run in the MapReduce framework. The three deployments of Hadoop used for benchmarking are destribed: Pseudo-Distributed mode, Fully-Distributed mode using Docker Compose, and Fully-Distributed mode using Docker Swarm. The benchmarking process is explained. Results are presented and discussed, and conclusions are drawn.

## 2 TECHNOLOGY USED

This section describes the technologies that have been utilized throughout the project. These technologies can be grouped into the following categories: Hadoop, Docker, personal computers, and cloud platforms.

### 2.1 Hadoop

Hadoop is a software library supported and maintained by The Apache Software Foundation. It allows for processing large data sets in a distributed computing environment. Inspired by two papers: *Google File System* [7] published in 2003 and *MapReduce: Simplified Data Processing on Large Clusters* [3] which followed in 2004, Hadoop 0.1 was released in 2006 and the official 1.0 release came in 2012 [32]. Hadoop is designed to allow users to leverage simple programming models to process large data sets on computer clusters. The software detects and handles failures on commodity hardware instead of requiring specialized hardware to handle deliver high-availibility. The most current version at the time of writing is 3.0.1 [27]. This project will leverage version 2.9.0, which is the current stable version.

The core of Hadoop include the following components: Hadoop Common, Hadoop Distributed File System (HDFS), Hadoop YARN and Hadoop MapReduce. There are also a large variety of Hadoop-related projects at Apache that cover machine learning (Mahout), data warehousing (Hive), data serialization (Avro), databases (Hbase, Cassandra), etc. All of these components are available via the Apache open source license [27]. A high-level diagram of the basic Hadoop architecture in a distributed cluster with one Master Node and two Worker Nodes is shown in Figure 1.

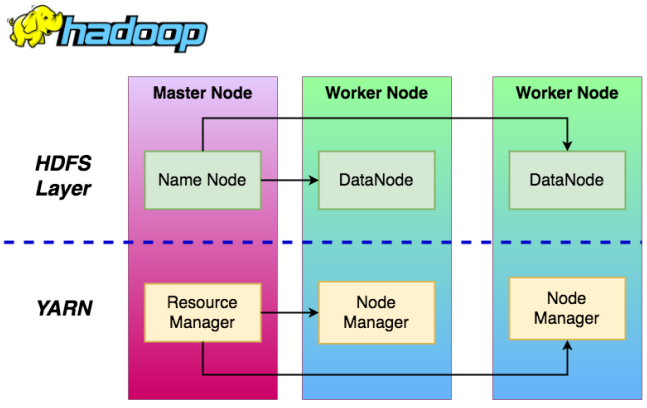


**Figure 1: Hadoop High Level Architecture [32]**

*HDFS.* HDFS stores metadata and application data separately: metadata are stored on the NameNode (usually sitting on the Master Node) and application data are distributively stored on Datanodes.

Datanodes communicate with each other through TCP-based protocols. Instead of storing one copy, data is replicated on multiple DataNodes to ensure high-availability [22].

*YARN.* Hadoop YARN provides resource management for the Hadoop cluaster [27]. As shown in Figure 1, based on application demand, priorities, and resource availability, the Resource Manager dynamically allocates containers to applications to run on particular nodes [30]. (Containers here meaning logical bundle of resources, not to be confused with Docker containers.) YARN interacts with Node Manager on each of the nodes: monitoring resource availability, reporting faults, and managing container lifecycles [30].

*MapReduce.* Hadoop MapReduce is an implementation of the MapReduce programming model that is typically used for processing large amounnts of data [27]. It relies on YARN for resource management. A map function is specified that generated an set of intermediate key-value pairs. The output of the map function is fed to a reduce function that combines these intermediate values into the final result [3].

*Hadoop-Streaming.* Although Hadoop and most of the applications in the Hadoop Framework are written in Java, Hadoop-Streaming, a java program that comes with the Hadoop distribution. Using Hadoop-Streaming, one can define mapper and reducer functions written in another language, and feed these into `hadoop-streaming.jar` as parameters to create and run MapReduce jobs [26]. In this project, all the mappers and reducers are written in Python and passed to Hadoop-Streaming. One drawback to this approach is that if the script being run is dependent on outside packages, they need to be zipped in a flat way and passed to each job containers for a special import before use. Because of this limitation, some of the linguistic features of the original version of the sentiment analysis algorithm used in this project are omitted due to the difficulties of transporting and loading necessary packages and corpora. This will be discussed in more detail in Section 4.

## 2.2 Docker

Docker is a technology that performs operating system-level virtualization, allowing applications to run in containers instead of full VMs [33]. It leverages control groups and name space isolation in the Linux kernel. Containers start up much faster than VMs and are supported by all the major public cloud vendors [6]. The most current stable release of Docker Community Edition available at the time of writing, 18.03.0-ce, is used for this project.

*Docker Compose.* Docker Compose is a tool which is distributed with Docker. It helps define and manage applications or clusters with multiple Docker containers [4]. Docker Compose provides multiple isolated environments on a single host. It is used in this project so that a Hadoop cluster can be deployed on a single VM or a single physical machine with the Master Node and Worker Nodes running in separate containers. The scalability feature of Docker Compose allows the size of the cluster to be adjusted at run time without stopping the cluster. When a service is stopped and restarted Compose will resuse the existing containers as long as the configuration files have not changed for those containers [4].

*Docker Swarm.* A group of physical machines or VMs running Docker Engine can be joined into a cluster called a Docker Swarm. Each machine is referred to as a node. One node in the cluster will be the head, or manager, of the swarm. The head node is the only node communicating with outside client and receiving execution instructions. It coordinates instructions and passes them to other worker nodes. On FutureSystem Echo, the cloud platform used for this project, five physical servers are in Docker Swarm mode with one head and four workers. Key features of Docker Swarm include: cluster management integrated with Docker Engine, decentralized design, scaling, load balancing, and rolling updates [5]. A stack of services can be deployed on a Swarm cluster using the same YAML file used by Docker Compose. The user can pass a single command to the head node to start, scale or stop the services. The head node then coordinates among the the other nodes in the Swarm cluster and attempt to balance the workload optimally.

## 2.3 VirtualBox

VirtualBox is software from Oracle that allows one to run x86 and AMD64/Intel64 VMs on personal computers. It is used in this project for running the VMs in the Hadoop setups and for burning custom images for the Raspberry Pis discussed in Section 10. Add-ons include Guest Additions, which allow one to cut and paste between the VM and the host, and the VirtualBox Extension Pack that allows mounting peripherals on USB 3.0 [15].

## 2.4 Personal Computers

*Macbook Pro (15-inch, 2017).*

- Operating system: macOS Sierra 10.12.6
- Processor: 3.1 GHz Intel Core i7
- Memory: 16 GB 2133 MHz LPDDR3
- Graphics: Intel HD Graphics 630 1536 MB
- Peripheals
  - HyperDrive - USB Type-C Hub (has microSD card slot)
  - Insignia - USB Type-C to Gigabit Ethernet Adapter (to connect to switch)
- Software
  - VirtualBox 5.2.8 with Guest Additions and the VirtualBox Extension Pack for USB 3.0 support.

*Macbook Pro (13-inch, 2015).*

- Operating system: macOS High Sierra 10.13.4
- Processor: 2.7 GHz Intel Core i5
- Memory: 8 GB 1867 MHz DDR3
- Graphics: Intel HD Graphics 6100 1536 MB

*Acer Laptop (Aspire 4830).*

- Operating system: CentOS Linux 7 (Core)
- Processor: Intel Core i5-2430M CPU @2.40GHz
- Memory: 16 GB 1600 MHz DDR3

*Asus Desktop.*

- Operating system: Windows 7 Professional
- Processor: Intel Core i5-4690K CPU @3.50GHz
- Memory: 16 GB 1600 MHz DDR3
- Software: VirtualBox 5.2.8

## 2.5 Cloud Platforms

*FutureSystem Echo.* FutureSystems is available as part of the Digital Science Center (DSC) infrastructure and resources located at Indiana University - Bloomington [31]. One of the resources, Echo, is a "SuperMicro distributed shared memory cluster with 192 CPU cores and 6TB total memory capacity" [31].

- Operating system: Ubuntu 16.04
- Processors: 16 SuperMicro X9DRW servers, each node with 2 6-core Intel Xeon CPU E5-2640 2.50GHz processors for a total of 192 CPU cores
- Memory: 6TB (384GB per node)

# 3 DATA

The data used for the project is the Polarity Data 2.0, which is a dataset of movie reviews first used by Bo Pang and Lillian Lee [16] [17]. The dataset includes 1000 positive and 1000 negative processed reviews that are labeled with respect to their overall sentiment polarity.

Each of the reviews is in the format of text file; each line in the files corresponds to a sentence; and every token (iėẇords, punctuation marks, and numbers) has been separated by a space. Figure 2 shows several lines in one of the movie reviews with line number on the left, illustrating the two features mentioned above. These features play an important role in the sentiment analysis algorithm, especially when using the MapReduce framework, which is discussed in detail in Section 4.

```
1 sometimes a movie comes along that falls somewhat askew of the rest .
2 some people call it " original " or " artsy " or " abstract " .
3 some people simply call it " trash " .
4 a life less ordinary is sure to bring about mixed feelings .
5 definitely a generation-x aimed movie , a life less ordinary has everything from claymation to profane
  angels to a karaoke-based musical dream sequence .
6 whew !
7 anyone in their 30's or above is probably not going to grasp what can be enjoyed about this film .
8 it's somewhat silly , it's somewhat outrageous , and it's definitely not your typical romance story , but for the
9 right audience , it works .
```

**Figure 2: Example Data File [13]**

The data is directly pulled from the source [13] and then split into training and testing sets according to a ratio of 8:2. The split algorithm maintains the original ratio of positive and negative reviews in both the training and testing data sets. As a result, there are 1600 reviews (800 positive and 800 negative) in the training data set and 400 reviews (200 positive and 200 negative) in the testing data set. The splitting process is done by using the random sort functionality in bash with a fixed random seed to ensure consistency of performance benchmarking on different platforms.

# 4 ALGORITHM

This section introduces the algorithm used to classify the overall sentiment of the movie reviews in the benchmarking process. The algorithm is a modified version of the one used by Pang and Lee [16] that is explained in details by Jurafsky and Martin [11].

## 4.1 Baseline algorithm

The baseline algorithm contains the following steps:

(1) Tokenization
(2) Feature Extraction
(3) Classification

*Tokenization.* As mentioned in Section 3, every token (words, punctuation marks, and numbers) has been separated by space in the text files. Becuase of this, the algorithm needs only to split the text strings with space to perform tokenization rather than using sophisticated tokenizers. This saves space and running time and avoids the complication of loading Python packages for Hadoop worker nodes in Hadoop-streaming. However, in more general settings with text data, the non-trivial tokenization step needs to be performed before other type of text processing.

*Feature Extraction.* The main feature that we use is the words contained in the documents. However, there are options mentioned in Jurafsky and Martin [11], including:

(1) All tokens
(2) Removing stoplist words
(3) Removing punctuation tokens
(4) Using only adjectives
(5) Unify tokens with lemmas
(6) Negation treatment

In this project, we applied stoplist words removal, punctuation tokens removal and negation treatment.

Stoplist is the list of words that are considered to be commonly used across documents regardless of sentiments and therefore add little value when performing classification tasks. We removed stoplist words according to the English stopwords module from the Python package: NLTK corpus [14]. Punctuation removal is done using the Python string module [18], supplemented by a regular expression written for this project. We applied a basic negation treatment by adding a NOT_ prefix to every token that follows a negation word and is not separated by any punctuation tokens. For example:

I really do not like the movie
becomes:
I really do not Not_like Not_the Not_movie.

This method is used by Das and Chen [2] as well as Pang, Lee, and Vaithyanathan [17].

We did not apply the option of using only adjectives based on the findings by Pang etàlthat using adjectives alone does not provide a better results than including all words [16]. In our own experiments, the accuracy dropped by an average of 2% in the testing phase when we implemented the algorithm. We also did not apply lemmatization for all the tokens. Although lemmatization would increase the accuracy by around 2% in one of the authors previous studies, it requires ontologies such as Wordnet. Including the Wordnet corpus with a lemmatizer would have complicated the implementation of Hadoop-streaming significantly. We tried to follow the instructions online [8] [12] to pass Python package NLTK and the corpus Wordnet as zipped files from name nodes to data nodes in the MapReduce tasks and found that the package could be passed but not loaded and corpus could not be passed at all. The issues we faced deserve further study in a future project.

Pang, Lee, and Vaithyanathan [17], also considered different set of features such as bigrams, combination (back up) of bigrams and unigrams, top-unigrams, etc. For simplicity, we chose to only use unigram models, iėsingle tokens.

*Classifier.* We used Naive Bayes as the classifier. The formulation is given as:

$$c_{NB} = \text{argmax}_{c_j \in C} P(c_j) \prod_{i \in positions} P(w_i|c_j) \quad (1)$$

where in this case, $j \in \{positive, negative\}$. Also, for this movie review dataset, we know the prior probabilities: $P(c_{positive}) = P(c_{negative}) = 0.5$

*Smoothing.* To avoid the problem that the probability of some token in the training/testing document is zero, we use the add-one smoothing, or Laplace smoothing. In this case:

$$P(w|c) = \frac{count(w,c) + 1}{count(c) + |V|} \quad (2)$$

Figure 3 is the algorithm we implemented as summarized by Jurafsky and Martin:

**function** TRAIN NAIVE BAYES(D, C) **returns** log $P(c)$ and log $P(w|c)$

**for each** class $c \in C$ # Calculate $P(c)$ terms
  $N_{doc}$ = number of documents in D
  $N_c$ = number of documents from D in class c
  $logprior[c] \leftarrow \log \frac{N_c}{N_{doc}}$
  $V \leftarrow$ vocabulary of D
  $bigdoc[c] \leftarrow$ **append**(d) **for** d $\in$ D **with** class $c$
  **for each** word $w$ in V # Calculate $P(w|c)$ terms
    $count(w,c) \leftarrow$ # of occurrences of $w$ in $bigdoc[c]$
    $loglikelihood[w,c] \leftarrow \log \frac{count(w,c) + 1}{\sum_{w' \text{ in } V} (count(w',c) + 1)}$
**return** $logprior$, $loglikelihood$, $V$

**function** TEST NAIVE BAYES($testdoc$, $logprior$, $loglikelihood$, C, V) **returns** best $c$

**for each** class $c \in C$
  $sum[c] \leftarrow logprior[c]$
  **for each** position $i$ in $testdoc$
    $word \leftarrow testdoc[i]$
    **if** $word \in V$
      $sum[c] \leftarrow sum[c] + loglikelihood[word,c]$
**return** $\text{argmax}_c$ $sum[c]$

**Figure 3: Naive Bayes for Sentiment Analysis [11]**

## 4.2 Implementation Using MapReduce

As illustrated in Equation 1, Equation 2, and Figure3, the Naive Bayes classification algorithm is based on comparing posterior probability. The likelihood of each single valid token can be calculated by aggregating the count of this token among the training documents for the two classes: positive and negative. The likelihood of a testing document is an aggregation of the likelihood of each valid token within that document. Therefore both training and testing process can be implemented in a MapReduce framework to facilitate the process of counting tokens and aggregation. The implementation is summarized in Figure 4

There are two MapReduce jobs in the training process, one for training on positive-labeled documents and the other for negative-labeled documents. $count(w,c)$ in Equation 2 shows the necessity to count the tokens according to the classes, which are positive and negative in this task. During the training on positive-labeled documents, the TrainingMapper Python code will read in each positive

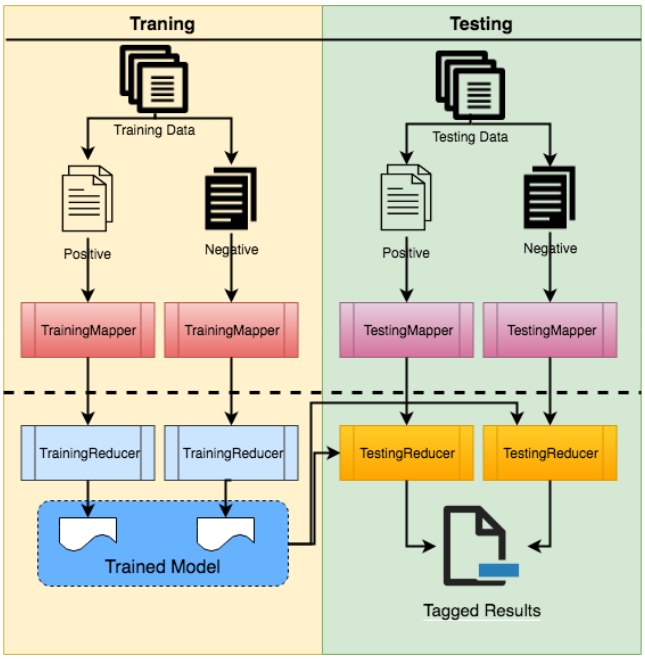


**Figure 4: MapReduce Implemenation of the Algorithm**

movie review in the training set through standard input, filter valid tokens (removing stoplist words, punctuation, and adding negation and described in Subsection4.1) and print to the standard output. The read-in process is done on a line-by-line basis using a Python generator. This is achievable because the data is formatted in so that each line is one sentence (see Section 3), thus no sentence is split by line breaks. The TrainingReducer Python code will take these standard output and perform aggregation by tokens. The result is a dictionary-like text file containing number of occurrence for each valid token in the positive training set. The negative training documents are processed in a similar way.

In the testing process, each testing document is analyzed separately. The TestingMapper reads in each document, perform the same filter to keep valid tokens, and print to standard output. The TestingReducer reads in both this standard output and the two results files from the training process, calculating the likelihood of each valid token in this document according to the smoothing formula given by Equation 2 before getting posterior probability through aggregation according to Equation 1. Then for each document there are two posterior probabilities, positive and negative. The document is assigned to the one with higher probability. The final output is a text file in which each line contains the testing document name and its classification label. In the actual implementation, we performed the testing process using two MapReduce jobs, one for the testing data that are actually positive, the other for the testing data that are actually negative. This is essentially the same as performing testing process on each document blindly and then verify the results. However, implementing it in this way makes it easier to calculate the accuracy of the algorithm.

In summary, we used Python codes to implement two mappers (TrainingMapper and TestingMapper) and two reducers (TrainingReducer and TestingReducer) to perform four MapReduce jobs for the training and testing processes. For the fixed random seed mentioned in Section3, we achieved the following results: out of the 200 positive testing data, 162 are correctly labeled; out of the 200 negative testing data, 168 are correctly labeled. Therefore, the overall accuracy of the algorithm is 82.5%. This is in line to the result achieved by Pang, Lee, and Vaithyanatha, which is illustrated in Figure5.

| | Features | # of features | frequency or presence? | NB | ME | SVM |
|---|---|---|---|---|---|---|
| (1) | unigrams | 16165 | freq. | **78.7** | N/A | 72.8 |
| (2) | unigrams | " | pres. | 81.0 | 80.4 | **82.9** |
| (3) | unigrams+bigrams | 32330 | pres. | 80.6 | 80.8 | **82.7** |
| (4) | bigrams | 16165 | pres. | 77.3 | **77.4** | 77.1 |
| (5) | unigrams+POS | 16695 | pres. | 81.5 | 80.4 | **81.9** |
| (6) | adjectives | 2633 | pres. | 77.0 | **77.7** | 75.1 |
| (7) | top 2633 unigrams | 2633 | pres. | 80.3 | 81.0 | **81.4** |
| (8) | unigrams+position | 22430 | pres. | 81.0 | 80.1 | **81.6** |

**Figure 5: Results from Pang, Lee, and Vaithyanatha (2002) [17]**

## 5 HADOOP DEPLOYMENT

For the entire project, we chose the current stable release of Hadoop at the time of the study: version 2.9.0. In order to make it convenient to benchmark across platforms we adopted a fixed set of configuration files for all of our installations. For example, the value of `dfs.replication` is set to 3, which means that 3 copies of the data will be stored in the HDFS. In addition, physical memory limit for each map task and reduce task is set to 1024MB, with the ratio of virtual memory to physical memory allowed set to 2.1. Therefore, the virtual memory limit for each map task and reduce task is 2150MB.

Hadoop can be configured to run in three modes: Standalone Mode, Pseudo-Distributed Mode and Fully-Distributed Mode [24].

- Standalone Mode: In this Mode, Local file system instead of HDFS is used. Resource Manager, Node Manager and all other processes are running not only in one virtual machine but in one single Java Virtual Machine (JVM).
- Pseudo-Distributed Mode: HDFS is used, NameNode, DataNode, ResourceManager and NodeManager are in the same virtual machine but in different JVMs.
- Fully-Distributed Mode: This is the mode that Hadoop clusters can be ranging from a few nodes to large clusters with thousands of nodes. Each node (worker) will be on different virtual/physical machines and they all have DataNode and NodeManager running as seperate JVMs.

In this project, we deployed Hadoop in Pseudo-Distributed Mode and Fully-Distributed Mode with the main focus on the latter.

### 5.1 Pseudo-Distributed

We developed Pseudo-Distributed Hadoop cluster in two ways as illustrated in Figure 6. The direct installation is on the left, where NameNode, DataNode, ResourceManager and NodeManager are sitting on the operating system of the host machine (wich could be a VM) as separate Java Virtual Machines. The right panel of Figure 6 shows the dockerized version where the four core components are wrapped in one Docker container and sitting on the Docker Engine.

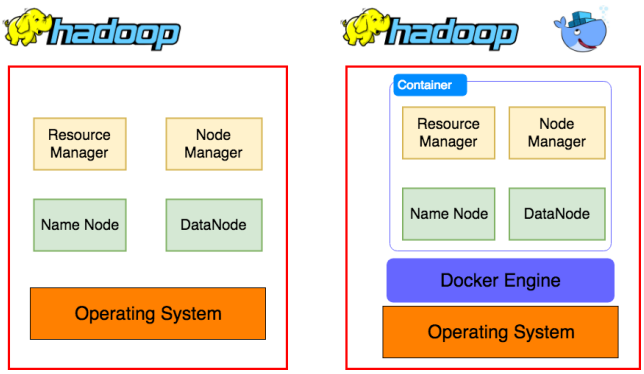


**Figure 6: Architecture of Pseudo-Distributed Hadoop**

*Direct Installation.* Direct installation of Hadoop is done based on instructions on Dominique Thiebaut's Wiki [29] and from The Apache Software Foundation [25]. First a virtual machine running Ubuntu 16.04 is setup on VirtualBox running on the 2017 Macbook Pro described in Section 2.4. It was decided to increase the memory to 2GB and the virtual hard drive to 10GB based on instructions provided in the Handbook [31]. Networking is configured using a bridged adaptor connected to the host's wifi port. OpenSSH, Curl, and the Java JDK 8 from OpenJDK are installed on the VM. Per Thiebault's instructions, ipv6 is disabled. A group called `hadoop` is created with a user called `hduser`. SSH keys are created using `ssh-keygen` and hduser's public key is added to `known_hosts` so that passwordless SSH to to localhost is possible. Pyenv is then installed and Python 3.6.2 is made the global version of Python for the VM. Hadoop 2.9.0 is downloaded from Apache and installed in `/usr/local/hadoop`. The following environment variables are set in hduser's `.bashrc` file:

```
export HADOOP_HOME=/usr/local/hadoop
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
```

The following directories are added to hduser's path:

```
$HADOOP_HOME/bin
$HADOOP_HOME/sbin
```

The rest of the configuration is done with modifications to xml files and scripts in `$HADOOP_HOME/etc/hadoop`: the java location needs to be set in `hadoop_env.sh`, the detault file system is set in `core-site.xml`, and the number of file system replications is set in `yarn-site.xml`. To make it easier to clean everything up and rerun the algorithm, the directory used for temporary files is also set in `core-site.xml`.

The sentiment analysis algorithm was run in pseudo-distributed mode on the VM with and without yarn as a resource manager. If yarn is used, a few more configuratons need to be done in the xml files in `$HADOOP_HOME/etc/hadoop`: the MapReduce framework

name needs to be set to yarn in `mapred-site.xml`, and `mapreduce_shuffle` needs to be set up in `yarn-site.xml`.

*Dockerized Installation.* We used a Dockerfile to build the image for the Pseudo-Distributed Hadoop container. The image is modified from the one on the sequenceiq GitHub repository [9]. The modification includes minor bug fixes, change of the Java and Hadoop version used, installation of Python and Pyenv, download of data and transfer of Python code for the use of Hadoop-Streaming. The container can be started up with a shell interface for user to input commands operating the cluster, or as an executable program which runs the whole classification algorithm and transfers the results, including running time, back to the host machine. In later case, the Docker Container needs to be started by initiating a customized start-up script that overrides the default scripts defined by the `COMMAND` or `ENTRYPOINT` settings in the Dockerfile.

## 5.2 Fully-Distributed using Docker Compose

A Fully-Distributed Hadoop cluster can be deployed on multiple VMs or physical machines. By using Docker Compose, one can deploy the same multi-node cluster on one single virtual machine. The architecture is illustrated in Figure 7, where the red rectangle stands for some virtual or physical machine. This could be a personal laptop, VirtualBox running Ubuntu or FutureSystem Echo. However, since Echo is in Swarm mode, Docker Compose will only deploy the hadoop cluster on the head node due to the restriction that the only node interacting with clients is the head node.

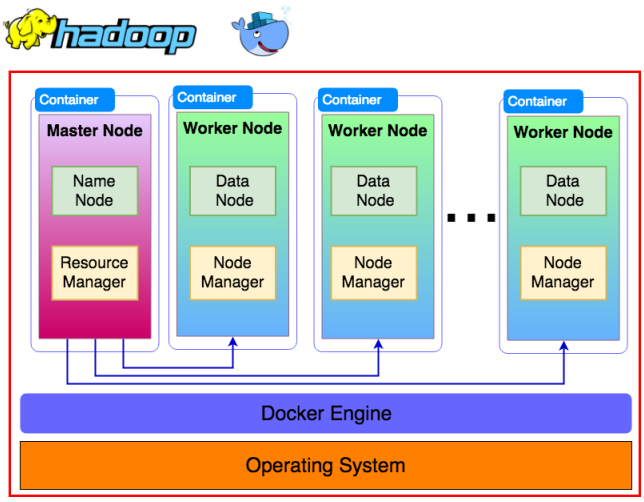


**Figure 7: Architecture of Fully-Distributed Hadoop Cluster Using Docker Compose**

In Figure 7, each Docker container acts as a node and the cluster contains one Master Node and multiple Worker Nodes. The ResourceManager and NameNode within the Master Node container can communicate with the all the NodeManagers and DataNodes respectively using TCP ports.

The configuration process differs from the Pseudo-Distributed Hadoop container in two ways. First, there are two different start-up scripts for Master Node container and Worker Node container respectively: the script for Master Node only starts ResourceManager and NameNode but not NodeManager or DataNode, and the script for Worker Node starts NodeManager and DataNode but not ResourceManager or NameNode. Second, a YAML file needs to be configured with information about the Master and Worker Node containers that includes which image to use, which ports to expose, the network, the container name, the start-up command, etc. One important feature Docker Compose provides is that one only needs to configure one Worker Node containers in the YAML file instead of multiple ones. When starting up the service, a Docker Compose scale command can be used to scale up the number of Worker Nodes as desired. Furthermore, the scale command can also be used while the cluster is running. In that case, the number of Worker Nodes is adjusted according to the scale command therefore the size of cluster can be controlled dynamically without the need of shutting down and restart the cluster.

After the cluster is started up, a user can log into any of the containers, check status, and run commands. The running of the sentiment analysis algorithm is automated in a similar way as the Pseudo-Distributed cluster using Docker Containers described in Subsection 5.1.

## 5.3 Fully-Distributed using Docker Swarm

Although Docker Compose provides a way to simplify the deployment of a Fully-Distributed Hadoop cluster, there is an obvious limitation: all the containers have to be on the same Docker Engine on an operating system in order to be within the scope of the same VM or physical machine. Therefore, the number of Worker Nodes and the computation and storage capability of the cluster is limited by the host machine. Docker Swarm, as introduced in Section 2, provides a way to deploy the cluster across different machines.

Instead of building a cluster of virtual machines with Docker Swarm enabled, for this project we used the FutureSystem Echo, where Docker Swarm has been fully implemented. The architecture of the Hadoop cluster is illustrated in Figure 8, where each red rectangle stands for a physical node other than the head node in the swarm cluster because Docker Swarm will not deploy any container on the head node. Although in this case, the Master Node is on a different machine from some of the Worker Nodes, the communication is enabled within the network which is automatically built as the first step of the cluster deployment. Similarly as Docker Compose, the Swarm Mode also allows scaling up of the cluster without shutting down the MapReduce process.

The YAML file for Docker Compose can be used to configure the cluster under the Swarm mode with some minor changes. Additional options, such as deploy, are available, but some options, such as container name, are disabled. One key difference is that we have to include some start-up scripts and use those as `Entrypoint` or `Commandto` make the sentiment analysis algorithm run automatically when the cluster starts. Unlike in previous methods of deployment, where we still have the option of logging into the container and pass commands through bash, in swarm mode we cannot log into the containers because they are all deployed on worker nodes of the Swarm cluster.

A challenge in implementing Swarm mode is determining how one gets the analysis results, which sits on some worker node in the

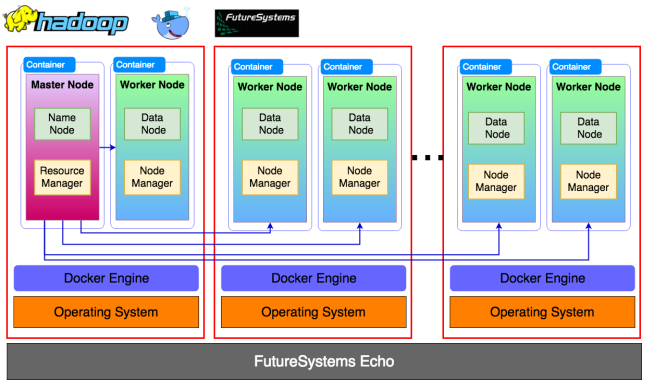


**Figure 8: Architecture of Fully-Distributed Hadoop Cluster Using Docker Swarm**

Swarm cluster, back to the head node. A stardard way of doing this is to use the WebHDFS REST API with the `curl` command. However, in our implementation, we had difficulty accessing different datanodes where the results are actually stored through WebHDFS. This may be caused by some port issue and configuration in the initial HDFS setting. As a workaround we put the results together with the log files in to the `logs` folder, which can be accessed through the YARN ResourceManager on a certain port (8088 by default).

## 6 BENCHMARKING PROCESS

The benchmarking has been done on several machines listed in Section 2: the 8GB RAM Macbook Pro 2015, the 16GB RAM Acer laptop running Centos, the 16GB RAM desktop running Ubuntu on VirtualBox, and FutureSystem Echo. For each of these machines, we used Docker to deploy Pseudo-Distributed Hadoop and Fully-distributed Hadoop using Docker Compose for worker number starting at 1. For each setting, 10 iterations of the whole process of setting up cluster and running algorithm were performed. We then calculated the mean of running time together with standard deviations and the coefficient of variation (which is the ratio of standard deviation and mean). On Echo, we also benchmarked the Fully-Distributed Hadoop using Docker Swarm in a similar fashion.

For some machines, there is a limit on the number of workers that can be established. For example, the Macbook can have at most two workers. When attempting to run more, the algorithm does not finish because the machine runs out of memory. The two 16GB RAM Acer laptops and VirtualBox on desktop could support 5 or 6 workers in a stable fashion. Stability here is defined as each iteration ending successfully and the deviation in running time is relatively small. Detailed results will be provided in Section 7.

## 7 RESULTS

This section elaborates the result of our benchmarking process. Deployment time and running time are discussed separately.

### 7.1 Deployment Time

All the Dockerized Hadoop clusters depend on Docker Engine and corresponding images. The deployment can be divided into three steps:

- Install of Docker
- Build or pull the image or images
- Start the cluster

*Install of Docker.* The installation of Docker on Linux and MacOS operating systems following the official guide on Docker website and can be done in around 1 minute.

*Build or pull the image or images.* Based on different Internet connection speeds we tried, one can build the images for deployment in 8 to 12 minutes. The images can also be pulled from DockerHub, which is the default cloud-based registry service of Docker. In our test, pulling in general is faster than building images by around 1 minute. We have four images throughout the whole project: pseudo, base, master and worker. The majority of layers are the same with some differences: Hadoop configuration files, startup scripts, and existence of data. Due to the sharing of layers, building 3 images (base, master and worker) for the Fully-Distributed cluster does not differ much from building only one. In the future, we could unify all images into one, with different scripts or options for the startup. It is noticed that on a single virtual machine if the iamges are pulled or built once, then later building process will utilize the cached layers and speed up the process.

*Start the cluster.* Starting a Pseudo-Distributed Hadoop takes less than 10 seconds. The start-up time for the Fully-Distributed cluster increases with the number of workers initialized. For worker numbers smaller than 10, the start-up takes less than 20 seconds. On FutureSystem Echo, under either Compose or Swarm mode, starting 40 workers takes around 30 seconds and starting 100 workers takes 40 seconds.

From the breakdown of deployment steps, we can see that the build or pull image step dominates the total time of the whole process. The total time is not more than 13 minutes in total when building from scratch. In our project, we also deployed a Pseudo-Distributed Hadoop directly (without Docker) on a VirtualBox. The process of manually setting up Hadoop with all dependencies and configuration files took much longer than the dockerized version. This illustrated the advantage of using Docker containers in cluster deployment.

### 7.2 Running Time

The running time is defined to be the time used to run the core script `runPythonMapReduce.sh`, which includes four steps. The first step is to split the data randomly into training and testing sets with the ratio 8:2 and while keeping the ratio of positive and negative reviews unchanged (0.5). The second step is to create directories on HDFS and put data into HDFS. The third step is the core MapReduce algorithm described in Section 4. The last step is to fetch classification results together with running time record from HDFS, and transfer these from the container to the host machine.

The results are summarized in Table 1, where each row represents the number of workers established in the cluster and each column is one particular deployment that is benchmarked. The number in the table is the mean running time of the 10 iterations in minutes (m) and seconds (s). The standard deviation is listed in the brackets. For each deployment options, the bold number is the least average running time that we observed, which can be considered as the

optimal Docker cluster choice for that deployment. The asterisk (*) indicates that the standard deviation is large. The threshold we used is that coefficient of variance (standard deviation divided by mean) equals to 0.1.

**Table 1: Running Time results**

| # Workers | Macbook Pro 2015 | Acer Laptop | Virtualbox on Desktop | Echo (Compose) | Echo (Swarm) |
|---|---|---|---|---|---|
| Pseudo | **99m14s (80s)** | **77m15s (8s)** | 43m34s (39s) | 22m25s (12s) | N/A |
| 1 | 104m34s (119s) | 82m8s (57s) | 44m47s (20s) | 33m42s (12s) | 20m49s (20s) |
| 2 | 108m50s (110s) | 84m35s (7s) | **42m47s (24s)** | 18m57s (6s) | 15m26s (17s) |
| 3 | N/A | 86m5s (6s) | 43m13s (17s) | 17m41s (2s) | 11m14s (4s) |
| 4 | N/A | 87m50s (11s) | 43m59s (58s) | **17m38s (2s)** | 9m24s (3s) |
| 5 | N/A | 90m1s (8s) | 48m46s (104s) | 17m52s (2s) | 8m47s (6s) |
| 6 | N/A | 92m32s (11s) | 58m57s (171s) | 18m9s (2s) | 8m25s (7s) |
| 7 | N/A | N/A | 85m19s (634s)* | 18m17s (2s) | 8m2s (4s) |
| 8 | N/A | N/A | 113m36s (2520s)* | 18m28s (2s) | 7m44s (2s) |
| 9 | N/A | N/A | N/A | 18m29s (1s) | 7m39s (2s) |
| 10 | N/A | N/A | N/A | 18m34s (2s) | **7m35s (2s)** |
| 15 | N/A | N/A | N/A | 18m56s (1s) | 7m37s (3s) |
| 20 | N/A | N/A | N/A | 19m12s (1s) | 7m44s (3s) |
| 25 | N/A | N/A | N/A | 19m34s (2s) | 7m52s (2s) |
| 30 | N/A | N/A | N/A | 19m50s (2s) | 8m1s (4s) |
| 35 | N/A | N/A | N/A | 20m2s (3s) | 8m16s (4s) |
| 40 | N/A | N/A | N/A | 20m18s (3s) | 8m20s (4s) |

# 8 OBSERVATION AND DISCUSSION

Based on Table 1, we have several observations:

## 8.1 Bottleneck of Memory

The first three deployments are done on personal computers, with memory from 8G to 16G. The N/A in the table indicates that the algorithm fails at run time due to Java exceptions (out of memory or lost of connection to MapReduce containers), which are good evidence of memory shortage. We observe that the Macbook, which has the least memory, can only run two Worker Node containers. The Acer laptop and desktop both have 16G memory, and they can sustain up to 6 Worker Node containers. Although we also get results for the desktop with 7 and 8 workers established, the running time increases significantly, and the variances are very large. Indeed, we could not get results on all 10 iterations with 8 workers, only 4 out of 10 executed successfully. During the execution with 7 or 8 workers, we can see from the terminal that some MapReduce containers are killed due to memory issue or connection issue and in that case ResourceManager will assign the failed jobs again until they are finished. During each iteration, different number of failed containers causes the big variation in the total running time.

Echo head node has almost 400GB of memory, so the iterations on Echo in Compose mode runs up to 40 Workers without any memory issue. When we deploy the cluster in Swarm mode, the total memory of the system is even larger (4 physical servers combined). However, since there may be other projects running on Echo, we did not increase the number of the Worker Node to test the limits of the system.

## 8.2 Pseudo vs One-Worker

We did not deploy Pseudo-Distributed Hadoop on Swarm because it would be only on one physical machine and does not utilize the advantages of Swarm. For all other deployment, we noticed that Pseudo-Distributed performs faster than the Fully-Distributed cluster with only one worker.

In these two settings, there will be exactly one instance of DataNode, NameNode, ResourceManager and NodeManager, the only difference is that in Fully-Distributed cluster with only one worker, these JVM runs in different containers. Given that both deployments share the same memory, computing power, and disks, the differences in the performance are likely related to the overhead of communication across containers. For example, message between DataNode and NameNode in the one-worker setting will have to pass through some TCP ports across containers while in Pseudo mode, the communication is much easier between JVMs directly. Therefore we see that for Pseudo-Distributed Hadoop outperforms Fully-Distributed Hadoop with only one worker.

## 8.3 Optimal Number of Worker

In each deployment with Fully-Distributed clusters, it seems that as the number of workers increases from one, the running time first decreases and then starts to increase. Therefore, each deployment has an optimal number of workers. The decrease and increase pattern in running time can be explained as follows: if the number of MapReduce containers (JVMs) within one Worker Node is limited by the settings in `mapred-site.xml`, then increase of Workers could lead to more mapper and/or reducers working at the same time thus reducing running time. However, since the number of mapper and reducer containers needed for the task is fixed (for example, 800 training files need at most 800 mapper), increasing number of workers does not really help because all of the workers are sharing the same set of processors and computing power. On the other hand, all workers also share the same hard drives, and the intensive file I/O could be the reason for the decline in performance as number of workers grows. (As discussed in Section 4, all mappers and reducers read files and write files to the disk extensively.) This is our conjecture based on the results observed and understanding of the algorithm; to make a conclusion of the root cause of this pattern, more specific testing with better control groups needs to be done.

Macbook and Acer laptop have the optimal number of workers to be one. According to previous discussion, one-worker clusters are slower than Pseudo-Distributed clusters. Therefore, running a Fully-Distributed cluster on these machines to solve the required sentiment analysis task is inefficient. The Desktop achieves the best result when the number of worker is 2. The Echo head node (when using Docker Compose) is optimal at 4 workers, and in Swarm mode on Echo the optimal worker number occurs within the range of 10 to 15.

## 8.4 Same Cluster on Different Platforms

If we fix the number of workers in the Fully-Distributed cluster, and compare the performance across these platforms. From Acer laptop, desktop, Echo with Compose and Echo with Swarm, the running time roughly reduces by half. We conjecture that the reason for the differences could have two possibilities, CPU power and Disk read/write speed. Given same number of Workers (without memory issue), the workload of each mapper and reducer is similar across

platforms, communication overhead should also be comparable, the differences could then be CPU power or hard drive writing speed or both. To disentangle these effect and draw a conclusion of the actual cause, we may need to design experiments controlling one of the effect and test the other. This could be future steps of the project.

Regardless of number of workers chosen, deployment under Swarm mode really outperforms the other. This illustrates the power of cloud computing in general. We could imagine gigantic clusters in Swarm mode hosting large number of containers, provides timely solution for Big Data applications.

## 9 CONCLUSION

Although its popularity is waning, Hadoop remains a useful tool for Big Data analysis. As one would expect, better hardware leads to better results. While many of the configurations tested are not ideal for the task, using a combination of Docker in its Compose and Swarm variants and Hadoop, provides a robust and versitile platform that can be deployed on a wide variety of hardware and software configurations with very little custom configuration. Startup is fast and the results, when there is sufficient memory available, are consisent. The authors would like to continue our exploration of Hadoop, comparing other cloud environments, including the Raspberry Pi cluster configuration that we began implementing and with larger datasets.

## 10 APPENDIX: HADOOP CLUSTER ON RASPBERRY PI

This section describes steps taken toward the goal of creating a Hadoop cluster using 5 Raspberry Pi 3 Model B computers in a repeatable, scalable way, so that the process could be replicated to create a much larger cluster. Successes and challenges are described and recommended next steps are outlined.

At the end of the process, each Pi has a unique hostname a new password. A computer outside of the cluster can log into any of the Pis via ssh and the Pis are configured to run MapReduce jobs using Apache Hadoop. The process of achieving these goals manually has been well documented in various blogs and message boards on the internet [21], but in order to make the process scalable, the confuguration described is automated to a large extent.

### 10.1 Hardware Used

The hardware used for the Raspberry Pi cluster consists of:

- Five Raspberry Pi 3 Model B computers
- One Waveshare 4in HDMI LCD touchscreen
- One Netgear model GS308 8-Port Gigabit ethernet switch
- One Anker PowerPort 6 USB power supply
- One Powtech 125V AC 15A 1875w adaptor with switch
- Five 1-foot ethernet cables
- Five 32 GB microSDHC UHS-I cards
- Six 6in USB 2.0 A-Male to Micro B cables
- 24 20mm by 5mm Hex Hexagonal Threaded Spacer Supports

A pinout diagram of the Pi 3B is shown in Figure 9.

### 10.2 Building a Pi Cluster

First, aluminium and copper heat syncs need to be attached to each Pi. The two aluminium heat syncs are attached to the Broadcom chip and the SMSC ethernet controller located on the top of the Pi. The blades of the heat syncs are parallel to the longer side of the Pi as shown in Figure 10.

The flat copper heat sync is attached to the Elpida RAM on the bottom of the Pi as shown in Figure 11.

After attaching the heat syncs, threaded hexagonal spacer supports are used to connect the Pis together. A fully-assembled 5-node Pi cluster is shown in Figure 12.

Each node of the cluster is then attached to the switch using an ethernet cables and to the power supply using a usb cables. The fully wired cluster is shown in Figure 13.

### 10.3 Creating a Custom Image

Each Pi uses a 32GB SD card as its hard drive. The operating system is to be burned on each SD card. For this cluster, the most current version of Raspbian Stretch Lite is chosen as the operating system. When downloaded, this image has ssh disabled and is configured with one user `pi` with the password `raspberry`. Every machine is named `raspberrypi`. In order to easily determine which SD card is in which computer in the cluster, the names are changed before burning the image to the SD card, and ssh is enabled on each card so the rest of the configuraton can be done from a remote computer without using a keyboard or monitor, commonly known as headless setup.

Another approach that the authors would like to explore at a later date would be to compile a custom image from the Raspbian source code. The code is freely available to download, either directly from raspbian.org [19] or on GitHub [20].

The computer used to create the custom images is the 2017 Macbook Pro that is described in Subsection 2.4. The script that creates the custom images downloads the latest image from Raspbian.org and mounts the two partitions so that they can be modified. The process used to mount the second partion only works on Linux, so a VM is created on which the script is run. This VM needs to have a big enough virtual disk for the base image times the number of custom images created. The current unzipped Raspbian Stretch Lite image at the time of this writing is 1.9GB. Instructions for setting up the VM provided in the Handbook were used [31].

If the images are going to be burned and verified on the VM, additional space will be needed equal to the size of the SD cards being burned because verificaton is done by copying the entire capacity of the SD card and comparing it to the image that was burned. The SD cards we are using are 32GB, so creating 5 images and verifying the burn will require around 44GB on top of the space required for Ubuntu. To be safe, the VM for creating the custom images is configured with at least a 65GB virtual disk image hard drive. It runs Ubuntu 16.04 on VirtualBox and is configured with 1 CPU, 2GB of memory, and USB 3.0 enabled. Networking is via a bridged network connected to the hosts WiFi.

The images are created using two scripts: `download_image.sh` - a simple shell script to download the latest Raspbian Stretch Lite image, and `modify_sdcard.py` - a Python script that will create the custom images. The Python script runs on Python version

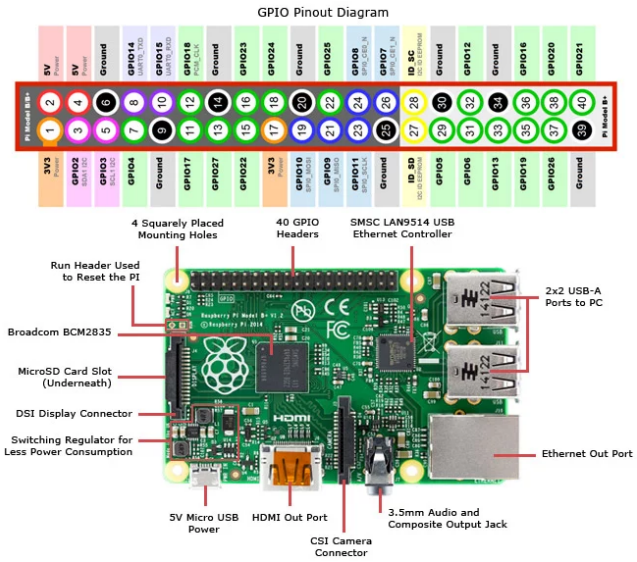


**Figure 9: Pinout diagram of Raspberry Pi 3B [10]. Note: the Pi used for this project has a Broadcom BCM2837, not the Broadcom BMC2835 shown in this figure.**

**Figure 10: Top view of a Raspberry Pi 3B with heat syncs attached.**

**Figure 11: Bottom view of a Raspberry Pi 3B with heat sync attached.**

2.7.12, which is the version that comes installed on Ubuntu 16.04. It requires one additional package, `pycryptodome`, which is needed to generate ssh keys. The script needs to be run as root to have permission to mount and modify the files in the Raspbian image.

The Python script creates the number of custom images specified with a flag. The names of the images are a basename followed by a three-digit number that increments sequentially from the specified starting number through that number plus the number of images created. The basename, starting number, and number of images can be specified with flags. The basename defaults to `snowcluster` and the starting number defaults to 0, so if three images were specified, the result would be three files: `snowcluster000.img`, `snowcluster001.img`, and `snowcluster002.img`.

When the script is run, the base image is copied the number of times specified by the user and the following modifications are made: * If the `--ssh` flag is set, or by default, the script will mount the first partition of the image and add a blank file named `ssh`. This enables ssh on the Pi on first boot. It also generates public and private RSA keys, creates a folder `//home/pi/.ssh/` and stores those keys in files called `id_rsa` for the private key and `id_rsa.pub` for the public key. It then appends the public key to the file `/home/pi/.ssh/authorized_keys`. If that file does not yet exist, it is created. This allows passwordless ssh login between any

**Figure 12: 5-node Pi cluster before wiring.**

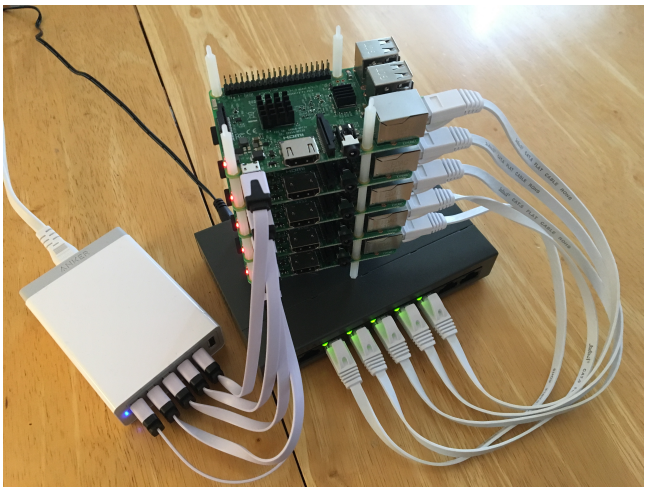

**Figure 13: Fully assembled 5-node Pi cluster.**

of the Pis in the cluster. * If the user specifies a public key with the `--sshkey` flag, the contents of that file is also appended to the file `/home/pi/.ssh/authorized_keys`. This allows passwordless ssh into the any of the Pis in the cluster from a setup machine.

After the images are created, they can either be copied to a shared folder and burned to the SD cards on the Mac using Etcher, or on the VM using dd. Instructions for doing this provided by The Raspberry Pi Foundation were tested on both platforms [28].

A known issue with the Python script is that because it needs to be run as root to mount and unmount the images and modify the contents, `~/.ssh` and its contents: `id_rsa`, `id_rsa.pub`, and `authorized_keys` are owned by root. This leads to a permissions error when connecting via ssh from one pi to another. Solutions for this issue need to be investigated.

In addition to fixing this permissions issue, other desired future enhancements include:

- Adding an option to specify paramaters in a yaml file as specifying them on the command line is cumbesome.
- Adding an option to create keys and adding them to `known_hosts` on all the images to bypass the verification on first ssh connection.
- Setting a fixed IP address on the head node if DHCP is to be used, or for all the Pis.
- Incorporating the functionality in `download_image.sh` into script, with option to download a different base OS (egfrom Dexter Labs).
- Adding an option to burn the images from the script.
- Adding a script to do post-boot configuraton.

## 10.4 Post-boot configuraton

Post-boot configuration was explored but not implemented. The goal for future development would be to write a script that would perform the remainder of the configuration, copy it to each of the Pis, and run on first login. This script will permanently enable ssh; change the password of all of the Pis; create or update a database containing IP addresses, Mac addresses, and hostnames; optionally configure the head node of the cluster as a DHCP server; and install and configure Hadoop on the Pis.

Permanently enabling ssh is necessary as the `ssh` file added in to the first partition is deleted after first login. It is done with the following commands:

```
sudo apt-get update
sudo apt-get install openssh-server
sudo systemctl enable ssh
sudo systemctl start ssh
```

Changing the password from the default password `raspberry` to something more secure can be done with this line [1]:

```
echo -e ``raspberry\\nsnowcluster\\nsnowcluster'' | passwd
```

The database of Mac addresses, IP addresses, and hostnames can be done in two steps: populate the hostnames and IP addresses during the creation of the VM images, and add the Mac addresses after booting up the Pis. Then the setup script can run `ifconfig -a` and parse the output to get the Mac addresses for both wireless and wired connections.

Setting up head node of the cluster as a DHCP server requires further investigation and testing. The process was followed and documented, but a new version of the DHCP server `isc-dhcp-server` is incompatible with the process followed [23].

## 11 WORK BREAKDOWN

Min Chen converted the sentiment analysis algorithm, did the Docker configurations and benchmarking and was the main author of these sections of the paper. Bertolt Sobolik did the Pi configuration and pseudo-distributed configurations on the virtual machines and was the main author of these sections of the paper. Min Chen and Bertolt Sobolik review and tested each other's work.

## ACKNOWLEDGMENTS

The authors would like to thank Dr. Gregor von Laszewski for his support and suggestions to write this paper, and Bo Feng for his guidance and help with Hadoop deployment in Docker Swarm mode.

## REFERENCES

[1] Elliot Beaumont. 2017. RPi-Distro/pi-gen. Stack Overflow. (June 2017). https://stackoverflow.com/a/44846864/7292730 Accessed: 2018-04-25.

[2] Sanjiv Das and Mike Chen. 2001. Yahoo! for Amazon: Extracting market sentiment from stock message boards. In *In Asia Pacific Finance Association Annual Conf. (APFA)*, Vol. 35. Bangkok, Thailand, 43. Accessed: 2018-04-11.

[3] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM* 51, 1 (Jan. 2008), 107–113. https://doi.org/10.1145/1327452.1327492 Accessed: 2018-04-25.

[4] Docker. [n. d.]. Overview of Docker Compose. Web Page. ([n. d.]). https://docs.docker.com/compose/overview/ Accessed: 2018-04-26.

[5] Docker. [n. d.]. Swarm mode overview. Web Page. ([n. d.]). https://docs.docker.com/engine/swarm/ Accessed: 2018-04-26.

[6] Ian Foster and Dennis B. Gannon. 2017. *Cloud Computing for Science and Engineering* (1st ed.). The MIT Press.

[7] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. 2003. The Google File System. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles (SOSP '03)*. ACM, New York, NY, USA, 29–43. https://doi.org/10.1145/945445.945450 Accessed: 2018-04-25.

[8] Jeff Hammerbacher. 2008. Sending Files to Remote Task Nodes with Hadoop MapReduce. Blog. (Nov. 2008). http://blog.cloudera.com/blog/2008/11/sending-files-to-remote-task-nodes-with-hadoop-mapreduce/ Accessed: 2018-04-1.

[9] Hortonworks. [n. d.]. hadoop-docker. Github Repository. ([n. d.]). https://github.com/sequenceiq/hadoop-docker Accessed: 2018-04-26.

[10] Jameco Electronics. [n. d.]. Raspberry Pi Pinout Diagram, Circuit Notes. Web Page. ([n. d.]). https://www.jameco.com/Jameco/workshop/circuitnotes/raspberry-pi-circuit-note.html

[11] Daniel Jurafsky and James H. Martin. 2009. *Speech and Language Processing (2Nd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA. Accessed: 2018-04-11.

[12] Nate Murray. 2009. How to use Cascading with Hadoop Streaming. Blog. (Nov. 2009). http://eigenjoy.com/2009/11/18/how-to-use-cascading-with-hadoop-streaming/ Accessed: 2018-04-1.

[13] NLP at Cornell. 2012. Movie Review Data. Web Page. (April 2012). http://www.cs.cornell.edu/people/pabo/movie-review-data/ Accessed: 2018-04-11.

[14] NLTK. 2015. Accessing Text Corpora and Lexical Resources. Web Page. (July 2015). https://www.nltk.org/book/ch02.html Accessed: 2018-04-17.

[15] Oracle. 2018. VirtualBox. Web Page. (April 2018). https://www.virtualbox.org/ Accessed: 2018-04-25.

[16] Bo Pang and Lillian Lee. 2004. A Sentimental Education: Sentiment Analysis Using Subjectivity Summarization Based on Minimum Cuts. In *Proceedings of the 42Nd Annual Meeting on Association for Computational Linguistics (ACL '04)*. Association for Computational Linguistics, Stroudsburg, PA, USA, Article 271. https://doi.org/10.3115/1218955.1218990 Accessed: 2018-04-11.

[17] Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. 2002. Thumbs Up?: Sentiment Classification Using Machine Learning Techniques. In *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing - Volume 10 (EMNLP '02)*. Association for Computational Linguistics, Stroudsburg, PA, USA, 79–86. https://doi.org/10.3115/1118693.1118704 Accessed: 2018-04-11.

[18] Python Software Foundation. 2018. string fi!? Common string operations. Web Page. (April 2018). https://docs.python.org/3/library/string.html Accessed: 2018-04-17.

[19] raspbian.org. 2018. Index of /raspbian/dists/stretch/. Web Site. (April 2018). http://archive.raspbian.org/raspbian/dists/stretch/ Accessed: 2018-04-25.

[20] RPi-Distro. 2018. pi-gen. GitHub. (April 2018). https://github.com/RPi-Distro/pi-gen Accessed: 2018-04-25.

[21] Vivek Seth. 2017. How to Bootstrap a Headless Raspberry Pi with a Mac. Medium. (Feb. 2017). https://medium.com/@viveks3th/how-to-bootstrap-a-headless-raspberry-pi-with-a-mac-6eba3be20b26 Accessed: 2018-04-25.

[22] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. 2010. The Hadoop Distributed File System. In *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST) (MSST '10)*. IEEE Computer Society, Washington, DC, USA, 1–10. https://doi.org/10.1109/MSST.2010.5496972 Accessed: 2018-04-25.

[23] Niranjan Tallapalli. [n. d.]. Hadoop and Spark Installation on Raspberry Pi-3 Cluster fi?! Part-3. Web Page. ([n. d.]). https://tekmarathon.com/2017/02/16/hadoop-and-spark-installation-on-raspberry-pi-3-cluster-part-3/ Accessed: 2018-04-27.

[24] The Apache Software Foundation. [n. d.]. Hadoop: Setting up a Single Node Cluster. Web Page. ([n. d.]). https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/SingleCluster.html Accessed: 2018-04-26.

[25] The Apache Software Foundation. [n. d.]. Hadoop: Setting up a Single Node Cluster. Web Page. ([n. d.]). https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/SingleCluster.html Accessed: 2018-04-27.

[26] The Apache Software Foundation. 2017. Hadoop Streaming. Web Page. (Nov. 2017). http://hadoop.apache.org/docs/r2.9.0/hadoop-streaming/HadoopStreaming.html Accessed: 2018-04-20.

[27] The Apache Software Foundation. 2018. Welcome to Apache Hadoop. Web Page. (April 2018). http://hadoop.apache.org/ Accessed: 2018-04-25.

[28] The Raspberry Pi Foundation. [n. d.]. Installing Raspberry Pi Images. Web Page. ([n. d.]). https://www.raspberrypi.org/documentation/installation/installing-images/ Accessed: 2018-04-26.

[29] Dominique Thiebaut. [n. d.]. Setup Virtual Hadoop Cluster under Ubuntu with VirtualBox. Wiki. ([n. d.]). http://www.science.smith.edu/dftwiki/index.php/Setup_Virtual_Hadoop_Cluster_under_Ubuntu_with_VirtualBox Accessed: 2018-04-27.

[30] Vinod Kumar Vavilapalli, Arun C. Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, Bikas Saha, Carlo Curino, Owen O'Malley, Sanjay Radia, Benjamin Reed, and Eric Baldeschwieler. 2013. Apache Hadoop YARN: Yet Another Resource Negotiator. In *Proceedings of the 4th Annual Symposium on Cloud Computing (SOCC '13)*. ACM, New York, NY, USA, Article 5, 16 pages. https://doi.org/10.1145/2523616.2523633 Accessed: 201-04-26.

[31] Gregor von Laszewski, Geoffrey C. Fox, and Judy Qiu. 2018. *Handbook of Clouds and Big Data – Theory and Practice* (2 ed.). Bloomington, IN 47408. http://cyberaide.org/papers/vonLaszewski-bigdata.pdf

[32] Wikipedia. [n. d.]. Apache Hadoop. Web Page. ([n. d.]). https://en.wikipedia.org/wiki/Apache_Hadoop Accessed: 2018-04-25.

[33] Wikipedia. [n. d.]. Docker (software). Web Page. ([n. d.]). https://en.wikipedia.org/wiki/Docker_(software) Accessed: 2018-04-26.

# REST Abstract File System Service

Swarnima Sowani
Indiana University
Smith Research Center
Bloomington, IN 47408
shsowani@iu.edu

Gregor von Laszewski
Indiana University
Smith Research Center
Bloomington, IN 47408, USA
laszewski@gmail.com

## ABSTRACT

We developed a REST Abstract File System Service which is a web services that implements and abstracts the underlying storage services and provides a uniform web services APIs for users to do file operations like, retrieving, storing, removing files on storage services. The service is provided to support storage engines such as Amazon Simple Storage Service, google drive and virtual machines. This service can be used by big data application clients to perform file operations with consolidated data view.

## KEYWORDS

hid-sp18-420, Volume: 9, Chapter: REST, Status: 100.
File system, abstraction, Amazon S3, Google drive, virtual machine

## 1 INTRODUCTION

File system operations are an integral part of the operating system as well as cloud services. They are a quintessential part of any big data applications that deals with storing, retrieving or reading the files. Modern distributed applications are likely to have data stored at different locations with different storage systems. Managing the data at the different locations is a difficult task.

There are numerous storage providers who facilitate their customers to store, read, write, retrieve files. However, integrating with each one of them is a potential tedious tasks since, understanding and implementing APIs for each service is a time consuming.

This motivates the development of our *REST Abstract File System Service*. It provides a layer of abstraction over the underlying file system so that the users or customers can integrate storage services as file systems like Amazon S3 [2], Google Drive [3] and other such services without even knowing their implementation details. Because of the abstraction, backend services can easily be switched while the client application uses the same API to access the various services.

The general layered architecture of the Service is depicted in Figure 1. Client applications can interact through an abstract file system with the file system services hosted by a variety of storage vendors. With the APIs, it is easy to perform file operations such as listing, uploading, downloading and deleting in a uniform way with supported storage engines. Consequently through the abstraction layer, all file operations will follow the same process for all storage engines and it will be as easy as performing operations on local system.

An example shown in Figure 2 identifies the ability to not only use the services in parallel, but also use them in conjunction. While files are stored on different backend systems, the REST file system abstraction makes them appear to be in the same file system.

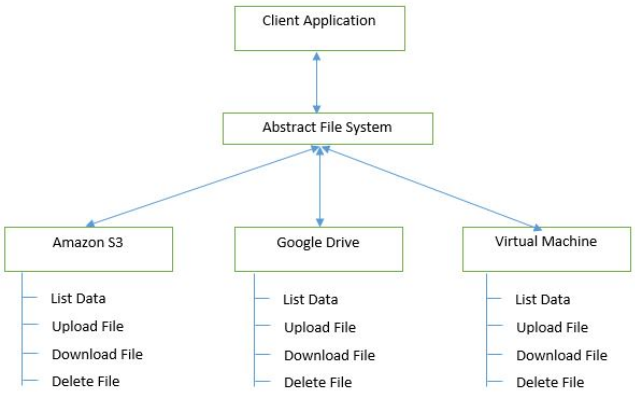


**Figure 1: Abstraction Layer**

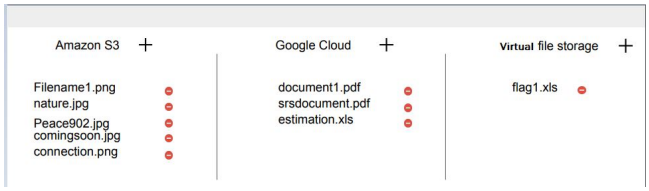


**Figure 2: Sample Client Application**

## 2 ARCHITECTURE

Figure 1 refered in introduction shows high level architecture of REST Abstract File System Service project. Client applications can interact with different storage systems in an abstract manner by using APIs provided by REST Abstract File System Service. Each storage system used in this project performs listing, uploading, downloading and removing file operations. There are three storage systems used currently. Hence, there are total twelve REST APIs used in this project with one API per file operation per storage system.

OpenAPI document is used to provide specifications of REST APIs. It is yaml document in JSON format. It contains details of all APIs such as path to access API, all input and output parameters for particular API and other specification details. We can define which parameters are required. This OpenAPI document is used by Swagger codegen to create documentation and generate code stubs in desired programming language. Python is used here as an underlying language for development. Java is pre-requirement to generate code using swagger. For this project we have used swagger version 2.0 and used the python scripts generated by swagger codegen to

develop REST API. It will be easy to integrate new storage engine in this system. To include new system it is required to provide API specification in yaml file and then using swagger codegen can be used to create service stubs in python where implementation logic can be written.

## 3 IMPLEMENTATION

REST Abstract File System Service project is built to provide an abstraction layer to perform file operations on different storage systems. It is currently supporting Amazon S3, Virtual machine storage and Google drive as its underlying storage engines. It is developed to perform file operations such as listing of existing files, removing a specific file, uploading a new file and downloading a specified file.

### 3.1 Technologies used

The REST Abstract File System Service project is developed based on different technologies. It is using Python as am implementation language and APIs are generated using Swagger Codegen by reading OpenAPI document.

*3.1.1 Python.* Python 2.7 is the underlying language that is used to implement all the APIs provided by the REST Abstract File System Service. There are 3 main python files having implementation of file operations to be performed with Amazon S3, Google drive and Virtual machine storage respectively. A number of python modules are used to support the implemented functionality. Some of the important modules are

**Boto3:** Boto3 is a software development kit (SDK) that provides AWS interface for Python applications. The REST Abstract File System Service is using Boto3 which is the latest version of the SDK provided. Boto 3 supports Python versions 2.6.5, 2.7 and 3.3 version [1]. It is used to support Amazon S3 interface with the REST Abstract File System Service.

**ftplib:** The ftplib module in Python is used to perform different file opearations on virtual machine by connecting to FTP server installed on that machine [4].

**yaml:** This library is used to read the confog.yaml file used in the REST Abstract File System Service project to store the configurations required by different storage engines.

**google-api-python-client:** This is a python specific client library provided by google to handle the Google drive APIs.

These python modules are used in different python files to improve implemetation of abstract file system.

*3.1.2 Swagger.* Our *REST Abstract File System Service* service is using Swagger Codegen to generate server side stubs based on the abstractFileSystem.yaml file with swagger specifications. This yaml file is an OpenAPI document which is in JSON format. It contains swagger version, and description of REST APIs. Swagger was then used to install and run REST service. It enabled APIs for accessing storage systems provided in this project.

### 3.2 Supported Storage Engines

The REST Abstract File System Service Service developed by us, is currently supporting three types of storage engines which are Amazon S3, Google drive and any virtual machine.

*3.2.1 Amazon S3.* Amazon S3 stands for Simple Storage Service. It is the most popular storage service provided by Amazon Web Service [2]. It provides a highly scalable, reliable, and low latency data storage infrastructure at low costs. Amazon S3 can be used to store and retrieve data of any kind and any amount from anywhere. Amazon S3 is known for its durability and stability. Amazon S3 SLAs claims to provide 99.999999999% durability for files stored in its durable storage [2]. Amazon S3 is popularly an object storage, where each file is treated as an object. Amazon S3 claims no cap on the amount of data that can be stored, that is why companies who needs scale on the go prefer to choose Amazon S3 as their file or object storage.

The REST Abstract File System Service encashes on the scalability and simplicity Amazon S3 provides by abstracting the underlying mechanism and providing a way to users to store their data into the durable S3 storage.

File operations supported by the REST Abstract File System Service includes,

- Listing of files inside a specified bucket (bucket logical segregation of files)
- Retrieving a specific file
- Storing file in the specified bucket
- Removing the file from the bucket

In order to use Amazon S3 as a storage system, first user needs to have an account with AWS system.

To create new root account with AWS,

(1) Go to AWS console and create new account.
(2) Give details such as email address, password and user name.
(3) Contact details: Give all specified contact details.
(4) Payment Information: Give credit card or debit card details for user verification by AWS side.
(5) Phone verification: AWS will make a call on the given number and give a 4 digit code to verify your phone number.
(6) Select a support plan and Continue.

It is required to create new user in AWS account and give programmatic access so that the account credentials can be used for configuring with project.

(1) Login to AWS account.
(2) Go to Services and select IAM from drop down. IAM stands for Access and Identity Management.
(3) Inside IAM resources, click on User.
(4) This will show the list of users. Click on Add User to add a new user.
(5) Provide user name and select programmatic access.
(6) On next page, provide optional permissions to user such as S3 Full access and similar.
(7) On next page, review all settings and click Next.
(8) Access key and secret key are displayed here. Save it somewhere safe. Key file can be downloaded here in ċsv format.

To use Amazon S3 as a storage engine within Abstract File System, access key and secret key credentials must be provided in the config.yaml file. The REST Abstract File System Service requires user to provide four parameters to configure and start using Amazon S3 service. (1) Access Key, (2) Secret Key, (3) AWS region, and (4) Bucket name.

Access key and secret key are required for the programmatic authentication of using S3 services. These keys are provided at the time of user account creation with a role to access Amazon S3. This can be completed by following steps to create new user in AWS account and give programmatic access section of this document. Amazon S3 stores data within resources called bucket. Buckets are logical segregation of files. Bucket name is configurable from the config.yaml file which specifies the location where all file operations can be performed. While creating a new bucket, an AWS region is assigned to that bucket. These buckets are globally unique and are accessible from anywhere. They are placed in specific AWS region. AWS region is an area provided by AWS where AWS resources available.

*3.2.2 Google Drive.* With 3 billion file uploads every day, Google drive is certainly one of the most popular and widely used storage provided by Google. Google drive offers its user a storage of 15 gigabytes for free and 100 gigabytes, 1 terabytes, 2 terabytes, 10 terabytes, 20 terabytes and 30 terabytes through optional paid plans. Files uploaded in Google drive can be up to 5 terabytes in size. Google drive provides 99.9% up-time guarantee [5].

The REST Abstract file system service provides integration with Google drive so that they can use their existing account or new one to store files seamlessly.

REST Abstract file system service provides integration with Google drive by the means of API keys.

File Operations Supported by the REST Abstract File System Service includes,

- Listing of files inside a drive
- Retrieving a specific file
- Storing file inside drive
- Removing the file from drive

REST Abstract file system service requires user to login to their Google account only for the first time of using this application and all later requests will not require to authenticate again. For the first request to Google drive API via REST Abstract file system service, it will open a browser for authentication. An existing account or new account with Google is required for authentication. User needs to perform authentication first and then Google drive will ask user to allow or deny Abstract File system project to access the Google drive.

Figure 3 shows the screenshot of how authentication is required by application to access the Google drive data. The project registered with Google drive API has name file-system. Hence, Figure 3 shows that Google drive is asking user to allow or cancel file access to file-system application.

Once user allows access to Abstract File system, user specific API keys will be generated and stored inside google-drive-credentials.json file. For all later accesses of Abstract file system drive API, these keys will be referred and there will be no need to login again.

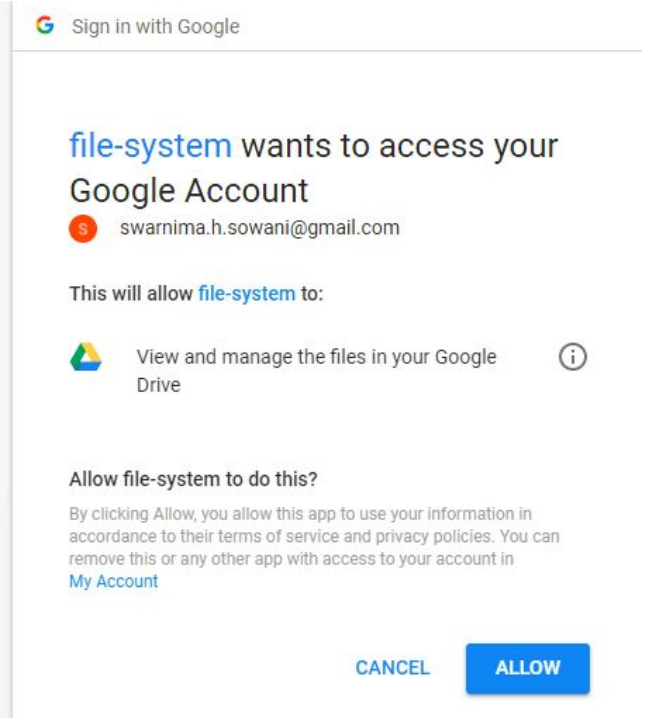


**Figure 3: Aythentication for Google Drive**

*3.2.3 Virtual Machine.* The virtual machine on any cloud service provider can be used as a storage. In certain scenarios, user needs to store the files on their specific servers for variety of reasons. REST Abstract file system Service facilitates storing the files on the server of their choice.

There can be multiple ways to access the storage provided by virtual machines.

One way is to write an http service accept and transfer file and storage on local file system and host this service on the virtual machine. Second way could be to do SSH or SCP by writing a sub-process for making a connection to machine using machines IP, port and authentication through username, password or `.pem` files. Third way is to do FTP installation on virtual machine and using the newly created FTP user in the program to access files on the virtual machine.

The REST Abstract File System Service facilitates integration with any virtual machine by the means of ftp installation. FTP is a file transfer protocol and is a standard mechanism to deal with remote file operations.

File Operations Supported by the REST Abstract File System Service include,

- Listing of files inside a specified folder allowed by ftp
- Retrieving a specific file
- Storing file inside specified folder
- Removing the file from the folder

To use storage of some virtual machine, it is important to perform FTP installation on that server.

3

Steps to install ftp on server [4] (Ubuntu Linux distribution is taken as example)

```
$ sudo apt-get install vsftpd
```

Edit the configuration file to allow writes to file system as well as set the default file system path for file operations as `/srv/ftp`

```
$ sudo service vsftpd restart
```

Make the ftp directory `/srv/ftp` writable

It is required to create a local file system user on the Virtual Machine, you can do that by using following command

```
local_root = /srv/ftp

write_enable=YES

$ sudo adduser your_user_name
```

This will ask for password and other details and will create a local user on the virtual machine which can access the ftp installations. Now everything is ready and REST Abstract file system Service can make connection with the file system of virtual machine.

Use `config.yaml` file in for REST Abstract File System Service to add the required details of Virtual machine. It will be required to provide details in config.yaml files to fill out hostname, port, username, and password. Host name is an IP address of the virtual machine which hosts the ftp installation. Port number is used to connect to the FTP on given port. By default port 21 is used for connection, and finally username and password of the FTP user created in FTP installation.

### 3.3 Project Artifacts

Thge REST Abstract File System Service uses different files. Each one of them is used for some specific function in this project.

**abstractFileSystem.yaml:** This file is used for giving swagger specification for all the APIs used in the REST Abstract File System Service. This file will be used by Swagger code generator to create python stubs as per specification given in the yaml file. This file contains swagger version, description, base path, path for each APIs respective input and output parameters of each API respectively.

**config.yaml:** This file is used to store the configurations required in the project. Config.yaml file is used to store access key, secret key, bucket name and region for integrating Amazon S3 in the system. It also stores host name, port, username and password required to integrate virtual machine as a storage engine in the REST Abstract File System Service project.

**client_id.json:** This file is used by Abstract File system to enable using google drive api in the project. This file is generated at the time of registering Abstract File System for API of Google drive. It has the key information that enables usage of google drive in this application.

**googledrivecredentials.json:** This file will store the keys of client who will be using Abstract File System. When any drive API is called for first time, it is required to login to the google drive and allow the REST Abstract File System Service to access the user drive. After approval, user specific keys will be stored in google-drive-credentials.json file

**MakeFile:** MakeFile has different targets and it is used to build the project. Make service is used for performing all the configurations required for the REST Abstract File System Service and then using swagger-codegen to generate code using swagger specification file abstractFileSystem.yaml. It will then copy all python files to the controller and download the requirements. Make start is used to run the service. Make clean is used to clean all the changes done while configuring the project.

MakeFile also supports docker implementation. Make container is used to call docker-build and docker-start target which will generate docker container and run project services in the new container. Make docker-clean is used to stop the service and remove the container.

**DockerFile:** DockerFile is used to generate a docker image for the REST Abstract File System Service so it can be run. It has specifications required for generating Ubuntu instance along with installation of python, java, curl, git, wget and required files to generate an image with all required installations ready. It then uses git repository by cloning inside image and then running a project based on targets available in the make file.

**all_drive_operations_controller.py:** This is a python file that contains the code to implement required file operations for using Google drive as a storage engine in the REST Abstract File System Service.

**all_s3_operations_controller.py:** This is a python file that contains the code to implement required file operations for using Amazon S3 as a storage engine in the REST Abstract File System Service.

**all_vm_operations_controller.py:** This is a python file that contains the code to implement required file operations for using virtual machine as a storage engine in Abstract File System.

## 4 RESULTS

When no configurations present in config.yaml file, no storage system will be accessible it will be displayed as message in API result.

Figure 4 shows an output generated when none of the storage engines is configured within the REST Abstract File System Service.

All files listed from all storage engines,

APIs used:

- /cloudmesh/file-system/S3/listData
- /cloudmesh/file-system/VM/listData
- /cloudmesh/file-system/drive/listData

Figure 5 shows result of output by running make test command. Make test calls this listData API of all storage engines. Hence, it is displaying files from all 3 storage engines which are Amazon S3, Google Drive and Virtual Machine.

To check file operations supported by the REST Abstract File System Service to integrate with Amazon S3 these APIs are used,

- /cloudmesh/file-system/S3/listData

```
root@ip-172-31-18-49:project-code# make test
curl http://localhost:8080/cloudmesh/file-system/S3/listData
"Amazon S3 is not configured correctly in /etc/config.yaml file"
curl http://localhost:8080/cloudmesh/file-system/drive/listData
"Google drive not configured to use Abstract File System"
curl http://localhost:8080/cloudmesh/file-system/VM/listData
"Virtual Machine is not configured correctly in /etc/config.yaml file"
```

**Figure 4: No storage system configured**

```
curl http://localhost:8080/cloudmesh/file-system/S3/listData
[
  "file.txt",
  "image1.JPG",
  "swarnima-paper.tex",
  "swarnima1.JPG",
  "welcome.txt",
  "vonLaszewskibigdata.pdf"
]
curl http://localhost:8080/cloudmesh/file-system/drive/listData
[
  "vonLaszewskibigdata.pdf",
  "vonLaszewskibigdata2.pdf",
  "abstractFileSystem-implementation-video.wmv",
  "abstractFileSystem-docker.wmv",
  "project",
  "welcome.txt",
  "swagger-code.wmv",
  "requirements.txt",
  "image.JPG",
  "Intermediate Project Results - Submission",
  "softwares_Astro.rar",
  "Dump20180402.rar",
  "swarnima.zip",
  "vonLaszewski-cloud-vol-7.pdf",
  "Blazelogo.PNG",
  "Download2s.rar",
  "IV_7March18.rar",
  "Cloud_7Mar18.rar",
  "Documents.rar",
  "Desktop.rar"
]
curl http://localhost:8080/cloudmesh/file-system/VM/listData
[
  [
    "file.txt",
    "hello_swarnima.txt",
    "image1.JPG",
    "setup.py",
    "welcome.txt",
"vonLaszewskibigdata.pdf"
  ]
]
```

**Figure 5: all files from all engines**

- /cloudmesh/file-system/S3/downloadFile/fileName
- /cloudmesh/file-system/S3/uploadFile/fileName
- /cloudmesh/file-system/S3/deleteFile/fileName

Figure 6 shows all file operation performed by REST Abstract File System Service with Amazon S3 as an underlying storage engines.

To check all file operations supported by the REST Abstract File System Service to integrate with Virtual Machine storage these APIs are used,

- /cloudmesh/file-system/VM/listData
- /cloudmesh/file-system/VM/downloadFile/fileName
- /cloudmesh/file-system/VM/uploadFile/fileName
- /cloudmesh/file-system/VM/deleteFile/fileName

Figure 7 shows all file operation performed by Abstract File System with Virtual Machine as an underlying storage engines.

To check all file operations supported by the REST Abstract File System Service to integrate with Google Drive these APIs are used,

- /cloudmesh/file-system/drive/listData
- /cloudmesh/file-system/drive/downloadFile/fileName
- /cloudmesh/file-system/drive/uploadFile/fileName
- /cloudmesh/file-system/drive/deleteFile/fileName

5

```
root@ip-172-31-18-49:/project-code# curl
http://localhost:8080/cloudmesh/file-system/S3/uploadFile/file.txt
"File uploaded"
root@ip-172-31-18-49:/project-code# curl
http://localhost:8080/cloudmesh/file-system/S3/listData
[
  "file.txt",
  "image1.JPG",
  "swarnima-paper.tex",
  "swarnima1.JPG",
  "welcome.txt"
]
root@ip-172-31-18-49:/project-code# curl
http://localhost:8080/cloudmesh/file-system/S3/downloadFile/file.txt
"file downloaded"
root@ip-172-31-18-49:/project-code# curl
http://localhost:8080/cloudmesh/file-system/S3/deleteFile/file.txt
"File deleted"
root@ip-172-31-18-49:/project-code# curl
http://localhost:8080/cloudmesh/file-system/S3/listData
[
  "image1.JPG",
  "swarnima-paper.tex",
  "swarnima1.JPG",
  "welcome.txt"
]
```

**Figure 6: all S3 operations**

Figure 8 shows output of file upload operation performed by the REST Abstract File System Service with Google Drive as an underlying storage engines.

Figure 9 shows output of file delete operation performed by the REST Abstract File System Service with Google Drive as an underlying storage engines. It first shows existing list of files then delete API call and then displays modified list of files.

Figure 10 shows output of file download operation performed by the REST Abstract File System Service with Google Drive as an underlying storage engines.

## 5 BENEFITS OF THE REST ABSTRACT FILE SYSTEM SERVICE

REST Abstract File System Service is a simple application providing different benefits.

(1) Since the REST Abstract File System Service abstracts the underlying storing mechanism, there is no need for client application to know the core implementation of the storage services like Amazon S3, Google Drive SDKs etc
(2) Switching between underlying storage mechanism is a hassle free task and can be done even by a newbie.
(3) Once abstract file system project is integrated into client applications, there is no code change required apart from only the configuration changes which are specific to underlying storage mechanism.
(4) Since the underlying storage mechanisms can be anything like Amazon S3, Google Drive which provides huge amount of data to be stored and retrieved on demand, scalability is the biggest benefit of abstract file system project
(5) Abstract File system is currently dealing with only 3 types of storage engines which are Amazon S3, Google Drive and Virtual machines. It is flexible to add other storage systems that can be configurable from config.yaml file and API specification provided in abstractFileSystem.yaml file.

## 6 BENCHMARKS

Abstract File system project uses Amazon S3, Google drive and Virtual machine as its underlying storage engines. There are total 4 APIs provided for each of the storage engine. The APis include listing of files, file upload, download and deletion respectively.

In order to make use of this project, it is required to have access to all these storage systems. To use Amazon S3, it is required to have an account in AWS with a bucket created in S3 where file operations can be performed. To use Google drive as a storage system in this project, user needs to have an account in Google and it is required to login to it when any of the Google drive API is called from the project for first time. To use Virtual machine as a storage system, it is required to complete FTP installation on the virtual machine and then the credentials can be provided in the config.yaml file present in the project to perform file operations on virtual machine. In case if any of the configuration is not available, abstract file system APIs call to the respective storage engine will provide a response message saying that this storage system is not configured.

The performance of all APIs is dependent upon the size of the file and the speed of internet. System performance was tested with internet speed of 4.29 mbps for upload and 6.1 mbps for download To test the performance of list API to list files from each of the storage engines, Amazon S3 took 0.01 seconds, Google drive took 0.02 seconds and Virtual machine took 0.01 seconds. To check the upload and download APIs for each of the storage engines, a file with 87,856 kilobyte is used.

Figure 11 shows records of time taken by file upload operation on different storage systems

```
root@ip-172-31-18-49:/project-code# curl
http://localhost:8080/cloudmesh/file-system/VM/listData
[
  [
    "file.txt",
    "hello_swarnima.txt",
    "image1.JPG",
    "setup.py",
    "welcome.txt"
  ]
]

root@ip-172-31-18-49:/project-code# curl
http://localhost:8080/cloudmesh/file-system/VM/uploadFile/von
Laszewskibigdata.pdf
"File uploaded"
root@ip-172-31-18-49:/project-code# curl
http://localhost:8080/cloudmesh/file-system/VM/listData
[
  [
    "file.txt",
    "hello_swarnima.txt",
    "image1.JPG",
    "setup.py",
    "vonLaszewskibigdata.pdf",
    "welcome.txt"
  ]
]
root@ip-172-31-18-49:/project-code# curl
http://localhost:8080/cloudmesh/file-system/VM/downloadFile/vsetup.py
"Unable to download file"
root@ip-172-31-18-49:/project-code# curl
http://localhost:8080/cloudmesh/file-system/VM/downloadFile/setup.py
"File downloaded"
root@ip-172-31-18-49:/project-code# curl
http://localhost:8080/cloudmesh/file-system/VM/deleteFile/setup.py
"File deleted"
root@ip-172-31-18-49:/project-code# curl
http://localhost:8080/cloudmesh/file-system/VM/listData
[
  [
    "file.txt",
    "hello_swarnima.txt",
    "image1.JPG",
    "vonLaszewskibigdata.pdf",
    "welcome.txt"
  ]
]
```

**Figure 7: all VM operations**

Table 1 shows outcome of the performance test with number of seconds taken to execute all four APIs by different storage engines respectively.

**Table 1: Processing time of all APIs for different storage engines**

| APIs | Amazon S3 | Google Drive | Virtual Machine |
|---|---|---|---|
| Listing | 0.01 | 0.02 | 0.01 |
| Upload | 0.14 | 0.24 | 0.05 |
| Download | 0.13 | 0.18 | 0.06 |
| Delete | 0.04 | 0.06 | 0.02 |

## 7 CHALLENGES AND LIMITATIONS

The REST Abstract File System Service project requires a strong internet connection. Performance of the system is dependent on the speed of internet.

The file name is passed as a URL parameter for APIs such as file upload, file download. REST Abstract file system Service is not supporting multiple file upload and download at a time. Only one file can be uploaded or downloaded or deleted with one API call.

No special security considerations are implemented other than those provided by the underlying subsystems.

## 8 CONCLUSION

The REST Abstract File System Service provides uniform APIs to perform file operations on different storage systems in an abstract way. The REST Abstract File System Service currently supports

```
root@ip-172-31-18-49:/project-code# curl
http://localhost:8080/cloudmesh/file-system/drive/listData
[
  "welcome.txt",
  "vonLaszewskibigdata.pdf",
  "vonLaszewskibigdata.pdf",
  "abstractFileSystem-implementation-video.wmv",
  "abstractFileSystem-docker.wmv",
  "project",
  "welcome.txt",
  "swagger-code.wmv",
  "requirements.txt",
  "image.JPG",
  "Intermediate Project Results - Submission",
  "softwares_Astro.rar",
  "Dump20180402.rar",
  "swarnima.zip",
  "vonLaszewski-cloud-vol-7.pdf",
  "Blazelogo.PNG",
  "Download2s.rar",
  "IV_7March18.rar",
  "Cloud_7Mar18.rar",
  "Documents.rar"
]
root@ip-172-31-18-49:/project-code# curl
http://localhost:8080/cloudmesh/file-system/drive/uploadFile/welcome2.txt
"File uploaded"
root@ip-172-31-18-49:/project-code# curl
http://localhost:8080/cloudmesh/file-system/drive/listData
[
  "welcome2.txt",
  "welcome.txt",
  "vonLaszewskibigdata.pdf",
  "vonLaszewskibigdata.pdf",
  "abstractFileSystem-implementation-video.wmv",
  "abstractFileSystem-docker.wmv",
  "project",
  "welcome.txt",
  "swagger-code.wmv",
  "requirements.txt",
  "image.JPG",
  "Intermediate Project Results - Submission",
  "softwares_Astro.rar",
  "Dump20180402.rar",
  "swarnima.zip",
  "vonLaszewski-cloud-vol-7.pdf",
  "Blazelogo.PNG",
  "Download2s.rar",
  "IV_7March18.rar",
  "Cloud_7Mar18.rar",
  "Documents.rar"
]
```

**Figure 8: Drive upload operations**

Amazon S3, Google drive and Virtual machine as its storage engine. It can be scalable to include other services as well.

Using the REST Abstract File system service allows to simplify the tedious task to manage files between different storage providers. This is achieved while using the same API by the client accessing different storage engines in a seamless way.

By using the abstraction layer, storing or retrieving files from different storage engines will be same process as being saved locally or transferred over the network to other storage system. Such abstraction makes it simpler to switch from one system to another without having to rewrite vast swathes of application code.

## ACKNOWLEDGMENTS

The author would like to thank Dr. Gregor von Laszewski for his support and suggestions to write this paper.

## REFERENCES

[1] Amazon. 2014. Boto 3 Documentation. Web Page. (2014). https://boto3.readthedocs.io/en/latest/
[2] Amazon. 2018. Amazon S3. Web Page. (2018). https://aws.amazon.com/s3/
[3] Google Developers. 2018. Python Quickstart. Web Page. (2018). https://developers.google.com/drive/v3/web/quickstart/python
[4] Python. 2018. ftplib - FTP protocol client. Web Page. (2018). https://docs.python.org/2/library/ftplib.html
[5] Wikipedia. 2018. Google Drive. Web Page. (2018). https://en.wikipedia.org/wiki/Google_Drive

## A README.MD

```
# REST Abstract File System SERVICE


The REST Abstract File System Service which is a web services that
implements and abstracts the underlying storage services and provides
a uniform web services APIs for users to do file operations like,
retrieving, storing, removing files on storage services. The service
is provided to support storage engines such as Amazon Simple Storage
```

```
root@ip-172-31-18-49:/project-code# curl
http://localhost:8080/cloudmesh/file-system/drive/listData
[
  "welcome2.txt",
  "welcome.txt",
  "vonLaszewskibigdata.pdf",
  "vonLaszewskibigdata.pdf",
  "abstractFileSystem-implementation-video.wmv",
  "abstractFileSystem-docker.wmv",
  "project",
  "welcome.txt",
  "swagger-code.wmv",
  "requirements.txt",
  "image.JPG",
  "Intermediate Project Results - Submission",
  "softwares_Astro.rar",
  "Dump20180402.rar",
  "swarnima.zip",
  "vonLaszewski-cloud-vol-7.pdf",
  "Blazelogo.PNG",
  "Download2s.rar",
  "IV_7March18.rar",
  "Cloud_7Mar18.rar",
  "Documents.rar"
]
root@ip-172-31-18-49:/project-code# curl
http://localhost:8080/cloudmesh/file-system/drive/deleteFile/welcome2.txt
"File deleted"
root@ip-172-31-18-49:/project-code# curl
http://localhost:8080/cloudmesh/file-system/drive/listData
[
  "welcome.txt",
  "vonLaszewskibigdata.pdf",
  "vonLaszewskibigdata.pdf",
  "abstractFileSystem-implementation-video.wmv",
  "abstractFileSystem-docker.wmv",
  "project",
  "swagger-code.wmv",
  "requirements.txt",
  "image.JPG",
  "Intermediate Project Results - Submission",
  "softwares_Astro.rar",
  "Dump20180402.rar",
  "swarnima.zip",
  "vonLaszewski-cloud-vol-7.pdf",
  "Blazelogo.PNG",
  "Download2s.rar",
  "IV_7March18.rar",
  "Cloud_7Mar18.rar",
  "Documents.rar",
  "Desktop.rar"
]
```

**Figure 9: Drive delete operations**

```
root@ip-172-31-18-49:/project-code# curl
http://localhost:8080/cloudmesh/file-system/S3/downloadFile/file.txt
"file downloaded"
```

**Figure 10: Drive download operations**

```
Service, google drive and virtual machines. This service can be used
by big data application clients to perform file operations with
consolidated data view.


File operations supported by Abstract File System for each storage
engine include:

1. Listing of files
2. Retrieving specific file
3. Storing file to specified folder
4. Removing file

## Prerequisites for Execution

In order to use different storage systems, it is required to have
all the systems available and there configurations provided in
etc/config.yaml file.

To use Amazon S3, it is required to provide access key, secret key,
bucket name is required.
```

```
Amazon S3 upload
Start time: 14:59:49
End time: 15:00:00
Time required: 0.11 seconds


root@ip-172-31-18-49:/project-code# date
Wed May  2 14:59:49 UTC 2018
root@ip-172-31-18-49:/project-code# curl
http://localhost:8080/cloudmesh/file-system/S3/uploadFile/von
Laszewskibigdata.pdf
"File uploaded"
root@ip-172-31-18-49:/project-code# date
Wed May  2 15:00:00 UTC 2018

Virtual Machine upload
Start Time: 14:48:26
End Time: 14:48:31
Time required: 0.05 seconds


root@ip-172-31-18-49:/project-code# date
Wed May  2 14:48:26 UTC 2018
root@ip-172-31-18-49:/project-code# curl
http://localhost:8080/cloudmesh/file-system/VM/uploadFile/von
Laszewskibigdata.pdf
"File uploaded"
root@ip-172-31-18-49:/project-code# date
Wed May  2 14:48:31 UTC 2018

Google Drive upload
Start Time : 15:05:12
End Time: 15:05:36
Time required: 0.24 seconds

root@ip-172-31-18-49:/project-code# date
Wed May  2 15:05:12 UTC 2018
root@ip-172-31-18-49:/project-code# curl
http://localhost:8080/cloudmesh/file-system/drive/uploadFile/von
Laszewskibigdata.pdf
"File uploaded"
root@ip-172-31-18-49:/project-code# date
Wed May  2 15:05:36 UTC 2018
```

**Figure 11: File upload performance test**

```
To use Virtual machine storage it is required to install FTP on that
machine and create a user with read write access to specified folder.
These specifications such as host name, port, FTP user name and password
should be provided in config.yaml file

To use google drive storage, it is required to have an Google account.
When any drive API is used for first time, it will open browser and ask
user to login to their google drive account and allow access to file-system
project.


## Test

To test the service follow these steps



### 1. Using Docker

git clone https://github.com/cloudmesh-community/hid-sp18-999.git

cd project-code

Use command   make container

This will internally call   make docker-build   and   make docker-start

Once docker build is successful and container is created, run

    make test

to check the services. Run

    make docker-clean

to remove docker image



### 2. Using make file

git clone https://github.com/cloudmesh-community/hid-sp18-999.git

cd project-code

Use command   make   to generate code

Use command   make test   to test the code

Use command   make clean   to clean the code


### 2. Using swagger code generator

To use Abstract File System, below python libraries needs to be installed

* pip install boto3
* pip install pyyaml
* pip install --upgrade google-api-python-client


Follow these steps to run Abstract File System project

git clone https://github.com/cloudmesh-community/hid-sp18-999.git
```

```
cd project-code

wget
http://central.maven.org/maven2/io/swagger/swagger-codegen-cli/2.3.1/swagger-cod
egen-cli-2.3.1.jar

java -jar swagger-codegen-cli-2.3.1.jar generate  -i abstractFileSystem.yaml
-l python-flask  -o server/file-system/flaskConnexion  -D supportPython2=true

This will generate server stubs. copy all_drive_operations_controller.py,
all_s3_operations_controller.py and all_vm_operations_controller.py files to
  /server/file-system/flaskConnexion/swagger_server/controllers

cd server/file-system/flaskConnexion

pip install -r requirements.txt

python setup.py install

python -m swagger_server

A message will be displayed as
           * Running on http://0.0.0.0:8080/ (Press CTRL+C to quit)


Test on broswer or make curl calls to test

     curl http://localhost:8080/cloudmesh/file-system/S3/listData
     curl http://localhost:8080/cloudmesh/file-system/drive/listData
     curl http://localhost:8080/cloudmesh/file-system/VM/listData

These calls will list all data in Amazon S3, Google drive and Virtual machine
respectively
#### Output

Video to display working of this project

<https://drive.google.com/file/d/11FCSg7vU61l9Mm6_M4P1JJxVi62Yn5f1/view>

Docker implementation with no data in config.yaml file

<https://drive.google.com/file/d/1L0dJSE4IQ0RI2_wcI4UTIxAKEyxSPs2K/view>
```

# Swagger Service for Openstack Resources

Arnav Arnav
Indiana University
Bloomington, IN 47408, USA
aarnav@iu.edu

## ABSTRACT

Openstack is one of the most important cloud coputing Infrastructure as a Service that allows users to setup and manage servers easily. It provides APIs that help users to manage these services through command-line and through code. The NIST Big Data Reference Architecture defines a reference architecture for various objects that are important in the context of cloud computing. In this project we aim to build a REST service using these object specifications that help users complete an openstack deployment easily.

## KEYWORDS

hid-sp18-503, Volume: 9, Chapter: REST, Status: 100.
Swagger, Openstack, REST-API, Cloud Computing

## 1 INTRODUCTION

As more and more appliations rely on data generated by their users, the importnce of cloud computing is rapidly growing due to the need to process this data and extract useful information. Openstack is a free and open source cloud Infrastructure as a Service (IaaS) that allows users to set up resources such as virtual servers for their customers. It is one of the most widely deployed cloud infrastructure and is constantly improving. It provides an openstack python software development toolkit (sdk) that contains python APIs for openstack services along with command line tools that allow users to write python code to remotely perform actions on the openstack cloud [21].

Representational State Transfer (REST) is an architectural standard for web applications that allows different applications to comunicate easily. It allows separation of clients and servers and the server funcionality can be esily changed without affecting the client's interface to the server. REST is highly scalable and easy to extend [3]. The NIST Big Data Reference Architecture defines standard properties for objects that that are important to Big Data ecosystem.

These objects can be easily accessed, and applications using these objects can comunicate seemlessly with the use of RESTful Applications that use REST verbs (GET, PUT, POST, UPDATE, DELETE) to manipulate these objects [3].

We aim to develop a REST service with the help of Swagger - a tool that converts a yml specification to code - that can be used by other applications to setup and manage Virtual Machines on openstack.

Doing this requires identification of various objects that are needed to accomplish such a task. Some of these are images, servers, keypairs, networks, and subnets. Each of these objects should be accessible from the REST application, so that they can be updated and stored as necessary.

## 2 NIST BIG DATA REFERENCE ARCHITECTURE

The field of big data is relatively new and there are many definitions of what big data technologies are. NIST Big Data Public Working Group (PWG) has been working on formalizing a standard of what consitutes a Big data ecosystem and what are the components that are needed to define and setup such an ecosystem. These components have been defined in the NIST Big Data Reference Architecture (NBDRA) document (Volume 6). In order to facilitate communication between these components, there is the need for a comon well defined interface, which has been defined in the Big Data Ineroperability Framework document (Volume 8) [1].

The document identifies what are the different requirements that such an interface should satisfy. The interfaces should be vendor agnostic, be reusable in different case specific scenarios and should allow users to plug in resources as needed. The document investigates the requirements of different stakeholders in a big data ecosystem and aims to develop a specification paradigm that fulfils most of these requirements. The document proposes speciications for various objects that are required for a big data ecosystem to function properly and provides examples of these specifications [1].

## 3 OPENSTACK

Openstack is an opensource cloud Infrastructure as a Service platform oiginally built by the colaboration of NASA and Rackspace hosting and is now maintained by Rackspace. The platform consists of various components that allow users to contro diverse multi-vendor computing and storage hardware with the help of web based user interface or through applications with the hellp of APIs. The Project was started in 2010 with the aim of creating an opensource platforms that would help organizations offer cloud services on standard hardware. The openstack community has grown to an active collaboration of more than 500 companies and works around a six month cycle with frequentmilestones [21].

### 3.1 Openstack Services

Openstack consists of various services that are responsible for different operations. Each of htese services are important from a users viewpoit and are described below:

- **Keystone**: Keystone is the identity management system for openstack. It is responsible for authentication and high level authorization. It is this service that generates authentication tokens a and identifies users with the help of passwords and provides the need for users to identify themselvs in their applications. Keystone can be integrated with third party services such as Lightweight Directory Access Protocol (LDAP) [17] [4].

- **Nova**: Nova is the compute service from Openstack, and is responsible for the management, creation and deletion of virtual machines with the help of various daemons running on linux. Nova allows administrators to choose the hypervisor of their choice while providing a common API to manage VMs. Nova requires Keystone, Glance and Neutron to work [17] [11].
- **Glance**: The service responsible for managing and cataloging images that are uploaded by the users. Glance was built to be the store for objects required by other services, and the support fot other objects can be added later [17] [15].
- **Neutron**: Neutron is a networking service provided by openstack. It is responsible for creating networks between VMs and in addition includes a firewall, load ballancer, and allows users to create Virtual Private Networks. Neutron providesvarious virtual abstractions such as subnets and routers which work in the same manner as the physical devices [17] [10].
- **Cinder**: Cinder is the openstck *block storage as a service* responsible for managing the external storage including external volumes and Network File storage (NFS). It is highly availabe, fault taulerant and is easily recoverable after failures [6].
- **Heat**: Heat is the cloud orchestration layer on opestack. It is responsible for providing Floating IPs, managing security groups, servers and many other services. Heat allows all this to be done with the help of a template file defined by the user that allows Heat to execute the correct API calls to provision the specied cloud [17] [12].
- **Horizon**: Horizon is the Graphical User Interface (GUI) dashboard that allows users to conect to openstack and and access openstack services using their web browsers [17] [7].

  Openstack provides differnet clients for each of these services that can be installed individually as needed. They provide a common unified API with the help of the openstack SDK that allows users to access most of thse sevices. Some of the services do not have complete functionality in the openstack SDK, which will be included in the near future [13].

## 3.2 Openstack SDK

The openstack SDK is a client library that alows users, and applications to interact with openstack services. It aims at providing all of the functionality that is provided by the services along with proper documentation and examples. The SDK provides an abstraction layer that abstracts away the complxities of the differences in specific openstack deployments, while also providing service specific tools when complex functions are to be performed [16].

The openstack SDK was built initially from three different librariesshade, os-client-config and python-openstacksdk. Shade was initially a part of the openstack nodepool project with some of its code written inside the Ansible openstack module. It was later consolidated and moved into *openstack.cloud* module. The philosophy from nodepool allows the users to change clouds they want to connect to in their code, without worrying much about the differences. The os-client-config was a library written to gather the configuration information from different deployments in a consistent manner. This was moved to *openstack.config* module which now reads the configuration from the *clouds.yaml* file (if available). The python-openstacksdk was a library that exposed the API to developers [16].

*3.2.1 Installing the SDK.* The sdk can be eaily installed from the Python Pagkage Index (PyPI) using the command:

```
pip install openstacksdk
```

The sdks for the individual services, if needed can be installed similarly using pip as follows:

```
pip install python-PROJECTclient
```

where PROJECT should be replaced with the name of the partcular service (such as nova)

To use the openstack sdk, and communicate with a cloud, first a connection must be created to the cloud. This can be done either by using a clouds.yaml file or by explicitly passing the variables to the *openstack.connect* method. Once a connection is made, the application can communicate to different services using the openstack sdk. These services are included in the modules named based on the service functionality for simplicity for users not familiar with openstack terminology. The neutron servce is abstracted in the module *openstack.network*, the nova service to *openstack.compute* module, the keystone service to *openstack.identity* module and so on.

## 3.3 Devstack

Devstack is a single node openstack installation which is a part of the openstack Quality Assurance (QA) project, an aims to provid tools required to install openstack from the source code to facilitate development and testing. Devstack provides a quick and easy way to install openstack for testing purposes, however it is not intended to be an openstack installer [5].

Devstack requires a minimum of of 8GB or RAM, two processor cores, an ethernet and internet access, an a supported linux version. Once Devstack has been installed, it can be used to show and test how a new functionality wroks by using the Horizon interface and through command line APIs [5].

It is suggested that devstack be setup on a virtual machine, and not natively on a developer's machine. Once a virtual machine that satisfies the requirements is set up, a new non-root user needs to be created that has sudo privileges. Following this the devstack repository can be cloned from GitHub. Executing the *stack.sh* script, installs openstack services with minimal configuration [14].

## 3.4 Openstack Container Service

In addition to providing servics to users for using virtual machines and virtual machine clusters, the Openstack Container Team developed an API called *magnum* that makes container orchestration availale to users as an openstack resource. MAgnum cluster orchestration engine uses Heat orchestration that uses an operating system image cintaining docker and kubernetes and runs that image on a Virtual machine with the help of Nova. It uses Neutron to network kubernetes containers, uses Cinder to provide volumes

for containers and allows users to chooese between Kubernetes, Docker and Apache Mesos as the cluster backend technology [8].

Magnum service consists of two parts, the magnum REST service, and the magnum-conductor. The REST server is horizontally scalable and can handle many requests at a time, while the conductor process is only single threaded at this time. All requests are stored in a message queue before magnum-conductor process can work on them [9]. All calls to the magnum service need to be authenticated with Keystone, and users can only see containers that arestarted by them. This provides an additional layer of security for the users. Magnum is intended for use by cloud providers who want to gicve users the option to create self service continers and container clusters just as easily as openstack Virtual Machines [8].

The magnum comandline and python clients can be easily installed from Python Package Index (PyPI) using pip by running the command:

```
pip install python-magnumclient
```

The magnum-client is not included in the python openstack SDK at this point and it has to be installed separately.

## 4 CHAMELEON CLOUD

Chameleon cloud is an experimental test bed for the FutureCloud project that is funded by NSF. Chameleon clou offers researchers across the US 5 PB of storage and 550 nodes physically separated across two different locations, the Texas Advanced Computing Center and the University of Chicago (UC) and encourages users to use lesser resources fr prototyping, automate their deployment and try the experiment on a larger cale to get more powerful results and hopefully publish their research [20].

Chameleon cloud comprises of a set of 12 Standsrd Cloud Units, each comprising of 42 compute nodes, 4 storage nodes with 128TB local storage and is connected to a network switch. Chameleon cloud is also equipped with numerous other hardware tyes to allow experimentation. The two locations TACC and UC are connected through a high speed internet connection [20].

## 5 SWAGGER

Swagger is an opensource set of tools for creatin APIs from openAPI specifications. Swagger project was initially owned by SmartBear and later donated to the linx foundation marking the start of the openAPI standard. Swagger follows an API first philosophy and simplifies the process of writing APIs for new and existing applications. The Swagger toolkit consists of various applications that include Swagger Codegen, Swagger Core, Swagger UI, Swagger Editor and Swagger Hub. While Swagger Codegen helps to build code stubs from the openAPI specificaton in many languages, the Swagger Inspector helps users to build openAPI Specifications from any API in the browser. The swagger community is separated from the openAPI comunity and focuses on building tools that can be used by the developers to build applications from the openAPI specification [19].

The openAPI specification is a standard and simple way to define an API that allows users to understand the functionality without having to read throughthe documentation od the entire code [18].

The openAPI standard allows specifications to be written in both yaml and JSON format. The API may include many differnet objects that can be defined in one file or in separate linked files [18].

## 6 APPROACH

The project uses all the technological concepts defined above. It comprses of a REST API developed from the openAPI specification that was used to generate code using *swager-codegen*. This allowed us to easily make updates and changes to the yaml structure without having to manually change a lot of the code.

The project in its entirity comprises of three parties - an openstack cloud (*chameleon cloud* in this case), a server running the REST API (*swagger server*) and a client. The client issues API calls to the endpoints on the swagger server, which in turn uses the openstack sdk to communicate to the cloud. The specification of the objects that is defined in the *devstack.yml* swagger specification file drives how these API calls are made, and thus is the most important part of the process.

### 6.1 Object Definitions

The following are the objects that are used in this project. These objects are inspired by the objects defined in the Nist Big Data Reference Architecture NIST (BDRA) document:

- **Flavor**: Flavors identify the different types of hadrdware available to the openstack users. A user can select a flavor based on their needs and the requirements o the image. The flavors include in their definition the RAM size, disk size, the number of virtual CPUs and the available swap memory, and are identified using a unique ID and a name.
- **Image**: An Image defines what types of operating systems are availabe to be used to create a virtual machine on openstack. Each of the images can have different minimum requirements that are needed to install the operating system and they must be included in the object.
- **Keypair**: A keypair defines a set of RSA keys that are generated by openstack to allow usere to correctly identify themselves and to log into the servers. It has two parts, a public key and a provate key and is identified with the help of a user defined name.
- **Server**: A server is another name for a Virtual Machine running on the openstack cloud. To define a server we must define the name of the server, the flavor and the image to be used, the secirity groups and networks that are needed for the server and a keypair that will be used to connect to the server in the future, along with a floating IP address.
- **Networks**: Openstack allows users to set up their own networks for their virtual machines, and alows them to create new VMs that use these networks. Networks can be internal or external and can have subnets created by the user.
- **Subnet**: Subnets allow users to identify and separate different parts of their openstack networks. A subnet object should contain a list of DNS servers and shoud identify whether it has a DHCP server or not. A minimal definition of the subnet oject is given in the appendix.

## 6.2 Available Endpoints

To run the application move into the project directory and provide correct credetnials in the clouds.yml file, thenrun the following code in the terminal:

```
make make run
```

The *make* command downloads *swagger-codegen* jarfile, runs swagger-codegen to generate the python stub for the project, and copies the *clouds.yml* and *default_controler.py* files at the right place. The *make run* command starts the server at localhost on port 8080.

The objects defined for this project can be accessed through REST API, as follows:

- **GET:** /cloudmesh/openstack/flavors: returns a list of all flavors that are available. It uses the openstack.compute module to perform this function.
- **GET:** /cloudmesh/openstack/images: returns a list of all available images. It again uses the openstack.compute module to perform this function.
- **GET:** /cloudmesh/openstack/image/name: returns the image with the specified name if there exists one.
- **GET:** /cloudmesh/openstack/networks: returns a list of all available networks. It uses openstack.network module for this API call, which communicates to the openstack neutron service.
- **GET:** /cloudmesh/openstack/networks/subnets: returns a list of all subnets available to the user.
- **GET:** /cloudmesh/openstack/keypairs: returns a list of keypairs that can be used by a user. The openstack.identity modulewas used for this function which in turn communicates to the keystone service.
- **GET:** /cloudmesh/openstack/servers: returns a list of servers (Virtual Machines) that are available on the openstack cloud
- **POST:** /cloudmesh/openstack/server/start: start the server with the given name, if the server (Virtual Machine) is present on the cloud.
- **POST:** /cloudmesh/openstack/server/stop: stop the server with the given name, if the server (Virtual Machine) is present on the cloud
- **POST:** /cloudmesh/openstack/keypair/create: create a new keypair for the nova service with the name specified in the request body
- **POST:** /cloudmesh/openstack/network/create: create a new network with the name specified in the request body
- **DELETE:** /cloudmesh/openstack/network/delete: delete the network with the name specified in the request body.
- **DELETE:** /cloudmesh/openstack/keypair/delete: delete the nova keypair with the name specified in the request body.

The project uses a Dockerfile that build an image for the service, which builds on top of the python3 image and installs all requirements Docker containers are lightweight and allow us to reproduce our project on a new machine without worrying about the specific environments and dependencies.

## 6.3 Significance

The Openstack services usuallly provide various endoints that support various REST API calls, but using these endpoints requires knowldge of openstack terminology and the knowledge of how these APIs should be used which can be complicated. Further there exist various vendors who provide openstack clouds as a service, and openstack allows administrators to make changes in how they want to deploy their cloud services, and thus the usage of endpoints for these services may differ across different openstack clouds. In such situations an application like ours provdes another layer of abstraction to the API and can allow users to use the same minimal API to manage various diffeent deployments of openstack.

Most cloud providers do not enforce user privilleges on the kinds of applications that can be run by a user in a VM. Adding another layer of abstraction can allow administrators to control, for instance, which users can install which softwares on these VMs, in an organizational setting where these softwares may be lisenced and expensive.

# 7 CONCLUSION

Openstack is an open source cloud IaaS and provides the openstack SDK that allows applications and users to commmunicate to the cloud. Swagger codegen is a tool that alllows quick development of software with the help of a yaml file that is written following the openAPI standard. Using the swagger spcification for some of the objects required for big data ecosystem, applications can be easily developed that use these objects. There exist various definitions of these objects, but it is important to develop an open standard that must be followed for consistency across applications.

# 8 IMPROVEMENTS

We define a basic application that uses the openAPI swagger specification for some of the objects that are required in a cloud based ecoystem. The project can be extended to include various other cloud services that are commonly used, while maintaining the same REST API.

The project API can be extended to provide services and functions that can be used to perform various actions, such as a Machine Learning API uses a trained model to allow users to autogenerate image captions [2].

# REFERENCES


[1] 2018. NIST Big Data Interoperability Framework: Volume 8, Reference Architecture Interface.

[2] Arnav. 2018. Image-Captioning. Github.com. (april 2018). https://github.com/seashiva94/Image-Captioning

[3] Codecademy. 2018. What is REST? Codecademy articles website. (2018). https://www.codecademy.com/articles/what-is-rest

[4] Openstack. [n. d.]. OpenStack Identity ("Keystone"). Openstack Wiki. ([n. d.]). https://wiki.openstack.org/wiki/Keystone

[5] openstack. 2016. DevStack. openstack wiki. (December 2016). https://wiki.openstack.org/wiki/DevStack

[6] Openstack. 2018. Cinder, the OpenStack Block Storage Service. openstack docs. (2018). https://docs.openstack.org/cinder/latest/

[7] Openstack. 2018. Horizon: The OpenStack Dashboard Project. openstack docs. (2018). https://docs.openstack.org/horizon/latest/

[8] openstack. 2018. Magnum. openstack wiki. (Feruary 2018). https://wiki.openstack.org/wiki/Magnum

[9] openstack. 2018. Magnum. openstack wiki. (April 2018). https://docs.openstack.org/magnum/latest/

[10] openstack. 2018. Networking (neutron) concepts. opensack docs. (2018). https://docs.openstack.org/mitaka/install-guide-ubuntu/neutron-concepts.html
[11] Openstack. 2018. OpenStack Compute (nova). Openstack Docs. (2018). https://docs.openstack.org/nova/latest/
[12] Openstack. 2018. OpenStack Orchestration. openstack heat wiki. (2018). https://wiki.openstack.org/wiki/Heat
[13] Openstack. 2018. OpenStack Python SDK. openstack docs. (2018). https://docs.openstack.org/mitaka/user-guide/sdk.html
[14] Openstack. 2018. Quick Start. Devstack docs. (April 2018). https://docs.openstack.org/devstack/latest/
[15] Openstack. 2018. Welcome to Glancefis documentation! openstack docs. (2018). https://docs.openstack.org/glance/pike/
[16] Openstack. 2018. Welcome to the OpenStack SDK! openstack docs. (2018). https://docs.openstack.org/openstacksdk/latest/
[17] oracle. 2015. OpenStack Services. docs.oracle.com. (2015). https://docs.oracle.com/cd/E64747_01/E64749/html/osusg-openstack-services.html
[18] Swagger. 2018. OpenAPI Specification. swagger.io website. (2018). https://swagger.io/specification/
[19] Swagger. 2018. What is Swagger? swagger.io website. (2018). https://swagger.io/getting-started/
[20] Gregor von Laszewski, Geoffrey C. Fox, and Judy Qiu. 2018. *Handbook of Clouds and Big Data* (2 ed.). Indiana University, Bloomington, IN 47408. http://cyberaide.org/papers/vonLaszewski-bigdata.pdf
[21] Wikipedia contributors. 2018. OpenStack — Wikipedia, The Free Encyclopedia. (2018). https://en.wikipedia.org/w/index.php?title=OpenStack&oldid=836750560 [Online; accessed 29-April-2018].

# GraphQL Web APIs

Averill Cate, Jr
The University of Indiana
Bloomington, Indiana 47408
acate@iu.edu

Gregor von Laszewski
Indiana University
Smith Research Center
Bloomington, IN 47408, USA
laszewski@gmail.com

## ABSTRACT

This project is for Advanced Cloud Computing, Indiana University Spring Semester 2018. Through this project, we will demonstrate the use of GraphQL and cloud computing services as a means for constructing and delivering web services. The project will rely on The Santa Rita Experimental Range (SRER) website and data sets maintained by the Univeristy of Arizona and second, related web site, the Walnut Gulch Experimental Watershed (WGEW) Online Data Access, maintained by the United States Department of Agriculure's, Agriculure Research Service (ARS). Neither of these existing sites provide modernized data services, yet both sites provide potentiall valuable data to the research community.

## KEYWORDS

hid-sp18-505, Volume: 9, Chapter: Messaging, Status: 100.
GraphQL, Web, Services

## 1 INTRODUCTION

What is GraphQL? GraphQL is a query language for web application programer interfaces (APIs)[6]. There have been been many other tools and methods used to develop APIs in the past. Some of those older tools are XML and SOAP, JSON and REST, plain old text or comma separated files. What makes GraphQL so different? First, as the name implies, GraphQL, is a query language. This is different compared to SOAP/XML and Swagger/REST in that those tools are a combination of data format and transport protocol.

Typically, SOAP/XML and Swagger/REST APIs are structured so that the API user needs to review the specifications for interacting with the API. For example, what functions or endpoints are available? What are the parameters for those functions and what data types apply to each parameter? GraphQL is different in that it provides a mechanism where an API becomes a domain specific language (DSL). This means that API consumers can focus on developing transactions with the API based on the data domain of the API. For example, a GraphQL API for stock data would allow API users to format queries something like the following:

```
{
   all_stocks {
     symbol, price, date
}
```

In this example we see a query request to a GraphQL API that is asking for all stock data with respective symbol, price, an date. Alternatively, a REST based query might look like the following:

```
http://server.host.com/api/v1/allstocks
```

In the REST based request we see information related to where the data are being requested from, the protocol used to make the request, the API version and finally, what the request is really after, all stock data. The latter could be considered more understandble to programmers or even to someone who is technically a advanced web users, but compared to the REST request, the GraphQL based request has a format more closely resembling the data intself and is less concered with how the data are requested.

A useful way to demonstrate the uses of GraphQL would be to apply GraphQL to the development of a real-world API that will transform existing web-based data sources created and maintained at United Stated Deparment of Agriculture web site and the University of Arizona. The first data resource is the Santa Rita Experimental which is an experimental range currently maintained by the University of Arizona. The experimental range was the first of its kind and was established in 1902[8]. The SRER website, in its current state, is not structured to deliver data as a service. The second data source is the Walnut Gulch Experimental Watershed, which is also an outdoor research facility located in southestern Arizona[7]. Like the SRER site, the WGEW preicpitation and runoff data site is not structured to delivered as services.

### 1.1 General Outline

(1) Web scraping existing data source to download, parse and clean existing data.
(2) Load the data into a database.
(3) Develop a small web application to provide two web services. The two services will deliver the same data. However, one of the services will use GraphQL[6] to deliver data and the other will use REST[9] to deliver its data.
(4) Develop a simple load testing tool that will be used to determine performance metrics for both services and compare the results of the load tests.
(5) Time permitting, create a simple Amazon Lambda service that will compute some metric related to the preciption/runoff data. The purpose is to demonstrate how cloud services can be used to off-load costs and dependencies on a distributed system like the one proposed in this project.
(6) Time permitting, develop a 3-node Raspberry Pi cluster to host the application. The purpose of this, is to demonstrate how a system like this one can be hosted and deployed on hardware infrastructure that is cheap and relatively simple to maintain.

## 2 DATA COLLECTION

Data were collected from the Santa Rita Experimental Range website. In Figure 1, we see the present day landing page for SRER. What is not apparent in the landing page is the plethora of data available data available in the site.

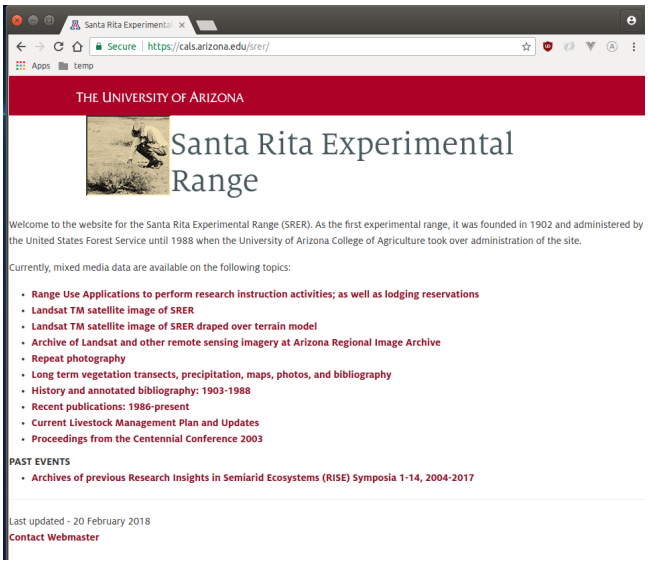


**Figure 1: SRER Page[8]**

In Figure 2 we have navigated one page deeper into the SRER site and as can be seen, meta-data and data are both availabe, but as raw text files. These data are consumable as is, but could be enhanced by being made machine readible by creating web services to deliver the data.

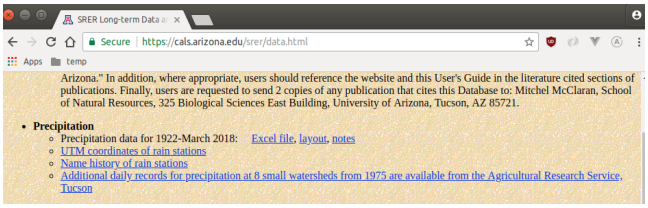


**Figure 2: SRER Precip.[8]**

## 3 DATA LOADING

The data sets downloaded for this demonstration were the raingage Universal Transverse Mercator (UTM) coordinate data files and the Excel file that contained all of the preciption data for the respective raingages. Some of the raingage data had to be cleaned mostly due to typographical errors and the UTM coordinates had to be converated to latitude and longitude values in case the data are rendered in mapping applications.

The precipitation data had to be converted from an Excel file to a CSV file and the value delimiter was changed from a comma to the pipe symbol. The reason for this is because the comma can occur in some of the descriptive text data for raingages, where as the pipe symbol is typically not used in written English.

## 4 WEB API DEVELOPMENT

After the data were prepared for import into a database, the foundation for a simple web API application was established using the following software:

(1) Docker - a container platform[5]
(2) Python - the programming language
(3) Graphene - a Python programming language library that is used to develop GraphQL APIs
(4) PostgresQL - the database server
(5) Django - a Python based model-view-controller (MVC) web development framework
(6) Django Rest Framework - a Django plugin that facilitates the development of REST APIs
(7) Graphene Django - a Django plugin that facilitates development of GraphQL APIs

### 4.1 REST/Swagger API

Generally, a REST based API endpoints, in combination with a tool like Swagger are combined to create a web interface like the one shown in Figure 3. The figure provides an example of the various API endpoints (e.g., /api/srer/v1.0/precipevents). Some endpoints require input parameters which are used to execute parameterized queries (e.g., queries that filter data) data, while other endpoints require no parameters and typically return a list of data entities.

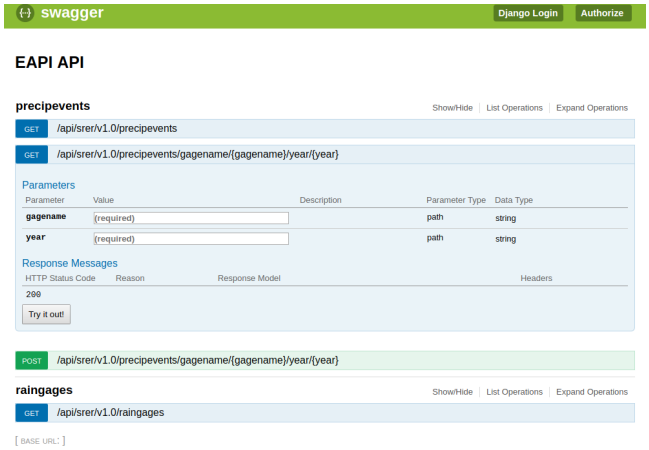


**Figure 3: REST/Swagger UI**

In Figure 4 we see an example of a REST request's response and the resulting JSON data packet. The JSON request is relatively clear. Based on the request url in Figure 4 we know the domain name (localhost), API context (srer), API version number (v1.0) and the data being requested (raingages). However, in a REST API unless well documented by the developers, developing properly formated parameterized request urls can be challenging. However, the difficulties are counter-balanced by the implementing a Swagger UI for the API and making sure proper documentation is added, so the end-user has as much information as possible.

The JSON response is a bit more intuitive. From firgure Figure 4 we can see the basic data structure and data types. There are many

programming libraries that automate the parsing of JSON data into domain objects that facilitate API client development.

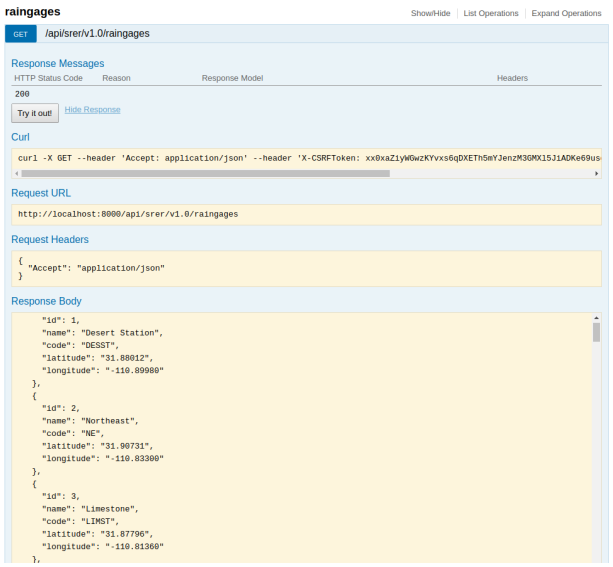


**Figure 4: REST/SWAGGER Response**

## 4.2 GraphQL API

Similar to the REST/Swagger user interface, many GraphQL APIs allow the developer to enable a user interface that serves as the GraphQL API's documentation. Figure 5 shows an example of the GraphQL API interface. At first glance, the GraphQL interface appears to not have nearly the amount of documentation that the Swagger equivalent has. However, the input portion (left half) of the interface has autocomplete features and the user simple has to hit the space bar or start typeing in order to get guidance from the interface.

Figure 6 demonstrates an example of a query in the GraphQL UI that is used to request a list of objects, in this case, all raingages. The left half of the user interface is where the query is created. The interface behaves similarly to many itegrated development environments (IDE) by being aware of the underlying data model and query request parameters. Consequently, as the user is developing the query, the UI provides autocomplete features that help expedite query development.

The query results are JSON data packet that is a list of raingage objects including attributes for each raingage. As with many other API's, this data packet can be consumed by a wide variety of clients including other web applications or thicker desktop clients like the statistical package R.

REST based API's are typically developed with create, retrieve, update and delete (CRUD) actions. GraphQL APIs offer the same type of actions, but not in the same format. Due to time constraints demonstration of those actions is outside of the scope of this paper.

## 5 SERVICE COMPARISON

New tools like GraphQL have an appeal to developers because they are new tools and provided entertaining opportunities to learn

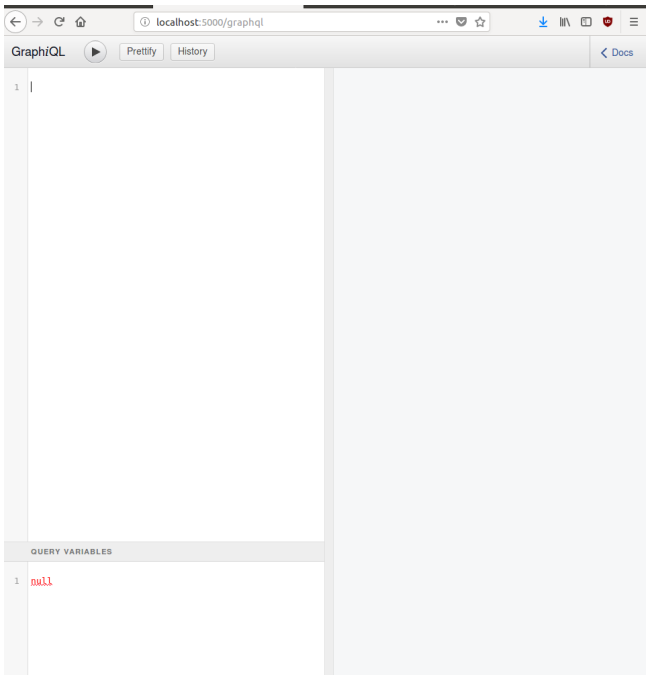


**Figure 5: GraphQL User Interface**

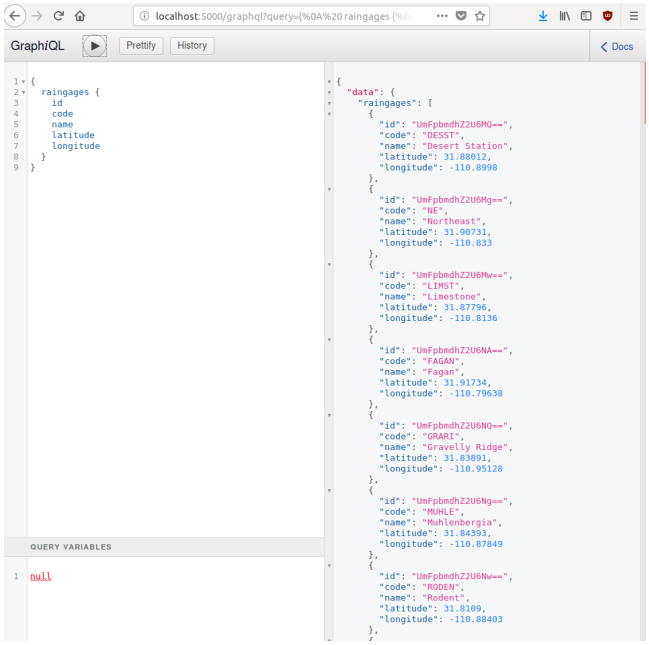


**Figure 6: GraphQL UI Query Example**

something new. However, it is important to validate the benefit of new tools by developing comparisons and measures against existing tools in order to assess qualitative attributes based on observed data and analysis.

Table 1 and Figure 7[10] demonstrate performance metrics and analysis based on their research. Their results indicate that GraphQL may have better performance rates.

**Table 1: Vazquez Research Findings**

[10]

| | REST API | GraphQL API |
|---|---|---|
| Total Request Size | 26.66 KB | 13.90 KB |
| Request 1 | 6.54s | 3.85s |
| Request 2 | 7.45s | 4.17s |
| Request 3 | 5.82s | 4.64s |
| Request 4 | 5.50s | 3.90s |
| Request 5 | 6.14s | 4.95s |
| Request 6 | 5.26s | 4.16s |
| Request 7 | 5.79s | 4.11s |
| Request 8 | 5.94s | 3.97s |
| Request 9 | 7.06s | 3.79s |
| Request 10 | 5.99s | 3.77s |
| Avg network response time | 6.15s | 4.16s |

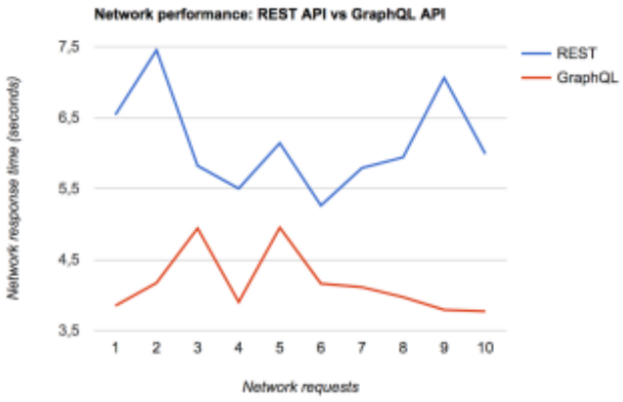


**Figure 7: GraphQL/REST Comparision**

In hopes of providing more effidence related to GraphQL performane in comparison to REST performance. A simple load test was created to test the REST API developed for this project against the GraphQL API created also created for this project.

The test environment consisted of:

(1) HP Spectre X360 Laptop with 16 GB of RAM
(2) Intel i7 Processor
(3) Ubuntu 16.04 Operating System
(4) PostgreSql 9.5 Database Server
(5) Python 3.6.4
(6) Django 2.0.3
(7) Locust Python load testing API and command line tool
(8) Graphene and Dango-Grapql plugins were used to create the GraphQL API
(9) Django Rest Framework was used to create the REST API

The source code repository for this project provides more detail on APIs and Python libraries used to develop the interface.

The load test was simple in each case, simulate 100 users, spawning 5 users per second, for 10 minutes. The Python load testing tool, Locust, was used to run each test under the described parameters and the web interface in Locust provided summy graphics showing the results of the test.

Figure **??** and Figure 9 demonstrate simple load-tests run against the GraphQL API developed for this project.

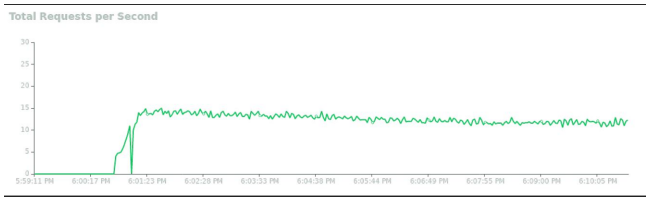


**Figure 8: GraphQL Total Requests per Second**

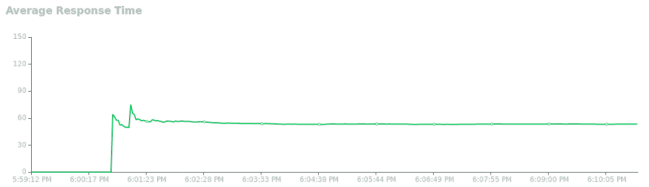


**Figure 9: GraphQL Average Response Time**

Figure 10 and Figure 11 show the results of the same test run against the REST API for the web application.

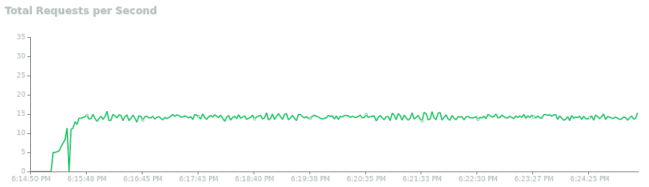


**Figure 10: REST Total Requests per Second**

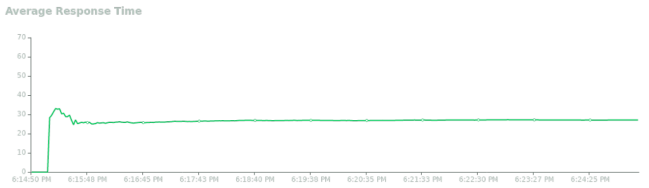


**Figure 11: REST Average Response Time**

Contrary to expectations the test demonstrates that the response times for the REST web service are lower than for the equivalent

GraphQL service. However, given that the particular test is simplistic and there could be many other factors affecting the test, there is no evidence in this study to make qualitative statements about the performance of eith GraphQL or REST. A more robust examination of the demo application and a rigorous load-testing design would be required in order to develop accurate insights into performance differences between the two technologies.

## 6 CLOUD INFRASTRUCTURE

In order to provide further proof-of-concept that the application and its respective data services and data are viable as web services in a near-production setting, a production like environemtn for the application was created using Amazon's Elastic Beanstalk (EB) cloud infrastructure. This paper does not serve as a tutorial on how to create an Amazon AWS account nor will we offer a complete tutorial on how to create an EB application. However, the following outlines and describes the experience and findings in the process of converting this application to a EB cloud-hosted application.

### 6.1 AWS Elastic Beanstalk

"AWS Elastic Beanstaok is an easy-to-use service used to deploy and scale web applications developed in programming languages like Java, Python, PHP, and others."[4]

The proceedure for creating and deploying an AWS EB application is relatively straight forward. A developer can either initialize an application using the EB web console or, if the user installs the EB command-line-interface (CLI) the application can be initialized locally. The developer can chose from several programming languages and specifies the programming language during project initialization. The initialization process requires the developer to create deployment configuration data so that AWS EB knows what programming language and what type of application is to be deployed and served over the internet. The application can exist locally before deployment and once properly configured the application can be deployed and almost run on first deployment.

However, most web applications rely on a backend database server like PostgreSql or MySQL. Amazon offers database servers throught its Relational Database Service (RDS)[2]. The application's database is created after the application is initialized and after EB knows about the application. The database is added as a RDS component to the EB application via online configuration. Once RDS is configured for the application, the appliation code has to be modified so that the codes has database connection information that applies to the production environment (EB) or local development environment.

A third feature that EB offers that facilitates development and deployment of web applications to a cloud based infrstructure is how EB handles static files. Web applications typically have many static files like images, cascading style sheets, javascript files and more. Those files tend to not change as frequently as application code does and ideally should be served seperately by a separate serve for security and efficiency reasons. However, dealing with that type of deployment can require quite a bit of time and increases complexity. EB has an elegant solution for this. A web application, once configured and initialized is automatically provide and Amazon S3 Bucket[3]. Amazon EB, automatically creates a storage location for the application's static files, copies those files when needed, to the storage location and serves those files for the web application. Typically, in a non-EB environment that is work the programmer has to develop and maintain.

## 7 CONCLUSION AND FINDINGS

The goal of this paper and project was, using precipitation and raingage data develop two different Web APIs and compare those APIs. The first API uses REST to create, read, update and delete data (CRUD), while the second API used GraphQL to offer the same CRUD functions. Due to time constraints both APIs were developed only for read and a simplified query operations. After both API's were developed a very simple benchmark test was executed against the endpoints the query for all raingage data in both APIs (REST, GraphQL). A simple benchmark comparison was developed to demonstrating request response rates for each API.

### 7.1 REST and GraphQL

The load tests cited earlier in the paper that the application's REST services tended to handle requests better than the GraphQL services. However, the Vazquez-Ingelmo, Cruz-Benito, Garcia-Palvo[**?** ] research demonstrates that GraphQL can produce improve request-response rates than REST. Overall, there are many things to consider when deciding to pick between the two web-service methods. Once possible consideration is caching. Because REST services tend to be more static in form that GraphQL services it may be easier to implement caching when needed for larger-scaled web-APIs.

### 7.2 Cloud Deployment

There were multiple reasons for chosing Amazon's Elastic Beanstalk as the deployment platform for the live version of the application. The possible deployment platforms to chose from where Heroku[1] and, as mentioned Amazon's Elastic Beanstalk. Heroku offers a simplified deployment environment that, as long as the project structure complies with one of the Heroku prescribed project structures, makes deployment a very simple process. Also, Heroku has excellent recovery process in place so that if an applcition is deployed and there is a failure in the deployment process, Heroku stops the deploy and allows the existing deployment to continue running. However, Heroku has a notably higher fee structure, which was the primary reason for choosing Amazon's Elastic Beanstalk.

Amazon's Elastic Beanstalk has a higher learning-curve than Heroku, but once the project is configured properly and deployed, work flow for making and committing application changes becomes streamlined. Also, Django applications are known for being easy to start in development, but difficult to deploy for testing and production. In particular a significant amount of work has to be done in order to make static files available. Elastic Beanstalk automatically handles static file deployment and serving static files if the application is configured propertly. After the developer completes the proper configuration, Elastic Beanstalk takes care of the rest with regrads to static files.

Another finding worth noting, while creating a cloud-based application like this one, was how to handle getting initial data sets into the application. This application relied upon data from the

Walnut Gulch Experimental Watershed and the Santa Rita Experimental Range. SRER precipitation data set was the largest and consisted of approximately 30,000 records, which is not a large data set. However, importing even 30,000 records proved to be resource intensive during the initial import. WGEW precipitation data set consisted of over 300,000 data points. This was a significant number to handle and proved problmetic during the deployment of after created API features related to WGEW. Initially, data import was handled during each deployment of the application. The databases tables were initialized and populated during each deployment. This was considered acceptable until all of the initial data was loaded and worked while the SRER data were the only data in the database. However, once the WGES data (300,000) records were incorporated into the deployment process, the deployment did not execute properly and left the application in a non-recoverable state in Amazon's Elastic Beanstalk. Fortunately, recovering from such a state and in the context of this project was a straight-forward process requiring re-setting some Elastic Beanstalk settings, removing the application from Elastic Beanstalk and re-deploying to a newly created application environment on Elastic Beanstalk. However, this meant figuring how to deploy the initial data sets since they still needed to be incorporated into the live application.

Elastic Beanstalk also provides remote database access to backend database servers being used as part of web applications like this one. Consequently, a secured, direct connection was created between the devlopment workstation and the production database where batch SQL-imports were used to complete the transfer of the initial data sets. This method ultimately proved to be a faster method than the application deployment method.

Finally, as of the completion of this paper the total costs incurred on Amazon's Elastic Beanstalk, which includes the deployed web application and Amazon S3 Bucket for storing static files and a PostgresQL database server with initial data import, and a few web requests to confirm the application is running amount to just under five dollars. The web application can be reviewed by opening a web browser to www.earthapi.net. A video documenting the AWS Elastic Beanstalk instance and console can be views at http://www.acatejr.com/cc.

## ACKNOWLEDGMENTS

The authors would like to thank Dr. Gregor von Laszewski for his support and suggestions to write this paper. The authors would also like to thank the following people. Dr. Mitch McLaran of the University of Arizona's School of Natural Resources and The Environment for providing access and support of the SRER website. Craig Wissler of the University of Arizona's School of Natural Resources and The Environment for providing feedback on related to the SRER data. Dr. David Goodrich and Dr. Phil Heilman, both from the United States Deparment of Agriculture's Southwest Watershed Research Center, for providing support and access to the Walnut Gulch Experimental Watershed website and data.

## REFERENCES

[1] 2018. Cloud Platform Application Heroku. (2018). https://www.heroku.com
[2] amazon.com. 2018. Amazon Relational Database Service. (2018). https://aws.amazon.com/rds/
[3] amazon.com. 2018. Amazon S3. (2018). https://aws.amazon.com/s3/
[4] amazon.com. 2018. AWS Elastic Beanstalk. (2018). https://aws.amazon.com/elasticbeanstalk/
[5] docker.com. 2018. Docker. (2018). https://www.docker.com
[6] Inc Facebook. 2018. GraphQL. (2018).
[7] The United States Department of Agriculture. 2018. Walnut Gulch Experimental Watershed. (2018). https://www.tucson.ars.ag.gov/dap/
[8] The University of Arizona. 2018. Santa Rita Experimental Range. (2018). https://cals.arizona.edu/srer/
[9] swagger.io. 2018. Swagger. (2018). https://swagger.io
[10] Andrea Vázquez-Ingelmo, Juan Cruz-Benito, and Francisco J. García-Peñalvo. 2017. Improving the OEEU's data-driven technological ecosystem's interoperability with GraphQL. In *TEEM*.

# Analyzing Twitter Activity to Identify Spam Accounts

Stephen Giuliani
Indiana University
Virginia, USA
sgiulian@iu.edu

## ABSTRACT

This paper provides an analysis of Twitter account activity to identify potential spamming bots as opposed to organic user activity. The approach uses a combination of web scraping in python, data storage within the MongoDB Atlas Cloud, and implementing a REST service to pull data from the cloud, expose it on a local system, and then archive the data for analysis.

## KEYWORDS

hid-sp18-507, Volume: 9, Chapter: Twitter, Status: 100.
Twitter, MongoDB, REST, TweePy

## 1 INTRODUCTION

The 2016 US Presidential election is a prime example of how social media can be used to push an agenda or advertise specific points of view in such a volume that discrediting the masses of inaccurate posts, shares, and re-tweets is a near impossible feat. Many individuals, organizations, as well as government entities, have utilized social media to flood various social mediums with repetetive posts and comments at the hands of automated bots, rather than real-user activity [11]. Since the unethical use of social media in this manor has come to light, post election, the US Department of State (DoS) and the Federal Bureau of Investigation (FBI) has been tasked to identify and address the use of fake accounts to push political agendas and manipulate the media as a whole [12].

Luckily for investigators, account activity for automated or fake users is typically characteristically different from the activity of organic or real users [3]. This paper will provide an analysis on Twitter users' accounts and user-habits to identify potential bots. This paper will also identify certain topics that are more likely to fall victim to the spamming tactics than less polar-opinion subject matter.

## 2 BOTS, SPAM, AND THE IMPACT

Automation is a norm in technology, especially social media. The purpose of the automated account, or bot, is what determines whether the account is considered spam versus a service. Examples of more favorable automated accounts include the 'Earthquake Bot', which automatically tweets warning information to areas where a 5 or greater magnitude earthquake is detected, the 'Netflix Bot', which will notify you each morning of the latest Netflix releases, and the 'Big Ben Clock', which just tweets 'BONG' every time London's Big Ben chimes [8]. The fun and useful bots are just a few examples of how Twitter accounts can be used for a collectively positive purpose. In contrast, there are accounts that serve only to like, re-tweet, or even produce original content, at a higher daily rate and often focus on subject that are hot-topics in the news or polarized views in politics. Within this analysis, we have concluded that certain topics are more likely to have a spam bot presence than others. For instance, '#LockHerUp', a notorious hashtag used in favor of putting Hillary Clinton in jail, is used by bots significantly more than '#SundayBrunch'.

The accounts that can be considered spam-based bots have similar account behavior to other bots and can be classified collectively via just a few data points. A spam bot is more likely to post at a much higher volume than a real user. Some bots rack up high counts of re-tweets with very few original posts, have a very unbalanced ratio of followers to people who follow back, and may achieve a high volume of activity within a few days of creation.

The use of these spaming accounts creates artificial interest, or hype in specific subjects. This tactic can keep topics in the limelight or falsely overpower reputable news sources ultimately controlling what we see online or in mainstream media [1]. Instances where the masses latched onto the volume of information rather without investigating source or information independently can lead to dangerous conclusions and sometimes unjustified courses of action. In 2012, the moments immediately following the unfortunate incident at Sandy Hook lead to an online hunt and prosecution of Ryan Lanza. Varieties of social mediums were quick to post and share than Ryan was the perpetrator of the terrorist attack, including death threats and pledges of revenge. However, as a short time passed and sources were verified and facts pushed through the social hype, Adam Lanza, Ryan's brother, was identified as the shooter–well after news outlets followed and falsely reported on socially-based conclusions [5].

## 3 ANALYTICAL APPROACH

In this section we will provide an overview of the primary steps involved in acquiring, storing, querying, and analyzing the subject Twitter data.

**Note to the reader:** In order to create, and therefore replicate, this paper, independent Twitter, Gmail, and Twitter Developer accounts were created. These accounts and credentials are necessary for access to Twitter's API. Additionally, a temporary access account was set up so that instructors and reviewers could access and replicate data storage and pulls within the MongoDB Atlas cloud service. The credentials and access information is provided within the project code and no changes should be made.

### 3.1 Technologies Used

**Twitter API**

Twitter uses a publicly available API for all of its users' Twitter activity and profile information. Free developer accounts are available and come with the ability to pull data from the past 7 days. Increased scope is available through paid-tiers of developer accounts.

**Python: TweePy**

Tweepy is a package developed for the purpose of interacting with Twitter's online API [9]. Queries are returned in JSON format for analysis or direct storage.

**MongoDB Atlas**

MongoDB is a popular NoSQL document database program that is open source and can be deployed almost anywhere. MongoDB Atlas is a cloud-based MongoDB service that can host documents through the services of Amazon Web Services (AWS), Microsoft's Azure, or Google's Cloud Platform (GCP) [6]. For purposes of this paper and analysis, we have used MongoDB Atlas to host the scraped Twitter data, using AWS, and expose the data through a RESTful service.

**Python: Flask (REST)**

Flask is a micro framework for Python that can be used to design and make RESTful queries. We use flask to make GET requests to the MongoDB Atlas Cloud data [4]. The data requested is used for the analysis itself. Once the data is scraped and stored within the cloud, python and Flask expose, or makes available, the data on a predetermined web address, or local endpoint for purposes of this paper.

### 3.2 Web Scraping, Twitter Data

In order to pull data from Twitter's API, you must first register for a developer account (free). Once registered, and after creating a default application, the account keys and credentials are necessary for authentication when using tweepy in python. A free developer's account has the ability to pull twitter data from the past 7 days only. For purposes of this analysis, these seven days are sufficient. Documentation on the data Twitter makes available to the public is available here [13]. Using tweepy, we are able to query Twitter's data via a number of characteristics: hashtags, words present in tweets, usernames, dates, and more. The JSON object that is returned (dependent on the structure of the query) includes information on the user's account since creation as well as meta-data on the tweet itself (geotags, platform used to post, links used, and more). The entirety of the returned JSON object is what is stored on the MongoDB Atlas cloud.

A typical Twitter query JSON object is less than 8 kB at a length of about 212 lines of code, based on a JSON dump of 100 tweet-data returns. Therefore, a high volume of twitter data can be pulled in stored within the cloud for analysis. However, according to Twitter's API policy [14], only 500 instances can be queried at once with a registered development user (paid options available for higher volume queries).

### 3.3 MongoDB Atlas, Storing Twitter Data

MongoDB's online web-hosting cloud service is available for free for testing purposes. The limitations of this service is a 3-cluster database with a storage limit of 512 Mb. Any requirements to use more than 3 clusters or storage needs greater than 512 Mb requires paid subscriptions, which are hourly-usage based. The data scraped from Twitter will be stored on this cloud and will be available to readers/reviewers after the analysis is concluded. For purposes of demonstration and this paper, the query max of 500 instances will be stored and used for the analysis. However, at an average of 8kb per tweet query, over 64,000 instances can be stored within the free 512 Mb of data MongoDB Atlas offers.

The ability to store data on a MongoDB database comes from the use of pymongo, a Python package developed for the administration of Mongo databases [15]. However, for testing purposes, we also used the mongo-shell capabilities to clear, start, stop, and monitor the cloud-server status. Documentation on pymongo and mongo-shell are available at https://api.mongodb.com/python/current/.

Connection to the cloud database requires ssl authentication, which will be published within this paper for testing purposes. Administration or viewing permissions will also be created. The JSON objects produced within Python using tweepy are stored in a created database collection on the Atlas cloud. This cloud database is also use as the connection for the RESTful service to pull specific data-points.

### 3.4 Flask, RESTful Service in Pulling from the Cloud

The data or JSON-format documents stored on the Atlas cloud can be connected to and authenticated via pymongo in python. Flask is used to bridge the connection and expose an available address for data to be displayed/used for analysis [10]. The RESTful API and paths created were created from scratch, grouping specific JSON attributes of the stored data. For instance, in order to determine the daily rate of posts per user, the GET request will pull the data for the user's total posts and the data including the creation date of the account. The JSON object produced via the GET request can be parsed and regrouped prior to exposing the data on flask's default local endpoint. See the code-documentation for the REST API used to query the Atlas cloud.

### 3.5 Data Analysis

As mentioned above, the data on the cloud can be pulled via GET requests. The data required is pulled via specific GET requests so that analysis can be done. MongoDB's Atlas cloud service has the ability to perform analysis on the cloud; however a free testing account is not able to perform this. Therefore, the analysis is conducted locally, via python objects. For purposes of data familiarization, the data is pulled and stored in a comma-separated values (CSV) format. This enables seamles interpretation by python's many analytical and visual packages. Metrics such as daily rates for friending other users and posting, friends-to-followers ratios, and account age are calculated. Medium, a popular online publication and editorial provides "Twelve Ways to Spot a Bot" [2] on Twitter; many of these points can be verified via Twitter's public API data. For analytical purposes, the GET request, hosted by flask through localhost endpoints, is downloaded as a raw JSON object and converted and stored locally as a CSV. This paper will analyze the scraped users' average posts per day, average favorites per day, as well as the ratio of friends to followers. Spam bots are known to constantly produce tweets or retweets and therefore we can expect a high number of average daily posts or favorites per bot-users. Similarly, a spam bot often will follow, or friend, a high volume of other twitter users so that its account expose increases. In contrary to the number of friends a bot may have, the number of users that follow back

are usually low; therefore a high ratio of friends to followers can indicate an automated account.

## 4 RESULTS

Evaluating the average daily posts, average daily favorites, and ratio of friends to followers revealed a majority of organic users; however there is still a significant number of users identified as a possible spam bot. The following tables, Table 1, Table 2, and Table 3, show the summary information on the daily post, daily favorites, and friend to follower ratio data, respectively, pulled from Twitter's user data.

**Table 1: Summary: Daily Posts**

| | |
|---|---|
| Count | 460 |
| Mean | 43.60 |
| Std | 72.19 |
| Min | 0.0014 |
| 25% | 4.95 |
| 50% | 17.49 |
| 75% | 50.57 |
| Max | 763.09 |

**Table 2: Summary: Daily Favorites**

| | |
|---|---|
| Count | 460 |
| Mean | 52.23 |
| Std | 89.20 |
| Min | 0.0024 |
| 25% | 7.84 |
| 50% | 23.02 |
| 75% | 59.53 |
| Max | 844.67 |

**Table 3: Summary: Friends per Followers**

| | |
|---|---|
| Count | 460 |
| Mean | 1.53 |
| Std | 1.77 |
| Min | 0.00 |
| 25% | 0.88 |
| 50% | 1.09 |
| 75% | 1.50 |
| Max | 18.50 |

Note that the number of instances within the CSV files totals only 460, rather than the 500 available within the GET requests. This is because 40 posts are associated with duplicate accounts that are already a part of the queried data; therefore their data is duplicative and only requires a single instance for evaluation.

Users' post, favorites, and ratio data indicates right-skewed data points because the minimum and maximum values are so far apart. The averages produced are likely inaccurate for organic-only accounts. The standard deviation from each of the means also indicates the wide range of account activity.

The histograms from Figures 1, 2, and 3 support the summary data and the conclusion that some accounts are not indicative of an average user.

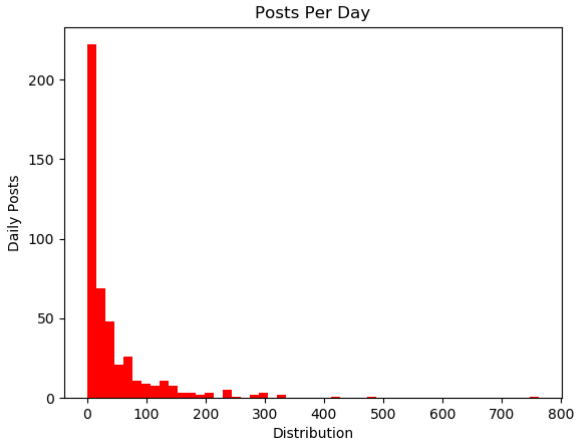


**Figure 1: Histogram: Posts per Day**

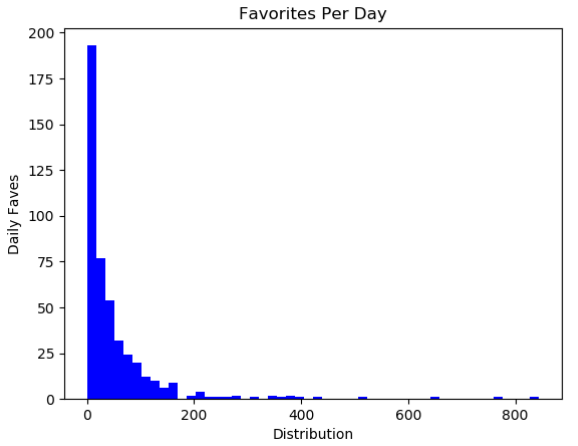


**Figure 2: Histogram: Favorites per Day**

Arbitrary metrics for differentiating activity of a spam-bot versus an organic user we used to re-evaluate the data to filter out potential bots, or outliers, in the queried data.

Daily posts of an organic user is considered to be at most 96 posts per day (a user who posts a single tweet every 10 minutes for 16 hours each day). Favorites per day was evaluated at a similar cap, only it was increased by 25 percent to account for the simplicity and efficiency of a favorite over a produced tweet. The ratio of friends to followers was capped at 2 times the standard deviation of the original data, or about 3.5. This indicates that an account may follow three and a half times as many users as are willing to follow them.

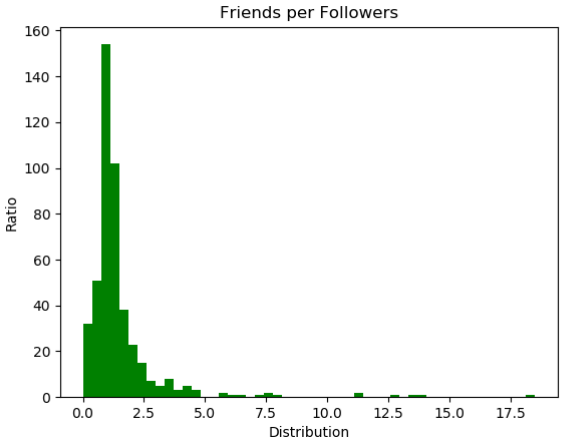


**Figure 3: Histogram: Friends per Followers**

After filtering out the users that don't fall within these thresholds, Table 4, Table 5, and Table 6 show the delta, or change in the data as impacted by the potential bot accounts.

**Table 4: Adjusted: Daily Posts**

| | |
|---|---|
| Count | 59 |
| Mean | 21.59 |
| Std | 48.79 |
| Min | 0.00 |
| 25% | 1.12 |
| 50% | 4.32 |
| 75% | 17.88 |
| Max | 667.62 |

**Table 5: Adjusted: Daily Favorites**

| | |
|---|---|
| Count | 47 |
| Mean | 22.34 |
| Std | 59.60 |
| Min | 0.00 |
| 25% | 1.55 |
| 50% | 3.67 |
| 75% | 13.18 |
| Max | 726.27 |

As shown in Table 7, Table 8, and Table 9, the average for daily posts and daily favorites fell by more than half the original mean. Of the 59, 47, and 28 accounts that were filtered out per each category, the top ten and the values that flagged them are shown below.

The analysis in each category was conducted independently. However, merging the Top 10 candidate bots based on daily activity shows 5 accounts sharing the top ten in both categories. When comparing the top 10 based on friend-follower ratios, none of these users share the top 10 with the daily-activity-based top 10 accounts.

**Table 6: Adjusted: Friends per Followers**

| | |
|---|---|
| Count | 28 |
| Mean | 0.35 |
| Std | 1.14 |
| Min | 0.00 |
| 25% | 0.04 |
| 50% | 0.03 |
| 75% | 0.09 |
| Max | 14.98 |

**Table 7: Potential Bots: Criteria = Daily Posts**

| Screen Name | Avg. Daily Posts |
|---|---|
| MaheshC78848209 | 763.09 |
| KatTheHammer1 | 475.44 |
| GaryWil29548846 | 421.00 |
| DonGibs22787443 | 330.10 |
| ARealPrincesa | 327.32 |
| notbuyingthat54 | 304.66 |
| nice-centurion | 294.95 |
| lynn85706529 | 291.30 |
| FrankJManrique3 | 287.52 |
| R98250729 | 279.81 |

**Table 8: Potential Bots: Criteria = Daily Favorites**

| Screen Name | Avg. Daily Favorites |
|---|---|
| siminuteman1776 | 844.67 |
| ImmoralReport | 772.75 |
| MaheshC78848209 | 644.52 |
| KatTheHammer1 | 522.71 |
| GaryWil29548846 | 427.00 |
| 4Freedom4ever | 395.86 |
| mickeyh54099794 | 380.63 |
| FrankJManrique3 | 374.21 |
| lynn85706529 | 368.90 |
| melinda63312935 | 350.65 |

**Table 9: Potential Bots: Criteria = Friends per Followers**

| Screen Name | Friends per Followers |
|---|---|
| Andy75213017 | 18.50 |
| KenSR-802VT | 13.75 |
| wesleyh23420956 | 13.44 |
| JamesChhetri7 | 12.58 |
| john316stv | 11.46 |
| Ikram-35 | 11.25 |
| JaredFu88957380 | 7.87 |
| LDuvall19 | 7.59 |
| nateweyand | 7.48 |
| DJLPX | 7.31 |

## 5 CHALLENGES AND OPPORTUNITIES

The purposes of this paper are to demonstrate the ability to scrape, store, access, and analyze data; however the analysis itself is only part of a much larger potential research topic; one that is outside the scope of this course.

### 5.1 Challenges

This paper demonstrates the ability to scrape and store data on the cloud where the data can be pulled, exposed locally, and then downloaded into analytical-friendly formats. Limitations on this analysis include Twitter's policy for query frequency and size of the data returned as well as MongoDB Atlas's limit on data storage. Both of these can be mitigated via paid respective accounts. Additionally, in order to perform the analysis independent of the provided credentials, a user must register themselves for these services. Also, depending on the query or subject matter of the requested information, the freely available 7-day history might not suffice. Sentiment analysis and bot filtering on past events would require the ability to pull tweets or archived user data.

### 5.2 Opportunities

In order to truly conclude a spam-bot over atypical organic user further analysis should be conducted. Analysis can include re-querying Twitter for specific users (such as the 5 that shared the top tens for daily activity) where the data on a specific-user's tweets can be evaluated with sentiment analysis, analysis on the repetition of tweets, other subjects the user promotes, and the platforms and languages used to tweet. Bots typically are based on programs that are not run using a mobile platform, such as the Twitter app on a phone; contrarily, most organic twitter users use their phone as the primary posting/sharing/favoring platform. Deep learning, out of scope of this paper is a great opportunity to parse through the dense JSON object pulled per user.

Additionally, some cloud services and platforms come with analytical capabilities in house. The lowest tier within the MongoDB Atlas cloud service does not offer data analytics; however should a cloud database platform be used with these capabilities it would eliminate the need to re-pull data to a local machine or server as the analytics, or even the necessary python files can be hosted and executed directly from the cloud.

## 6 CONCLUSION

Clearly, automation within Twitter is prevalent. According to a study conducted by the University of Southern California and Indiana University, up to 15% of twitter accounts could be bots [7]. Lately, the news, as well as the FBI, have focused on the use of these bots as a way of manipulating media and public perception. By pulling data from just the past seven days from Twitter, based on a query for a hot-topic hashtag, '#LockHerUp', and determined by activity that deviates from typical organic user activity, over 100 accounts appear to be automated and in support of the politically-charged campaign.

The ability to scrape, and directly store-then-pull data on MongoDB's Atlas cloud service was seamless, efficient, and was possible at no cost to the user. The 500 records pulled from Twitter and stored in the cloud occupied less than 1% of the available memory cap and therefore allows for opportunities to vastly increase data volumes for analysis.


## ACKNOWLEDGMENTS

The authors would like to thank Dr. Gregor von Laszewski and the Associate Instructors for their support and suggestions while writing this paper.


## REFERENCES


[1] Wikipedia contributors. 2018. Twitter bot — Wikipedia. Web Page. (Apr 2018). https://en.wikipedia.org/wiki/Twitter_bot Accessed: 2018-04-05.

[2] Digital Forensic Research Lab. 2017. #BotSpot: Twelve Ways to Spot a Bot. Web Page. (Aug 2017). https://medium.com/dfrlab/botspot-twelve-ways-to-spot-a-bot-aedc7d9c110c Accessed: 2018-04-01.

[3] Supraja Gurajala, Joshua White, Brian Hudson, Brian Voter, and Jeanna Matthews. 2016. Profile characteristics of fake Twitter accounts. Web Journal. (2016). http://journals.sagepub.com/doi/pdf/10.1177/2053951716674236 Accessed: 2018-04-08.

[4] Anthony Herbert. 2017. MongoDB Backed RESTful — mongo.py. Code Repository. (Jan 2017). https://github.com/PrettyPrinted/mongodb_backed_restful/blob/master/mongo.py Accessed: 2018-04-05.

[5] Kashmir Hill. 2012. Blaming The Wrong Lanza: How Media Got It Wrong In Newtown. Web Page. (Dec 2012). https://www.forbes.com/sites/kashmirhill/2012/12/17/blaming-the-wrong-lanza-how-media-got-it-wrong-in-newtown/#55e6a9327601 Accessed: 2018-04-01.

[6] MongoDB, Inc. 2018. MongoDB Atlas. Web Page. (2018). https://docs.atlas.mongodb.com/ Accessed: 2018-04-05.

[7] Michael Newberg. 2017. As many as 48 million Twitter accounts aren't people, says study. Web Page. (Mar 2017). https://www.cnbc.com/2017/03/10/nearly-48-million-twitter-accounts-could-be-bots-says-study.html Accessed: 2018-04-08.

[8] Jannis Redmann. 2017. Awesome Twitter Bots. Code Repository. (Aug 2017). https://gist.github.com/derhuerst/1cb20598b692aa87d9bb Accessed: 2018-04-01.

[9] Joshua Roesslein. 2009. Tweepy. Web Page. (2009). http://www.tweepy.org/ Accessed: 2018-04-01.

[10] Armin Ronacher. 2018. Welcome — Flask (A Python Microframework). Web Page. (2018). http://flask.pocoo.org/ Accessed: 2018-04-01.

[11] Eli Rosenberg. 2018. Twitter to tell 677,000 users they were had by the Russians. Some signs show the problem continues. — Washington Post. Web Page. (Jan 2018). https://www.washingtonpost.com/news/the-switch/wp/2018/01/19/twitter-to-tell-677000-users-they-were-had-by-the-russians-some-signs-show-the-problem-continues/ Accessed: 2018-04-08.

[12] Chris Strohm. 2018. FBI Task Force to Expose Russian Social Media Manipulation. Web Page. (Jan 2018). https://www.bloomberg.com/news/articles/2018-01-10/fbi-plans-task-force-to-expose-russian-social-media-manipulation Accessed: 2018-04-08.

[13] Twitter, Inc. 2018. API Reference Index — Twitter Developers. Web Page. (2018). https://developer.twitter.com/en/docs/api-reference-index Accessed: 2018-04-01.

[14] Twitter, Inc. 2018. Rate Limiting — Twitter Developers. Web Page. (2018). https://developer.twitter.com/en/docs/basics/rate-limiting.html Accessed: 2018-04-01.

[15] Robert Walters. 2017. Getting Started with Python and MongoDB. Blog. (Apr 2017). https://www.mongodb.com/blog/post/getting-started-with-python-and-mongodb Accessed: 2018-04-01.

# Edge Computing and Big Data Processing using Raspberry Pi

Naveen Kaul
Indiana University
Bloomington, IN 47408, USA
nkaul@iu.edu

## ABSTRACT

Edge computing is commonly described as generation, collection, and analysis of data at the actual site, where data is generated as opposed to storing and analyzing the data at a centralized location like cloud or a data center. This data can be stored in a centralized location for a historical purposes, but the main storage and analysis is done on site. The technology typically consists of sensor-generated data, robotics, automated machines, smartphones and tablets, and distributed analytics servers that are used for computing and analysis on the spot. In this project we aim to explore a framework built on Raspberry Pi cluster that can be used to process data stored on the cluster and discuss advantages and limitations.

## KEYWORDS

hid-sp18-510, Volume: 9, Chapter: Edge Computing, Status: 100. Edge Computing, Big Data, Spark

## 1 INTRODUCTION

"Edge computing is a method of optimizing cloud computing systems by performing data processing at the edge of the network, near the source of the data" [13]. It is typically used along with IoT (Internet of Things) devices that usually collect data, sometimes in massive amounts. Edge computing allows devices to store and analyze data locally, hence reducing the traffic to the central repository/cloud.

Purpose of this project is to explore various applications of Raspberry Pi which can be seen as next gen IoT device power source and it becomes important to understand its limitations and applications. This project would also implement a solution to archive old and large files from the local cluster to a centralized cloud location, which is a very common use case industry wide.

## 2 TECHNOLOGIES USED

This section describes various tools and technologies used in the project and their configuration and implementation steps.

- Raspberry Pi
- Apache Spark
- Google Cloud
- Apache Kafka
- Apache Spark mllib

### 2.1 Raspberry Pi

"A Raspberry Pi is a small credit card-sized computer originally designed for education, inspired by the 1981 BBC Micro" [8]. It was originally created by Eben Upton, whose goal was to create a low-cost device that can be primarily used for improving programming skills at pre-university level. But owing to its low cost and flexibility, it was quickly adopted by several different people including, DIYers, makers and electronic enthusiasts who need more than a basic microcontroller device[8].

**Table 1: Configuration**

| Model Name | Processor | Cores | RAM | Price |
|---|---|---|---|---|
| Pi Zero | 1GH 32 bit | 1 | 512 MB | $5 |
| Pi 1 Model A+ | 700 MHz 32 bit | 1 | 512 MB | $20 |
| Pi 1 Model B+ | 700 MHz 32 bit | 1 | 512 MB | $25 |
| Pi 2 Model B | 900 MHz 32 bit | 4 | 1 GB | $35 |
| Pi 2 Model B+ | 1.2 MHz 64 bit | 4 | 1 GB | $35 |

The Raspberry Pi was designed for the Linux operating system, and hence many Linux distributions are coming up with a version compatible with Raspberry Pi. Raspbian and Pidora being two of the most commonly used operating systems.

Two of the most popular options are Raspbian, which is based on the Debian operating system, and Pidora, which is based on the Fedora operating system. Each of these work well for beginners and can be installed either directly on SD Cards or via NOOBS (New Out Of Box Software) also available to download free [8].

Different models available

### 2.2 Apache Spark

"Apache Spark is an open source big data processing framework built around speed, ease of use, and sophisticated analytics" [10]. Originally developed in 2009 in UC Berkeleyfis AMPLab, and was later open sourced in 2010 as an Apache project [10].

Characteristics of Apache Spark

- Spark provides a comprehensive, unified framework to manage big data processing requirements with support for variety of heterogenous data sets such as (text data, graph data etc).
- Spark provides streaming capabilities in addition to batch processing.
- Spark can improve the performance of applications up to 100 times faster than in Hadoop by utilizing in memory processing even when running on disk.
- Spark provides support for programming languages like Java, Scala, or Python. It also comes with a pre configured set of over 80 high-level operators. Spark also provides a capability to interactively query data using spark shell.
- It also support SQL queries, machine learning algorithms and graph data processing that can be used as standalone processes or stages in a single data processing pipeline.

### 2.3 GoogleCloud

Google Cloud Platform is a type of public cloud, which consists of a variety of infrastructure hardware such as compute and storage

devices, and virtual resources, such as virtual machines (VMs), Virtual Private Network etc that are contained in Google's data centers around the globe. Each data center is located globally within in a region. Currently, Google cloud regions include Central US, Western Europe, and East Asia. "Each region is a collection of zones, which are isolated from each other within the region" [3]. Each zone is identified by a name that combines a letter identifier with the name of the region. For example, zone a in the East Asia region is named asia-east1-a.

Any GCP resources that a user wants to allocate or use must belong to a project and hence project is a unique umbrella containing everything required from hardware to lambda services. It is easier for resources within a single project to collaborate and work together, for example via an internal network, subject to the regions-and-zones rules.

Google Cloud platform provides a lot of services that can be broadly classified under following categories:

- Computing including hardware, containers and virtual machines
- Storage services
- Networking services
- Big data processing tools and technologies
- Machine Learning resources that also include support for Natural Language Processing and Computer vision related use cases.
- Data Analysis Services
- Application Platform services commonly known as PAAS (Platform as a Service)

### 2.4 Apache Kafka

"Apache Kafka is a publish-subscribe messaging system"[4]. This means that a software through which applications can send messages to each other. Applications can either create messages or subscribe to the messages that are created by other applications. In Kafka, each message can be broadly classified under a name called topic and kafka cluster consists of brokers which are nodes that transfer those messages. Producers publish data (push messages) into Kafka topics defined within the broker and then consumer applications consume those messages from the topic.

Messages are byte arrays that can store any object in any format. As mentioned earlier, all Kafka messages are organized into topics and hence any message needs to be sent or received by connecting to a specific topic.

### 2.5 Apache Spark MLlib

The machine learning library offered by Apache Spark is called MLlib, this is a very powerful tool to perform machine learning at scale since this library provides support for distributed processing and also provides a lot of machine learning algorithms out of box [7]. MLlib very neatly into Spark's APIs and interoperates very well with NumPy in Python and R libraries, it can be easily used for any type of data source and efficiently plugged into Hadoop workflows [1].

As of latest release machine learning algorithms included in MLlib are

- Classification: logistic regression, naive Bayes

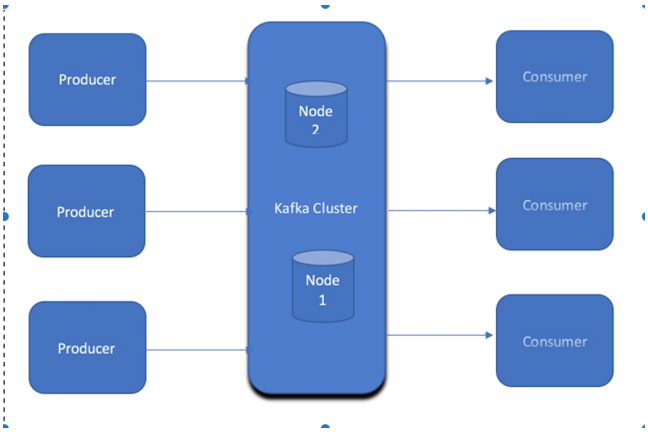


**Figure 1: Kafka Architecture**

- Regression: generalized linear regression, survival regression
- Decision trees, random forests, and gradient-boosted trees
- Recommendation: alternating least squares (ALS)
- Clustering: K-means, Gaussian mixtures (GMMs)
- Topic modeling: latent Dirichlet allocation (LDA) Frequent itemsets, association rules, and sequential pattern mining

MLlib also provides support for building data pipelines and saving those pipelines and models for further use, providing better reusability. It also has started providing support for third party algorithms and packages like scikit-learn, xgboost etc [2].

## 3 IMPLEMENTATION STEPS

Below listed are the steps to be implemented as part of this project.

- Build a hadoop/spark cluster on a cluster of Raspberry Pis.
- Install Zookeeper and Apache Kafka on the server
- Use spark cluster to train and test a machine learning algorithm
- Use python program to push the files to central cloud like Google Cloud or AWS.

## 4 SETUP AND INSTALL RASPBERRY PI CLUSTER

Raspberry Pi is a small computer but has a lot of potential due to its cost and applications in common day to day routines. More powerful version of Pi can be used in industries for performing various autonomous tasks like drones, image sensing etc. So it makes a good use case to install big data processing tools on Pi and test its application.

### 4.1 List of Components Used

The Pi cluster was built based on three raspberry Pis that have 1 GB RAM and 1.2 GHz processor.

- 3 Raspberry Pi B+ having 1 GB RAM and 1.2 GHz processor speed
- 32 GB Sandisk Micro SDXC Card
- Common Acrylic Case

- Anker Power Supply

## 4.2 Installation steps

The setup starts from procuring the hardware and then downloading the image file from raspberry.org website. For purpose of this project, raspbian operating system was used, which is build on debian linux platform and comes installed with some preliminary software.

## 4.3 Bootstrapping and Initial Setup

Once the image was downloaded from raspberrypi.org, there was some bootstrapping done so that it's easier to log into the Pis in absence of a monitor, also sometime called as headless installation. Below changes were done after the image was installed Create an SSH file Edit wpa_supplicant.conf file to add WIFI SSID and password so that Pi can connect to Wifi as soon as it powers on. All these steps need to be done after the Pi has been formatted and image has been flashed on to Pi, which can be done easily using Etcher or command line tools.

```
touch /Volumes/boot/ssh
# This will enable the ssh on Pi

nano vi /Volumes/boot/wpa_supplicant.conf
ctrl_interface=DIR=/var/run/wpa_supplicant \
GROUP=netdev
update_config=1

network={
        ssid="<network name>"
        psk="<password>"
}
# This will allow Pi to automatically
log on to the wireless network.
```

Once the Pi is powered on, as soon as it logs on to wireless network, its IP address can be noted and then used for logging in to Pi. Alternatively, Each Pi can also be updated with static IP addresses or host names. Its a good idea to provide each Pi different hostname so that they can be identified easily.

```
#Run this command after logging into each Pi
sudo vi /etc/hosts
192.168.0.1 pimaster #master node in the cluster
192.168.0.2 pislave01 #slave node1
192.168.0.3 pislave02 #master node2

#Edit the hostname as well for each Pi
sudo vi /etc/hostname
192.168.0.1 pimaster
```

## 4.4 System setup and configuration for Spark and Hadoop

One very important setup if Spark needs to be installed on Pi, if not done spark will not run and fail with memory issues. This also needed to be done on all devices since spark cluster was failing repeatedly due to memory issues.

```
sudo vi /etc/dphys-swapfile
CONF_SWAPSIZE=1000
#Pis require a reboot for the change to take effect.
```

Another important step in setting up the cluster is to correctly create and copy public keys across the cluster. It is advisable to create another user for performing all hadoop and spark related tasks and processes.

```
sudo addgroup hadoop
sudo adduser --ingroup hadoop hduser
sudo adduser hduser sudo
su hduse
```

Run the updates on each Pi and also install required softwares

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install python-dev python3-dev \
python-numpy python-scipy python-pandas python-matplotlib
```

Next step was to create and copy keys across the cluster, for this ssh-key-gen was run, this needs to be done for all the Pis in the cluster so that they can communicate without any issues among each other using SSH.

```
sudo su hduser
ssh-keygen -t RSA
ssh-copy-id hduser@pimaster
ssh-copy-id hduser@pislave01
ssh-copy-id hduser@pislave02
```

Hadoop was installed directly using wget command and then copying the folder to the desired location. All the configurations were done as appropriate and some of them proved to be very important.

Various websites were used as reference to performing all these installs and some of them are listed below.

- tekmarathon [12]
- Billion Taxi Rides [6]
- Install Spark [5]

# 5 ALTERNATIVES TO USING ETCHER

Since manually bootstrapping and installing all images is manageable but this task can become cumbersome if there are a lot of nodes in a cluster hence some mechanism of taking back up of Pi images need to be explored.

- Latest raspbian operating system come preinstalled with SdCardcopier which is a GUI based interface to backup the image from a SDCard loaded on a PI to another external storage device, but even though this is convenient, this process still can not be performed via command line and needs the user to have access to Raspberry Pi via a monitor.
- One way is to perform the same task as above through command line and even though some people have used rsync, or dd command on linux, they all have their disadvantages.
- There is a command line version of SDCard copier called pi-clone, whose source code is available on github, but this should be used with caution and used only by very seasoned users [11].

## 6 INSTALLING SPARK

Spark can also be installed correctly from apache website and version used in this project was 2.3.0. One very important thing to keep in mind while installing Spark is that spark needs a lot of memory to work that can not be provided by Raspberry Pi as they only have 1 GB Ram for B+ and even less than that in all other models so there are some configurations that needs to be modified before Spark can function properly. spark-env.sh located at $SPARK_HOME needs below settings

```
SPARK_MASTER_HOST=pimaster
SPARK_WORKER_MEMORY=512m
SPARK_EXECUTOR_MEMORY=512M
SPARK_DRIVER_MEMORY=512M
HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
SPARK_DAEMON_MEMORY=512m
```

After successful installation of hadoop and spark, their activity can be monitored through the URLs. Default url for spark is //master-node:8080

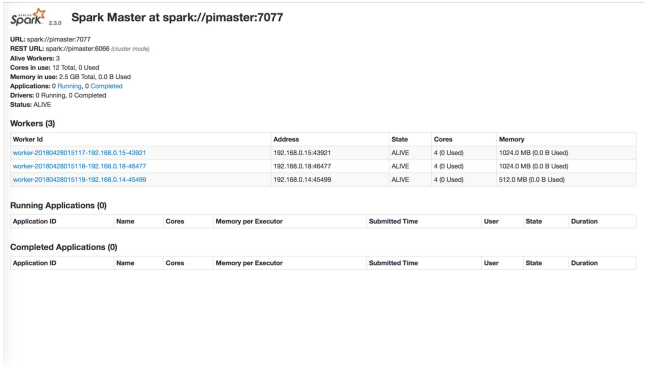


**Figure 2: Sample SPARK Url**

### 6.1 Running Spark

There are several ways that Spark can be run either through spark-submit or spark-shell mode. In an order for Spark to run in cluster mode, HADOOP_CONF_DIR or YARN_CONF_DIR need to be set so that Spark processes know how to communicate with Hadoop and Yarn.

Spark can be started by running

```
$SPARK_HOME/bin/start-master.sh
$SPARK_HOME/bin/start-slaves.sh
```

Once started, verification can be done using spark-shell or pyspark commands if the pyspark is installed.

Sample command line view

```
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.3.0
      /_/

Using Scala version 2.11.8 (Java HotSpot(TM)
Client VM, Java 1.8.0_65)
Type in expressions to have them evaluated.
Type :help for more information.
```

## 7 MACHINE LEARNING ON PI CLUSTER

The main topic of this project is to determine whether we can use Raspberry Pi for performing any AI or Machine learning related task since its small, cheap and yet powerful so some machine learning and few deep learning algorithms were attempted on the cluster and their performance was monitored and compared. One thing can be said without doubt that Raspberry Pi can not be yet used for Deep Learning capabilities with limits on memory and processing power. Some deep learning applications were tried to classify images of flowers and Pi master node became unresponsive while the model was run and hence had to be killed. A normal machine learning algorithm runs within seconds on a cluster if 3 Pis but does not scale very well even with Spark installed. Main issue being available memory since swap space alone can not accomodate for less memory.

The machine learning algorithm run was a linear regression algorithm to predict the median price of houses in different districts of California. The job took around 1 minute to train on a dataset of 20,000 records, which barely takes seconds when run on a powerful personal computer like Mac or Windows.

### 7.1 Predicting House prices

Dataset [9] needs to be downloaded from the website directly. This dataset can be stored locally and then loaded in SPARK RDD by running below command

```
rdd = sc.textFile('/path/cal_housing.data')
```

Various commands can be run to analyze contents of the dataset like

```
describe()
collect() Running collect is not advisable since that can take
  up all the driver memory and cause Spark to fail
show()
take()
```

Since looking at data it seems that the data is not uniformly distributed so it becomes very important as with any other machine learning algorithm to standardize the data using Scaler functionality of Spark MLlib. It standardizes features by either scaling data to unit variance and/or scaling data based on the man and standard deviation of the dataset.

```
standardScaler =
StandardScaler(inputCol="features", outputCol="scaled_features")
scaler = standardScaler.fit(df)
scaled_df = scaler.transform(df)
```

Data can then be split into training and test dataset randomly usually in a 80%-20% split, model is trained on 80% of data while tested on rest of 20%.

A few other things that can come handy for machine learning using edge devices are

- Using pre-trained model which can be fitted or trained on high performance machines and then deployed on these devices, these devices can then use those models to make predictions and also send any new data back to the central location
- MLlib provides an option of online learning where machine learning can also be applied on the data being streamed which can reduce latency and burden on the processing power by taking into account all the data points but at different times.

Sample code snippets for machine learning were utilized from DataCamp[14].

## 8 DATA STORAGE/ARCHIVAL

Since storage is very limited on the pi cluster and even the edge devices that are commonly used, they can not afford to store a lot of data so one good option is to archive data at regular intervals or based on the size to a central cloud location like AWS, Azure or Google Cloud. In this project, various options provided by Google Cloud were explored.

Google Cloud platform provides different storage solutions based on different applications and workloads requirements.

Some of the main products that make sense from the topic at hand are

- Persistent Disk - Fully managed storage used for taking backup of virtual machine, this can be a good option for the edge devices that have some container installed on them.
- Google Cloud Storage - This is a good option for storing images and media files.
- Google Cloud Big Table - This is also a good option for IoT devices that do not have standard data structures like rows and columns since this storage can also use noSQL databases etc.

### 8.1 Methods to transfer Data

Google Cloud storage platform provides various ways to transfer data between various cloud resources and other servers. These include JSON or XML based APIs or even command line utilities to perform such tasks.

### 8.2 JSON APIs

Any cloud service including Google Cloud Platform provide JSON APIs that can be called programmatically to store objects in the cloud remotely using HTTP.

Steps required to upload objects in Google Cloud using JSON APIs

- Get an authorization access token from the OAuth 2.0 Playground. Configure the playground to use your own OAuth credentials.
- Add the object's data to the request body.
- Use cURL to call the JSON API with a POST Object request

```
curl -X POST --data-binary @file_name \
    -H "Authorization: Bearer Token_Key" \
    -H "Content-Type: Object" \
```

### 8.3 Command Line Utilities

Google also provides command line utility called gsutil which can be used to perform several options in cloud storage.

Creating a bucket which is a storage container on Cloud

```
gsutil mb -l us-east1 gs://my-bucket/
```

Copying objects from local folder to cloud

```
gsutil cp my-local-folder/my_file gs://my-bucket/my-folder/
```

## 9 CHALLENGES FACED

While working on this project, there were several challenges that had to be overcome in order to achieve the final goal. Some of them are listed below.

### 9.1 Change in filename from slaves to workers on hadoop 3.1.0

There has been a change in the file name for defining the host names of slaves and master in Hadoop from 3.1.0, workers file has to be used compare to slaves file in previous versions, this created some confusion and as a result workers not being started on the hadoop cluster.

### 9.2 Spark cluster failing due to memory issues

Spark cluster kept failing with memory issues initially and created a lot of issues in running any machine learning algorithms. This problem was only solved after referencing a lot of forums and online discussions to understand the configuration changes. Changing swap file size helped too.

### 9.3 Running deep learning libraries on Pi cluster

Running deep learning algorithms like Theanos, Keras and TensorFlow on Pi cluster was also unsuccessful primarily due to memory size and incompatibility of some libraries with Pi operating system. A lot of attempts were made to have these libraries work but could not be completed successfully within the available time.

### 9.4 Issue in running jupyter with spark on pi

There were also some compatibilities with jupyter and spark on Pi since some libraries were not compatible with each other. Also in order to run jupyter on any of Pi, it can be done by using VNC and remotely logging into the server or connecting a monitor to the Pi.

## 10 NEXT STEPS

While, this project started with setting up a hadoop and spark cluster on raspberry pi, an effort was made to also try various deep learning libraries on spark like Keras and TensorFlow but those objectives could not be achieved due to memory limitations of Raspberry Pi. Below are some of ideas that can be used to enhance the project.

### 10.1 Running DeepLearning on Pi cluster

As discussed earlier, running deep learning libraries on Pi cluster can be very beneficial from the use case of edge computing so

making those work successfully is a good step in moving towards broadening the usage of Pi.

### 10.2 Streaming Data from Sensors

Apache Kafka can be easily used to stream data from several add-on sensors on Pi and this data can then be used to perform some actions using the Pi. For example, if sensor detects heat, Pi can send a message or email to user in order to alert them, similarly a camera can be easily installed on Pi and then used for streaming images and making decisions based on the images.

## 11 CONCLUSION

While Raspberry Pi individually or as a cluster has a lot of potential to be used for edge devices, it can not be treated as a one stop shop for all kinds of problems. It can be used very well for use cases where we need a device which is more complex and flexible than a simple microcontroller but costs less than a server, but use cases where there is lot of data processing involved, which requires a lot of in memory processing, it may fall short of in achieving its full potential. Nevertheless Raspberry Pi is a very nifty tool, but it has a long way to be used in the industry.

## REFERENCES


[1] Apache. [n. d.]. Spark MLlib. Web Page. ([n. d.]). https://spark.apache.org/mllib/

[2] DataBricks. [n. d.]. MLlib and Machine Learning. Web Page. ([n. d.]). https://docs.databricks.com/spark/latest/mllib/index.html

[3] Google. [n. d.]. Google Cloud Storage. Web Page. ([n. d.]). https://cloud.google.com/storage/

[4] IntelliPaat. 2016. Apache Kafka. Web Page. (December 2016). https://intellipaat.com/blog/what-is-apache-kafka/

[5] Darren JW. 2015. Installing Apache Spark on a Raspberry Pi 2. Web Page. (April 2015). https://darrenjw2.wordpress.com/2015/04/17/installing-apache-spark-on-a-raspberry-pi-2/

[6] Mark litwintschik. 2017. 1.1 Billion Taxi Rides With Spark 2.2 and 3 Raspberry Pi 3 Model Bs. Web Page. (September 2017). http://tech.marksblogg.com/billion-nyc-taxi-rides-spark-raspberry-pi.html

[7] Xiangrui Meng. [n. d.]. MLlib: Scalable Machine Leaning on Spark. Web Page. ([n. d.]). https://stanford.edu/~rezab/sparkworkshop/slides/xiangrui.pdf

[8] OpenSource.com. [n. d.]. What is Raspberry Pi. Web Page. ([n. d.]). https://opensource.com/resources/raspberry-pi

[9] R. Kelley Pace and Ronald Barry. [n. d.]. California Housing. Web Page. ([n. d.]). http://www.dcc.fc.up.pt/~ltorgo/Regression/cal_housing.html

[10] Srini Penchikala. 2015. Big Data Processing with Apache Spark. Web Page. (January 2015). https://www.infoq.com/articles/apache-spark-introduction

[11] rasberrypi.org. 2017. raspberrypi-ui. Web Page. (July 2017). https://github.com/raspberrypi-ui/piclone/blob/master/src/backup

[12] Niranjan Tallapalli. 2017. Hadoop and Spark Installation on Raspberry Pi-3 Cluster. Web Page. (February 2017). https://tekmarathon.com/2017/02/15/hadoop-and-spark-installation-on-raspberry-pi-3-cluster-part-2/

[13] Wikipedia. 2018. Edge Computing. Web Page. (April 2018). https://intellipaat.com/blog/what-is-apache-kafka/

[14] Karlijn Willems. 2017. Apache Spark Machine Learning Tutorial. Web Page. (July 2017). https://www.datacamp.com/community/tutorials/apache-spark-tutorial-machine-learning

# Automated Apache Spark Cluster Deployment on AWS EC2 using Ansible

Sandeep Khandelwal
Indiana University
Bloomington, IN 47408, USA
skhande@iu.edu

## ABSTRACT

This project is about developing a module for automated deployment of Apache Spark Cluster on AWS (Amazon Web Services) EC2 (Elastic Compute Cloud) instances on a single click and performs various data processing related tasks on Apache Spark Cluster. This project provides option to deploy Apache Spark Cluster and terminate the Apache Spark Cluster once data processing task is completed. This project also does the benchmarking of deployment, data processing and termination of the instances.

## KEYWORDS

hid-sp18-511, Volume: 9, Chapter: Ansible, Status: 100.
AWS, EC2, Apache Spark, Apache Hadoop, Ansible

## 1 INTRODUCTION

This module does infrastructure related tasks of provisioning of AWS [5] EC2 [6] instances, setting up the public/private key pair for secure connection with EC2 [6] instance and configure inbound-/outbound traffic rules on provisioned machines to allow connectivity on various ports. This module also perform Apache Spark [7] installation on the provisioned EC2 [6] instances using master/-worker model and perform all required configuration. One of the machine will work as a master node and other machines will work as worker node. User can do various data processing related tasks once Apache Spark [7] Cluster is setup. Apache Spark [7] Cluster can be terminated once data processing related task complete.

The module expose command line options for Apache Spark [7] Cluster deployment and termination of machines once the data processing related task complete. User can connect to Apache Spark [7] Cluster and perform various data related tasks. Benchmarking is done for measuring the deployment, data processing and termination of the instances.

Ansible [1] automation tool is used in this project to provision EC2 [6] instances, deploy Apache Spark [7] Cluster on EC2 [6] instance and terminate AWS [5] EC2 [6] instances once the data processing related task complete.

## 2 TECHNOLOGIES USED

This section describes the technology used in the project including AWS [5] (see Section 2.1), Apache Spark [7] (see Section 2.2), and Ansible [1] (see Section 2.3).

### 2.1 AWS

AWS [5] is a secure cloud service provider that provide various on-demand services. AWS [5] provides services in Compute, Storage, Database, Migration, Networking and Content Delivery, Developer Tools and many more. These services are easy to setup and can be configured in few minutes. All the services are on demand service and AWS [5] charge only for the duration service is being used. This is very effective and suitable for both small and large customers.

The following AWS configuration is used in this project:

- `project_name: ec2_spark`
- `region: us-west-2`
- `env: stg`

*2.1.1 EC2.* EC2 [6] is AWS secure and resizable compute service. EC2 [6] instances can be created on demand with different configuration of memory, CPU power and hard disk. EC2 [6] instances can be added and terminated at run time based on the load of application. Adding or terminating EC2 [6] instances is very easy and can be done in few minutes.

Two EC2 [6] instances are created in this project with following configuration. One of the instance is used for master node and another one for worker node.

Table 1 provides the EC2 [6] configuration details.

**Table 1: Configuration**

| Node | AMI | CPUs | RAM | Disk |
|---|---|---|---|---|
| Apache Spark Master | ami-4e79ed361 | 1 | 2 GB | 8GB |
| Apache Spark Worker | ami-4e79ed361 | 1 | 2 GB | 8GB |

*2.1.2 Security Group.* A *Security Group* restricts the inbound and outbound traffic for EC2 [6] instance. We can define the allowed incoming and outgoing traffic using Security Group and assign it to the instance. Security Group works as firewall and allow only permitted traffic on EC2 [6] instance.

We create the security group `ec2_spark_stg_security_group` with the following inbound and outbound ports open in this project and assign it to the EC2 [6] instances created.

- `Inbound: 80, 8080, 22, 7178, 8181, 7077, 443`
- `Outbound: all`

*2.1.3 Key Pair.* A *Key Pair* is used to connect EC2 [6] instance securely. We need to create Key Pair and provide the private key to connect the EC2 [6] instance.

`ec2_spark_stg_key` Key pair is created in this project. The private key with the name `ec2_spark_stg_key-private.pem` is created and saved on the user machine and this can be used for making the ssh connection to EC2 [6] instances.

### 2.2 Apache Spark

Apache Spark [7] is analytics engine for large scale data processing. Apache Spark [7] has high performance engine, very easy to use

and provide option to plug-in third party components. Apache Spark [7] is a fast, in-memory data processing engine. It efficiently execute streaming, machine learning or SQL workloads that require fast access to data. Developers can create applications using dataset in Hadoop [2] to use Apache Spark [7].

The following configuration is used in this project for Apache Spark

- spark_version: 2.3.0
- spark_hadoop_version: 2.7
- spark_temp_dir: /tmp
- spark_working_dir: /var/lib/spark
- spark_install_dir: /opt/spark
- spark_mirror: http://apache.claz.org/spark/spark-2.3.0/
- spark_master_memory_mb: 1024
- spark_master_work_port: 7077
- spark_master_ui_port: 8080
- spark_worker_memory_mb: 1024
- spark_worker_work_port: 7178
- spark_worker_ui_port: 8181

## 2.3 Ansible

Ansible [1] is open source software for the infrastructure automation, configuration management and application deployment. Automation tasks are defined in Ansible Playbook using YAML language. For communication to hosts, Ansible [1] uses OpenSSH.

For this project Ansible [1] is used to create and setup EC2 [6] instances, deploy Apache Spark [7] Cluster on EC2 [6] instance and perform various data processing related tasks on Apache Spark [7] Cluster and terminate Apache Spark [7] Cluster.

# 3 DEPLOYMENT

This section describes the setup (see Section 3.1) required for the project, steps for the deployment (see Section 3.2) of Apache Spark [7] Cluster on EC2 [6] instances, explanation of data processing (see Section 3.3) tasks that can be done on Apache Spark [7] Cluster and termination steps (see Section 3.4) of EC2 [6] instances once the data processing tasks complete.

## 3.1 Setup

This section describes the setup steps for Ansible, AWS Key, Git Repository and various configuration options that can be updated as per the project requirements.

*Ansible.* Download and install Ansible [1] on the machine where deployment script will be executed.

Ansible [1] can be downloaded from https://www.ansible.com/resources/get-started

Download and install Ansible from the above URL and validate the installation by typing the following command on Unix console:

```
$ ansible-playbook --version
```

You should see the following output. Version number and directories could be different in your case.

```
ansible-playbook 2.5.0 version = 2.7.13
(default, Jan 24 2018, 21:48:31) [GCC 5.4.0 20160609]
```

*AWS key.* Create AWS account if not exist already and download the AWS [5] ID and secret key. These keys will be used in authentication with AWS [5].

Enter the following command with key values on Unix console:

```
export AWS_ACCESS_KEY_ID='<AWS_ACCESS_KEY_ID value>'
export
AWS_SECRET_ACCESS_KEY='<AWS_SECRET_ACCESS_KEY value>'
```

Validate the variable by typing the following command on Unix console:

```
echo AWS_ACCESS_KEY_ID
echo AWS_SECRET_ACCESS_KEY
```

You should see the values as output which were previously set by export command.

*Git repository.* Setup the git repository:

```
export HID=hid-sp18-511
mkdir -p ~/github/cloudmesh-community
cd ~/github/cloudmesh-community
git clone
https://github.com/cloudmesh-community/$HID.git /project-code
```

Validate code has been cloned into the directory.

*Configuration options.* There are various configuration option that can be updated based on the requirement.

**AWS related configuration options:**

```
cd ~/github/cloudmesh-community/$HID.git/project-code/
group_vars/all
```

Open main.yml file and update the following information as per the requirement:

```
project_name: <specify the project name>
region: <specify AWS region>
env: <deployment environment>
```

**EC2 related configuration options:**

```
cd ~/github/cloudmesh-community/$HID.git/project-code/
roles/provisionec2/defaults
```

Open main.yml file and update the following information as per the requirement

```
ami_image: <EC2 AMI image type>
instance_type: <Instance type>
```

**Apache Spark [7] related configuration options:**

```
cd ~/github/cloudmesh-community/$HID.git/project-code/
roles/sparkmaster/defaults
```

Open main.yml file and update the following information as per the requirement for Apache Spark [7] master

```
spark_version: <Spark master version>
spark_hadoop_version: <Hadoop version>
spark_temp_dir: <Spark master temprary directory>
spark_working_dir: <Spark master working directory>
spark_install_dir: <Spark master installation directory>
spark_mirror: <Spark master mirror URL>
spark_master_memory_mb: <Spark master memory>
spark_master_work_port: <Spark master work port>
spark_master_ui_port: <Spark master UI port>
```

```
cd ~/github/cloudmesh-community/$HID.git/project-code/
roles/sparkworker/defaults
```

Open main.yml file and update the following information as per the requirement for Apache Spark [7] worker

```
spark_version: <Spark worker version>
spark_hadoop_version: <Hadoop version>
spark_temp_dir: <Spark worker temprary directory>
spark_working_dir: <Spark worker working directory>
spark_install_dir: <Spark master installation directory>
spark_mirror: <Spark worker mirror URL>
spark_master_work_port: <Spark master work port>
spark_master_ui_port: <Spark worker UI port>
spark_worker_work_port: <Spark worker work port>
spark_worker_ui_port: <Spark worker UI port>
```

## 3.2 Deploy Apache Spark Cluster

Execute the following command on the Unix console to deploy Apache Spark Cluster [7].

```
cd ~/github/cloudmesh-community/$HID.git/project-code/
ansible-playbook site.yml --tags `provision'
```

This command will perform following tasks:

(1) Create Security Group in AWS
(2) Create Key Pair in AWS
(3) Provision AWS EC2 instance for master
(4) Provision AWS EC2 instance for worker
(5) Create Spark user and group on master
(6) Setup Spark specific directories on master
(7) Download and unarchive Apache Spark on master
(8) Download and install Java on Spark master
(9) Setup Apache Spark configuration files on master
(10) Start Apache Spark master service on master
(11) Create Apache Spark user and group on worker
(12) Setup Apache Spark specific directories on worker
(13) Download and unarchive Apache Spark on worker
(14) Download and install Java on worker
(15) Setup Apache Spark configuration files on worker

*Security Group:* Figure 1 shows security group created in the deployment.

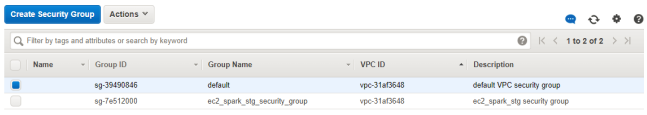


**Figure 1: Security group**

*Key Pair:* Figure 2 shows Key pair created in the deployment.

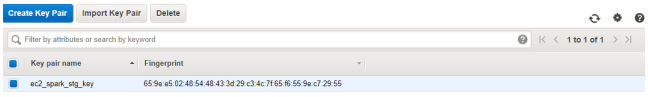


**Figure 2: Key pair**

**EC2 [6] instances:**


**Figure 3: EC2 instance**

Figure 3 shows EC2 [6] instances created in the deployment.

```
cd ~/github/cloudmesh-community/$HID.git/project-code
/inventory
```

Open hosts file and find out the IP address in [sparkmaster] and [sparkworker] section.

Connect to the Spark worker node using ssh:

```
ssh -i ec2_spark_stg_key-private.pem
ubuntu@<Spark worker IP adddress>
```

Once you connect to the Spark worker, execute the command:

```
sudo start-slave.sh spark://<Spark master IP address>:7077
```

Validate Spark Apache Cluster up and running by typing the below url in browser

```
http://<Spark master IP address>:8080
```

Figure 4 shows Apache Spark [7] Cluster URL.

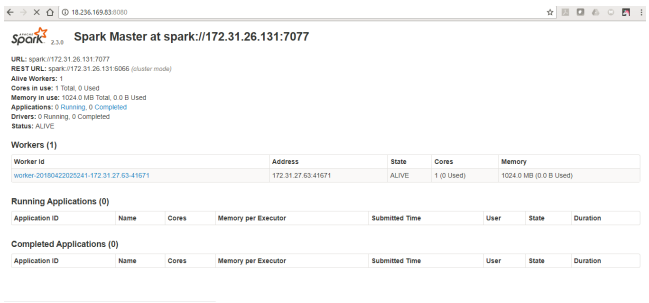


**Figure 4: Spark Cluster URL**

Connect to Apache Spark [7] master and worker and validate Apache Spark shell is functioning properly. Run the following command on both master and worker node.

```
cd /opt/spark/spark-2.3.0-bin-hadoop2.7/bin
./spark-shell
```

Figure 5 shows Apache Spark [7] shell command prompt.

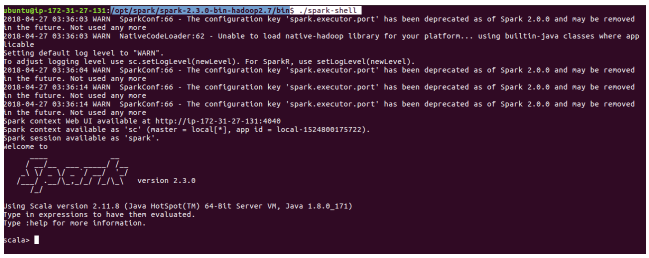


**Figure 5: Spark Shell**

### 3.3 Data processing on Apache Spark Cluster

Apache Spark [7] Cluster can be used for various data related tasks. Apache Spark [7] can be used for the Batch processing, Streaming, Machine learning etc. These data processing related tasks can be written Java, R, Python or Scala.

Apache Spark [7] distribution come along with multiple example code. These example code can be used as a starting point for the hands-on. These example code are available in Java, R, Python and Scala language.

In this section we will see how data processing related tasks written in Python language can be submit to Apache Spark [7] Cluster. We will use Python streaming example code in Apache Spark [7] distribution to demonstrate the submission of application to Apache Spark [7] Cluster.

We will use network word count example for the demonstration. This example code can be found at the following path in both master and worker node.

```
/opt/spark/spark-2.3.0-bin-hadoop2.7/examples/src/main \
/python/streaming/network_wordcount.py
```

This program can be executed using the following command.

```
cd /opt/spark/spark-2.3.0-bin-hadoop2.7
sudo bin/spark-submit --master spark://172.31.27.119:7077 \
examples/src/main/python/streaming/network_wordcount.py \
localhost 9999
```

This streaming program require hostname and port number as argument. This streaming program will connect to the TCP server specified using the hostname and port number argument to receive the data.

Execute the program to listen to the TCP server on local host at port 1000 using the following command.

```
cd /opt/spark/spark-2.3.0-bin-hadoop2.7
sudo bin/spark-submit --master spark://172.31.27.119:7077 \
examples/src/main/python/streaming/network_wordcount.py \
localhost 9999
```

Once the program is executed, it will wait for the data on TCP server on localhost at port 9999.

We could create a virtual TCP server using the following command.

```
sudo nc -lk 9999
```

The above command will create the TCP server on local host at port 9999. Any data specified in the console will be received by the streaming program in batches of 1 second interval.

The streaming application will count the number of different words in each batch and print the initial 10 words on the console.

Figure 6 shows the command to run Python stream program and submit to Apache Spark [7] Cluster.

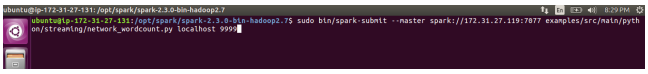

**Figure 6: Python Spark stream run command**

Figure 7 shows the command to run TCP server on a particular host and port number. We are running the TCP server on localhost at port 9999 in this demonstration. We have ingested few words into the server.

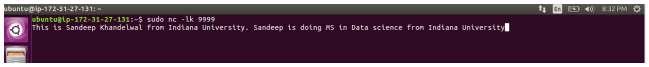

**Figure 7: Python stream TCP server**

Figure 8 shows the output of stream once few of the words are ingested into the TCP server.

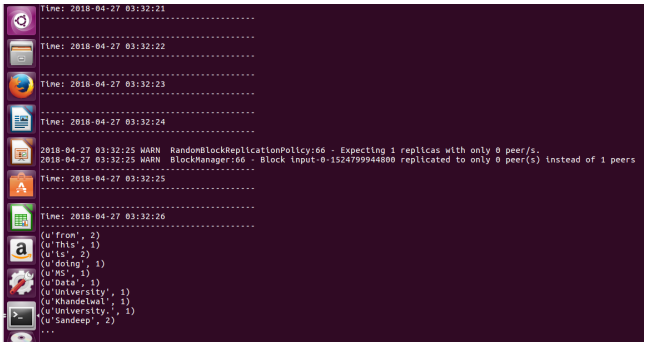

**Figure 8: Python stream output**

### 3.4 Terminate Apache Spark Cluster

Execute the following command on the Unix console to terminate Apache Spark Cluster [7].

```
cd ~/github/cloudmesh-community/$HID.git/project-code/
ansible-playbook site.yml --tags `terminate'
```

This command will perform following tasks:

- Terminate Apache Spark master node
- Terminate Apache Spark worker node

Validate Apache Spark [7] master and Apache Spark [7] worker nodes have been terminated by login to AWS [5] console.

Figure 9 shows EC2 [6] instances terminated.

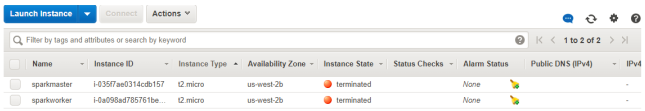

**Figure 9: EC2 instance termination**

## 4 PROJECT VIDEO

Two videos to demonstrate the Apache Spark [7] automated deployment has been created and is made available at `Video_Part_1` [3] and `Video_Part_2` [4].

The first and second videos are sequential and has to be watched in the order.

This video walk through all the prerequisites required for the project, demo of Apache Spark [7] cluster deployment on EC2 [6] instances and termination.

## 5 RESULTS

Table 2 provide the duration for deployment of Spark [7] Cluster on two EC2 [6] instances. Apache Spark [7] master is deployed on one EC2 [6] instance and Apache Spark [7] worker is deployed on another EC2 [6] instance.

**Table 2: Deployment and Termination Results**

| Setup | Duration (in minutes) |
|---|---|
| Apache Spark Cluster deployment | 21 mins |
| Apache Spark Cluster termination | 1 min |

Table 3 provide the duration for processing of stream data of 10 words once it has been typed on the TCP server.

**Table 3: Data Processing Results**

| Data Processing | Duration (in second) |
|---|---|
| Data processing with 2 nodes | 1/10 seconds |

## 6 CONCLUSION

We are successfully able to deploy Apache Spark [7] Cluster on AWS [5] EC2 [6] instances using Ansible [1] deployment and configuration tool on a single click. The amount of duration for end to end deployment was only few minutes instead of hours/days which was the case in earlier days when Cloud was not there. We were able to do streaming using deployed Apache Spark [7] Cluster. This could be used for any kind of data processing need. We were also able to terminate the instances on a single click only in 1 minute.

This project could be further enhanced to add other AWS [5] EC2 [6] services. We could create service for virtual network, storage etc as per requirement. This project could be updated to provide options for specifying the number of nodes to be setup in the cluster

## ACKNOWLEDGMENTS

The authors would like to thank Dr. Gregor von Laszewski for his support and suggestions to write this paper ans providing the technical assistance.

## REFERENCES

[1] Ansible. 2018. Welcome to Ansible documentation. Web Page. (2018). https://www.ansible.com/ Accessed: 2018-04-02.
[2] Apache Hadoop. 2018. Welcome to Apache Hadoop documentation. Web Page. (2018). http://hadoop.apache.org/ Accessed: 2018-04-02.
[3] Sandeep Khandelwal. 2018. Apache Spark deployment demo part 1. Web Page. (2018). https://d1b10bmlvqabco.cloudfront.net/attach/jbkvbp3ed3m2ez/j6r57sr2IDo/jgenhk1lk5ly/hidsp18511_AWS_EC2_Deployment_1.mp4 Accessed: 2018-05-01.
[4] Sandeep Khandelwal. 2018. Apache Spark deployment demo part 2. Web Page. (2018). https://d1b10bmlvqabco.cloudfront.net/attach/jbkvbp3ed3m2ez/j6r57sr2IDo/jgeniacbvn7m/hidsp18511_AWS_EC2_Deployment_2.mp4 Accessed: 2018-05-01.
[5] Amazon Web Services. 2018. Welcome to AWS documentation. Web Page. (2018). https://aws.amazon.com/ Accessed: 2018-04-02.
[6] Amazon Web Services. 2018. Welcome to AWS EC2 documentation. Web Page. (2018). https://aws.amazon.com/ec2/ Accessed: 2018-04-02.
[7] Apache Spark. 2018. Welcome to Apache Spark documentation. Web Page. (2018). https://spark.apache.org/ Accessed: 2018-04-02.

# Service for Managing the Public Key

Uma M Kugan
Indiana University
107 S. Indiana Avenue
Bloomington, Indiana 43017-6221
umakugan@iu.edu

## ABSTRACT

SSH keys are used to control access to any system using public and private key pair. The user creates the private keys and store it securely on his machine and shares the public key. Usually IT admins distributes and manage SSH keys which creates more hassle for the admins to manage. In this paper, we are going to create a rest service to manage their public key.

## KEYWORDS

hid-sp18-513, Volume: 9, Chapter: Security, Status: 100.
Security, Key Management

## 1 INTRODUCTION

SSH is a way for anyone to connect to any servers in a more secured way. Any information that is exchanged between host computer to the server is encrypted which prevents someone from snooping the data. There are two ways by which user can authenticate to connect to the servers: one by providing user name and password combination or by using SSH keys. SSH keys are used in most organizations. The problem with SSH keys is keeping track of whose SSH key has been placed on which machines and making sure that the keys get changed every couple of days/weeks so that it is hard for anyone to guess what the key Is and try to attack the server by forging your identity. One of the main issues is that IT admins have to spend a lot of time visiting each server, changing out the private keys then go back to each user and have them make changes on their laptop [6]. The rest api provides the easier solution to exchange keys between the servers and in this project we have created few api to transfer keys between servers and also to properly manage them.

### 1.1 What is SSH?

The SSH protocol encrypts to secure the connection between a client and a server. All user authentication including commands, results and file transfers are encrypted to protect against attacks in the network.

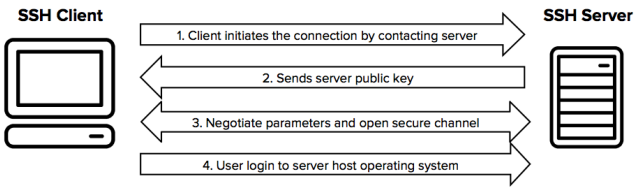


**Figure 1: SSH Protocol [5]**

## 2 HOW SSH WORKS

SSH uses a client-server model of computing, allowing users to establish secure communications between two hosts, whether UNIX, Windows or virtually anything else. SSH is also increasingly important as the use of cloud computing grows, helping organizations limit exposure while connecting to a cloud-based virtual machine over the Internet by providing a local gateway-style system via an SSH endpoint. The server enables incoming SSH connections to a host and handles user authentication, authorization and related tasks. The client connects to an SSH server and makes requests after authenticating to the server. An SSH session is the ongoing connection between a client and a server begins after the client successfully authenticates to a server and ends when the connection terminates. Once the system authenticates a user, one or more channels may be opened within the connection, with each channel acting as an individual data link and separate pathway for information [2].

## 3 ROLE OF SECURE SHELL

The Secure Shell (SSH) protocol, developed in the mid-1990s, and it used everywhere both in traditional and virtual environments to authenticate Unix users and their applications to remote, internal systems and encrypt the resulting session traffic. In order to authenticate, an user or an application must share its private SSH key file to a target system that possesses its corresponding public SSH key file. Once the key pair is validated, the user or application is granted access to the protected account. In effect, this means that any user or application with access to a private key may access any target system such as Unix and Linux systems, virtual machines, network devices, or file transfer solutions that contains the corresponding public key. In a one-to-many environment, in which one private SSH key is used to attain privileged access to many target systems, organizations can no longer afford to ignore these privileged private keys. To protect the heart of the enterprise,organizations must proactively secure and manage all of their privileged account credentials, including both passwords and SSH keys, to achieve consistent, comprehensive protection against advanced attacks [1]

## 4 SSH KEY

SSH keys are basically files that are generated by the OpenSSH program on any computer. An SSH Key consists of two parts. A public key and a private key. An SSH key is an access credential in the SSH protocol in which keys are primarily used for login to one server from other server without entering the passwors and it is also used in automated processes and used by system administrators for implementing single sign-on.

### 4.1 Private and Public Key

The private key file is kept hidden usually in the home directory which can be accessed when connecting to a server. The public key is placed on the server that we are trying to get access to. Now, when we try to connect to the server, computer presents information to the server via the SSH key that proves to the server its really you trying to get access. Without the public SSH key on the server and private key on the computer nobody can access your account using SSH [5].

### 4.2 Authorized Key

An authorized key in SSH is a public key used for granting login access to users and this authentication mechanism is defined as public key authentication. Authorized keys is the file that contains public key of the other server from which passwordless sign on can be initiated and it is configured separately for each user. It is usually present in `.ssh` under home directory. Key location and name are editable in SSH server configuration files. To properly secure and manage, it is usually named same as root owned location [5]. A key pair can be created using the ssh-keygen in openssh and then the public key is copied to server using ssh-copy-id tool.

## 5 WHAT IS SSH-KEYGEN

Ssh-keygen is a tool for creating new authentication key pairs for SSH. SSH supports several public key algorithms for creating authentication keys.

### 5.1 RSA

It is an old algorithm based on the difficulty of factoring large numbers. It is quite possible the RSA algorithm will become practically breakable in the foreseeable future and hence it is advisable to choose some other algorithm. All SSH clients support this algorithm [5].

### 5.2 DSA

It is an old US government Digital Signature Algorithm which is based on the difficulty of computing discrete logarithms. DSA in its original form is no longer recommended [5].

### 5.3 ecdsa

A new Digital Signature Algorithm standardized by the US government, using elliptic curves. This is probably a good algorithm for current applications. Most SSH clients now support this algorithm [5].

### 5.4 ed25519

This is a new algorithm added in OpenSSH. Support for it in clients is not yet universal [5].

## 6 SSH-COPY-ID

ssh-copy-id installs an SSH key on a server as an authorized key. Main use of SSH Copy Id is to provision access without prompting the user for the password. The ssh copy id command edits the authorized keys file on the server. It creates the `.ssh` directory if it does not exist and it also creates the authorized keys file if it does not exist and then ssh key is copied to server.

## 7 SSH CLIENT CONFIGURATION FILES

The ssh program on a host receives its configuration from either the command line or from configuration files:

- `~/.ssh/config`
- `/etc/ssh/ssh_config`

ssh command gets the configuration data from the below three mentioned sources in the order they are stated:

- command line options
- user-specific configuration file (`~/.ssh/config`)
- system-wide configuration file (`/etc/ssh/ssh_config`)

[5].

## 8 SSH-COPY-ID

ssh-copy-id installs an SSH key on a server as an authorized key. Main use of SSH Copy Id is to provision access without prompting the user for the password. The ssh copy id command edits the authorized keys file on the server. It creates the `.ssh` directory if it does not exist and it also creates the authorized keys file if it does not exist and then ssh key is copied to server.

## 9 SSH AGENTS

If private key is encrypted with a passphrase, then passphrase must be entered every time to connect to an SSH server using public-key authentication. Whenever the ssh or scp command is executed, the server will need the passphrase to authenticate and to decrypt the private key before it lets into the remote server.

An SSH agent is a program used to decrypt the private keys and provide access to SSH client programs. User need to provide passphrase once, when adding your private key to the agent's cache. This is used when frequent SSH connections are made. An agent is typically configured to run automatically upon login and persist for the duration of your login session. SSH-Agent is the default agent included with OpenSSH [5].

## 10 SSH KEYS: THE HIDDEN CREDENTIALS

SSH keys are always the major concern to security teams because these credentials can be easily created, and are then difficult to track, manage or control. In any environment without any proper controls in place, any user with access to any machine can generate an SSH key pair that will forever grant that user direct access to the system. And, because there is no built-in oversight to SSH keys, no one may ever know. Further, because SSH is commonly used in automated application to application authentication, SSH key pairs can be generated, distributed and never thought of again, leaving applications and application servers vulnerable to attacks using unmanaged, outdated SSH keys [1]. In the enterprise environment with hundreds of IT users and thousands of systems managing the SSH challenge becomes exponential. A large enterprise may have thousands or even millions of valid SSH key pairs, many of which may no longer be needed by authorized users or applications. As a result, attackers can exploit this vulnerability and gain privileged

access to critical systems. Without central management or knowledge of who is accessing what, organizations cannot see where their risks and vulnerabilities lie much less take action to address them [1].

## 11 SSH KEYS VULNERABILITIES

Vulnerabilities with SSH range from weaknesses in the protocol itself to configuration, implementation and management complexities that can lead to dangerous mistakes. Poor SSH key management practices can result in compromise of administrative privileges, expanded breadth of attack surfaces and other issues that could easily be avoided by proper training and more effective processes backed by automation and tools [2].

### 11.1 Configuration Issue

Application developers must learn how to keep both configuration and private key files secure. With these files, a malicious individual can easily impersonate an authorized user (or host) and easily connect to a remote host and application. Before deploying any new devices in a networked environment, it is always the best practice to change default passwords for all applications, operating systems, routers, firewalls and all other systems [2].

### 11.2 Gaining Access with SSH Key

Lack of defined governance for SSH key-based trust relationships can allow an attacker who compromises one system to quickly pivot from one system to another and extend a breach into other parts of an organization. Enough keys may be stolen, leaked or misused without having terminated their trust relationships to pose a serious, ongoing threat to an organization. Attackers once get access to these keys, they will generates a new key pair and adds the new authorized key to authorized keys file which is not usually monitored or audited. The December 2014 Sony hack included the leak of SSH keys as well as password lists, which led to the compromise of related services and accounts; mishandled SSH keys may have even facilitated the initial compromise [2].

## 12 HOW TO MANAGE SSH RISKS?

To reduce the threat and comply with regulatory requirements, organizations should take a proactive, end-to-end approach to SSH key security and management. Such an approach should include proactive controls to secure, manage and monitor the creation, storage and use of all privileged SSH keys. By employing a layered approach that includes the critical security measures outlined below, organizations can build out a proactive SSH key management program that reduces risks and helps meet regulatory requirements.

### 12.1 Secure Key storage

To reduce the risk of stolen SSH keys, organizations should store private user and application keys in a highly-secure centralized repository that supports strong access controls. In an unmanaged state, SSH keys are stored as system files and can easily be moved or copied. As a result, critical systems are only as secure as the machines or devices on which the private keys are stored. To mitigate this risk, organizations should consider removing SSH keys from vulnerable endpoints and systems and instead storing them in a highly secure, highly available central repository that supports access controls such as automated workflows for elevated-privilege requests and strong authentication to quickly verify user and application identities. With a secure centralized model, organizations can remove critical credentials from individual machines, protect all credentials equally and centrally track all usage of SSH keys by users and applications [3].

### 12.2 Proactive key rotation

Similar to static passwords, static SSH keys pose an on-going risk to organizations. Compromised private SSH keys can provide unauthorized users with permanent backdoor access into critical systems. To mitigate this risk, organizations should rotate all SSH key pairs at regular intervals, just as they rotate privileged passwords today. In unmanaged environments, with keys dispersed across devices, the prospect of frequently rotating thousands of keys can be daunting. However, once all user and application SSH keys are consolidated under one central management system, key pairs can be automatically rotated and public keys can be automatically distributed to target systems. With central management and automated key rotation, organizations can better secure SSH keys and the target systems they protect, implement best practices and comply with regulations without burdening the IT team[3].

### 12.3 Privileged Session monitoring

Proactive protection of privileged credentials is a critical element in privileged account security, but proactive controls must be complemented with equally strong detection and response capabilities. To minimize any potential damage caused by advanced external and inside attackers, organizations should proactively monitor all privileged sessions, including those that occur via SSH. When abnormal activity is detected, security teams should have the ability to remotely terminate the suspicious session to disrupt the potential attack. Similarly, organizations must also record all privileged session activity. When an incident is detected, response teams require immediate access to session recordings and detailed audit logs to determine exactly what happened and what steps must be taken to re-mediate the incident. Access to these privileged session recordings and audit logs can also be granted to auditors to help prove compliance with regulations [3].

## 13 MEASURES TO BE TAKEN FOR SECURING SSH KEYS

Both internal and external auditors must add Secure Shell key scanning and management to their checks. Proper controls and tools must be put in place for managing Secure Shell keys. The real issue is authorized keys, as they are the ones that grant access. No matter how much you try to protect private keys, it is of no help until the millions of existing authorized keys have been sorted out. Regulators must establish firm deadlines for enterprises and execute the revoking of all access when no longer needed. However, implementation and audit guidelines should be clarified to ensure Secure Shell keys are taken into account. Boards, audit committees, Security and Governance and risk management officers must ensure Secure Shell key-based access is properly accounted for in their organizations to avoid civil and criminal liability [7].

### 13.1 Use Passphrase

It is always recommended that keys used for single sign-on should have a passphrase to prevent use of the key if it is stolen or compromised. It is the best practice to use keys without passphrase for fully automated jobs such as backups.

The ssh-agent and ssh-add programs can be used to avoid having to enter the passphrase every time the key is used.

### 13.2 Command Restrictions

A command restriction is a `command=<permitted command>` option added to the beginning of the line in the target server authorized keys file. The copy-id tool does not default any restrictions but it is highly recommended when the key is used for automating operation.

## 14 NIST ISSUES GUIDANCE ON SSH KEY MANAGEMENT

US National Instute of Standards and Technology (NIST) as issued guidance on SSH key management as NIST IR 7966.

### 14.1 REGULATORY COMPLIANCE REQUIRES SSH KEY MANAGEMENT

Typical requirements for compliance include:

- Managing Identities and Credentials - SSH keys are access credentials
- Provisioning and Termination Process for Access - including access based on SSH keys
- Segregation of Duties - elimination of key-based access from test and development systems into production
- Disaster Recovery - limiting attack spread from primary systems to disaster recovery sites and backup systems
- Privileged Access Controls - SSH keys are often used to bypass jump servers
- Boundary definition and documentation of connections for systems that contains sensitive data such as payment systems, financial data environments, patient data environments, or between government information systems
- Incident Response and Recovery

[4].

## 15 PYTHON - API

Python is a high-level object-oriented programming language which is most popular and widely used due to its minimal setup and high availability of large collection of free libraries. The project uses Python 2.7 that is available in the native version of Ubuntu 16.07.

### 15.1 Flask - Rest Service

Flask is a micro framework developed for web development. The framework is widely popular because its light weight and has minimal dependencies. Flask was originally written in Python. Some of the key features of Flask are integrated rest service, secure cookies and unit testing support. Flask-RESTful is an extension of Flask that supports quick and easy API deployments with minimal setup.

## 16 DEPLOYMENT

The Project uses few API to effectively manage the keys and their user authorization in Sqlite Database. Three tables have been created to maintain access request and also the list of users in each server, their groups and their access level.

### 16.1 Setup

*Git repository.* Setup the git repository:

```
export HID=hid-sp18-513
mkdir -p ~/github/cloudmesh-community
cd ~/github/cloudmesh-community
git clone
https://github.com/cloudmesh-community/$HID.git
/project-code
```

Technology: Python, Flask API, Sqlite3
Install **Python 2.7.13** via PIP
Install **pip install flask-restful**
Database: sshkeymgmt.db
Tables: usergrp, user access, user_req_access
API: /`request_access`
/`validate_and_action`
/`create_ssh_key`
/`copy_ssh_key`

User will call the API request access to request for sharing the key from one server to other server with host name and user id of the source and target and also the location of public key in source host.

API validate and action will validate if the user have a login in the target server. If not the request will be declined and the status will be updated on the table.

Once the request is approved, the third API will actually copy the keys to target server. Before it copies, API will validate if the user calling the API have access to target server or whether he is any group and or has admin access.

If key is not present, `create_ssh_key` api can be used to create the ssh key.

Getstatus api is used to get status of the request at any point, user need to pass the status: APPROVED, DENIED, COMPLETED, FAILED.

## 17 SSH KEY MANAGEMENT PRODUCTS

There are several SSH Key Management Products that are readily available in the market. These products will help to generate, manage, store and rotates the key periodically. SSH.com's Universal Key Manager is the industry's leading product which provides risk assessments, process driven, full control and all in one view. There are other products such as FOXPASS which can integrate with LDAP server.

## 18 BENCHMARKS

SSH key management project uses Virtual machine and SQLite as its data storage. There are in total 5 APIs and once all these APIs are running, these services can be called from anywhere. It creates the database sshkeymgmt and three tables in it. All the request to copy key from one server to other server is made via the API and the

information is stored in the table. Second API will either approve and deny all the access request based on whether user have the login in target system. At any point status API will give the list of request for whatever the status that API is called for. Create SSH key api will create the SSH Key if the key is not already present in any server. COPY SSH Key API will copy the key from source to target server only when the request is approved and once if the key is copied successfully it will update the status to request fulfilled. All these API were running very fast and there is no significant delay.

## 19 LIMITATIONS

In this project, we have define a very basic api to manage the ssh keys. This project can be further enhanced and extended to include active directory or LDAP integrations.

## 20 CONCLUSION

SSH is an widely used important protocol that provides encrypted, authenticated communications in a variety of configurations. Public key authentication is very vital to any organization, as it automates access and can easily provide ease of login to accounts with elevated privileges. However, SSH is also especially vulnerable to complexities that involve configuration, implementation and management. In order to reduce risk, proactive monitoring and implementation of best practices both during implementation and afterward is needed.

In a big organizations, there may be huge volumes of keys in use and remediation of an environment where SSH keys and trust relationships have never been fully managed is complex, labor-intensive and time-consuming. But there are frameworks available in the industry which can be used or the project has few basic API to manage the key in smaller environment which can be further enhanced to larger scale.

## ACKNOWLEDGMENTS

The author would like to thank Dr. Gregor von Laszewski for his support and suggestions in writing this report and succesfully completing the project.

## REFERENCES

[1] Cyberark. 2014. *Discover the Keys to Securing SSH.* Technical Report. Cyberark. https://lp.cyberark.com/rs/cyberarksoftware/images/wp-discover-the-keys-to-securing-ssh-10-20-2014-en.pdf

[2] Barbara Filkins. 2015. *Securing SSH with the CIS Critical Security Controls.* Technical Report. SANS,Venafi. https://www.venafi.com/sites/default/files/2016-10/Venafi_Securing_SSH_CSCs_final.pdf

[3] P. Gutmann. 2004. Simplifying public key management, In Computer. *Computer* 37, 101 – 103.

[4] SSH Communications Security Inc. 2018. SSH Key Management. (2018). https://www.ssh.com/iam/ssh-key-management/

[5] SSH Communications Security Inc. 2018. SSH Secure Shell. (2018). https://www.ssh.com/ssh/

[6] Tatu J Ylonen. 2013. User key management for the Secure Shell (SSH). (2013). https://patentimages.storage.googleapis.com/ca/b8/57/0a724a1196261d/US20130117554A1.pdf US Patent App. 13/723,204.

[7] Tatu Ylnen. 2015. *NIST publishes guidelines for SSH key management: What happens next?* Technical Report. Network World. https://www.networkworld.com/article/3005365/network-security/nist-publishes-guidelines-for-ssh-key-management-what-happens-next.html

# Openstack deployment using Swagger and Libcloud

Shagufta Pathan
Indiana University
Bloomington, IN 47408
spathan@iu.edu

## ABSTRACT

This work focuses on leveraging the capability of Swagger Codegen and Apache libcloud for allowing users to easily deploy instances on OpenStack without worrying about the underlying architecture. The users can create an instance, list instances, add floating ip to the instance, add ssh keypair and more, provided the user has permission to perform these activities. Using the REST API endpoints, a fully-fledged instance can be created instead of clicking on the UI form and filling in the details to get the instance created. We also demonstrate the use of cloud-init to send userdata on instance creation by executing some basic shell commands. To achieve this, we choose OpenStack as the cloud infrastructure since it is ever-growing and one of the most popular cloud platform.

## KEYWORDS

hid-sp18-516, Volume: 9, Chapter: REST, Status: 100.
Swagger, REST, Apache libcloud, Python

## 1 INTRODUCTION

OpenStack is an open-source cloud-computing service and is available for free that provides virtual servers and resources to customers. It is mostly deployed as infrastructure-as-a-service [10]. Whereas Apache Libcloud is a Python library that allows to interact with several popular cloud service providers with the help of a unified API to interact with different cloud services. The compute component of libcloud allows to manage cloud and virtual servers offered by different providers [4]. Libcloud helps users to deploy one or more virtual machines on OpenStack and allows to run commands on the virtual machines. The motivation behind using Libcloud is to ensure that the application is portable in terms of the cloud service provider without having to rewrite the Swagger specification file.

Figure 1 shows the high level architecture. The application consists of a REST server implemented using Swagger. The server provides several APIs which allow provisioning and manipulation of different resources on the OpenStack cloud. This implementation of the REST server uses libcloud to abstract the communication with the cloud.

## 2 TECHNOLOGIES USED

This section lists the technologies that were used:

- Python 2.7
- OpenStack
- Swagger Codegen 2.0
- Apache Libcloud 2.3.0
- Chameleon Cloud/Horizon

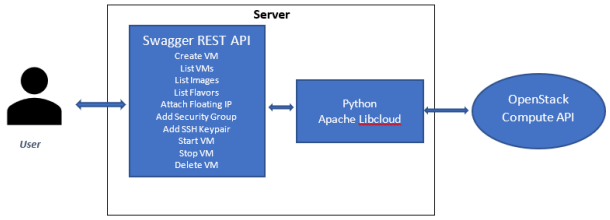


**Figure 1: Project Architecture**

## 3 BACKGROUND ON TECHNOLOGIES USED

### 3.1 OpenStack

OpenStack is a set of software tools to build and manage public and private clouds using pooled virtual resources. OpenStack, managed by OpenStack Foundation, controls large pools of compute, storage and networking resources throughout a datacenter. OpenStack is available to users via OpenStack API, web-based dashboard or through RESTful web services [14]. It allows users to easily deploy virtual machines that can handle a variety of tasks to manage a cloud environment on the fly. OpenStack is made up of a number of components. But there are mainly 9 key components [5]:

**Nova:** This is the cloud computing fabric controller, which is the main part of an IaaS system. It is used for managing and deploying large number of instances to handle computing tasks.

**Swift:** This is the object Storage system which is scalable horizontally by simply adding new servers. It stores files and objects on multiple disk drives and OpenStack is responsible for ensuring data replication and integrity.

**Cinder:** This provides block-level storage with OpenStack Compute Instances and is important in scenarios where data access speed is the most important consideration.

**Neutron:** This provides networking capability for all the OpenStack components and ensures that each component is able to communicate effectively and efficiently.

**Horizon:** This is the dashboard and the only GUI to OpenStack. It allows system administrators to easily see what is going on in the cloud and manage it as needed.

**Keystone:** This maintains authentication service across the cloud operating system and manages a central directory of users mapping to the OpenStack services that they can access. It supports different types of authentications including username and password credentials, token-based and AWS style logins.

**Glance:** This provides image services to OpenStack and allows these images to be used as templates when deploying new virtual machine instances.

**Ceilometer:** This provides telemetry services/billing services to individual users of the cloud. It provides all the information on the usage reporting which keeps track of user's system of usage of the cloud.

**Heat:** This provides orchestration service that orchestrates multiple composite cloud applications using templates.

We focus on using the Compute, Keystone, Horizon and Glance components which is made available through the libcloud APIs.

### 3.2 Apache libcloud

Apache libcloud is a Python library that hides the differences between different cloud provider APIs and gives users the ability to manage different cloud resources using a unified API. The users do not have to worry about the underlying architecture of the cloud provider. Libcloud supports more than 50 cloud providers and allows users to manage resources such as the following categories [4]:

- Cloud Servers and Block Storage
- Cloud Object Storage and CDN
- Load Balancers as a Service
- DNS as a Service
- Container Services
- Backup as a Service

For the purposes of this work, we are only going to focus on Cloud Servers and Block Storage category. Libcloud's compute component that falls under the Cloud Servers and Block Storage category, can manage cloud and virtual servers of different cloud providers. Libcloud provides support for Python 2.7, PyPy, and Python 3 [2].

Libcloud for OpenStack driver constructor takes different arguments with which the OpenStack installation can be done. Arguments describing things such as authentication service API URL, authentication service API version and so on can be specified. Note that majority of the arguments to the driver constructor are optional [3].

To install the latest stable version of apache-libcloud, open a terminal and simply run [4]: **pip install apache-libcloud**

### 3.3 Swagger Codegen

We use Swagger Codegen 2.0 to design the specifications of the services developed and to generate the code based on the specifications developed. Swagger is a tool for developing API specifications based on the OpenAPI Specification (OAS). It allows not only the specification, but the generation of code based on the specification in a variety of languages. "OpenAPI Specification is an API description format for REST APIs. An OpenAPI file can be used to describe the entire API such as available endpoints (/users) and operations on each endpoint (GET /users, POST /users), operation parameters Input and Output for each operation, Authentication methods and contact information, license, terms of use and other information" [11].

Swagger Codegen is one of the major swagger tools that is used to generate server stubs and client libraries from the OpenAPI specification [11]. We are going to use Python (flask) as the preferred language. The specification is written for OpenStack VM deployment and the code has been generated using Swagger Codegen version 2.0. The same specification can be used for any other provider instead of OpenStack as we are using libcloud. The generated code was used to implement useful REST APIs using python scripts. Make sure to have Java 7 or 8 installed on your system for Swagger Codegen to work.

### 3.4 Chameleon Cloud

Chameleon is an experimental testbed funded by the NSF FutureCloud program. It is built over two sites, University of Chicago (UC) and Texas Advanced Computing Center (TACC). It offers a total of over 550 nodes and 5 PB of space in twelve Standard Cloud Unit (SCU) racks and about a quarter of the testbed is configured with OpenStack KVM. Chameleon provides an installation of OpenStack version 2015.1 (Kilo) using the KVM virtualization technology [13]. For the purposes of this work, we are going to interact with Chameleon for TACC for monitoring the creation of virtual machines and other tasks performed on the VM. The easiest way to interact with Chameleon is via the GUI Horizon dashboard as described earlier. To log in to the dashboard, we need to have Chameleon username and password which can be created on the Chameleon portal.

## 4 METHODOLOGY

We are going to focus on identifying resources related to OpenStack for deploying an instance of a virtual machine using the REST APIs instead of using the GUI dashboard by selecting multiple options. The resources identified can be considered as part of NIST Big Data Reference Architecture (NBDRA). The methodology can be divided into documenting various resources, using these resources to provide interfaces to access or create individual resources using the OpenStack libcloud API.

### 4.1 NIST Big Data Reference Architecture (NBDRA)

The NBDRA defines interfaces related to Infrastructure as a Service frameworks. This includes specific objects useful for OpenStack, Azure, and AWS, as well as others. It defines different objects that supports functions such as starting, stopping, suspending, resuming, migration, network configuration, assigning of resources for the virtual machines [1]. The following resources have been identified for interacting with the OpenStack framework:

**List Images:** This service lists all the public and private images hosted on Chameleon Cloud. Images contain a virtual disk that has a bootable operating system installed on it which are useful for creating instances within the cloud. The GET method is used to retrieve all the images of the cloud.

**List Flavor:** This service lists all flavors/sizes that are used to create an instance. Flavors are useful for creating instances within the cloud as they define a number of parameters like sizes of RAM, disk, number of cores etc̀o specify what type of virtual machine to run. The GET method is used to retrieve all the flavors of the cloud.

**Create an instance:** This service creates an instance on the TACC datacenter of Chameleon cloud. The POST method is used to create an instance. The attributes such as instance name, ssh keypair name, security group name, flavor id and image id are sent as payload in the POST request. Only the instance name is a required parameter, all other parameters are picked up from the config file if they are not specified.

**Import SSH Keypair:** This service allows to import the SSH Keypair with the contents of the public key onto OpenStack. Keypairs are SSH credentials that are injected into an instance when it is launched. The POST method is used to import the SSH Keypair name and Public key file location. All the parameters are sent as payload in the JSON format.

**Attach Floating IP:** This service is used to associate the floating IP to the instance from an existing pool of addresses. Once the instance is created, in order to access the instance, a floating IP is required. An instance has a private, fixed IP address and can also have a public or floating IP address. Private IPs are automatically assigned on instance creation but need to assign public IPs manually in order to communicate with networks outside the cloud, including internet [8]. POST method is used for this and instance name is sent as payload.

**Security Group:** This method creates a security group on the cloud, if it is not present already. Security group rules are sets of IP filter rules that are required as they enable users to ping and use SSH to connect to the instance. Security groups define networking access and are applied to all instances within a project [7]. The security group name is sent as payload in the POST request.

**Start an instance:** This method starts a stopped instance. POST method is used, and instance name is sent as payload.

**Stop an instance:** This method stops a running instance. POST method is used, and instance name is sent as payload.

**Delete an instance:** This method destroys the specified instance. DELETE method is used and instance name is sent as the path parameter.

## 4.2 Swagger RESTful APIs

The objects described in the above section can be accessed by the REST service using its own base path. Base paths for the objects of OpenStack will be *cloudmesh/openstack* followed by the object name. Following shows the list of APIs developed and implemented inside the Swagger service:

**/images:** lists images available on the cloud
**/flavor:** lists the flavors available on the cloud
**/listInstances:** lists all the instances available on the cloud
**/createInstance:** creates an instance on the cloud
**/deleteInstance:** deletes a specified instance
**/stopInstance:** stops a specified instance
**/startInstance:** starts a specified instance
**/addkeypair:** imports the keypair name and content on the cloud
**/floatingIP:** associates a floating IP with the specified instance
**/securitygroup:** creates a security group name with the security rules

## 4.3 Custom Configuration File

In order to access OpenStack from libcloud, authentication service API url is required. This is specified in a configuration file defined as *class.yaml*. The authentication credentials including username, password, auth_url, region_name, project_name, version_number and so on can be specified in this yaml file. In order to make the code portable with other clouds, different cloud provider details can also be added to this yaml file. By default, we only support *TACC* as the cloud provider.

Reading the configuration parameters from this file and using the REST APIs, users can easily create instances on the specified cloud without having to either use the command-line tools or the GUI form to enter the details one by one. The configuration details entered in the yaml file are obtained by downloading the environment file called *openrc.sh* file. This file provides project-specific details and credentials that the OpenStack services can use. It can be downloaded from the web interface via the *Access & Security* link, then clicking on the *API Access* tab and clicking on *Download OpenStack RC File* on the top of the page [9].

The class.yaml file used for this work was provided in the class handbook [13]. It has been updated with the latest details that are needed for this work.

## 4.4 Using Cloud-Init with OpenStack

Cloud-init is the Ubuntu package that can handle early initialization of the cloud instance. It is a set of python scripts and utilities that is installed in the Ubuntu Cloud Images. To use cloud-init, we need to use a cloud-init enabled image and most Openstack based public cloud providers support it. Some of the things it can configure are [12]:

- setting up default locale
- adding ssh keys to user's *.ssh/authorized_keys* which can allow them to log in
- setting up mount points
- Configuring network devices
- setting up a hostname
- generating ssh private keys

Cloud-init can be configured via user-data. User-data can be provided during instance launch time. User data can be specified in a number of input formats such as [12]:

**Gzip Compressed Content:** Content compressed as gzip data will be uncompressed and treated like it were not compressed. Compression of data is useful since there is a limit on the amount of user data that can be sent. It is limited to 16384 bytes.

**User-Data Script:** This begins with *#!* or *Content-Type: text/x-shellscript*. This script is executed during first boot at *rc.local-like* level, which is very late in the boot sequence.

**Cloud Config Data:** This content is *cloud-config* data and it begins with *#cloud-config* or *Content-Type: text/cloud-config*.

**Mime Multi Part archive:** If the user wants to specify more than one type of format for example, both user data script and cloud-config, it can be done using this format.

**Include File:** This file is an *include* file and contains a list of urls, one per line. The content read from each of the URLs can be gzipped, mime-multi-part or plain-text. It begins with *#include* or *Content-Type: text/x-include-url*.

**Upstart Job:** Like any other upstart job, the content is placed into a file in /etc/init and will be consumed by upstart. This format begins with *#upstart-job* or *Content-Type: text/upstart-job*.

**Cloud Boothook:** This begins with *#cloud-boothook* or *Content-Type: text/cloud-boothook*. This is the earliest hook available. The content is placed under /var/lib/cloud and then executed immediately.

**Part Handler:** This is a *part-handler* and begins with *#part-handler* or *Content-Type: text/part-handler*. The content is python code that contains a list_types method and a handle_type method and is written to a file in /var/lib/-cloud/data based on its filename.

We are going to demonstrate the use of user data via user-data script. Libcloud for OpenStack allows to use cloud-init using the *ex_config_drive* and *ex_userdata* arguments to create_node function. The user data we are demonstrating is a list of shell commands that will be executed on instance creation and the result is saved to an output file called *output.txt* in the user's home directory. This userdata is configured in the util.py file.

Following is an example showing how the cloud-init arguments can be provided in the libcloud create_node function:

```
userdata = ```#!/bin/bash
echo 'System Information: $(uname -a)' | tee /home/cc/output.txt
echo 'The time is now $(date -R)!' | tee -a /home/cc/output.txt
echo 'The hostname is $(hostname)' | tee -a /home/cc/output.txt
echo 'ifconfig details: $(ifconfig)' | tee -a /home/cc/output.txt
'''
node = driver.create_node(name='vm_name', image=image, size=size,
                    ex_userdata=userdata, ex_config_drive=True)
```

**Figure 2: User-Data Scripts [12]**

Following is another example of user data using cloud-config format. This example just installs nginx and starts it when the instance is created.

As explained above, other user data formats can be sent as cloud-init user data. Other cloud-config examples include Including users and groups, adding a yum repository, Install and run chef recipes, Setup and run puppet, Additional apt configuration etc. Cloud-init can prove to be very useful to set up the cloud instances on creation depending on the user's requirement [12].

## 4.5 Libcloud functions

The libcloud libraries for compute object were imported using the following commands 4:

The following table 1 lists the libcloud functions that were leveraged during the development of the REST APIs [6].

```
cloud_init_config = ```
#cloud-config
packages:
 - nginx
runcmd:
 - service nginx start
'''
node = driver.create_node(name='cloud_init', image=image, size=size
                    ex_userdata=cloud_init_config, ex_config_driv
```

**Figure 3: Cloud Config Example [6]**

```
from libcloud.compute.types import Provider

from libcloud.compute.providers import get_driver
```

**Figure 4: Libcloud libraries [6]**

**Table 1: Libcloud functions leveraged**

| Description | Function |
|---|---|
| To instantiate the driver | libcloud.get_driver (Provider.OPENSTACK) |
| To list images | list_images () |
| To list flavor | list_sizes () |
| To list instances | list_nodes () |
| To list floating IPs | ex_list_floating_ips () |
| To list floating IP pools | ex_list_floating_ip_pools () |
| Attach floating ip to instance | ex_attach_floating_ip_to_node () |
| To list keypairs | list_key_pairs () |
| To create security group | ex_create_security_group () |
| To create security group rules | ex_create_security_group_rule () |
| To get image from id | get_image (img_id) |
| To get flavor from id | ex_get_size (flvr_id) |
| To create instance | create_node () |
| To delete instance | destroy_node () |
| To start instance | ex_start_node () |
| To stop instance | ex_stop_node () |

# 5 ARTIFACTS

The Figure 5 lists the artifacts provided as part of this work:

```
Makefile
Dockerfile
util.py
default_controller.py
openstack.yaml
class.yaml
requirements.txt
```

**Figure 5: Project Artifacts**

Code details

4

*Makefile*: Makefile contains different targets that perform different tasks like generate Swagger server-side code, copy files to the desired location, start and stop the Swagger service, import required packages for the REST services code, test different Swagger REST APIs, build docker image, run the docker container, start and stop the docker container.

*Dockerfile*: Dockerfile contains all the requirements needed to build the docker image such as installing Ubuntu, python 2.7, setting up pyenv, downloading swagger-codegen jar file, setting up the work directory, copying all the files to the desired work directory and starting the swagger service.

*openstack.yaml*: This is a yaml document file which contains the definitions of the REST services for OpenStack deployment conforming to Swagger/OpenAPI 2.0.

*class.yaml*: This file contains the authentication information and other configuration details related to OpenStack. As some information like username, password is sensitive information, it needs to be configured in this file before execution. Default parameters for image, flavor, security group, keypair name can also be specified in this file.

*util.py*: python script for reading the configurations from class.yaml and cloud-init configuration for OpenStack.

*default_controller.py*: This is a Swagger controller file with the actual implementation of the REST services for OpenStack.

*requirements.txt*: This file lists all the dependency python packages that are needed for execution.

## 6 RESULTS

The Swagger REST services can be deployed as a standalone service or as a docker container using the commands specified in the Makefile. The service can be accessed through browser or CURL utility to verify the execution. After running each test target specified in the Makefile (test_VMs, test_images, test_flavors, test_createVM etc.), we can log on to Chameleon cloud to verify and validate the result. Some screenshots of the test execution have been provided.

Figure 6 shows the execution of the REST API for listing images of OpenStack available on the Chameleon cloud. As you can see, all the images, both public and private that are available on the cloud for OpenStack have been listed.

Figure 7 shows the execution of the REST API for listing flavors of OpenStack available on the Chameleon cloud. As you can, all the flavors that are needed for instance creation are listed, and the user can choose any flavor from this list that is required for creating the instance.

Figure 8 shows the execution of the REST API for creating instances and its availability on Chameleon cloud. As you can see, we were able to successfully create two instances *test-VM1-516* and *test-VM2-516* and verify that it is listed on the UI.

Figure 9 shows the execution of the REST API for listing all the instances of OpenStack available on the Chameleon cloud. As you can see, all the instances that have been created by different users are listed, including the ones we created previously.

Figure 10 shows the execution of the REST API for associating floating IP of instance *test-VM2-516* and its result on Chameleon

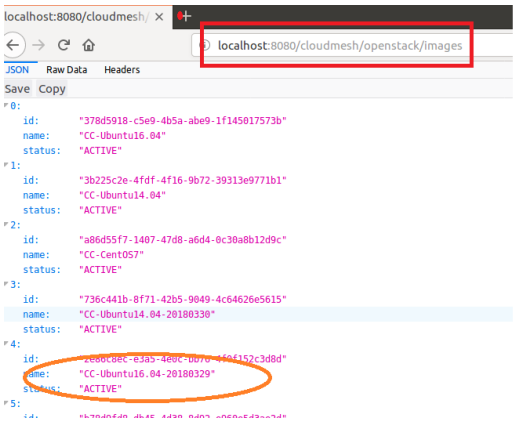


**Figure 6: List Images**

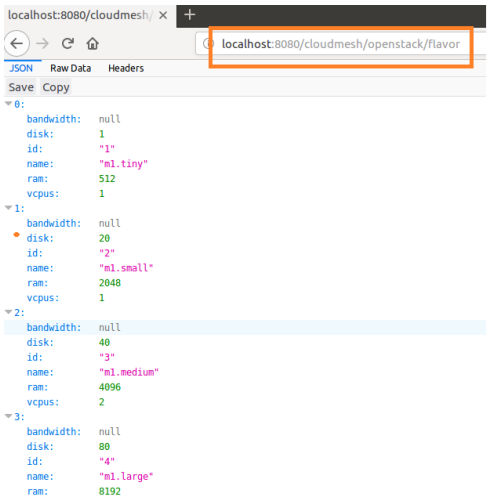


**Figure 7: List Flavors**

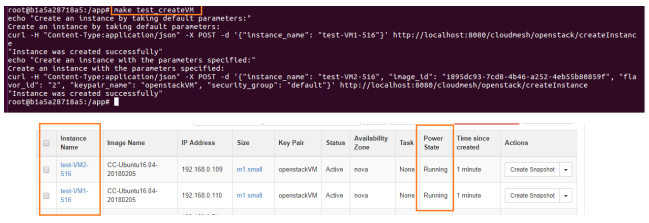


**Figure 8: Create Instance**

cloud. By default, only private IP is assigned on instance creation, assigning floating IP will allow user to access it outside the cloud.

Figure 11 shows the execution of the REST API for deleting instance of OpenStack. Listing all the instances again should not display the deleted instance.

Figure 12 shows the execution of the REST API for stopping instance of OpenStack and its result on the Chameleon cloud. Once the instance is stopped, you can check its status on the UI. It should

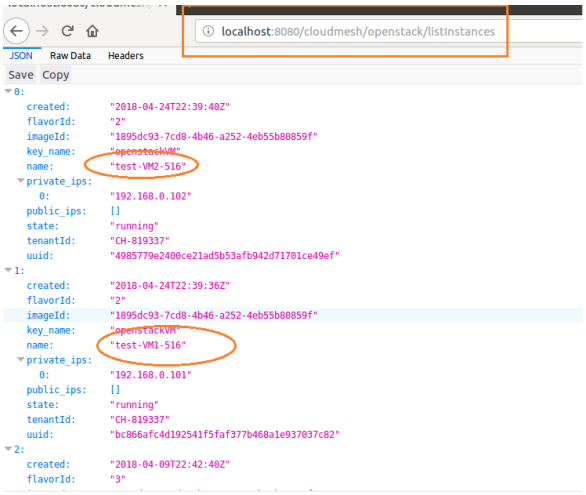


**Figure 9: List VMs**

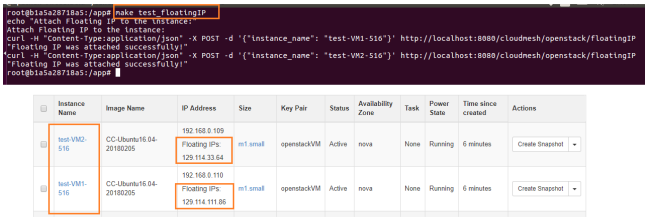


**Figure 10: Floating IP**

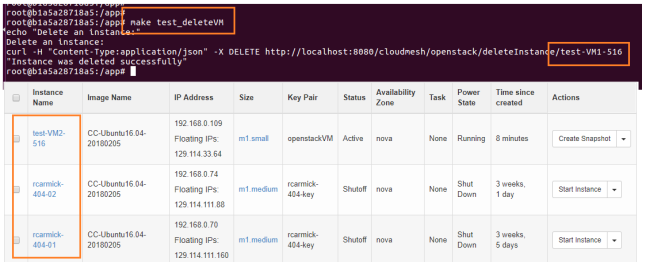


**Figure 11: Delete Instance**

display *Shutoff* against the instance name. Or you can also list the instances again and check its status.

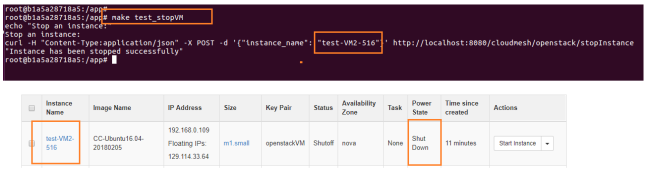


**Figure 12: Shut Off Instance**

Once the instance is up and running, we can log in to that instance via *ssh* using the floating IP associated with the instance. The user *cc* allows automatic login to the instance. We have a REST API that allows associating floating IP to the instance as seen in the screenshots shown earlier. Following is the command to login to the instance: *ssh cc@floatingIPofVM*

Once logged in, we can navigate to /home/cc/ and view the output of the commands executed via cloud-init. The result is saved in output.txt.

A short video explaining the execution details of this work has been created and the link has been added to the README.md file present inside the project-code folder.

## 7 CONCLUSION

We were able to successfully show how the OpenStack instances can be deployed using Apache libcloud with the help of Swagger generated code. We showed how Apache libcloud can be leveraged to perform different tasks like create/delete/start/stop instances or attach floating IP and so on on the OpenStack cloud using the unified API abstracting away the underlying details of the cloud provider. We also demonstrated how the userdata can be sent on the cloud during instance creation using cloud-init configuration and how it can be used to perform different user-defined tasks. All this was achieved with the help of python scripts that leveraged libcloud into RESTful APIs using the Swagger specifications that was created in OpenAPI format. Currently, we are using TACC as the default testbed for interacting with Chameleon cloud. A substantial future work would be take input from user to specify on which testbed/cloud the user would like to deploy their instances.

## ACKNOWLEDGMENTS

The author would like to thank Dr. Gregor von Laszewski for his support and suggestions to write this paper.

## REFERENCES

[1] 2018. NIST Big Data Interoperability Framework: Volume 8, Reference Architecture Interface.
[2] Apache Libcloud. 2013. Compute. Web Page. (2013). http://libcloud.readthedocs.io/en/v0.14.0/compute/index.html Accessed: 2018-04-01.
[3] Apache Libcloud. 2013. OpenStack Compute Driver Documentation. Web Page. (2013). http://libcloud.readthedocs.io/en/v0.14.0/compute/drivers/openstack.html Accessed: 2018-04-01.
[4] Apache Libcloud. 2018. Welcome to Apache Libcloud's documentation! Web Page. (2018). http://libcloud.readthedocs.io/en/latest/index.html Accessed: 2018-03-26.
[5] opensource.com. 2018. What is OpenStack? Web Page. (2018). https://opensource.com/resources/what-is-openstack Accessed: 2018-04-20.
[6] OpenStack. 2013. OpenStack Compute Driver Documentation. Web Page. (2013). http://libcloud.readthedocs.io/en/latest/compute/drivers/openstack.html Accessed: 2018-04-01.
[7] OpenStack. 2018. Configure access and security for instances. Web Page. (2018). https://docs.openstack.org/ocata/user-guide/cli-nova-configure-access-security-for-instances.html Accessed: 2018-03-17.
[8] openstack. 2018. Manage IP addresses. Web Page. (2018). https://docs.openstack.org/ocata/user-guide/cli-manage-ip-addresses.html Accessed: 2018-03-17.
[9] OpenStack. 2018. Set environment variables using the OpenStack RC file. Web Page. (2018). https://docs.openstack.org/ocata/user-guide/common/cli-set-environment-variables-using-openstack-rc.html Accessed: 2018-03-20.
[10] OpenStack. 2018. Software. Web Page. (2018). https://www.openstack.org/software/ Accessed: 2018-03-08.
[11] Swagger. 2018. What Is OpenAPI? Web Page. (2018). https://swagger.io/docs/specification/about/ Accessed: 2018-02-01.
[12] Ubuntu. 2014. CloudInit. Web Page. (03 2014). https://help.ubuntu.com/community/CloudInit Accessed: 2018-04-20.

[13] Gregor von Laszewski, Geoffrey C. Fox, and Judy Qiu. 2018. *Handbook of Clouds and Big Data* (2 ed.). Indiana University, Bloomington, IN 47408. http://cyberaide.org/papers/vonLaszewski-bigdata.pdf
[14] Wikipedia. 2018. OpenStack. Web Page. (04 2018). https://en.wikipedia.org/wiki/OpenStack

# Leveraging REST for cloud portability

Michael Robinson
Indiana University
Bloomington, IN 47408, USA
micbrobi@iu.edu

Sushant Athaley
Indiana University
Smith Research Center
Bloomington, IN 47408, USA
sathaley@iu.edu

Harshad Pitkar
Indiana University
Bloomington, IN 47408, USA
hpitkar@iu.edu

## ABSTRACT

Our research measures how the portability of an application is impacted by decisions made early in the software lifecycle when native cloud provider APIs are used. We demonstrate how efforts to leverage scalable, reproducable solutions like boto and libcloud are advantageous. We derived reproducable code using Swagger and Python to show how libraries like boto and libcloud can be used to migrate an application from one cloud provider to another with a measurable reduction in human error and with less time to execute. Additionally, we leveraged Swagger to develop RESTful APIs to further improve the gains. We captured the results of the research to share the derived code and concluded how the research can be applied to existing and new development efforts intending to leverage cloud providers.

## KEYWORDS

hid-sp18-518, hid-sp18-402, hid-sp18-517, Volume: 9, Chapter: REST, Status: 100.
libcloud, boto

## 1 INTRODUCTION

Cloud portability is a growing area of research due to the increased profiliferation of cloud providers [4]. Each provider has unique APIs and tools to their cloud environments which can disincentivize portability as it influences a consumer to stay with their existing solution provider. Efforts to standardize cloud portability like TOSCA have made progress yet participation by cloud providers is constrained due to the competitive nature in the space. Each cloud provider is looking to retain their userbase and there is also a desire by each provider to become the de facto standard by being the market leader. To fill the gap, solutions like Apache libcloud and boto have delivered an abstraction solution to developers to design applications that are easy to port.

Developers are already confronted with a lack of transparency on which cloud provider is optimal for the long-term sustainability of their application. Additionally, attempts to abstract away from cloud providers are helpful yet their non-standardization still potentially locks you into the solutions provided by Apache or communities like boto. We will deliver a continuation of that abstraction concept with an accepted standard, REST, to extend libcloud and boto. By leveraging REST, we introduced a standardized implementation that leverages cloud portability libraries to manage the diversity of cloud applications.

## 2 TECHNOLOGY REVIEW

The National Institute of Standards and Technology defines cloud portability as "data that can be moved from one cloud system to another and that applications can be ported and run on different cloud systems at an acceptable cost." [5] The concept of portability can be extended to encompass the full application stack from the web service to the underlying hardware itself. Portability can also simply mean the ability to ensure high availability where you only are looking to protect against one cloud provider being a single point of faiure.

In NIST Special Publication 500-293, the United States government has defined a strategic roadmap that includes ten formal recommendations for all cloud usage. Out of the ten requirements, eight of them reference portability and interoperability. The Standards Acceleration to Jumpstart the Adoption of Cloud Computing (SAJACC) is an initiative under the guidance of NIST 500-293 that is to define "qualitative testing of specifications against interoperability, security, and portability requirements [3]."

To define portability further, we have to differentiate the tiers of cloud service and where portability may be needed. Cloud providers have generally grouped service into the following four types.

- Infrastructure as a Service - IaaS
- Platform as a Service - PaaS
- Software as a Service - SaaS
- Functions as a Service - FaaS

Each grouping has dependencies that can make portability more difficult. For example, AWS Lambda, which is a FaaS solution, is highly specialized and the APIs in use are specific to that vendor. While solutions like libcloud and boto attempt to include all providers, the speed of the market makes it challenging for portability libraries to include the latest and great cloud provider offerings [1]. Another dependency is the complexity of what needs to be ported. IaaS is the closest cloud offering to bare-metal and dependencies for hardware-specific requirements are not a consideration for most portability offerings. As the adoption of containers and functions increases, legacy implementations of cloud solutions that leveraged IaaS will be more difficult to port over.

The work by the Irish Centre for Cloud Computing provided a "qualitative comparative of current open-source IaaS frameworks" which is in contrast to vendor offerings which tend to lack in portability [8]. The study was limited to the five top open-source providers of IaaS and the derived outcome of the comparison was a breakdown over twenty categories that included portability to vendor IaaS. The summary was that each solution is tailored towards a specific need and while portability is possible, there are other challenges that are considered when choosing when and which of many cloud providers you will end up using.

The research by Kostoska, Gusev and Ristov further highlights the challenges with portability, open-source and standards. The researchers were "motivated by several open research questions about

cloud solutions, such as how to wisely choose a cloud host for services and how to change the cloud provider in an easy manner."[7] The paper stipulates that not only is it difficult to choose which cloud provider to use but that the community of cloud providers and what differentiates them continues to grow. The work concludes that no standard exists and their own efforts are only to "offer a possibility for a documented service exchange."

An interesting example of where multiple private sector providers can ensure interoperability is illustrated by Wired journalist Joe Weinman. In his writings, he expresses how air travel is easy for a consumer to determine where to fly out of, how to get through security, and to have confidence their bags will arrive. The history of aviation is one of consolidation and resistance to standards yet the market ultimately did accept some level of standardization. [10] Weinman concludes with "The Internet took decades to go from a vision of packet switching to where it is today. Between the IEEE, industry, and academia, one can hope that the vision of an Intercloud is now getting the attention it deserves."

In summary, multiple efforts by government and academia are encouraging standardization. For cloud providers to capture the market opportunity of Federal, State and local institutions, they will be encouraged to comply with the newly developed standards. The counter is that private sector will continue to encourage specialization and consumer demand continues to show exponential growth. As Weinman states, the Internet is now a blend of decades of collaboration, incentives and mistakes and cloud portability will likely be the same.

## 3 BACKGROUND ON LIBCLOUD

The concept of LibCloud began in 2009 to address the growing challenge of API diversity between the dozens of cloud providers. The effort eventually became a top level Apache project in 2011. LicCloud provides support for the three Cloud providers we tested, Amazon Web Services, Microsoft Azure and Google Cloud and also supports dozens more.

The solution breaks down support between seven primary aspects, which are Compute, Storage, Key Pair Management, Load Balancing, Container, Backups, and DNS. The support for each aspect varies depending on the native support by the provider and the project status. Another important consideration is region support which is typically restrained to the primary region. This may limit the usefulness for projects that leverage cloud resources globally.

Each aspect has a varying degree of functions available that also may not be fully available to leverage. For example, the supported methods for Compute are fully implemented for EC2 and Azure Virtual Machines yet the deploy node method is not yet available for Google Compute Engine. An example that demonstrates the challenge with migrating to a newer service provider is key management. Amazon EC2 fully supports all methods for key management yet Azure and Google Cloud Engine do not support even one.

Another challenge is that each cloud provider does not offer a testing platform for the use of LibCloud. To test the use of the APIs, you will have to leverage production services to determine the solution works. We leveraged free trials to develop the use of libcloud which is suboptimatal for long-term development.

## 4 BACKGROUND ON BOTO3

Boto3 is the underlying technology that powers the command line interface for AWS. It is a software development kit that is freely available for customized use if you wish to have raw access to the AWS cloud APIs. Underneath boto3 is botocore, which is a framework that can be used to extend into other cloud providers. The concept of a Python SDK for managing cloud services provided in AWS began by Mitch Garnatt in 2006 and has grown exponentially since. [9]

Botocore itself is driven by JSON blobs that define the feature and its use. This concept makes it easy to integrate into other coding languages and also makes the platform simple to extend. The native instance of boto groups the available features into resources, collections, clients, paginators and waiters. The features in resources and collections are the core tools that allow you to manipulate and enumerate cloud resources. The other three feature groups are for ease of use for aspects like session management and grouping of data. [2]

Boto3 strengths also lead to some of its limitations. The library grants in-depth ability to manage AWS resources and is a mature offering that stays current with the rapid pace of AWS feature releases. While AWS features are provided into boto3 for AWS customers, the extensions into other cloud providers is more limited as they are community-driven. This has created a situation where boto3 is more likely to be used solely for AWS.

## 5 ARCHITECTURE

The summary of technologies used were implemented as scripts or markups to leverage multiple methods available in libcloud and boto3. The solution was optimized to provide a turnkey implementation to easily manage the deployment and interaction with resources in three of the primary cloud providers. The summation is a deployable Swagger instance that provides a RESTful API to use the underlying python scripts which use libcloud and boto3 to interact, all with little to no knowledge required by the user.

Since this is a webservice, it provides a capability to access it programmatically which is advantageous as multiple VMs can be created within a few minutes as compared to manually creating through AWS console which is time-consuming if we have to do it in mass. It was helpful also when there is a desire to create VMs with the same configuration for multiple users as this webservice can be invoked through any program. Default VM configuration can be used during creation of a VM and avoid defining configuration for each. It also simplifies start, stop and termination of all VMs as a list of VMs can be provided for iteration through that list to perform the desired operation. Since it is configuration driven through automation, human error was minimized. Our service provided flexibility to work with different clouds (currently 3) and scalability to add more cloud implementations. Service will encapsulate various cloud implementations regardless of their operating difference and provides a one-stop solution.

A high-level depiction of the architecture along with requisite functions are shown in Figure 1.

A layered approach was used so that a new implementation could be easily integrated into the solution. Figure 2 depicts the layered approach for this service.

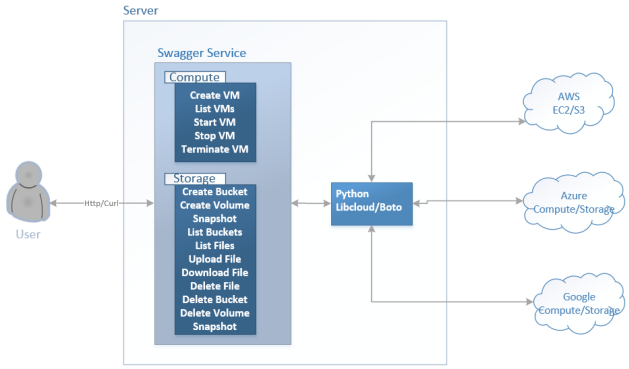


**Figure 1: Project Architecture**

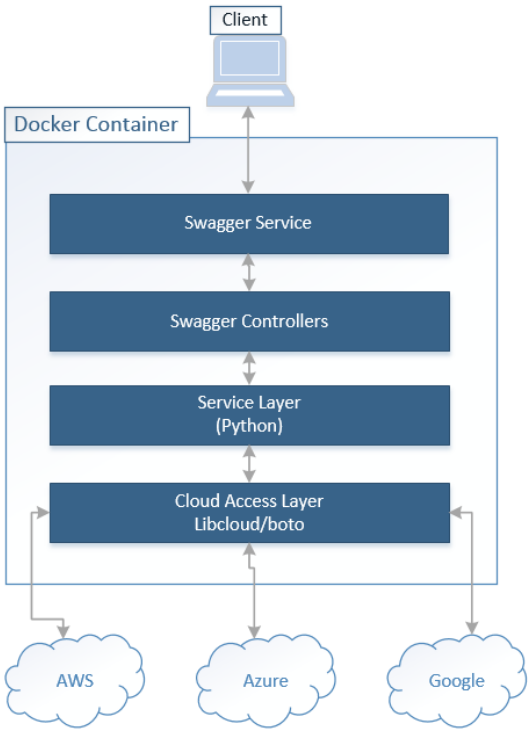


**Figure 2: Layered Architecture**

## 6 OTHER TECHNOLOGIES USED

The other technologies leveraged for implementation are listed below. The solution leverages these technologies yet the paper does not describe their use in detail.

**Python.** Python was the underlying language that was used to bridge the use of libcloud and boto into the Swagger platform. Multiple scripts were written in python that would handle credential handling to the methods that would interact with Compute or Storage resources. Both libcloud and boto3 are available as native libraries for Python and it is also salient that Python was a language of choice. Other languages do offer cloud SDKs yet they were not explored.

**Swagger.** Swagger and Swagger Codegen was used to generate the documentation and design to enable a RESTful API implementation of the python scripts. Swagger was leveraged to implement the REST interfaces to align with course content. There are other frameworks available such as Flask or Django that were not assessed.

**Amazon Web Services.** AWS natively supports boto3 which provided robust options on what could be delivered via REST. Also, as AWS is the industry leader, there is significant open-source development available that provided examples of how to use libcloud and boto3. AWS also was the most prominent when it came to features available and features mapped into libcloud for usage.

**Google Cloud.** GCE has an in-depth integration available with libcloud and Apache provides very detailed examples on how to use libcloud. Google Cloud is missing some APIs compared to AWS and as such, they are not available for portability testing.

**Microsoft Azure.** Azure is not natively supported by libcloud or boto3 yet libcloud has extensive support for most of the available API calls. Microsoft Azure has grown in market share and there is a high likelihood that it would be a target for porting to from an AWS developed solution.

**R.** R was utilized to generate the box plot visualizations and the summary statistics from benchmarking data.

## 7 METHODOLOGY

Python libraries exist now that help you abstract your project from your cloud provider. This may be important if you think you may need to use multiple cloud providers or if you want to ensure you take steps to avoid provider lock-in. There are multiple providers in this space and one of them is Apache Libcloud [1] To put it simply, this library allows you to code simple resource management that is independent of cloud provider specific API calls.

Our research is intended to prove that their is significant value to leveraging frameworks that provide portability. By comparing the native porting of an application to a defined RESTful API, we theorize the porting of a service between cloud providers will be more efficient. The combination of open-source portability python libraries, the speed of Swagger, and the standardization of REST will be a combination that will drastically improve portability.

The Cloud Security Council has broken cloud portability into five measurable aspects of instruction, syntactic, metadata, behavior and policy. We will leverage the instruction aspect, where the correct application instructions are followed when ported, and the syntactic facet where porting between providers is transparent to the end user. This implies our testing is more closely aligned with real-world adoption but assumes aspects like policy are left unattended. [4]

The instruction facet looks to assess whether the intended target that is being ported to is capable of "understanding and executing the instructions contained in the executable artifacts of the application." [4] This is a strong consideration if you are running code that depends on an underlying runtime environment, especially if it

depends on a specific environment such as the difference between Java Runtime 7 or 8. When the porting is poorly executed, it may result in execution failures, poor performance or even worse, the instruction appears to run yet gives inaccurate results that must be found by regression testing.

The syntactic element looks to identify the ease of the solution to alleviate knowledge of the underlying syntax for the cloud provider API or needing to manually convert different unit sizes. This is salient when providers may use custom naming conventions or if the cloud provider expects steps to be executed in a specific fashion. This is also important at higher abstraction layers such as the application data itself, where the "PaaS service may provide instances of databases ready-to-use, in which case the actual databases provided may be sensitive to the data syntax of the customer data." [4] The syntactic element is also where we will highlight issues for cross-platform support in instances where the target provider either does not have API support or libcloud does not support yet.

To experiment, we leveraged the following resources.

- Swagger development to design/build API services for UI
- Python development to leverage libcloud
- Amazon Web Services, Azure, and Google Cloud documentation
- Swagger API documentation

To reflect on the three cloud providers chosen, the results were expected to skew towards the strengths and weaknesses of each provider. AWS is considered the most mature provider with a very deep API offering where solutions like Azure are making progress. Alternatively, Data Motion states "Google developed the Kubernetes standard that AWS and Azure now offer and GCP specializes in high compute offerings like Big Data, analytics and machine learning [6]."

To determine the effectiveness, we have chosen to measure the cycle time on porting a solution without libcloud to two alternatives, leveraging native libcloud only and then our RESTful implementation of libcloud. Additionally, we will measure the lead time metrics on using a RESTful API compared to the command line python libraries to determine if removing the expectation for portability can improve application development speed.

## 7.1 Code Organization

The code related to the project is organized as described in Figure 3

```
code
    - Makefile
    - Dockerfile
    - compute-storage.yml
    - benchmarking.sh
    - etc
        - credentials.yml
        - google key file
    - aws
        - python scripts
  -  azure
        - python scripts
    - google
        - python scripts
```

**Figure 3: Code Structure**

Code details

- *Makefile* - Makefile script contains commands to generate Swagger REST service code, import required packages, copy files on the desired location, start Swagger service, test Swagger service, build docker container, start and stop docker container.
- *Dockerfile* - builds docker container
- *compute-storage.yml* - contains Swagger REST service specification
- *etc* - this directory contains all configuration related files. *credentials.yml* file contains authentication information required by AWS, AZURE, and Google cloud. This directory also contains key file required by Google cloud for the connection.
- *aws* - this directory contains Swagger controllers and python scripts for various operations on Amazon AWS cloud services
- *azure* - this directory contains Swagger controllers and python scripts for various operations on Azure cloud services
- *google* - this directory contains Swagger controllers and python scripts for various operations on Google cloud services
- *benchmarking.sh* - this script contains commands to generate data for benchmarking

## 7.2 Deployment

The Swagger service can be deployed as docker container or standalone service on any server type. The Makefile contains commands to build and run the docker container. Makefile also contains commands to build the Swagger service and start Swagger service as a standalone application. Once the service is deployed, it can be accessed through any browser or curl-like utility.

## 7.3 Usage

There are certain pre-requisites for each cloud to use this REST service.

- *AWS* - AWS account along with access key and a secret key is required to connect AWS cloud.
- *Azure* Azure tenant ID, subscription ID, application id and password is required to connect Azure cloud.
- *Google* Google service account ID, project ID along with key file is required to connect Google cloud.

This information should be kept confidential and need to be configured in the yml file under the etc directory. The key file generated for Google cloud will also need to be kept in this directory. Default VM parameters like image, size, and region can be specified in the configuration and Swagger specification file.

Table1 describes functions provided by the Swagger service.

**Table 1: Service Calls**

| Path | Description |
|---|---|
| /compute/aws/ec2 | Creates AWS VM |
| /compute/aws/ec2/findByRegion | List AWS VMs based on region |
| /compute/aws/ec2/{vmname} | Get AWS VM details by VM Name |
| /compute/aws/ec2/{vmname}/start | Start AWS VM |
| /compute/aws/ec2/{vmname}/stop | Stop AWS VM |
| /compute/aws/ec2/{vmname}/terminate | Terminate AWS VM |
| /storage/aws/s3/bucket | List all AWS S3 buckets |
| /storage/aws/s3/bucket | Post method to create S3 bucket |
| /storage/aws/s3/bucket | Delete method to delete S3 bucket |
| /storage/aws/s3/{bucketName} | List all files inside the bucket |
| /storage/aws/s3/{bucketName}/ deleteFile | Delete file from bucket |
| /storage/aws/s3/{bucketName}/ uploadFile | Upload file to AWS S3 bucket |
| /storage/aws/s3/{bucketName}/ downloadFile | Download file from S3 to the /download directory |
| /compute/azure/createvm | Create VM on Azure cloud |
| /compute/azure/deletevm | Delete or teminate Azure VM |
| /compute/azure/startvm | Start Azure VM |
| /compute/azure/stopvm | Stop Azure VM |
| /compute/azure/listvm | List Azure VMs |
| /storage/azure/createVol | Create Azure volume |
| /storage/azure/createVolSnap | Create Azure volume snapshot |
| /storage/azure/deleteVol | Delete Azure volume |
| /storage/azure/deleteVolSnap | Delete Azure volume snapshot |
| /compute/google/createvm | Create VM on Google cloud |
| /compute/google/deletevm | Delete or teminate Google VM |
| /compute/google/startvm | Start Google VM |
| /compute/google/stopvm | Stop Google VM |
| /compute/google/listvm | List Google VMs |
| /storage/google/createVol | Create Google volume |
| /storage/google/createVolSnap | Create Google volume snapshot |
| /storage/google/deleteVol | Delete Google volume |
| /storage/google/deleteVolSnap | Delete Google volume snapshot |

## 7.4 Leveraging libcloud library with Python

To use the libcloud library in your python environment, you install the apache-libcloud package using the Python package management system, pip. Libcloud currently supports Python versions 2.5, 2.6, 2.7 and Python 3. To use libcloud as a library, leverage Python and import libcloud like any other library. A useful feature to use in Python is the help function which will provide you a simple manual on further use of libcloud.

As an example, we leveraged libcloud to configure and manage EC2 in AWS. We first needed to configure credentials for an active cloud provider account and then configured libcloud provider to use that account. To be able to access AWS S3 from libcloud, we need the access key to be specified in the call. An access key can be setup on AWS console by navigating to My Security credentials, Encryption Keys, and then Access Keys.Next, local variables were defined to store the credentials to be used. Once that was set up, we defined the EC2 driver with our region preference, which was AWS us-east-1, the most full featured region. This is because there are additional tasks that must be taken with SSH keys. We used our browser to review EC2 and finalize the setup.

A Python example for leveraging libcloud to list available containers in an AWS S3 instance is shown below. The key steps are importing the libraries and leveraging the functions that enable connectivity, authentication and authorization. Once that is complete, you can pull or push information using the family of functions that are available in Libcloud for your flavor of cloud provider, in this instance which is AWS.

```
from libcloud.storage.types import Provider
from libcloud.storage.providers import get_driver


cls = get_driver(Provider.S3_US_EAST2)
driver = cls('api key', 'api secret key')

d = driver.list_containers();

print d;
```

**Figure 4: Leveraging Libcloud [1]**

The following table2 is a breakdown of other functions that were leveraged in Libcloud during our development. The list is tailored towards only what we used directly. If a feature is specific towards a certain cloud provider, as in it was not completely abstracted by Libcloud, the description will point out the customized use.

## 7.5 Leveraging boto3 library with Python

In order to leverage boto3 within a python console, you simply can install using pip like we have done with libcloud. Boto3 was developed with Python 3 in mind yet backwards compatibility is available to Python 2.7 and 2.6.5. A salient point to stress is that boto3 is different than boto, like how Python 2 is different than Python 3. The authors of boto stress that boto3 should be used as boto is only supported for older implementations. The features are similar yet boto will eventually be deprecated and developers will have to migrate to boto3. [9]

As with all interactions with cloud providers, the aspect of credentials and privileges must be addressed before successful use of the libraries. Since boto3 powers the use of the AWS CLI, we found the use of the AWS CLI to be the most effective way to configure

credentials. The credential space is shared between boto3 and AWS CLI and there is less error when using the user-friendly AWS CLI. Credentials can be hardcoded as well and examples are provided by Amazon. [2] The last step that must be taken is define the location you wish boto3 to focus on such as AWS regions. Multi-region support is limited and if you do not define an alternate region, it will use us-east-1.

Below is an example of how we leveraged boto3 to start and stop an instance defined in AWS EC2. The python script is passed two parameters, On or Off, and the instance ID you wish to change. The parameters are ultimately passed to boto3 to connect to your EC2 instances and perform the action.

```
\# To start the instance
$ python boto_ec2.py <on> <instance id>

\# To stop the instance
$ python boto_ec2.py <off> <instance id>

\# Add below code to a file named boto_ec2.py

import boto3

action = sys.argv[1].upper()

ec2 = boto3.client('ec2')
```

**Figure 5: Leveraging Boto3 [2]**

Once boto3 is imported, the object ec2 is leveraged for ease of use. The client method is defined with the intended cloud provider environment, in this case it is AWS EC2. The calls in Python using the variable are then vendor agnostic. For the methods for starting and stopping instances, boto3 allows the author to follow a common syntax structure that will map the parameters into the underlying vendor API calls. Last, boto3 will capture the returned output provided by the vendor and will make it available for your use. In this python script, we have captured the response into a variable for later use and to show that it works as defined, the output is sent to the console.

## 8 USE CASE: REST API CALLS

The final outcome of the code development was an ability to leverage an HTTP call to request usage of cloud services. For example, below is a sample of an API call where elements are passed with POST to provide the parameters needed for libcloud or boto to initiate the call. When the call is completed, the API returns a string on what was requests. The parameters are sent in JSON format and are extensible into other platforms. Some feature enhancements for the REST API call would be error handling if libcloud or boto have issues executing the call.

Using a utility like curl, you can push a JSON blob using POST. The RESTful API is organized into folders, for example, where computer resources are under a URI structure prefaced with the intended use, like compute, and then the cloud provider you wish to use, like ec2. The JSON blob contains the parameters that will be in the body of the POST that will be used by Python and follows normal HTTP syntax.

By leveraging an HTTP call allows, we have created an opportunity to extend the solution for automation or to provide a browser-based user experience. Both of those types of solutions would improve the scalability of what can be done compared to cloud provider UI or the provided command-line tools. Both solutions are also less prone to error by incorrect data entry or mistakes by overly broad permissions.

The use of Swagger has also helped define the RESTful API into a well-known standard. The markup file for the solution contains the documentation for use and defines what type of field is expected, from strings to integers. The benefit from Swagger is that a very thorough explanation of how to use the API is provided and the efforts to generate the documentation is very light. Additionally, it is automated and the accuracy is complete compared to a manual effort to capture how to use the RESTful API to call libcloud or boto3.

The API call maps to the following Python script which would build a virtual machine. The method requires multiple parameters that are mapped into the RESTful API calls. The JSON blob would pass the paramenters into the Python script to leverage the `get\_image` and `create\_node` features in libcloud.

```
def
createAzureVM(vmname,vmsize,image_name,resource_group,storage_acco
blob_container):

\#print region

urn='Canonical:UbuntuServer:16.04-LTS:latest'

image=driver.get_image(urn, location=None)

print image
```

**Figure 6: Python script example**

The error handling by the script provides the user feedback on what happened and how to resolve. A future enhancement would be to map the method return back to the RESTful API and the error could be presented to the user in the user interface they are using to leverage the API. When the method works correctly, the parameters passed into the method could be leveraged to confirm the correct virtual machine was created and also used to enhance the message sent back to the user.

## 9 BENCHMARKING

A benchmarking shell script is provided that tests the timing performance between the cloud providers when leveraging libcloud. The script would initiate the measurement when the API call is made and then consider the request complete when libcloud interprets the cloud provider response and returns it. For the benchmarking, a series of ten tests were performed per method per the three cloud

providers. The output for the benchmarks were captured for later visualizations to interpret how well libcloud works for different types of requests and for different providers. All three providers successfully generated virtual machines, as expected, with the parameters provided. Additionally, libcloud abstracted the syntax successfully for all three providers. Table3 shows the benchmarking test results.

## 9.1 Virtual Machine - Start

It was also observed that all three providers were able to execute successfully across all 10 tests. AWS was benchmarked as the leader of the three with an average execution time of 16.4 seconds. AWS was surprisingly able to still be the leader on average with a significant outlier in the dataset where one test took 148 seconds to complete. The other two providers, Google and Azure, started up in 23.4s and 26s respectively. Out of the five benchmarks, starting a virtual machine when using libcloud was one of two metrics that had two providers average within a reasonable range of each other. Figure 7 shows the details.

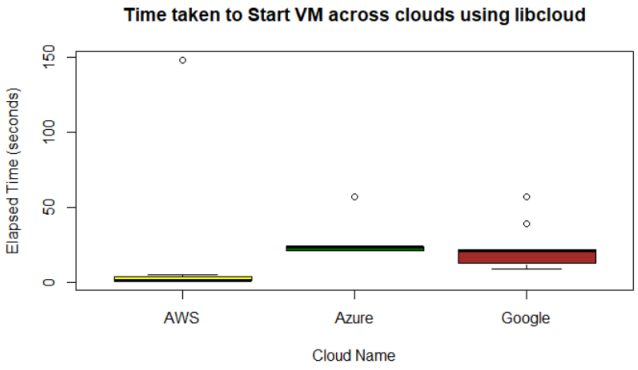


**Figure 7: VM starting benchmark**

## 9.2 Virtual Machine - Stop

For the stopping of virtual machines, AWS again was the clear leader in performance. Libcloud, on average, reported a successful stop of a virtual machine in 21.9s. For Azure, the average result was 60.9s which was marginally better than the average for Google of 62.4s. Similar to the starting of virtual machines, there was a clear leader and the other two providers compared evenly with each other. The same can bee seen in figure 8.

## 9.3 Virtual Machine - Create

As you can see from figure 9 that creation of virtual machines is where a new leader is presented with Google. This was expected as marketing for Google implies they are very efficient at horizontal scaling and rapid deployment of net new virtual machines and containers. Google was found to be able to generate a new virtual machine in 12.4s, which is half the time it takes compared to 27.5s from Azure. AWS trailed both with an average creation time of 37.6 seconds.

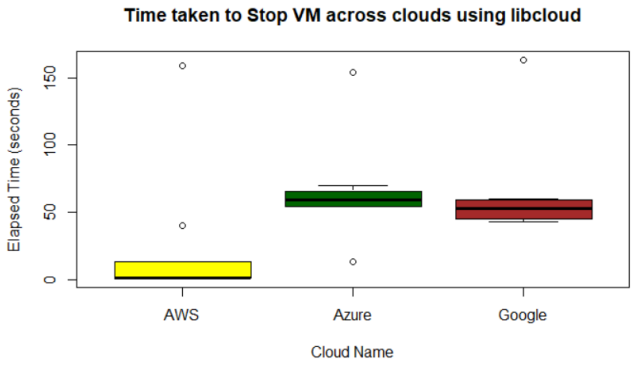


**Figure 8: VM stopping benchmark**

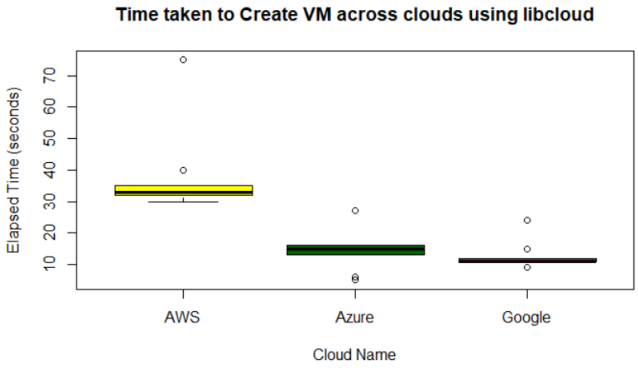


**Figure 9: VM creation benchmark**

## 9.4 Storage - Create

For the creation of storage, AWS was significantly faster than the other two providers. AWS was able to create a 1 GB volume and return success within 1.8s. Azure was slower at 15.4s yet Google was markedly much slower with an average speed of 126.7. Where Google was the fastest solution to create a new virtual machine, the ability to rapidly deploy additional storage was found to be a weakness of the Google cloud provider. We do not perceive it to be an issue with how Libcloud requests storage from Google and it is attributed to Google directly. The visualization of the same can be seen in Figure 10.

## 9.5 Virtual Machine - Delete

For the rapid decommissioning or clean up of virtual machines, AWS was also found to be remarkably fast. Libcloud had success with all three vendors with AWS on average reporting a virtual machine had been deleted in 3.2s. Google came in next with an average of 87s and Azure was the slowest at 199.3s. Figure 11 shows the details of the benchmark.

# 10 RESULTS

We had success leveraging libcloud to generate and control the execution of cloud resources with Amazon Web Services and Microsoft Azure. We were able to abstract away from native API

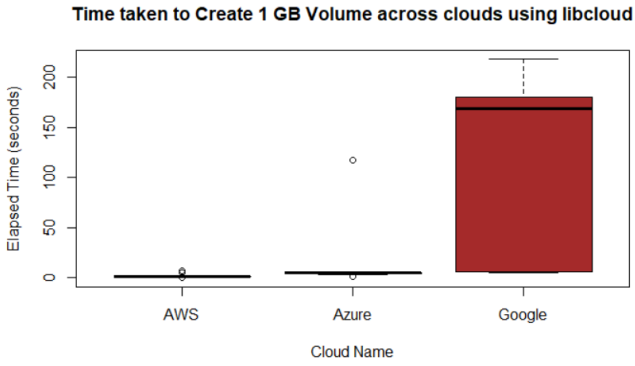


**Figure 10: Volume creation benchmark**

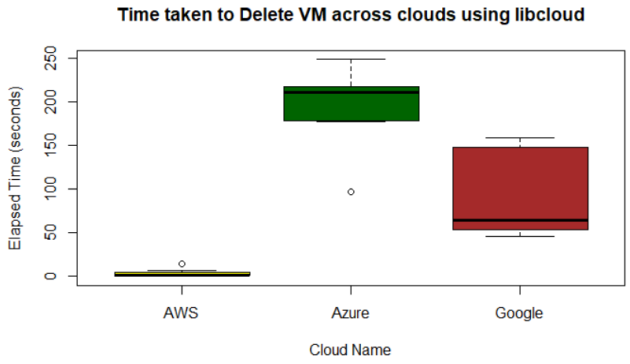


**Figure 11: VM deletion benchmark**

calls for a cloud provider. Additionally, we were able to map the Python scripts leveraging libcloud into RESTful APIs that were well-documented leveraging Swagger. The solutions combined to provide a solution that hides the intricacies of knowing how to use each vendor's API and the verbiage they use to explain different types of cloud resources.

While libcloud works well with all the three vendors Amazon, Microsoft and Google, there are certain functions that are available only for AWS such as uploading a file or downloading a file. Libcloud integration with Google is straightforward and the API functions operate over string parameters whereas for Azure a Node object has to be passed and returned in most cases. Implementation with Azure especially required the use of a hidden or internal function usage in one of the cases. Due to this constraint, time to integrate Azure was significantly higher as compared to AWS or Google.

The use of boto3 became challenging in relation to Google Cloud. In concurrency with what the technology review stressed, the use of proprietary utilities like gsutil limited what we can develop. Boto3 development was successful with AWS resources yet at boto3 is the underlying technology for AWS utilities, the value-add was limited to converting boto3 calls for AWS into a RESTful API.

## 11 CONCLUSION

Libraries available for abstracting the cloud provider from your code development were found to be effective. The solutions are syntactically and instructionally accurate. The solutions also had negligible impact on performance and were easy to extend for custom usage, such as benchmarking. The cloud providers were found to be distinct, even with an abstraction layer such as libcloud and boto3. There were challenges with authorization and object handling between providers that had to be handled independently.

The project exhibited that it is possible to ensure your application could be ported yet with significant work. Further research would be necessary to see if a manual port of native use of cloud provider APIs is more intensive than leveraging something like libcloud. Our perception is that efforts to ensure portability are incomplete and if the three primary providers could incur cost beyond API calls, there is still significant work to be done to align with the NIST guidance of ensuring portability to get beyond where "consumers must seek to understand cloud services through a customized and product specific view presented by each service provider [3]."

## 12 ARTIFACTS

Code is checked-in in Github at location *https://github.com/cloudmesh-community/hid-sp18-518/tree/master/project- code*

## ACKNOWLEDGMENTS

The authors would like to thank Dr. Gregor von Laszewski for his support and suggestions to write this paper.

## REFERENCES

[1] Apache. 2018. *Apache LibCloud.* https://libcloud.readthedocs.io/en/latest/index.html
[2] AWS. 2018. *Boto3 and botocore.* http://boto3.readthedocs.io/en/latest/index.html
[3] L. Badger, D. Bernstein, M. Iorga R. Bohn, F. de Vaulx andM. Hogan, J. Messina J. Mao, K. Mills, E. Simmon, A. Sokol, J. Tong, F. Whiteside, and D. Leaf. 2014. *High-Priority Requirements to Further USG Agency Cloud Computing Adoption.* https://dx.doi.org/10.6028/NIST.SP.500-293
[4] Cloud Security Council. 2017. *Interoperability and Portability for Cloud Computing.* http://www.cloud-council.org/deliverables/CSCC-Interoperability-and-Portability-for-Cloud-Computing-A-Guide.pdf
[5] NIST Cloud Computing Standards Roadmap Working Group. 2013. *NIST Cloud Computing Standards Roadmap.* https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.500-291r2.pdf
[6] C. Harvey and A. Patrizio. 2017. *AWS vs. Azure vs. Google: Cloud Comparison.* https://www.datamation.com/cloud-computing/aws-vs.-azure-vs.-google-cloud-comparison.html
[7] M. Kostoska, M. Gusev, and S. Ristov. 2014. *A New Cloud Services Portability Platform.* https://doi.org/10.1016/j.proeng.2014.03.118
[8] T. Lynn, G. Hunt, D. Corcoran, J. Morrison, and P. Healy. 2015. *A Comparative Study of Current Open-Source Infrastructure as a Service Frameworks.* https://www.researchgate.net/publication/278665353_A_Comparative_Study_of_Current_Open-Source_Infrastructure_as_a_Service_Frameworks
[9] Amazon Web Services. 2015. *AWS SDK boto3 and botocore.* https://aws.amazon.com/blogs/aws/now-available-aws-sdk-for-python-3-boto3/
[10] J. Weinman. 2013. *Will Multiple Clouds Evolve Into the Intercloud?* https://www.wired.com/insights/2013/10/will-multiple-clouds-evolve-into-the-intercloud/

## 13 WORK CONTRIBUTIONS

- AWS Compute and Storage coding by Sushant Athaley
- Azure Compute and Storage Coding by Harshad Pitkar
- Google libcloud research by Michael Robinson
- Google Compute coding by Harshad Pitkar
- Google Storage Coding by Sushant Athaley
- Dockerfile by Sushant Athaley

- Makefile by Sushant Athaley
- Architecture explanation by Michael Robinson
- Architecture diagrams by Sushant Athaley
- Benchmarking script by Harshad Pitkar
- Bencmarking plots by Harshad Pitkar
- Benchmarking explanation by Michael Robinson
- Code packaging and management by Sushant Athaley
- Code explanation by Michael Robinson
- Swagger Spec by Sushant Athaley and Harshad Pitkar
- Service Testing by Sushant Athaley and Harshad Pitkar
- Code Merging by Sushant Athaley and Harshad Pitkar
- Project Paper by Michael Robinson and Sushant Athaley and Harshad Pitkar

**Table 2: LibCloud features leveraged**

| Command | Description |
|---|---|
| `get_driver()` | How to choose the primary cloud provider that will be used by following functions. We leveraged Amazon Web Services, Microsoft Azure and Google Cloud as targets for this method. |
| `dns()` | Used to configure DNS for compute resources. This is an example of a late-stage call to further define how a virtual machine should be configured. |
| `get_driver_lb()` | Used to configure load balancing for compute resources. This field is a great example of the power of cloud computing and RESTful API as redundant, highly available virtual machines can be built quickly and if you choose, by default. |
| `list_nodes()` | Used to enumerate the nodes already created within the cloud provider. This call can be automated to routinely audit usage of virtual machines to identify trends for usage. |
| `list_sizes()` | Used to enumerate the size of the VM and supporting components. This method is commonly used to reduce costs and identify VMs that may have grown beyond a comfortable level. |
| `list_images()` | Used to enumerate the avaiable, deployable images the cloud provider makes available. This parameter is typically used before a call to generate a virtual machine to ensure the image is still provided by the cloud provider. |
| `get_image()` | Used to pull the unique identifier for the human-readable name for the chosen image |
| `ex_get_node()` | Used to pull the unique identifier for one or many VMs. |
| `create_node()` | Used in conjuection with image listing to deploy a new VM. |
| `ex_start_node()` | Used to tell a previously built VM to boot. |
| `ex_stop_node()` | Used to direct a VM to begin shutdown gracefully. |
| `destroy_node()` | Used to delete a VM and will return success. This method is one where human error can lead to significant damages and leveraging REST, the solution can confirm deletion with a user or restrict deletion. It also can simply ensure virtual machines actually are deleted after a certain threshold as it is common for human-generated virtual machines to be left running that leads to unwarranted costs. |
| `ex_get_volume()` | Used to enumerate the unique identifier for cloud storage typically used by a VM. |
| `create_volume()` | Used to add addtional storage to a VM. |
| `destroy_volume()` | Used to delete storage or snapshot to recover storage space. |

**Table 3: Benchmarking Test Results**

| Cloud Name | Operation Type | Mean(s) | Median(s) | Standard Deviation(s) |
|---|---|---|---|---|
| AWS | Create VM | 37.6 | 33 | 13.41806411 |
| AWS | Create Vol/Bucket | 1.8 | 1 | 2.299758441 |
| AWS | Delete S3 File | 0.4 | 0 | 0.516397779 |
| AWS | Delete VM | 3.2 | 1 | 4.391911758 |
| AWS | Delete Vol/Bucket | 0.7 | 1 | 0.483045892 |
| AWS | Download File from S3 | 0.5 | 0 | 0.527046277 |
| AWS | Get VM Details | 7.8 | 1 | 17.25495098 |
| AWS | Get Vol/Bucket Details | 1.8 | 0 | 3.735713527 |
| AWS | List VM | 23.1 | 1 | 56.68715316 |
| AWS | List Vol/Bucket Details | 0.4 | 0 | 0.516397779 |
| AWS | Start VM | 16.4 | 1 | 46.26301619 |
| AWS | Stop VM | 21.9 | 1 | 49.73362377 |
| AWS | Upload File to S3 | 1.1 | 1 | 1.791957341 |
| Azure | Create VM | 27.5 | 15 | 42.41658481 |
| Azure | Create Vol/Bucket | 15.4 | 5 | 35.7217643 |
| Azure | Create Volume Snapshot | 20.7 | 11 | 38.01768594 |
| Azure | Delete VM | 199.3 | 210 | 42.57294394 |
| Azure | Delete Vol/Bucket | 4.3 | 6 | 2.983286778 |
| Azure | Delete Volume Snapshot | 2.8 | 3 | 1.87379591 |
| Azure | List VM | 2.5 | 2 | 2.677063067 |
| Azure | Start VM | 26 | 23 | 10.98483804 |
| Azure | Stop VM | 60.9 | 59 | 38.66221239 |
| Google | Create VM | 12.4 | 11 | 4.402019738 |
| Google | Create Vol/Bucket | 126.7 | 168 | 85.77496397 |
| Google | Create Volume Snapshot | 26.1 | 13 | 35.63222636 |
| Google | Delete VM | 87 | 64 | 46.64761516 |
| Google | Delete Vol/Bucket | 6.6 | 6 | 0.699205899 |
| Google | Delete Volume Snapshot | 23.8 | 4 | 42.56185042 |
| Google | List VM | 9.6 | 5 | 10.5746027 |
| Google | Start VM | 23.4 | 21 | 14.42374585 |
| Google | Stop VM | 62.4 | 53 | 35.89552742 |

# Interacting With Data Services on AWS

Scott Steinbruegge
Indiana University
Bloomington, IN 47408, USA
srsteinb@iu.edu

## ABSTRACT

Amazon Web Services provides a wide variety of data related services that allow a user to immediately start working on utilizing and interpreting their data instead of trying to obtain or provision their own hardware beforehand. By using a data set that contains Medicare hospital patient survey information, we showcase how REST APIs can be used to directly interact with multiple AWS services that are used specifically for data related use cases. We provide some examples of how to interact with these services through the AWS Management Console, but the AWS Python SDK called Boto is the main method used in our project in order to be able to work programmatically with the Amazon APIs. We start with presenting how to get raw data onto the AWS platform directly from the web and landing that data in S3 for use by other AWS services such as Data Pipeline, RDS, Redshift, DynamoDB and Athena. Along the way we provide explanations and hands on coding examples for each service in order to gain insight into how to effectively interact with AWS while performing a variety of data related tasks.

## KEYWORDS

hid-sp18-521, Volume: 9, Chapter: Amazon, Status: 100.
Amazon, AWS, Swagger, AWS Data Services

## 1 USER INTERACTIONS WITH AWS

Between the AWS Management Console, SDKs (software development kits) and AWS CLI (command line interface), AWS provides multiple ways to interact with their services. While some of the work we perform is done through the Management Console, the majority of it is performed by the Python SDK for AWS named Boto. Using one of the AWS SDKs allows developers to be able to write their own software that interacts directly with AWS services through their own set of APIs. There are two levels of APIs that AWS provides, resource APIs and client APIs. The resource APIs provide an object-oriented interface to the services which expose multiple attributes and methods per service at a higher level than the lower level client API calls [4]. Client APIs, also called low-level APIs, provide methods that map one-to-one with the service APIs. No level of abstraction is provided by the client APIs, the developer needs to explicitly provide all information about the targeted resource for every client API call [3]. When available, the resource APIs are preferred since they abstract the majority of the backend details away from the developer and they can focus on interacting directly with the service using the necessary parameters. Boto also provides the ability to use waiters, which allow for polling the status of specific AWS resources within your code. An example of when a waiter could be used is when you run code trying to provision a new resource and moving onto the next step cannot be done until that new resource has been setup successfully. Using the waiter would allow your code to continuously poll the status of that dependent service [2].

## 2 UTILIZING DATA SERVICES ON AWS

To provide examples on how to work with data on several AWS platforms, we utilized a sample data set from the Medicare data website that contains "a list of hospital ratings for the Hospital Consumer Assessment of Healthcare Providers and Systems (HCAHPS). HCAHPS is a national, standardized survey of hospital patients about their experiences during a recent inpatient hospital stay [20]." This data set was provided in both CSV and JSON format which allowed us to choose which data format was best suited for the specific AWS service we worked on. Preliminary data analysis on the raw file showed 23 columns and approximately 264,000 rows of data. For each AWS service we chose to work with, we attempted to perform a different type of analysis on this data in order to show how a user would be able to gain insights from data residing within different AWS locations.

The AWS Management Console was used during the initial setup of our project environment in order to provision certain services. We felt it would be best to keep our costs low for this project by not exposing the ability to provision high cost AWS services through a REST API. Interacting with each of these services though was still done through Boto in order to keep the majority of the tasks done in programmatic fashion. At a high level, our project uses multiple Swagger REST APIs which then call Python functions that use Boto and additional libraries to interact directly with AWS service specific APIs to perform different types of data interactions. In order to be able to programmatically interact with AWS, an IAM user needed to be created and permissions had to be assigned for all of the AWS services it would be interacting with.

An IAM user is a login created on the AWS platform that can then be used to access multiple AWS services based on the permissions assigned to it [16]. This user creation was accomplished through the AWS Management Console so that access keys could then be easily obtained upon new user creation which then allowed this user to have programmatic access to AWS services [15]. In order to create our IAM user, we logged into the console and selected IAM from the services menu. On the left side of the next screen we then selected the menu option users which took us to the user information page where we selected the add user button. We then chose the user name i524user and selected only the check box that allows the user to have programmatic access through an access key ID and secret access key and then selected the permissions button to proceed. On the permissions page we then chose to attach existing policies to this new user in order to grant it permissions to the AWS services it would be using. During initial user creation only one existing policy can be attached, so we selected the AmazonS3FullAccess policy which grants full access to all S3 buckets for the entire AWS

account the IAM user exists under. While permissions could have been granted more granular, for this project, this policy would do what we needed. After the existing policy was selected, we clicked the review button which took us to a page showing an overview of the user options. We were satisfied with our selections and clicked create user. The next page showed the access key ID and secret access key that we would need to save in a secure location in order to use within our Python code later on. The secret access key information is only presented once after the user has been created. If it is lost it cannot be retrieved and a new set of keys would need to be generated for this user [15]. We were then able to go back to the user menu and select the name of the new user we created to bring up the permissions page. From there we selected the add permissions button and attached existing policies like we did during the initial user creation for the remaining AWS services we would be working with later on: Data Pipeline, RDS, Redshift, DynamoDB and Athena. Now that we had our user created and the proper permissions assigned to it, we were then able to get started on setting up and interacting with each of the AWS data services. The final list of policies assigned to our user are displayed in Figure 1.

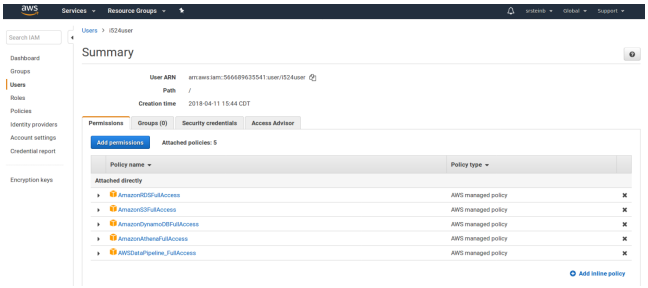


**Figure 1: IAM user permissions**

Programmatically interacting with services on AWS requires the code to utilize both the access key ID and secret access key generated upon IAM user creation. Along with these keys, several other data services require that a separate user login be created directly within the service itself, such as Amazon RDS and Redshift. In order to be able to store and reuse the credentials needed for this project, we stored these values in a YAML configuration file. Using the YAML Python library we could then import the YAML configuration file into our Python code, store it in a list variable and then assign user names and passwords to variables for each service needing credentials. The majority of connections to the AWS services utilized need only the IAM access keys but connecting to the MySQL instance on RDS and the Redshift database both required separate user names and passwords. We chose to store the access key ID in the k1 variable and the secret access key in the k2 variable to better mask what they really are since it is best to keep these keys secure as possible [15]. The Python code used to pull connection information from the YAML file is shown in Figure 2.

## 2.1 S3

Once our IAM user was created, we were ready to start ingesting our raw data into AWS. As a starting point, we chose to land our

```
with open(os.path.expanduser(
~/.cloudmesh/configuration-awsdataservices.yml"), 'r') as ymlfile:

config = yaml.load(ymlfile)

mysql˙user = config['cloudmesh']
['aws-data-services']['mysql']['user']['name']

mysql˙password = config['cloudmesh']
['aws-data-services']['mysql']['user']['password']

redshift˙user = config['cloudmesh']
['aws-data-services']['redshift']['user']['name']

redshift˙password = config['cloudmesh']
['aws-data-services']['redshift']['user']['password']

k1 = config['cloudmesh']['aws-data-services']['misc']['k1']

k2 = config['cloudmesh']['aws-data-services']['misc']['k2']
```

**Figure 2: Python code to import YAML configuration settings**

raw CSV and JSON files in S3. "Amazon S3 stores data as objects within resources called buckets. You can store as many objects as you want within a bucket, and write, read, and delete objects in your bucket [1]." We created our bucket through the AWS Management Console by browsing the services menu and selecting S3 from the services list. From there we selected the create bucket button, entered a bucket name of hid-sp18-521 and left all of the other options to what they were defaulted to since our IAM user has full access to S3 for this project and then created the bucket.

With our S3 bucket in place, we were then ready to start writing our Python code to interact with the AWS services. The first function we implemented was to pull the raw Medicare data files in both CSV and JSON format directly from the web into our S3 bucket. This was accomplished by using the Python library smart open. This library could be used to stream large data files to and from a variety of data sources including S3 and HTTP [25]. To connect to the S3 bucket we had to specify the destination file and bucket name as well as the access key ID and secret access key of our IAM user. A for loop is then used to import all of the rows in the raw file directly from the HTTP location where the file is stored and writes it to the S3 destination file provided. If the copy was performed successfully, the response returned is the total size of the file that was written into S3. Code to perform this task for both the CSV and JSON import into S3 were exactly the same except for the file extension. Both of these functions could be rerun as needed and the S3 file would just get overwritten if it already existed. The Python code used to import the raw data files into S3 is shown in Figure 3.

Once the raw data was in place in S3, we wanted to verify that the files we copied actually existed in our S3 bucket. To do this, we connected to the S3 resource API by assigning a variable to the S3 resource object and provided the parameters for AWS service name, region and access keys. The S3 resource API contained the class named bucket which was used to specify which bucket name we were interested in using [8]. We were then able to setup a for loop to use the bucket class method called object to get the names of all files located within our S3 bucket and append them to a list variable.

```
with smart_open.smart_open('s3://' + k1 + ':' + k2
+'@hid-sp18-521/PatientSurveyData.csv', 'wb') as fout:

  for line in smart_open.smart_open('https://data.medicare.gov/resource/
  rmgi-5fhi.csv'):

    response = fout.write(line + '"n')

return response
```

**Figure 3: Python code to import raw data files into S3**

This list variable would then be returned to our API to show all of the names of the files located within our project specific bucket. The Python code used to list all files located in the S3 bucket is shown in Figure 4.

```
file_names = []

s3 = boto3.resource('s3', region_name='us-east-1'
, aws_access_key_id=k1, aws_secret_access_key=k2)

bucket = s3.Bucket('hid-sp18-521')

for object in bucket.objects.all():
  file_names.append(object.key)

return file_names
```

**Figure 4: Python code to show all files located in S3 bucket**

## 2.2 RDS

Now that our raw data files were landed on the AWS platform within S3, we then had more options to be able to extract, transform and load this data to various other AWS data services. Our next task was to import our CSV data file into an Amazon RDS database. Before importing the S3 data an RDS instance needed to be provisioned. RDS is a "managed relational database service which offers six database engine choices and allows users to setup, operate and scale relational databases within the AWS cloud [9]." The six database platforms RDS works with are Amazon Aurora, MySQL, MariaDB, Oracle, SQL Server, and PostgreSQL. For the purposes of this project, we chose to implement a MySQL instance since it fell within our AWS account free usage tier. When we refer to the term instance in regards to working with RDS, it can be thought of as a database server, which would then contain multiple databases within that instance [9]. Since exposing access to create RDS instances was not something we wanted to do in order to make sure our costs for the project remained low, we decided to perform a one-time setup an RDS MySQL instance through the AWS Management Console.

To access the RDS console, search to the RDS service in the AWS services list which brings up the RDS dashboard. The dash displays information about the existing RDS resources that are currently allocated. Since we have not created any RDS instances yet, we selected the instances menu on the left side of the dashboard which took use to the page where we could then select the launch DB instance button to get started. The first choice to make was which engine to use. At the bottom of this screen there was a checkbox we selected that only showed the engine options available for free tier usage. We chose MySQL since it is a very popular open source database engine and clicked the next button. The following page asked us to specify our instance specifications and settings. Due to free tier restrictions, we did not change any of the default instance specifications and were just able to provide a name for our instance, iu-sp18, as well as a master user name and password. Moving onto the next page, we chose to leave all of the defaults in place for the network and security options and made sure that the option was selected to setup the instance to be publicly accessible. Under database options we could then select a default database name to be created on the instance, which was named I524, and then left the remaining options on this page in place as their defaults since they served our needs to the project use case. We were then able to select the launch DB instance button at the bottom of the screen and AWS started the provisioning of the MySQL instance which took about 5 to 10 minutes total to come online and ready for connections [9].

Once the instance was running, it was time to log into the instance using the master user and password we setup upon creation in order to create an additional user login and the table we would use to import our data into. Interacting with a database residing on RDS is accomplished the same way a user would connect to an on-premise database server. There is a free GUI based application provided by MySQL named MySQL Workbench that we used to perform our first set of interactions with the instance. In order to connect to the instance, we needed the name of the endpoint and port number, which could be found under the instances menu on the RDS dashboard page by clicking the name of the instance in the list. About half way down the instance details page in the connect section was the endpoint and port number, which are always needed when setting up any connections to this instance, either through a GUI application or programmatically. Once we had all of the connection information needed, we were able to successfully login to the RDS instance through MySQL Workbench. From here, we executed standard SQL statements against the I524 database to create a new database user, named IUuser, which we would use for programmatically connecting through our APIs and created the empty table schema named PatientSurveyData that would contain the CSV data imported from S3 [9]. The location of the RDS endpoint and port number within the AWS Management Console is shown in Figure 5.


**Figure 5: RDS server information**

## 2.3 Data Pipeline

In order to get the data into our RDS instance from S3, we utilized AWS Data Pipeline, which "is a web service that you can use to automate the movement and transformation of data [19]." In broader IT terminology, it could be considered an AWS specific extract, transform, load (ETL) tool. It consists of multiple pieces of data driven workflows that are dependent upon successful competition of one another. The user can specify parameters to interact with

transformations run against various sets of data and related services and Data Pipeline will handle the application of this logic. Data Pipeline consists of three main components, the first being a pipeline definition, which specifies the logic to applied during data transformations and management. Next is the pipeline itself, which handles the scheduling and execution of the defined tasks by creating an EC2 instance to perform the work specified by the pipeline definition. Lastly, the task runner handling the polling between tasks within the pipeline and performs those tasks as needed. Task runner is also installed and set to run automatically on all resources used by your pipeline definition [19].

The AWS Management Console provides a web-based interface that allowed us to create and interact with pipelines. As we were novices with the service, this was the best place to start working in order to gain an understanding on how to properly setup and execute a pipeline. To start creating our pipeline to import data from our S3 bucket in the MySQL RDS database, we had to navigate to the AWS Data Pipeline service listed under the services menu on the main dashboard of the AWS Management Console. From there, since we had yet to create any pipelines, it presented an intro screen and we selected the button get started now to proceed. The next page is where we specified the pipelines name and source. Under the source options, there were options to build using a template, import a definition or build using Architect. Using the template option for our use case made the most sense, especially since it provided a specific load S3 data into RDS MySQL table template to build our pipeline from. Once the template was selected, we were then prompted to provide information related to the source S3 file we wanted to utilize as well the connection information and location of the MySQL RDS destination we wanted to import into. For S3, only the file path was needed, but for RDS we needed to supply the RDS instance name, user name and password to connect with, the standard SQL based insert statement we'd use to insert data in our MySQL table we created earlier named PatientSurveyData and an optional statement which we utilized to clear out all of the data in the existing MySQL table before we imported the data from the S3 file [14].

After specifying all of these pipeline parameters, we then scrolled down to the scheduling section where we chose to run our pipeline on activation only, meaning we didn't want it to run on any set schedule, only whenever it was manually started. The rest of the options below this we left as default and proceeded to the bottom of the page and clicked the activate button to create and start our pipeline. The pipeline creation did not produce any errors and we were taken to a screen where we could monitor the execution details of our pipeline. By using the refresh button on the right side of the screen we were able to track the status of each step within the pipeline and determine where the process was currently at. Our pipeline consistent of two components, the first provisioned the EC2 instance which performed the execution of the SQL code against our RDS database to delete data from the existing table and create it if it does not already exist. Upon successful completion of that step, the pipeline then starts a component to load the data from S3 into the RDS database which is executed from the same EC2 instance that was created in step one. Once both components completed with a status of successful, we were ready to query the MySQL table and interact with the Medicare data within a relational database in the AWS cloud [14]. Figure 6 shows the layout of how the data pipeline looks within Architect as well as the parameters we specified upon creation of the pipeline.

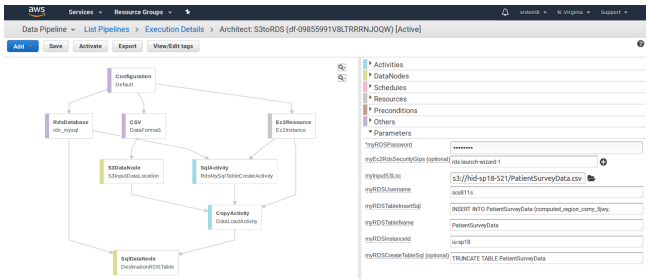


**Figure 6: Data Pipeline layout and parameters**

With pipeline in place and tested to work successfully, we wanted to implement functionality within our code that would allow a user to both activate and check the execution status of our data pipeline on demand. Boto would allow us to accomplish these tasks through our Python code by utilizing the Data Pipeline API. We assigned a variable to the client API connection for data pipeline by providing the resource name, region, access key ID and secret access key in the same manner we did for the S3 code. We could then use the activate pipeline function and pass in the pipeline ID, which can be retrieved from the list pipelines screen on the Data Pipeline page in AWS Management Console. This code then allowed us to activate our pipeline through an API call whenever we wanted. A similar approach was taken when pulling the status of an existing data pipeline. But for this one we needed to use the describe pipelines method and return the string value of a key named @pipelineState from the list of pipeline descriptions for this pipeline in order to present the current runtime status. So now we were able to monitor the status of the pipeline job we were just able to start within another API [6]. The Python code used to both start the pipeline and check the status of the pipeline is shown in Figure 7.

```
pipeline = boto3.client('datapipeline', region˙name='us-east-1',
aws˙access˙key˙id=k1, aws˙secret˙access˙key=k2)

response = pipeline.activate˙pipeline(pipelineId='df-09855991V8LTRRRNJOQW')

return response
---
pipeline = boto3.client('datapipeline', region˙name='us-east-1',
aws˙access˙key˙id=k1, aws˙secret˙access˙key=k2)

pipeline˙status =
      pipeline.describe˙pipelines(pipelineIds=['df-09855991V8LTRRRNJOQW'])

fields = pipeline˙status['pipelineDescriptionList'][0]['fields']

for field in fields:
  if field['key'] == '@pipelineState':
    return field['stringValue']
```

**Figure 7: Python code to start and return status of data pipeline**

Once the pipeline execution was in place within our code, we then felt comfortable enough to develop an API to run a query

against the data we imported in our MySQL RDS database. When programmatically interacting with an RDS database, an Amazon specific API is not necessary. We could use any library that we'd normally use to connect to a MySQL database whether it was cloud based or not. PyMySQL was the library we chose to utilize for this project. After importing the library into our code, we were then able to define a function that first assigned a variable to a MySQL connection object by providing the host name, which was the end-point name listed within RDS, a user name and password, which was stored in the variables imported from the YAML configuration files, a database name, charset and cursor class, which we specified to return any results as a dictionary data type. We were then able to assign this connection object to a cursor object which would allow us to further interact with the MySQL database. To query a specific subset of data we were interested in, we developed a SQL select query to return all of the rows from the table where the star rating for the hospitals was listed was five. We allowed the function to accept a parameter of starRating and used in within the SQL statements where clause in order to filter the data we were interested in returning. The SQL statement was then run by calling the execute method using the SQL query variable and starRating parameter as arguments. The fetchall method was then invoked to collect all of the rows returned in the results and return them to our API for presentation to the user [22]. The Python code used to query the MySQL RDS database is shown in Figure 8.

```
connection = pymysql.connect(
host='iu-sp18.cgnrvgmckfic.us-east-1.rds.amazonaws.com',
user=mysql˙user,
password=mysql˙password,
db='I524',
charset='utf8mb4',
cursorclass=pymysql.cursors.DictCursor)

cursor = connection.cursor()
sql = SELECT * FROM PatientSurveyData WHERE patient˙survey˙star˙rating = %s"

cursor.execute(sql, (starRating))
result = cursor.fetchall()
return result
```

**Figure 8: Python code to query the MySQL RDS database**

## 2.4 DynamoDB

Since we had access to our data set in JSON format, we wanted to explore the idea of being able to ingest this raw data into a NoSQL database for additional analysis. DynamoDB is a fully managed database service that supports document and key-value storage models. There is no need to create or manage individual databases, all tables created and utilized will be placed in whatever region is specified upon table creation. Like RDS, the database software and hardware provisioning are handled by AWS, allowing to user to focus solely on managing their data. DynamoDB runs only on SSD drives in order to provide low-latency response times for all database operations on the platform. The data model within DynamoDB consists of tables, items and attributes. Tables are a collection of items and can store an infinite number of items. In relational database terms, an item can be thought of a row of data, which consists of a primary key and any number of additional attributes, there is no limit. An attribute is a specific element of data within an item that consists of an attribute name and value. As the platform is schema less, each item does not need to have the same number or name of attributes. Every table created must have a primary key assigned which can consist of either a single attribute or multiple attributes and the primary key must exist for every item. The only limitation on an item is that it must be under 400 KB in size. As there is nothing to setup ahead of time other than allowing our IAM user access to use the DynamoDB service, we could jump right into using Boto for our interacts with this database platform [11].

Creating a DynamoDB table was our first step which we were able to do within Boto by using the DynamoDB resource API. To creation a service connection, we needed to pass in the region name, access key ID and secret access key. From there, we were then able to call the create table method and provide several required arguments. The table name was provided as well as the key schema, which identifies what attributes would be the primary key and the key type for each attribute used. Our data had a key in place already named provider ID which was used as the primary key for the DynamoDB table as well. There were two options for key type, hash or range. A hash key, also called a partition key, which is used to handle distribution of the data across multiple partitions, and a range key, also called a sort key, which indicates how the data stored in DynamoDB will be sorted. For single attribute primary key, a hash key needed to be used as the key type. We also needed to supply the definitions of the attributes we chose to use as our key. We specified this as provider ID and assigned it a data type of string. The last argument we needed to supply in order to create a table was provisioned throughput, which was the amount of read and write activity the table could support [11].

If the table we created was in a highly transactional environment we would need to take time to consider how to calculate what values we would need to set the read and write capacity units to. These throughput settings are used to take into account how many systems resources would need to be reserved in order to meet the I/O requirements. The values are submitted in terms of capacity units, or amount of data the table needed to handle per second. To be able to calculate the amount of capacity units we needed to determine our data sets average item, or row, size. By taking the total file size of our raw JSON file, which was 937 KB and dividing that by the total number of rows objects within the file, which was 1000, to get an average item size of approximately 1 MB. "One read capacity unit represents one strongly consistent read per second, or two eventually consistent reads per second, for an item up to 4 KB in size. If you need to read an item that is larger than 4 KB, DynamoDB will need to consume additional read capacity units [17]." Same concept applies for write capacity units, except it is for an item up to 1 KB in size. Since our item size and I/O footprint was low for our project use case, we decided to use just 5 read and write capacity units to keep our costs down. With all of the required arguments gathered, we could now submit the resource API call to AWS to create a new table. Creation of the table itself took several minutes and once it was ready to use we were then able to start inserting data into our new DynamoDB table [17]. The Python code used to create the DynamoDB table is is shown in Figure 9.

```
dynamodb = boto3.client('dynamodb', region˙name='us-east-1',
aws˙access˙key˙id=k1, aws˙secret˙access˙key=k2)

response = dynamodb.create˙table
  (TableName='PatientSurveyData',
   KeySchema=[−'AttributeName': 'provider˙id', 'KeyType': 'HASH'˝, ],
   AttributeDefinitions=[−'AttributeName': 'provider˙id',
    'AttributeType': 'S'˝],
   ProvisionedThroughput=−'ReadCapacityUnits': 5, 'WriteCapacityUnits': 5˝,
)

return response
```

**Figure 9: Python code to create the DynamoDB table**

The resource API for DynamoDB provided a class named table which allowed us to execute table specific functions by providing a table name argument. We also utilized the Python library urlib to import the JSON file from its source on the web directly into our DynamoDB table [24]. The urlopen function was called against the HTTP path of the JSON file and stored into a variable. This variable containing the file data could then be read into a loading function provided by the Python JSON library to convert the data into proper JSON format that could be accepted by the DynamoDB put item method [7]. The Python code used to insert into the DynamoDB table is is shown in Figure 10.

```
dynamodb = boto3.resource('dynamodb', region˙name='us-east-1',
aws˙access˙key˙id=k1, aws˙secret˙access˙key=k2)

table = dynamodb.Table('PatientSurveyData')

url = https://data.medicare.gov/resource/rmgi-5fhi.json''

response = urllib.urlopen(url)

data = json.loads(response.read(), parse˙float=decimal.Decimal)

for row in data:
  response = table.put˙item(Item=row)

return response
```

**Figure 10: Python code to insert into the DynamoDB table**

Upon successful insert of the data into our DynamoDB table, we could then write code to pull back data from the database using the table class from the resource API. Normally we would just want to query this table for the information needed, but DynamoDB has different requirement for using specific query functionality than a normal relational database engine does. A query can only be used when a table has a composite primary key, which consists of both a partition key and sort key or has a secondary index that contains a composite key. Since our example utilized a simple primary key consisting of only a partition key with no additional indexes created on the table, we needed to use the scan function in order to access the data. The main different between a query and scan is that the scan always searches through the entire table to find the data and then filters out the necessary data based on the filter expressions provided. Using a query removes the additional step of having to filter out the data and can directly access the data required based on either the composite key or indexes that were created on the table, making it a better choice if faster response times are needed for larger sets of data. Scan worked for our specific needs in this project and worked similarly to query code wise in that the main required parameters needed to retrieve data were table name and filter expression [7]. The DynamoDB API provides a set of conditions that can be used to pull back data based on whatever criteria and attributes you specify through the filter expression parameter. With our sample dataset, we were interested in returning all of the hospitals from the patient survey data that were located in the state of Missouri. To do this, we utilized the eq condition run against the attribute state in order to return all items in the state of MO [10]. The Python code used to scan data from the DynamoDB table is is shown in Figure 11.

```
dynamodb = boto3.resource('dynamodb', region˙name='us-east-1',
aws˙access˙key˙id=k1, aws˙secret˙access˙key=k2)

table = dynamodb.Table('PatientSurveyData')

response = table.scan(FilterExpression=Attr('state').eq('MO'))

items = response['Items']

return items
```

**Figure 11: Python code to scan data from the DynamoDB table**

Using either the resource or client API, we could then delete the DynamoDB table once it was no longer needed. It was equally simple to delete a table using either API, so we wanted to show how the client API could be used to delete a table. It's used similar to other services in that we provide the service name, region and key parameters to allow us to connect to the service and then we could run the delete table method and just supply the table name as a parameter in order to delete our table [7]. The Python code used to delete the DynamoDB table is is shown in Figure 12.

```
dynamodb = boto3.client('dynamodb', region˙name='us-east-1',
aws˙access˙key˙id=k1, aws˙secret˙access˙key=k2)

response = dynamodb.delete˙table(TableName='PatientSurveyData')

return response
```

**Figure 12: Python code to delete the DynamoDB table**

## 2.5 Redshift

We've touched on several operational data services provided by AWS in the sections above and next wanted to get our hands on their data warehouse platform, Redshift. Redshift is a cloud based, fully managed data warehouse service that allows the utilization of standard SQL to analyze data on the platform. A Redshift data warehouse consists of one or many nodes making up a cluster, with a cluster containing one or more databases. One of the key features of the platform is its columnar data storage, which allows Redshift to enhance query performance by requiring fewer I/Os since only the columns involved in each specify query are utilized as opposed to the entire row. It also uses multiple parallel processing which spreads the data and query executions across multiple nodes of a cluster to keep query performance high [13].

Keeping our costs in mind, we decided to not expose the ability to create a cluster through our APIs and opted for setup through AWS Management Console. After navigating to the Redshift dashboard page in the same manner that we did for other AWS services, we selected the launch cluster button and were then prompted to provide information such as cluster identifier, database name and port and master user credentials. The next page asked us which settings we wanted for the node configuration. For our purposes we only needed to use the minimally sized node type, dc2.large, and just a single node cluster. On the additional configuration page, we left all of the defaults in place and made sure that the cluster was set to be publicly accessible since we wanted to query the Redshift database from our API. We then proceeded to the review page and opted to then launch our cluster [13].

After waiting several minutes for our cluster to come online, we were then ready to connect to the cluster in order to setup our new database, user and table for our data set to be imported into. Since Redshift is based on the PostgreSQL platform, any SQL client tools compatible with PostgreSQL would allow us to connect to Redshift. Using the GUI tool SQL Workbench/J [21], which is recommended by AWS, would allow us to connect to Redshift using a JDBC connection. Under the configuration tab of the cluster on the Redshift dashboard, there is a tab named database properties, which contains the JDBC URL we would need in order to connect. With this information we were able to setup a connection profile in SQL Workbench/J and assign the Redshift JDBC driver to the profile. The settings in the profile that we needed to provide were the JDBC URL, master user information and check the autocommit box. We were then able to successfully connect to Redshift and start running queries against it [13]. Figure 13 shows the Redshift connection information we provided to the SQL Workbench/J GUI in order to access our Redshift cluster for the first time.

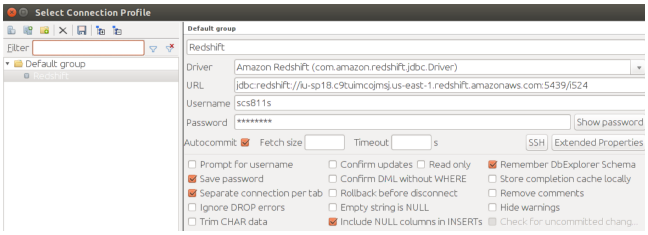


**Figure 13: SQL Workbench/J to Redshift**

We used a variation of the code for the one-time run SQL queries we executed to create our RDS database, user and table. We also wanted our API user to only be able to run queries on the data in our Redshift table rather than being able to drop or create them. After setting up the Redshift environment for us to be able to interact with our Python code, our next step was import our data from S3 into Redshift. By using the Python library, psycopg2, we could interact with Redshift using the PostgreSQL Python functionality [23]. Using this library was very similar to the MySQL library, as we had to specify the database name, Redshift endpoint, port and user name in order to setup a connection and cursor object. We could then run a query that utilized the copy command to load our data from S3. This command required that we supply the Redshift table name, path to the CSV file located in S3, our access key info, and options to ignore the header row, remove the quotes from around the values and specify the comma delimiter in the source file as a comma. Executing this command within our API would then import the CSV file with our patient survey data on S3 directly into our Redshift table [13]. The Python code used to import the CSV data from S3 into Redshift is shown in Figure 14.

```
redshift = psycopg2.connect(dbname= 'i524',
host='iu-sp18.c9tuimcojmsj.us-east-1.redshift.amazonaws.com',
port='5439',
user=redshift˙user,
password= redshift˙password)

cur = redshift.cursor()

cur.execute(COPY PatientSurveyData
  FROM 's3://hid-sp18-521/PatientSurveyData.csv' ''
ACCESS˙KEY˙ID ''' + k1 +
' SECRET˙ACCESS˙KEY ''' + k2 + ' ''
ignoreheader 1 ''
removequotes ''
delimiter ',';'')

return redshift.commit()
```

**Figure 14: Python code to insert S3 data into the Redshift table**

Then we wanted to provide the ability to query this data from within our code utilizing a SQL select statement. For our example use case, we coded a query to pull each state and their average star rating from the survey data to see how each state did overall at a high level. The execute function then submitted the query from our API and used the cursors fetchall function to store the query results returned back from Redshift into a variable. The Python code used to select data from the Redshift table is shown in Figure 15.

```
redshift = psycopg2.connect(dbname= 'i524',
host='iu-sp18.c9tuimcojmsj.us-east-1.redshift.amazonaws.com',
port='5439',
user=redshift˙user,
password=redshift˙password)

cur = redshift.cursor()

cur.execute(select location˙state,
avg(patient˙survey˙star˙rating)
from PatientSurveyData
where patient˙survey˙star˙rating in (1,2,3,4,5)
group by location˙state
order by location˙state'')

sql = cur.fetchall()
redshift.commit()

return sql
```

**Figure 15: Python code to select data from the Redshift table**

Code to delete all data from the table was also implemented in order to promote reusability of our Redshift data loading if we ever had the need to reload the table with data from a new version of the S3 patient survey data file. This required just a simple delete statement to remove all rows from our Redshift table. The Python code used to delete data from the Redshift table is shown in Figure 18.

```
redshift = psycopg2.connect(dbname= 'i524',
host='iu-sp18.c9tuimcojmsj.us-east-1.redshift.amazonaws.com',
port='5439',
user=redshift˙user,
password=redshift˙password)

cur = redshift.cursor()

cur.execute(DELETE FROM PatientSurveyData'')

return redshift.commit()
```

**Figure 16: Python code to delete data from the Redshift table**

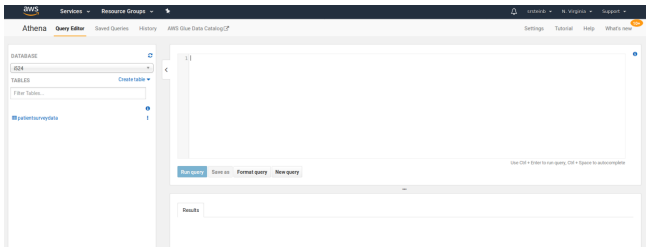

**Figure 17: Athena query editor**

## 2.6 Athena

The last AWS data related service we touched was Amazon Athena, one of the newer data services on the AWS platform. "Amazon Athena is an interactive query service that allows users to analyze data directly in Amazon S3 using standard SQL [18]." Its main use case is to allow the use of ad hoc SQL queries directly on files stored in S3 without having to ingest or aggregate this data in other data services in order to be able to perform analysis on it. The platform is serverless meaning there is no infrastructure to provision and scaling of query workload is handled automatically. Athena is able to query many different file formats and can handle querying structured, semi-structured and unstructured data. Before being able to run queries, metadata needed to be gathered on the files we would want to query in S3. "In Athena, tables and databases are containers for the metadata definitions that define a schema for underlying source data. For each dataset, a table needs to exist in Athena. The metadata in the table tells Athena where the data is located in Amazon S3, and specifies the structure of the data, for example, column names, data types, and the name of the table. Databases are a logical grouping of tables, and also hold only metadata and schema information for a dataset [18]." This meant we needed to first create a database to hold our metadata as well as a table structure that matches the patient survey data file we wanted to query on S3. We opted to use the internal Athena data catalog for our use case, but the option to use the AWS Glue Data Catalog also exists, which would allow Athena to automatically use the object metadata already collected and stored from other AWS data services that use AWS Glue [18].

We started working with Athena by using the AWS Management Console functionality in order to gain an understand on how the service works. The main dashboard page for Athena consists of database and table information on the left side of the screen and an interactive query window in the middle. Athena has a default database in place for storing metadata, but we wanted to create our own. DDL queries run on Athena need to be submitted in HiveQL, which uses a simple create database command that other SQL based languages use and allowed us to create a new metadata database name i524. Since this would just be a one-time database creation, this functionality was not implemented into our APIs [18]. Figure 17 shows the Athena query editor which contains an input window where we could run our HiveQL code to create our metadata database and run interactive queries as needed.

Next, we coded a create table DDL statement using HiveQL which would setup the metadata structure needed to be able to query the S3 file directly from Athena. Since we were creating the metadata table manually, the CSV file we wanted to use needed to exist in its own bucket separate from the rest of the files since the input parameters did not allow us to specify a direct S3 file name to use. It only reqiored a file path which would then look for the file based on the source file type we indicated in the create table code. We used the resource API in Boto to establish a connection to Athena as we did for the other services we used in our APIs above. From there we could then use the start query execution method to run our DDL code [5]. We opted to make this create table statement part of our API code in order to show how we could execute Athena queries programmatically. The create table command checks to see if the table already exists and if not, creates it. Additional properties about the source data file format, column separator, skipping the header row and quote identifiers were supplied, as well as the S3 bucket where the source file was located and the S3 bucket where we want the query results to be output to [12]. The Python code used to import CSV data from S3 into Athena is shown in Figure **??**.

```
athena = boto3.client('athena', region˙name='us-east-1', aws˙access˙key˙id=k1,
aws˙secret˙access˙key=k2)

with smart˙open.smart˙open ('s3://' + k1 + ':' + k2 +'@hid-sp18-521/
athena-input/PatientSurveyData.csv', 'wb')
as fout:
  for line in
    smart˙open.smart˙open('https://data.medicare.gov/resource/
    rmgi-5fhi.csv'):
      response = fout.write(line + '"n')

response = athena.start˙query˙execution(QueryString=CREATE EXTERNAL TABLE
IF NOT EXISTS i524.PatientSurveyData (
...columns excluded from example, too many...
)
ROW FORMAT SERDE
'org.apache.hadoop.hive.serde2.OpenCSVSerde' WITH SERDEPROPERTIES (
'separatorChar' = ',',
'quoteChar' = '"'
)
STORED AS TEXTFILE
LOCATION 's3://hid-sp18-521/athena-input'
TBLPROPERTIES (skip.header.line.count"=1");",
QueryExecutionContext=–'Database': 'i524'",
ResultConfiguration=–'OutputLocation': 's3://hid-sp18-521'")

return response
```

**Figure 18: Python code to insert S3 CSV data into Athena table**

For our example select query, we wanted to produce the records in the data set for a specific city that is specified by a user parameter in our API. Since we were not using the GUI to execute our select

query, our results would need to be stored in a CSV output file in the folder parameter we provided during the create table statement. This functionality is by default and would then allow us to use the Athena query output in CSV format to send on to other data services for additional analysis if needed [5]. The Python code used to query data from the Athena table is shown in Figure 19.

```
athena = boto3.client('athena',
region˙name='us-east-1', aws˙access˙key˙id=k1, aws˙secret˙access˙key=k2)

query = 'SELECT * FROM patientsurveydata WHERE city = "'%s"' % (city)

response = athena.start˙query˙execution(QueryString=query,
  QueryExecutionContext=−'Database': 'i524'",
  ResultConfiguration=−'OutputLocation': 's3://hid-sp18-521'")

return response
```

**Figure 19: Python code to query the Athena table**

In order to be able to re-run our Athena test cases, we included a function to delete the Athena table so that we could repopulate it on demand as needed and used the same method start query execution that the other two Athena functions do in order to interact with the service [5]. The Python code used to drop the Athena table is shown in Figure 20.

```
athena = boto3.client('athena', region˙name='us-east-1', aws˙access˙key˙id=k1,
aws˙secret˙access˙key=k2)

response = athena.start˙query˙execution(QueryString=
DROP TABLE IF EXISTS PatientSurveyData'',
QueryExecutionContext=−'Database': 'i524'",
ResultConfiguration=−'OutputLocation': 's3://hid-sp18-521'")

return response
```

**Figure 20: Python code to drop the Athena table**

## 3 CONCLUSION

AWS offers cloud-based services for many different use cases and data platforms to meet a wide variety of user needs. By taking most of the backend administrative work out of managing and provisioning data platforms, we were able to spend the majority of our research time on interacting with each of the individual services through the AWS Management Console and AWS SDK for Python, called Boto. In the cases where exposing service setup to an end user didn't make sense, we learned how to perform configuration through the GUI so that we could then work programmatically with the data itself later on in our project. This was accomplished through the use of several Swagger APIs that called Python functions that used Boto functionality to be able to interact with the AWS resource and client APIs provided for each individual service we touched on. AWS provided stellar API documentation for Boto that allowed us to successfully write our Python code to extract, transform, load and query our Medicare patient survey data using six of AWS many data related services: S3, RDS, Data Pipeline, DynamoDB, Redshift and Athena.


## ACKNOWLEDGMENTS

The authors would like to thank Dr. Gregor von Laszewski for providing us with an automated Latex document checker so that we could easily ensure that our final paper was correctly formatted to proper Latex standards and layout.


## REFERENCES


[1] AWS. 2018. Amazon S3 Features. Web Page. (April 2018). https://aws.amazon.com/s3/features/

[2] AWS. 2018. AWS SDK for Python. Web Page. (April 2018). https://aws.amazon.com/sdk-for-python/

[3] AWS. 2018. Boto 3 Low-level Clients. Web Page. (April 2018). http://boto3.readthedocs.io/en/latest/guide/clients.html

[4] AWS. 2018. Boto 3 Resources. Web Page. (April 2018). https://boto3.readthedocs.io/en/latest/guide/resources.html

[5] AWS. 2018. Boto Athena. Web Page. (April 2018). http://boto3.readthedocs.io/en/latest/reference/services/athena.html

[6] AWS. 2018. Boto Data Pipeline. Web Page. (April 2018). http://boto3.readthedocs.io/en/latest/reference/services/datapipeline.html

[7] AWS. 2018. Boto DynamoDB. Web Page. (April 2018). http://boto3.readthedocs.io/en/latest/reference/services/dynamodb.html

[8] AWS. 2018. Boto S3 Bucket Class. Web Page. (April 2018). http://boto3.readthedocs.io/en/latest/reference/services/s3.html?highlight=s3#bucket

[9] AWS. 2018. Creating a MySQL DB Instance and Connecting to a Database on a MySQL DB Instance. Web Page. (April 2018). https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_GettingStarted.CreatingConnecting.MySQL.html

[10] AWS. 2018. DynamoDB Conditions. Web Page. (April 2018). http://boto3.readthedocs.io/en/latest/reference/customizations/dynamodb.html#boto3.dynamodb.conditions.Attr

[11] AWS. 2018. Frequently Asked Questions About Amazon DynamoDB. Web Page. (April 2018). https://aws.amazon.com/dynamodb/faqs/

[12] AWS. 2018. Getting Started With Amazon Athena. Web Page. (April 2018). https://docs.aws.amazon.com/athena/latest/ug/getting-started.html

[13] AWS. 2018. Getting Started with Amazon Redshift. Web Page. (April 2018). https://docs.aws.amazon.com/redshift/latest/gsg/getting-started.html

[14] AWS. 2018. Getting Started with AWS Data Pipeline. Web Page. (April 2018). https://docs.aws.amazon.com/datapipeline/latest/DeveloperGuide/dp-getting-started.html

[15] AWS. 2018. Managing Access Keys for IAM Users. Web Page. (April 2018). https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_access-keys.html

[16] AWS. 2018. Overview of Identity Management for Users. Web Page. (April 2018). https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction_identity-management.html

[17] AWS. 2018. Throughput Settings for Reads and Writes. Web Page. (April 2018). https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ProvisionedThroughput.html

[18] AWS. 2018. What is Amazon Athena? Web Page. (April 2018). https://docs.aws.amazon.com/athena/latest/ug/what-is.html

[19] AWS. 2018. What is AWS Data Pipeline? Web Page. (April 2018). https://docs.aws.amazon.com/datapipeline/latest/DeveloperGuide/what-is-datapipeline.html

[20] CMS. 2018. Patient survey HCAHPS - Hospital. Web Page. (April 2018). https://data.medicare.gov/Hospital-Compare/Patient-survey-HCAHPS-Hospital/dgck-syfz

[21] Thomas Kellerer. 2018. Installing and starting SQL Workbench J. Web Page. (April 2018). http://www.sql-workbench.net/manual/install.html

[22] Yutaka Matsubara. 2016. PyMYSQL CRUD Examples. Web Page. (2016). https://pymysql.readthedocs.io/en/latest/user/examples.html

[23] Kostas Pardalis. 2016. Access your data in Amazon Redshift and PostgreSQL with Python and R. Web Page. (May 2016). https://www.blendo.co/blog/access-your-data-in-amazon-redshift-and-postgresql-with-python-and-r/

[24] Python Software Foundation. 2018. Urllib open arbitrary resources by URL. Web Page. (April 2018). https://docs.python.org/2/library/urllib.html

[25] Radim Rehurek. 2018. Smart Open. Web Page. (April 2018). https://pypi.org/project/smart_open/

# Report: Big Data in Healthcare

Hao Tian
Indiana University
School of Informatics, Computing and Engineering
Bloomington, IN 47408, USA
tian4@indiana.edu

## ABSTRACT

Health, as the most important factor that influences the happiness with life, people pay great attention and put much effort into it. The term "healthcare" is the result of efforts to maintain a healthy condition in the general public. However, the growing population brings many challenges to the healthcare field. To provide people with a good healthcare environment, the traditional healthcare field needs to cooperate with cutting-edge technologies in order to fit the population.

Healthcare today is an important topic throughout the world, and as the population is growing, human health needs to be protected by more advanced methods and technology. For tracking health in the population, each country needs a complete medical care system, which could monitor general health conditions and offer valuable information to medical services, such as hospitals, disease research institutions, and even national security agencies. However, as the population stands now, the traditional methods and technologies cannot fully support the requirement to satisfy the healthcare needs for most people.

To satisfy the entire population's healthcare needs with a functioning healthcare system, the cutting edge technology concepts - Big Data - provide the tools needed to develop better healthcare systems. The basic concept of Big Data is to process and analyze a large amount of data, including data mining, data visualization, and using machine learning knowledge to predict future situations. With the large amount of data needing processing in the healthcare fields, using the knowledge for processing massive amount of data is very helpful in dealing with issues that healthcare systems are facing today.

This paper discusses the Big Data technologies and applications that are currently being used in healthcare fields, and the possible technologies that might be used in the future in order to improve the quality of health care. The serious illnesses, such as cancers, AIDS and other deadly diseases, are also included in the discussion about the technologies concerned with healthcare.

## KEYWORDS

hid-sp18-hid524, Volume: 9, Chapter: Application, Status: 100.
Big Data, healthcare, Medical Care, Data Mining, Cancer.

## 1 INTRODUCTION

### 1.1 Healthcare

In starting this paper, we need to define what healthcare is. According to the definition from the Merriam Webster dictionary: healthcare is an effort to maintain people in a healthy condition using treatment given by trained and licensed medical professionals [11]. The healthy condition indicates that both physical and mental status are healthy, and they are basic factors that lead people to live a happy life.

Each country in the world has different healthcare systems to maintain citizens' health and protect citizens from getting diseases. The countries like Canada, the United Kingdom, and China have uniform healthcare systems, like national healthcare services provided to all the citizens [6, 33, 48]. The United States of America, as a developed country, has a very unique status regarding healthcare: the U.S. does not have a uniform healthcare system and had no universal healthcare coverage for U.S. citizens until 2016 [27]. The free market for medical care in the U.S. brought great value to the U.S. economy [28], but the lack of government management has caused trouble accessing health care for some people [26].

### 1.2 Big Data

Since the start of the second decade in the twenty-first century, the amount of information that people see and read each day has grown to be much higher that before, and the traditional methods and technologies for data processing are inadequate for this period of dramatically growing data sizes [41]. In this situation, people have realized the significance of developing more advanced data processing methods and technologies.

Big data is a term referring to the technologies that are used to process large amounts of data, and it is also a major concept used in the future technologies. The word "Big Data" has been given three characters. It has also been called 3 Vs [19]:

- Variety: the categories of data are more varied; the data could be in different forms, such as videos, sounds, and texts. Today, data processing techniques should be able to gain information from different types of data.
- Velocity: the speed of data processing has become more demanding; people need more efficient ways to collect data or process data. "Real-time" indicates the immediate updating of data.
- Volume: data size is incredibly big, the traditional computation methods will be abandoned because they cannot handle it. More efficient methods or applications need to be created.

The future technologies need to fit the rapidly increasing pool of data, and the data size is still growing. More than traditional data processing, such as data cleaning, and raw data processing to convert unreadable data into data that humans can read, the data size of today also brings another opportunity for people to gain valuable information from the data. Indeed, the Big Data-related technologies have more jobs available than the traditional technologies; analyzing data and predicting outcomes from data become very important parts of the Big Data idea. Despite the fact

that analysis and prediction are not new, the growing amount of data enhances the importance of those theoretical concepts from mathematics and statistics [41].

### 1.3 Healthcare and Big Data

Today, the human population on earth is 7.6 billion [30], and most of the population is concentrated in Asia, in countries such as China, India, and Southeast Asian countries [3]. Each person could be seen as a dataset with different variables concerned with health conditions, and the whole world is a huge, unformatted database, which includes all of that datasets. For each healthcare system, whether it contains public healthcare systems or private healthcare companies, this amount of data challenges every healthcare organization, and they need to develop adequate methodologies and use advanced technologies to achieve the purpose of helping people to maintain good health.

Big Data concepts have been implemented in many fields, including the healthcare area. The Big Data-related technologies, such as real-time data analysis, cloud computing, and data visualization, cut the cost of traditional data processing methods, which brings efficiency to the health care area and also lowers the cost compared to the traditional healthcare field [12].

### 1.4 Paper Layout

The paper is organized according to the following structure:

(1) The beginning of the paper discusses the trends within the healthcare field, as well as how the Big Data concepts and technologies affect the traditional healthcare field.
(2) Then the paper talks about the elementary component of the current healthcare industry, the data. The data transformation from paper to digital is the key to apply the newest technologies to the healthcare field, and it is also one of the major obstacles that limits the evolution of the healthcare field.
(3) The Internet of Things technology (IoT) is a great application in today's healthcare industry, because it provides real-time data to healthcare systems, and it also makes the healthcare service more efficient, despite the fact that it is not perfect.
(4) The cloud computing infrastructure of the common healthcare systems that combine the Big Data technologies is an important part of today's healthcare industry. Because of the distribution system infrastructure used in the healthcare field, today's healthcare services can process a large amount of data and serve numerous patients at the same time.
(5) After discussing those important technologies used in the healthcare field, the paper talks about the healthcare services that are used today. The important aspects in the healthcare field will be discussed, such as data analysis of digital medical data, gene prediction on potential health threats, and powerful systems that are used for healthcare services.
(6) After all the discussions about technology, the paper starts to put forth a new perspective: the healthcare in the future. In this section, the paper discusses new thoughts and new technologies that might be used in the future in order to make the healthcare services better.
(7) Summary of the paper.

## 2 TRENDS IN HEALTHCARE TODAY

The growing population has become a challenge for the traditional healthcare area, but as the amount of data grows, it also brings numerous opportunities for the health care area to provide better services. As time goes by, the healthcare area has made many improvements and has also improved its methods and technologies. Carol McDonald, an experienced medical and health insurance-related application developer, mentions some trends showing how Big Data is changing the healthcare area, which could be summarized as three major changes [16]:

- **Data:** In the healthcare field, the data quality will be better in its relevance and accuracy in the future. In order to provide better health care services, the information about each patient will be more closely related to the important factors that go into health condition evaluation [29]. More crucial data will be included in the database, and the size of false data is decreasing [29].
- **Methods:** The "methods" here indicate the theoretical statistical and theoretical mathematical methods that have been used in healthcare to provide data analysis services. As the date size is growing each day, the opportunity has come to many fields. The growing data brings the healthcare companies a great chance to improve healthcare service quality by offering patient healthcare plans. Healthcare companies could use statistical, mathematical and artificial intelligence knowledge to help patients to make plans about their healthcare investments.
- **Technologies:** The clinical service is improving, and more and more advanced technologies have been used in hospital, medical care and elder care settings. This trend helps improve technologies in health care. One of the most classic examples is the usage of IoT (Internet of Things) in healthcare [31]; the medical devices such as real-time medical monitors and trackers could update the health data on patients, which is a efficient method for providing more rational health services, even if it somehow compromises the privacy slightly.

## 3 DATA IN HEALTHCARE

The digital data growth in healthcare is remarkable: according to the Dell EMC Digital Universe research team, the digital data in the healthcare field is 4.4 zettabytes, with a 40 percent growth rate annually [36]. Compared with other fields, healthcare is one of the fastest growing segments in digital data in the world, and the healthcare applications are pushing the growth even more [36].

Despite the fact that the data growth in the healthcare field brings information to healthcare companies, the data quality is not guaranteed because of data format and source issues. Especially for the medical data, because medical data is so complex (due to concerns about scale of data dimension, missing values in databases, biased data and so on); even slight inconsistency in the databases could require much time and effort to fix [37]. However, people

have come up with various solutions for improving the overall quality of data that would be used in the healthcare industry.

## 3.1 EHR - Electronic Health Records

Electronic Health Records (ERC) is the name for digital health condition data, and it stores patient health data [49]. Storing the digital data of each patient is a basic concept that exemplifies applying Big Data ideas into healthcare, since digital data is the foundation of applying data analysis, data machines, and machine learning methods and technologies. Figure 1 shows how the EHR collaborates with other segments of healthcare systems [43].

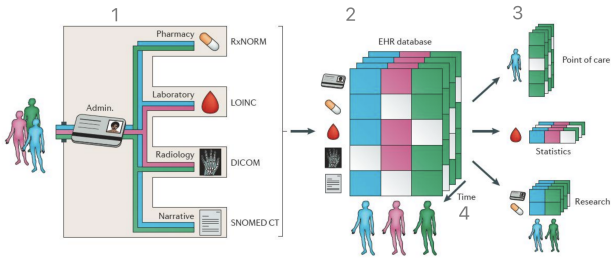


**Figure 1: How does a classic EHR system work**

Figure 1 shows a stream chart showing how a classic EHR system works. In general, it could be separated into four components (each component's order numbers follow the number in Figure 1) [49]:

(1) First, patients provide their health data in a certain format, and the format depends on the EHR system. The components include different types of information, and each type of information will be used for different purposes.
(2) The EHR database collects all the data. The data will be stored for future usage.
(3) Base on the need the patient has, the EHR system uses appropriate data for different needs. Some EHR systems need human support for making plans and managing the healthcare for patients, but some systems do not need human support, depending on the design and implementation of the system.
(4) Patients get their results from the system.

In the United States, the United Kingdom, and other developed countries in Europe, the EHR systems have been put into wide use in hospitals, research institutions, and healthcare companies [44]. The EHR database can be shared by the patients, healthcare providers, and others who have permission to access the data on certain patients [43]. The EHR data that is open to the public (including non-sensitive information regarding patients) could be used as the material for multi-purpose studies on public disease distribution, making healthcare plans and decisions for patients, and also predicting future circumstances regarding health conditions in certain patients or the general public [44].

Despite the fact that EHR systems have been widely used by many countries and many organizations, because of the natural complexity of the medical data, the EHR has limited ability to preprocess and analyze it [44]. Although EHR systems have been implemented by many healthcare institutions in the United States, even today, the United States does not have a uniform EHR system; the constraints of local EHR databases cannot provide the qualified data analysis. Furthermore, because different institutions use different EHR systems, the connections between them cannot form a uniform system, which could provide a more powerful healthcare system for the country. More than that, the privacy of the EHR data is also a controversial topic; many patients refuse to publish their health data due to privacy concerns, which is a non-negligible problem, because some of the information is crucial for certain data analysis jobs [44].

## 3.2 KP HealthConnect - A successful EHR system

KP HealthConnect is a successful EHR system model developed by Kaiser Permanente. Kaiser Permanente is a massive healthcare company founded in California, U.S., and KP HealthConnect exemplifies a successful pattern of EHR system use [15]. KP HealthConnect covers some states in the U.S. (most of them are on the West Coast), and the feedback from both patients and physicians sides is quite positive [15].

Just like how the general EHR system works (Section 3.1), the KP healthcare needs patients to provide their EHR data to the system first, then physicians or medical specialists, and the system itself, will process the data for analysis and deep learning, but why could KP HealthConnect be called "a complete EHR system" with high evaluation scores from users? According to the study from The Commonwealth Fund (a healthcare company), the key components of KP HealthConnect's success are as follows [40]:

- **Lowers the communication cost:** Compared with other EHR systems, KP HealthConnect lowers the communication cost between patient and doctors. Patients can fill online surveys to report their health conditions without communicating with their doctors or health care providers ordinarily.
- **Provides more access to patients:** the system provides patients with more privileges to manage their own data in the EHR system; they can choose the data that they would like to provide to the system, to doctors, or to the public.
- **Learning Models:** The EHR data in the system could be used for providing patients with the future healthcare plans. The study model in the KP HealthConnect system can use the data that patients provide and the medical knowledge in the model to create a special health plan for each patient.

Indeed, the KP HealthConnect is a great EHR system that only needs slight human assistance for the system to work, but the system only covers fewer than ten states in the United States, so the EHR database is not large enough for the learning models to come up with rational decisions. More than the size of the EHR database, KP HealthConnect still has issues with data consistency: the databases in different data centers could be different, and it is usually caused by delayed data updating [47]. The issue indicates that the distribution system in KP HealthConnect is not well formed or implemented.

### 3.3 Cancer Biomedical Informatics Grid

Cancer has became one of the most lethal diseases in the world, and there are no treatments proven to cure cancer. Even today, people are still short on knowledge about it, and the trend of today is that more and more people are getting cancer. In order to gather information about cancer for the purposes of learning about cancer and preventing cancer from occurring, the National Cancer Institute (NCI) developed the Cancer Biomedical Informatics Grid (caBIG) [32].

The caBIG is a clinic database and system used for cancer study; specifically, scientists and medical specialists from the NCI use the data collected from different resources to study cancer. The infrastructure of the caBIG system allows scientists and medical specialists to study the data and use the applications offered by the system to conduct studies concerned with predicting cancer growth, forming patient treatment plans, and communicating data and study results [32]. The caBIG is a great step for people to fight against cancer by gathering information and concatenating data, systems, and scientific professional resources together to study it.

However, in reality, the outcomes of the research conducted by using caBIG database and system are not optimal, due to many reasons [32]:

- The infrastructure of the caBIG system is too complicated because of the nature of data needed for cancer studies. Additionally, funding the system is too expensive. Most of the money is not used in the laboratory; instead, most of the cost has been used to build the heavy load database and the program management, because the contents in the caBIG system are too complex and the communication between the research community and the system leader is not efficient.
- Since the NCI is a life science institution, the actual coding program implementation of the caBIG is not NCI. NCI sells the job out to some technicians to complete it as contractors. However, the contractors building the project do not really understand cancer research. The project does not actually meet the requirements for the research.
- The commercial usage of the caBIG is unsustainable, since the complexity of the system restrains the research, and it also limits the actual services that NCI could provide to the public and all potential users, even though the caBIG is free and open source.
- Because of the complexity of the caBIG system, poor communication between technicians and researchers, and the slow pace at which the research happens, the caBIG system leader layer does not have a goal that matches the purpose of using the caBIG. Instead, the caBIG becomes a huge, sophisticated data collection device that has little functional value.

Despite that in reality, the caBIG does not provide desirable results to the public, it is undeniable that caBIG represents a valiant attempt in building a massive database system to fight against cancer.

### 3.4 Limitation of healthcare Data today

Even though the digital health data has become more common in practice, the issues and the potential risks still exist in the data. From the general EHR system for all diseases, and the caBIG for the cancer specifically, the healthcare data is always complicated, huge in memory, and lacking in structured data. According to Dan LeSueur, a experienced technician in the healthcare industry, there are several difficulties the healthcare industry encounters in using the data [38]:

- The healthcare data is complex, and each data owner has a different form of data storage. The complex nature of healthcare data is the numerous dimensions recorded for each patient's data, and as more and more patient data has been added to a database, the time complexity of exploiting the database has become harder and harder.
- The healthcare data is too widely distributed in different places: different hospitals, healthcare providers, and even doctors hold various data about patients, as well as information about diseases from different places. The difficulty about distributed data is more than geographic, but more about the fact that the source of the data is distributed by different sources.
- Data in the healthcare industry is not formatted; there are too many unstructured data pieces that do not fit in the structured databases. According to the experiences from EHR system implementations within all different organizations, the unstructured data is a significant obstacle to the future data analysis. Even the high-performing EHR systems, like KP HealthConnect, cannot provide good learning results from unformatted data collections.
- The inconsonance of definitions and standards about the data influences the treatments. To be more specific, for example, different doctors have different standards about a criterion: some level of the criterion can be seen as the clue to a certain disease, but doctors might disagree with that. Furthermore, the standards concerned with health care and disease are always changing, and the new observations of those standards cannot always be updated and published, which poses more obstacles to building healthcare applications.
- Missing data is a big issue in the healthcare data. It is usually caused by the privacy concerns from the data providers; the sensitive data is not usually provided by the patients, or the sensitive data is not published to the public, although some of that sensitive data is crucial for the data analysis [44].

Given the current quality of the healthcare data, it is known that the process of improving the healthcare data quality is a long term project. Additionally, as the life science knowledge is improving, the data that will be used in the healthcare industry will be more complex.

### 3.5 Public-Key Cryptography for Security of the Health data

In previous sections, it is known that the privacy of the data is one of the major concerns that prevents people from offering complete health data records, since the sensitive information always appears in the health data. More than sensitive information, the IoT (Internet of Things) devices for health care also have the probability of

being attacked by adversaries (for example, the Fitbit in Section 5.3.).

Although, as the emergence of internet technologies has enabled people to share information with each other, in the healthcare field, people can now have more access to their personal health data, and they can more easily share it with other people. As this situation appears, the security demand with the privacy of health data has become inevitable. For this demand, applying public-key cryptography methods could be a great practice for protecting the privacy of the healthcare data [18].

Public-key cryptography is one of the greatest inventions in the history of mathematics. By using two math-related keys (public key and private key), it makes it more secure to encrypt data, since only the private key can decrypt the data that is encrypted by using the paired public key. The security of public-key encryption makes it a good candidate for securing personal health data [18].

The more detailed process as to how the public-key encryption can be used is as follows:

- Every patient that goes to a hospital can have his/her own private key, and then a public key is generated based on that private key. The private key will be kept secure by the patient.
- After the patient goes to the hospital, the new medical data will be generated by the hospital, and then that data will be encrypted using the patientfis public key. Then it can be stored in the hospital data system.
- Whenever a doctor feels he needs to have access to that medical data, he can reach out to the patient again, and the patient can use the private key to decrypt that medical data.

The usage of public-key cryptography can not only work in a hospital data system, but it can also work whenever people are going through an activity that generates the personal health data. For example, when people buy a new IoT device that helps detect the blood glucose levels, their blood data will be generated. After that data is generated, it can then be encrypted by using the userfis public key and then it can be stored anywhere, because only the user can have access to it by using the private key.

## 4 IOT PRODUCTS IN HEALTHCARE

The technologies used in the healthcare field have significantly changed the way people think about healthcare. With the trend of Big Data generation, many new technologies and products have been created in order to serve the growing health data.

IOT (Internet of Things) is a new technology field, which combines cloud system and data analysis knowledge. It is a great area that offers the real- time data transfer, and the real-time data is the basic component of high quality data analysis. The implementations of IoT have been used in multiple fields, such as communications, transportation, online shopping, and others. The most important improvement that IoT brings to the world is connecting the data that is stored separately all over the place into a center for better data use.

IoT technology and products have been used in the health care industry that healthcare providers are using today to help people make wise decision about their health plans. IoT could provide real-time medical services to people, and it could also lower the cost of data collecting.

### 4.1 IoT Implementation in healthcare

The development of Internet of Things (IoT) has brought a lot of convenience to our lives. Self-flying drones have been used frequently in outdoor activities. Home management devices like Alexa and Google Home have brought a new interface for users to have a better experience at home.

For healthcare, the IoT also has great potential. In many hospitals, there are machines that monitor patients' health 24 hours a day to ensure that patients are in good condition. But nowadays, as the IoT devices become more and more portable, there are far more interesting and promising possibilities. The emerging portability of IoT devices makes it easy for people to carry them without feeling that the devices are a pain [31].

This portability means patients can carry the device 24 hours a day without even feeling it. This means it becomes even easier to collect medical data for everyone. For some disease like diabetes, it is very critical to monitor a patient's blood glucose levels, so a portable IoT device is born to do such kinds of work.

Other than portability, the ability to collect and transfer data is also important for such kinds of IoT devices. The Bluetooth technology enables the device to connect with cellphones so that the patient can have an overview of their medical data. Furthermore, using cellphones, they can share this data with their doctors. This series of data collecting and data sharing actions is a big deal for patients.

### 4.2 IoT example 1: Livongo

Before the healthcare IoT came into use, a patient with obesity needed to take their blood sample and bring it to their doctors so that they could know what was going on within their body, but right now, with IoT devices, they can collect this information within seconds and they can react to emergent situations in a more timely way.

For example, a company called Livongo is providing IoT devices and services to patients with chronic conditions to monitor their conditions. Patients with obesity can use Livongo's palm-sized device to collect blood samples, and the device can instantly analyze the blood sample and gather information on the patient's blood glucose levels [25].

Connecting to the patient's cellphone through Bluetooth, the device can then send that data to the cellphone. On the cellphone, the patient can view the data and make adjustments to their diet. If necessary, they can also share that data with their doctors. Livongo also has a strong data infrastructure to process the patient's behavior data and medical data, and it uses artificial intelligence to find patterns among that data so that they can provide better advice to the patient to help them live a healthier life [25].

### 4.3 IoT example 2: Fitbit

In the practice of performing IoT, Fitbit is a great company that produces IoT healthcare devices. People using Fitbit hand bands can generate a huge amount of personal data through their hand band. That data is then stored in Fitbitfis servers. Then, as more

and more people use hand bands, more and more personal health data is generated every day [10].

The abundance of health data brings many benefits. For example, more data means we can generate more insight for the people using the devices. Big data and machine learning can exploit that data and generate more insights to help people live a healthy life [10].

However, the Fitbit application is not perfect on security, since the abundance of personal data is also a double-edged sword when it comes to data breaches. Services like Fitbit can never guarantee that the user data can be perfectly safe. Once the data breach happens, the privacy issue can be very severe. Once third parties get access to that data through data breaches, they can sell that data on the black market, and then millions of people will have to face privacy issues. There is even the risk of being hacked by a third party, but in general, the security perspective in Fitbit devices performs well [35].

### 4.4 Barriers with the IoT in healthcare

In the examples, it is clear that the IoT devices provide real-time services to patients, and they also collect large amounts of valuable data from patients. It saves time from both the patient side and the medical specialist side, and it could help create learning models that use the data to produce a better result. However, the IoT in the healthcare field is not perfect, and there still many problems that the current IoT in the healthcare field needs to address:

- Inconsonance of IoT data and EHR databases: the data collected from the IoT devices is hard to coordinate with EHR databases, and it is mainly because of the complexity and lower integration level of the EHR. So even as IoT devices collect many pieces of data from patients, if the data cannot be updated to the database properly, the work of collecting data is meaningless [45].
- Split databases and no centralized IoT device: when an IoT device collects health data, it should collect data using multiple criteria, such as heart rate, glucose levels and others. Unfortunately, each IoT device only provides the partial data to the system. For example, IoT devices only report the glucose level of a diabetes patient, but the rest of the data will become worthless. This is a big barrier preventing the wide usage of IoT, and this would generate extra cost for producing IoT devices for use in different settings [45].

Still, the nature of the health data is a factor that lowers the performance of IoT devices in healthcare. Furthermore, non-centralized IoT services today are also preventing more people from using IoT devices to monitor their health conditions.

## 5 CLOUD COMPUTING INFRASTRUCTURE IN HEALTHCARE

As what has been mentioned in Section 3.4, the limitations of the health data today will influence the quality of data in the healthcare field, such as data mining, data analysis, and predicting learning. Despite that we cannot change the quality of the healthcare data immediately, we can work more on the other aspects of healthcare systems. For example, the infrastructure of data processing could make up for the shortage in the health data. There could be many solutions for organizations to use to offer high quality healthcare.

### 5.1 Impacts from the Cloud Computing

Today, cloud computing has become a great tool for processing computation tasks involving the large data size. The cloud computing uses the distributed system principle of gathering computational power via a connected network or the internet. By applying the cloud computing to the healthcare industry, it could make the problem of dealing with a large amount of complex health data much more scalable.

Current market demands in the healthcare field are becoming more related to real-time service and prevention: people want their healthcare to perform their functions when people need t hem, and they also want it to prevent them from getting lethal diseases. However, with the aforementioned trends about data complexity, the prevalence of chronic illnesses and the aging population, the demands from the public seem extremely hard to achieve. In order to satisfy those demands, the healthcare organizations need the power of cloud computing [46].

For building a cloud service for a healthcare organization, the security of the data needs to be guaranteed by the cloud system. From this premise, the cloud computing system could use computational power to process the data. With the secure data as the premise, the cloud server built for dealing with a large scale of data could bring economic benefits and time efficiency to the organizations [46].

Sections 4.2 and 4.3 give two applicable solutions for using the cloud server to achieve great performance in the healthcare field. In those sections, the implementations of using multiple aspects of the Big Data technologies were shown, such as distributed environment, machine learning, and data pipeline.

### 5.2 Scalable data solutions from Government healthcare

For many years, the healthcare system in the U.S. has been associated with formidable and unaffordable medical bills. Many people bear the fear that they will get a disease that will cost their life savings. For the most risky populations, the elderly, disabled, and low-income individuals in the U.S., they can rely on government healthcare programs. For these people, whether they can receive the healthcare on time is essential to their lives. To ensure a better management of patient healthcare data and ensure the patient can receive the aid on time, the government needs more scalable data solutions to manage the programs.

There are several components that constitute scalable data solutions:

(1) First there should secure data storage.

As it is known, the federal government manages hundreds of millions of people's medical data. It is essential that this data does not fall into the hand of those who can exploit this data in the terrible way that can expose people's privacy and put them in danger. So a distributed content management system (CMS) is suitable in this case. The distributed cloud environment can eliminate the machine breach that can happen in the government's legacy servers. Additionally, the CMS can enable government officers to

update and review that data in an iterative and timely way, so that the efficiency in the process can be ensured [9].

(2) When the distributed CMS is ready, it becomes possible to make complete use of that medical data to make better medical decisions.

For example, the ETL data pipeline can be built into this distributed environment to get insight into the medical history data. Insight like how much aid is sufficient for this kind of disease can be gathered from the data analysis conducted through this data pipeline. Once this information can be gathered, the government can make a better disposal of that medical aid to ensure everyone gets enough money for their situation [5].

## 5.3 Scalable data solutions from Companies healthcare

Other than the government healthcare programs, the employer healthcare program is also important for people. For a lot of people, where they want to work usually depends on whether they can get a good healthcare plan from their employers. So it becomes clear that the employers also need to make good decisions on their healthcare plans for their employees. For many companies, the money they can save on the healthcare for their employees will ultimately go into their profits for the year [7].

However, there is an obstacle for employers to make better decisions when it comes to the healthcare plan for their employees.

The problem is that these companies lack a suitable infrastructure to keep track of the healthcare data of their employees. Many of these companies will follow the conventional way of using Excel data sheets to keep track of this data. This methodology becomes limited when the organization wants to get insight into this data. However, a more advanced data analysis system for this data is difficult without the infrastructure. Therefore, to enable these analyses, these companies should invest in a more scalable data infrastructure for their healthcare data. With a scalable data infrastructure, the data can have higher quality compared to simply using Excel. For example, a scalable data pipeline can be comprised of several components, which have their own specific job, like data cleaning, duration, and normalization. The remaining data that has gone through this pipeline is clean and has a much higher quality, which is extremely helpful when it comes to doing data analysis on this data [7].

Once the infrastructure is ready, the companies can use the data analysis to drive insight. For example, they can now have answers to questions like what factors are driving their healthcare expense higher and what services their employees are spending their money on.

## 5.4 Scalable Data Pipeline

The implementations of both solutions in Section 4.1 and 4.2 need to apply the data pipeline in order to achieve the scalable performance. Without the data pipeline, the performance will be terrible because of the complexity of the healthcare data.

Before we actually get in touch with the usage of data pipelines, its definition needs to be clarified. According to the definition from the Dataconomy [20]:

*"The data pipeline is an ideal mix of software technologies that automate the management, analysis and visualization of data from multiple sources, making it available for strategic use."*

A scalable data pipeline for healthcare systems should be responsive and reliable so that patients can receive what they need in time. Here are several components of this kind of scalable data pipeline [20]:

- Source: the source of the data pipeline can be diverse. The data can come from a persistent data storage form like MySQL, PosgreSQL, MongoDB, and others. Additionally, it can also be other web services that try to push this data into the data pipeline. If it is the latter, then the communication can be through either HTTP or RPC services.
- Message Bus: Since the data pipeline involves moving data from place A to place B, there should be a component that can provide buffering to the grouping when the data is transferred from one place to another. Think of a message bus as an exact pipeline. The data goes in from the left, and it can be stored persistently on the message bus, or it can be pulled out from the right and go to other components in the data pipeline.

  As a persistent message queue, the data can be processed in order and eliminate the pressure of outage from the consumer that tries to pull out the data. Some widely used message queues include Kafka and Redis. There are a few differences between Kafka and Redis. Since Redis is a memory database, it will require as much space in memory as there is data in the pipeline. Also, the data can be lost in this non-persistent environment.

  Another difference is that, for Kafka, instead of pushing data to consumers, like Redis does, the consumers in the data pipeline keep track of the most recent data they have read and can ask for the next set of data. This way, Kafka enables consumers to read data in real time and in a streaming way [42].
- Data Router: Since the message bus does not have the capability to distribute the data to other services, there should be another service that reads data from the message bus that distributes to other services that will utilize the data for work like data analysis. This consumer service can be a router that routes different data from the message bus to other services.
- Data analytic platform: This component is where the data analysis work happens. This platform subscribes data from the data router and does all the data analytic work to generate insight from data that has flown through the data pipeline.

Furthermore, for optimizing the data pipeline in actual applications, using virtualization tools would lower the cost of usage. For example, using Docker to deploy a rooter service would be very convenient and save a lot of time, since the servers might need to update manually. Also, using Swagger to generate rooters would cut the communication cost between the server and clients; in this case, the server could be the healthcare provider, and the clients would be patients who are using the server [34].

With data infrastructure like this, the data can be cleaned when it flows through the data pipeline. Also, the usage of the message bus can enable very scalable data processing among different services. The data analysis with huge amounts of available data can generate better insight with this scalable data infrastructure.

# 6 BIG DATA AND HEALTHCARE TODAY

Today, there are many possible uses of Big Data technologies in the healthcare industry. Technologies such as statistical learning models, cloud computing, and IoT enrich the possibilities of what the healthcare field could achieve. The implementations of those technologies create many profitable enterprises, and they also bring better service to people for maintaining healthy lives.

As this paper has mentioned before, big data is widely used in the healthcare industry currently, and it gives a variety of supports and solutions in helping take care of human health better and curing certain diseases. The professionals in big data can find important rules or trends by gathering information from different symptoms and analyzing its similarities and differences, which is hard to observe in other ways. Since all the industries in the healthcare field realize the great progress that big data brings to human health, they have started to invest more funds and put more facilities into it. Therefore, there are already some successful examples that combine healthcare and big data technologies.

## 6.1 Big data driven healthcare - Data analysis

With the quality of the data improving as it has been, the data analysis will become more and more valuable in the healthcare field.

The action for building data-driven healthcare organizations is the unstoppable trend of the field. As mentioned in the IBM software white paper, "Data-driven healthcare organizations use big data analysis for big gains", it mentions the guideline for establishing the a efficient data-driven healthcare organizations [8].

For establishing a efficient data-driven healthcare organization, the pre-request is to obtain high quality data. However, having only the high quality data is not enough; the analysis work is the key to achieving optimal results. According to the white paper, the analysis work in a health care organization needs to be able to perform analysis from two sides [8]:

- Clinical analysis: The organization needs qualified medical specialists to provide health analysis from medical perspectives. The medical team needs be able to define the standards of different criteria to define the health condition of its patients.
- Advanced analysis: The organizations need to provide the machine- based analysis applications in order to process the huge amount of data. The advanced analysis is crucial for the organizations, because it can offer patients real-time feedback about their health conditions. With growing data amounts, this analysis will become significant in affecting the quality of the health care services.

By combining these two aspects, a healthcare organization's ability to analyze data would become optimal. Compared to the traditional one-perspective healthcare analysis, the cooperation between both sides would offer patients rational analytical results. Other than good analytical performance, other aspects of a healthcare organization also need attention, such as data integrity, data security, and application performances [8].

## 6.2 Evidence-Based Medicine for Healthcare

Evidence-Based Medicine (EBM) is a great practical approach that uses the principles and methods of data analysis to study the disease and make optimal medical decisions by not only using scientific knowledge, but also by using medical data. This approach cuts down on the cost of human analysis. Even the EBM is strongly dependent on the digital data, but it combines the knowledge in the medical field and the technologies from the data science field. It follows the principles in Section 6.1, and it would give a satisfying result for the medical plan [39].

The traditional hospitals and clinics use "cookbook medicine" methods to determine which diagnosis patients get, which means that doctors use the same battery of tests to identify the cause of symptoms [2]. Cookbook medicine is a practical approach of medicine and only gives weak recommendations to patients. However, there exist too many diseases that appear similar in their symptoms at the beginning, so it is impossible or inaccurate for doctors to make a decision during the primary period. Therefore, in order to improve the accuracy and time of disease determination and reduce the pressure on hospitals, evidence-based medicine (EBM) was introduced and it has been used widely around the world to offer strong recommendations for patients.

Evidence-based medicine is an approach to medical practice intended to optimize decision-making by emphasizing the use of evidence from well-designed and well-conducted research. In other words, instead of using the same process of tests to figure out the type of disease a person has, EBM offers millions of patients' data to help doctors match these symptoms to the most similar example so that it can generate the most accurate result. More importantly, with the development of big data and smart devices, the technology is also open to patients. Patients can use smart device applications of evidence-based medicine to identify their illnesses [39].

In practice, Beth Israel Deaconess Medical Center gathers two million patients' data, stores them in the cloud, and provides smart phone applications to users to help those users diagnose their symptoms. For example, users could input "high blood pressure", which can match another term called "elevated blood pressure" [2].

## 6.3 Study DNA Sequencing to Prevent Diseases

Human genes carry more information about our lives than we think. The discovery of DNA technology unlocks a lot of possibilities in healthcare. The DNA discovery technology unlocks the myths inside genes. People can know how likely they are to be exposed to the danger of obesity and genetic diseases. But most of these benefits can be only seen within hospitals or organizations that provide this DNA service.

The DNA service has not been very affordable and convenient for most people until recently. As the technology in DNA discovery advances, it becomes cheaper and cheaper to order a DNA test. Some services that can provide affordable DNA services have emerged, and the usage of DNA could be a potentially promising area for the healthcare industry.

A service called Helix is a great example. Helix uses the new DNA sequencing technology and machine learning to enable a more precise and affordable DNA test. The new DNA sequencing technology is different from the conventional genotype technology. Compared with the conventional DNA genotype that only reads a snapshot of DNA, DNA sequencing reads base DNA pairs so that it can process more DNA pairs and provide a more precise test result [17].

This service brings unprecedented convenience to DNA testing because users only need to buy a test kit online. Once the test kit is delivered, users only need to take a drop of saliva and mail it back to Helix. Once the DNA test is done and the result is ready, users can then view their genetic information online and see those insights into how to live a better life based on that genetic information [17].

## 6.4 Big Data Analysis In Predicting Deadly Infections

With the widespread usage of IoT and sensors in healthcare, more and more medical data is being collected. With such an abundance of medical data, the need to make full use of this data is imminent.

Some promising ways of exploiting this medical data include analyzing medical records of patients with the same disease, and then generating insight from that data so that the those diseases can be detected in the early stages. For example, a disease called sepsis is an inflammatory infection that kills around a quarter million people in the US every year. The disease is very hard to spot in its early stages. Furthermore, once a patient is infected with such a disease, it is very hard to turn things around. However, with big data analysis solutions, such detection in the early stages can be possible.

Dignity Health, the largest hospital provider in California, is employing big data and advanced analysis platforms to detect such sepsis cases at an early stage. By monitoring 120,000 lives per month in its 34 hospitals and managing 7,500 patients with potential sepsis per month, Dignity Health collects data from those people's electronic medical records. It then uses natural language processing to monitor factors that could indicate a sepsis infection [4].

Once a potential sign of infection occurs, it then notifies doctors and physicians to take action. Since it started utilizing the big data and its predictive analysis system, Dignity Health has seen a drop of five percent in the sepsis mortality rate. Since the hospital can now better predict how likely the patient is to be infected, it can take the most appropriate action so that the patient can save a lot in medical costs [4].

## 6.5 Big Data and Preventing Healthcare Fraud

Other than being useful in predicting epidemics, curing disease and improving life quality, big data can also be helpful in allowing the healthcare industry to work more efficiently. In the U.S., most individuals rely on healthcare insurance to cover their medical bills. Along with the prevalence of medical insurance also comes insurance fraud. Like detecting epidemics, big data can also be helpful in predicting these instances of insurance fraud.

United Healthcare, the biggest healthcare insurer in the U.S., is using big data and advanced analytics to detect insurance fraud and waste. United Health gathers data from its insurance members, insurance claims, hospitals, healthcare providers and clinicians, and then it uses advanced analytics to find insight from that data, build solutions based on insight, and achieve better management of medical resources [4].

## 6.6 Streaming System of Record for Healthcare

For the current healthcare industry, more and more data needs to be stored so that people can make better use of that data. Therefore, it is important to look for the best ways to access, compare, and analyze the data. Now, this health data is accessible to healthcare providers all over the world because the system's owners have implemented a streaming system, which helps integrate consistent and real-time patient data by using the health devices. It provides a huge streaming capability that allows it to deal with high-volume data like Facebook and Ebay. In the meantime, healthcare providers can access patient data with the most effective and efficient methods.

The stream is an unbounded sequence of events carried from a set of producers to a set of consumers [13]. Currently, MapR Stream is implemented by the largest number of healthcare companies. MapR Stream could build a bridge between producers and consumers to exchange and update data in real time on the Apache Kafka 0.9 API. The MapR Stream has two strong advantages that provide efficient healthcare services:

- First Advantage - Partition Provides Good Concurrency

  Data from patients is divided into a variety of partitions according to the topics within it. And this data is transmitted in a parallel manner across multiple servers. The advantage is to speed up the transmission of data and put data into different categories. Figure 2 shows the infrastructure of how the data transfers among healthcare providers and patients [13]:

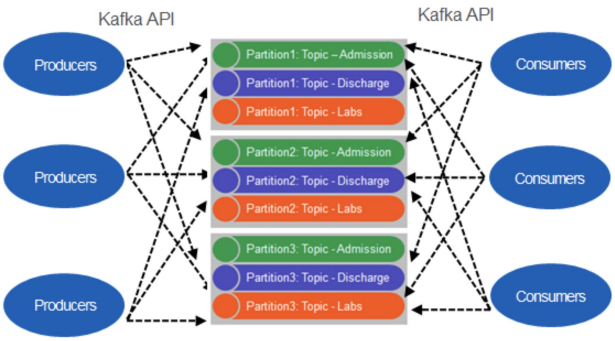


**Figure 2: The infrastructure of the MapR Stream**

- Second Advantage - Partition Serves as a Similar Function of Queue

  Each partition offers similar functions with a queue as shown in Figure 3 [13]:

  Messages with the specific topic will be appended to the corresponding partition and arranged in order according to the time they entered the partition (old messages with the high priority and a small number and new messages with the low priority and a big number). However, data

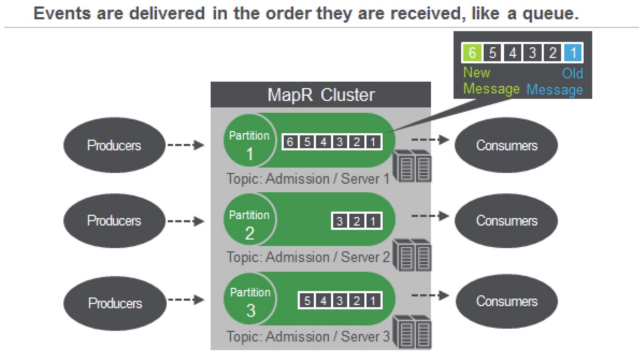


**Figure 3: Partition Servers of the MapR Stream**

in the healthcare field is not only for one-time use, but it needs to be used multiple times. The point is that data needs to be delivered towards multiple destinations, unlike a queue, so it will remain in the partition for a certain time to be accessible to other users in the system. This feature provides an effective means of reusing and delivering data.

Other than MapR, there is another company called Liaison Technologies that uses the stream system to provide healthcare services. Liaison Technologies has become the leading company that uses this type of streaming technology for healthcare records.

Liaison uses its own platform called ALLOY to help customers analyze and transform data. Most of the time, a piece of a patient record needs to be observed in different formats, such as a graph, a document, or an analysis, and offered to various venues like clinics, hospitals, and pharmaceutical facilities. The ALLOY platform can transform this data in real time into various formats and update them to the cloud. More importantly, during the transformation process, the platform creates an analysis automatically based on its data analysis system, which provides customers the most useful information and helps them better understand the patient records, thereby helping the customers make the most correct decisions.

Liaison Technologies is a company that provides cloud-based solutions to help organizations integrate, manage, and secure data across the enterprise [1]. One of the most widely used solutions they offer is to deal with health data in the healthcare and life sciences industry by using the MapR Stream system. With the implementation of MapR Stream, they have already solved two big difficulties of big data - the data and the system; it can meet HIPAA compliance requirements and the proliferation of data formats and representations [1].

## 7 FUTURE OF HEALTHCARE

Today, the health care industry is driven by more and more medical data, and in the future, the data will be even more complex, as the life science and medicine fields improve. As this paper mentions in Section 3, the data in the healthcare industry has not been stored in a uniform format, and the complexity of the health data is also another obstacle that complicates the data usage in the healthcare industry. Unfortunately, as the population on the earth grows, more and more health data will appear, with more complicated and potentially unusable data structures. It adds more difficulties to using health data for analysis purposes. Furthermore, the cost of collecting health data, maintaining integrity of the database, and correcting imperfect data (such as missing and false values) will be greater. All of those trends apply to different healthcare systems, since the health data for people across the world is pretty much the same [21, 22].

Despite the idea that the data for healthcare fields will still be complicated and chaotic,the technologies in the distribution system, cloud computing, and the IoT will help remedy the shortage of usable health data. In the future, the distribution system used for healthcare service will includes multiple kinds of devices, such as wearable devices that collect the heath data directly from patients and update the data to the database directly. The IoT could be an important factor that helps remedy the imperfection of the health data by collecting the data in a uniform format (it depends on what the healthcare system needs), which significantly cuts the extra costs of collecting data and modifying that data [14, 22]. The better the data quality is, the more precise dataanalysis results healthcare providers will have.

More that the health data, cloud computing, and IoT that serve the general public independently, in the future, hospitals could use these technologies together to build a concrete, multi-purpose healthcare service system. A hospital is a giant database that can collect the data directly from patients, so hospitals could collect data by themselves and then use them to design treatments for each patient, or use that data for research. In the Mayo Clinic, in Minnesota, doctors could use health data to design special treatments for each patient and use advanced AI base study models to modify the treatment. Additionally, the medical researchers from the Mayo Clinic use the health data from patients to study diseases, which accelerates the pace at which they can find methods of curing certain diseases with firsthand data from patients [23]. In the future, the public and private hospitals could use the same mechanism of collecting and using data to provide better healthcare services to patients.

In addition to the data and technologies changing the healthcare field today, the brain science field will also change the machine learning model in the future. As people learn more about the brain, a better study model will be created, which helps the neural networks of some machine learning models to become more advanced. The improvement of brain science will give machine learning models a promising avenue for classifying and predicting jobs in every field, which also includes the healthcare field. Healthcare services or systems could use the more advanced study models to make decisions and create better health plans [24].

## 8 SUMMARY

This paper talks about how Big Data shapes the healthcare field today, and it frames the health data as the fundamental element in the healthcare field, including how traditional data is different from the modern digital data and what issues in health data are still problematic. The summary of the health data that is used today has the following properties:

- The health data will be more and more predominantly digital. The traditional health data will be transferred to digital data with the digital healthcare service growth.
- Data that is used in the healthcare field is complex, and given that more health data will be added, the volume and the complexity of data will make this health data harder to use.
- The health data could be less integrated due to many reasons: for example, the missing data could be caused by the patient being unwilling to offer it because of the sensitivity of the data or for security reasons.

Furthermore, the technologies related to the healthcare field and Big Data, such as IoT and cloud computing systems, have been introduced to provide better healthcare services. Those technologies help remedy the shortage of health data. For example, IoT could collect data directly from patients with a well-formed data format, which avoids creating missing data. Additionally, the cloud computing systems and distribution system could be used for storing and processing a massive amount of data. The development of Big Data technologies has brought the following improvements to the health care field:

- IoT: gives the possibility of collecting real-time data directly from patients and makes better use of the health data. The real-time data updating might compromise patients' privacy, but for providing better services and preventing emergency health problems, this is a sufficient method. In the future, the IoT devices will become more suitable for wearing.
- Cloud Computing and Distribution System: Cloud computing and distribution system offer the computing power of collecting and processing massive amount of data, which is supporting the current and future healthcare field. The learning model that used in the computing system is the core that helps people to make the rational healthcare plan.

In this paper, some instances of healthcare applications and companies are mentioned, those instances give people a great idea of how the healthcare services serve people from different aspects, for example, using data analysis to make healthcare plan, or predicting diseases from DNA and so on. Additionally, those instances are also provide people a more advanced ideas that how the healthcare should improved in the future.

In conclusion, the Big Data concepts and technologies will be applied more in the healthcare fields with advanced technologies, formatted database and IoT devices that gives people conveniences for planning their healthcare. The optimized views of healthcare comes from the massive investment on making data more meaningful to people. With the improvement in Big Data related technologies, the healthcare fields will provide better services to people for maintain good health.

## 9 ACKNOWLEDGMENTS


## ACKNOWLEDGMENTS

The authors would like to thank Dr. Gregor von Laszewski for his support and suggestions in writing this paper, as well as Nicole DiPaolo for copyediting the final draft.


## REFERENCES


[1] [n. d.]. 5 Big Data Production Examples in Healthcare — MapR. ([n. d.]). https://mapr.com/blog/5-big-data-production-examples-healthcare/

[2] [n. d.]. 7 Big Data Use Cases for Healthcare. ([n. d.]). http://www.ingrammicroadvisor.com/data-center/7-big-data-use-cases-for-healthcare

[3] [n. d.]. BBC - Intermediate 2 Bitesize Geography - World Population Distribution : Revision. ([n. d.]). http://www.bbc.co.uk/bitesize/intermediate2/geography/human{_}environments/world{_}population{_}distribution/revision/1/

[4] [n. d.]. Big data in health care — SAS. ([n. d.]). https://www.sas.com/en{_}us/insights/articles/big-data/big-data-in-healthcare.html

[5] [n. d.]. Building a Real-Time Streaming ETL Pipeline in 20 Minutes - Confluent. ([n. d.]). https://www.confluent.io/blog/building-real-time-streaming-etl-pipeline-20-minutes/

[6] [n. d.]. Canada's Health Care System - Canada.ca. ([n. d.]). https://www.canada.ca/en/health-canada/services/health-care-system/reports-publications/health-care-system/canada.html

[7] [n. d.]. Data Analytics for Employee Benefits — Nuna. ([n. d.]). https://www.nuna.com/employers/

[8] [n. d.]. Data-driven healthcare organizations use big data analytics for big gains. ([n. d.]). https://www-03.ibm.com/industries/ca/en/healthcare/documents/Data{_}driven{_}healthcare{_}organizations{_}use{_}big{_}data{_}analytics{_}for{_}big{_}gains.pdf

[9] [n. d.]. Federal Data Solutions for Medicaid and Medicare — Nuna. ([n. d.]). https://www.nuna.com/government/

[10] [n. d.]. Fitbit Official Site for Activity Trackers & More. ([n. d.]). https://www.fitbit.com/home

[11] [n. d.]. Health Care — Definition of Health Care by Merriam-Webster. ([n. d.]). https://www.merriam-webster.com/dictionary/healthcare

[12] [n. d.]. How Big Data Is Changing Healthcare. ([n. d.]). https://www.forbes.com/sites/bernardmarr/2015/04/21/how-big-data-is-changing-healthcare/{#}7d93d5028730

[13] [n. d.]. How Stream-First Architecture Patterns Are Revolutionizing Healthcare Platforms — MapR. ([n. d.]). https://mapr.com/blog/how-stream-first-architecture-patterns-are-revolutionizing-healthcare-platforms/

[14] [n. d.]. Is Digital Health The Future Of Healthcare? ([n. d.]). https://www.forbes.com/sites/quora/2018/04/04/is-digital-health-the-future-of-healthcare/{#}557ef1292556

[15] [n. d.]. Kaiser Permanente's EHR System Healthcare Economist. ([n. d.]). http://healthcare-economist.com/2010/12/20/kaiser-permanentes-ehr-system/

[16] [n. d.]. MapR Guide to Big Data in Healthcare. ([n. d.]).

[17] [n. d.]. Our Science - Helix. ([n. d.]). https://www.helix.com/our-science/

[18] [n. d.]. Public Key Encryption. ([n. d.]). https://www.tutorialspoint.com/cryptography/public{_}key{_}encryption.htm

[19] [n. d.]. The 3Vs that define Big Data - Data Science Central. ([n. d.]). https://www.datasciencecentral.com/forum/topics/the-3vs-that-define-big-data

[20] [n. d.]. The Data Pipeline fi?! Analytics at the Speed of Business - Dataconomy. ([n. d.]). http://dataconomy.com/2017/04/data-pipeline-analytics-business/

[21] [n. d.]. The Future of Healthcare — Roswell Park Comprehensive Cancer Center. ([n. d.]). https://www.roswellpark.org/partners-practice/white-papers/future-healthcare

[22] [n. d.]. The future of healthcare: Finding the opportunities that lie beneath the uncertainty — McKinsey & Company. ([n. d.]). https://www.mckinsey.com/industries/healthcare-systems-and-services/our-insights/the-future-of-healthcare-finding-the-opportunities-that-lie-beneath-the-uncertainty

[23] [n. d.]. THE NEXT MAYO / REMAKING A MEDICAL GIANT MAYO SEEKS TO DOMINATE WITH DATA. ([n. d.]). https://www.mayoclinic.org/documents/mayo-seeks-to-dominate-with-data/doc-20139628

[24] [n. d.]. THE NEXT MAYO / REMAKING A MEDICAL GIANT MAYO SEEKS TO DOMINATE WITH DATA. ([n. d.]). https://www.mayoclinic.org/documents/mayo-seeks-to-dominate-with-data/doc-20139628

[25] [n. d.]. The People Powering IoT. ([n. d.]). https://cdn2.hubspot.net/hubfs/338908/Doc/Livongo{_}Case{_}Study.pdf?t=1488557837557

[26] [n. d.]. The Politics of Healthcare - Hillsdale College. ([n. d.]). https://www.hillsdale.edu/educational-outreach/free-market-forum/2008-archive/the-politics-of-healthcare/

[27] [n. d.]. The U.S. Health Care System: An International Perspective - DPEAFLCIO. ([n. d.]). http://dpeaflcio.org/programs-publications/issue-fact-sheets/the-u-s-health-care-system-an-international-perspective/

[28] [n. d.]. U.S. Health Care Industry Statistics and Market Size Overview - Industry Statistics - Plunkett Research Ltd. ([n. d.]). https://www.plunkettresearch.com/statistics/Industry-Statistics-US-Health-Care-Industry-Statistics-and-Market-Size-Overview/

[29] [n. d.]. Which Healthcare Data is Important for Population Health Management? ([n. d.]). https://healthitanalytics.com/news/which-healthcare-data-is-important-for-population-health-management

[30] [n. d.]. World Population Clock: 7.6 Billion People (2018) - Worldometers. ([n. d.]). http://www.worldometers.info/world-population/
[31] 2014. Understanding the Internet of Things (IoT). (2014). https://www.gsma.com/iot/wp-content/uploads/2014/08/cl{_}iot{_}wp{_}07{_}14.pdf
[32] Andrea Califano, Arul Chinnaiyan, Geoffrey M Duyk, Managing Director, Sanjiv S Gambhir, D K Ludwig Professor Director, Tim Hubbard, David J Lipman, Lincoln D Stein, Jean Y Wang, Executive T Secretary Olivia Bartlett, and Claire L Harris. [n. d.]. Ad hoc Working Group on the NCI Cancer Biomedical Informatics Grid (caBIG ® ) Program. ([n. d.]). https://deainfo.nci.nih.gov/advisory/bsa/archive/bsa0311/caBIGfinalReport.pdf
[33] Josh Chang, Felix Peysakhovich, Weimin Wang, and Jin Zhu. [n. d.]. The UK Health Care System History of UK Healthcare System. ([n. d.]). http://assets.ce.columbia.edu/pdf/actu/actu-uk.pdf
[34] Weill Cornell Medical, Harvard TH Chan Brad Chapman, Amos A Folarin, Richard JB Dobson, and Stephen J Newhouse. 2015. Open Peer Review NGSeasy: a next generation sequencing pipeline in Docker containers [version 1; referees: 3 approved with reservations]. (2015). https://doi.org/10.12688/f1000research.7104.1
[35] Britt Cyr, Webb Horn, and Daniela Miao. [n. d.]. Security Analysis of Wearable Fitness Devices (Fitbit). ([n. d.]). https://pdfs.semanticscholar.org/f4ab/ebef4e39791f358618294cd8d040d7024399.pdf
[36] Corporation EMC and IDC. 2015. Vertical Industry Brief: Digital Universe Driving Data Growth in Healthcare. (2015), 16. https://doi.org/10.1002/ejoc.201200111 arXiv:arXiv:1011.1669v3
[37] Choong Ho Lee and Hyung-Jin Yoon. 2017. Medical big data: promise and challenges. *Kidney research and clinical practice* 36, 1 (mar 2017), 3–11. https://doi.org/10.23876/j.krcp.2017.36.1.3
[38] Dan LeSueur. [n. d.]. 5 Reasons Healthcare Data Is Unique and Difficult to Measure. ([n. d.]). https://www.healthcatalyst.com/5-reasons-healthcare-data-is-difficult-to-measure
[39] Izet Masic, Milan Miokovic, and Belma Muhamedagic. 2008. Evidence based medicine - new approaches and challenges. *Acta informatica medica : AIM : journal of the Society for Medical Informatics of Bosnia & Herzegovina : casopis Drustva za medicinsku informatiku BiH* 16, 4 (2008), 219–25. https://doi.org/10.5455/aim.2008.16.219-225
[40] Douglas McCarthy, Kimberly Mueller, and Jennifer Wrenn. 2009. Case Study Kaiser Permanente: Bridging the Quality Divide with Integrated Practice, Group Accountability, and Health Information Technology. *Commonwealth Fund* 17, June (2009), 1–45. http://www.commonwealthfund.org/{~}/media/Files/Publications/CaseStudy/2009/Jun/1278{_}McCarthy{_}Kaiser{_}case{_}study{_}624{_}update.pdf
[41] José Moura and Carlos Serrão. [n. d.]. Security and Privacy Issues of Big Data. ([n. d.]). https://arxiv.org/pdf/1601.06206.pdf
[42] Nithin and Ashish. 8720. Evaluation of Apache Kafka & Redis as a potential backend for Ascoltatori & Ponte to enable highly resilient & fault tolerant pub-sub mechanism. (8720). http://www.ics.uci.edu/{~}cs237/ProjectSlidesSpring2016/15.pdf
[43] Chandan K Reddy and Jimeng Sun. 2013. Big Data Analytics for Healthcare. (2013). http://dmkd.cs.wayne.edu/TUTORIAL/Healthcare/
[44] M K Ross, W Wei, and L Ohno-Machado. 2014. "Big data" and the electronic health record. *Yearbook of medical informatics* 9, 1 (aug 2014), 97–104. https://doi.org/10.15265/IY-2014-0003
[45] Cloud Standards Customer Council. 2017. Impact of Cloud Computing on Healthcare Version 2.0. (2017). http://www.cloud-council.org/deliverables/CSCC-Impact-of-Cloud-Computing-on-Healthcare.pdf
[46] Cloud Standards Customer Council. 2017. Impact of Cloud Computing on Healthcare Version 2.0. (2017). http://www.cloud-council.org/deliverables/CSCC-Impact-of-Cloud-Computing-on-Healthcare.pdf
[47] Walter G Suarez. 2013. An Overview of Health IT @ Kaiser Permanente NIST Health IT Symposium Series fi?! Gaithersburg, MD. (2013). https://www.nist.gov/sites/default/files/documents/healthcare/KP-Health-IT-OVERVIEW-Dr-Suarez-April-2013.pdf
[48] Yuelian Sun, Hans Gregersen, and Wei Yuan. 2017. Chinese health care system and clinical epidemiology. *Clinical epidemiology* 9 (2017), 167–178. https://doi.org/10.2147/CLEP.S106258
[49] Paul C Tang and Clement J Mcdonald. [n. d.]. Electronic Health Record Systems. ([n. d.]). http://eknygos.lsmuni.lt/springer/56/447-475.pdf

# CMENV: Deployable Cloudmesh Containers

Tim Whitson
Indiana University
Smith Research Center
Bloomington, IN 47408, USA
tdwhitso@indiana.edu

Gregor von Laszewski
Indiana University
Smith Research Center
Bloomington, IN 47408, USA
laszewski@gmail.com

## ABSTRACT

In order to meet the needs of modern cloud computing environments, there is a need for flexible, language-independent computing resources. To meet this need, we present *cmenv*, a modularized system of REST APIs housed within easily-deployable Docker containers. Each individual API stands alone within the program and we discuss the structure for APIs. A custom resolver is created to route HTTP requests back to the API controllers. Various REST services are given as standard for maintaing the program.

## KEYWORDS

hid-sp18-526, Volume: 9, Chapter: REST, Status: 100.
cloudmesh, rest, swagger

## 1 INTRODUCTION

Docker is a program that provides containerization (virtualization within the operating system). Docker is quickly becoming standard for software development and deployment within the cloud. According to the 2016 Docker Survey, 60% of users deploy Docker as part of their "cloud strategy"[1].

REST APIs are another crucial cloud component. In fact, many important software programs, such as Kubernetes, come with REST APIs built-in. An important benefit of REST APIs is that they are language-independent. All web languages have methods for interacting with, and creating, REST APIs. Therefore, REST APIs play a crucial role in *cmenv*.

We combine all required services into a single REST API that can be deployed easily within a Docker container. Each API stands on its own as a modular system.

## 2 OPENAPI

It is important, especially when working with modular systems, that all components of the system adhere to the same standard. For this project, we use OpenAPI. According to their website, OpenAPI "defines a standard, programming language-agnostic interface description for REST APIs, which allows both humans and computers to discover and understand the capabilities of a service without requiring access to source code, additional documentation, or inspection of network traffic."[2]

The specific OpenAPI standard we use is Swagger 2.0. Swagger 2.0 allows for simple resource definitions. We will not require any operation handling, as all routing will be done by a custom resolver, discussed later.

## 3 CONFIGURATION

Each instance of *cmenv* uses its own configuration file, *config.yml*. This is due to the fact that the containers need to be composable. Currently, the configuration allows for the selective use of services. Either a list of services or "all" can be provided in the configuration to determine which services are running. The container will then run only the APIs which are listed. Currently, the setup also requires the *Dockerfile* to be in the install directory.

In order to launch the program, the following directory structure must be adhered to:

```
.
|-- config.yml # CMENV configuration
|-- Dockerfile # Dockerfile
```

- **config.yml** This file contains the configuration for *cmenv*, such as which services will be run.
- **Dockerfile** This Dockerfile is provided. It should not be modified except in special circumstances.

## 4 PROGRAM FUNCTIONALITY

The first role of the program is to generate the Swagger specification. In order to do this, all requested services must be gathered. Then the program combines the paths of each module from its *swagger.yml* specification. After all services are combined, a final Swagger specification is created with required fields (title, version, etc.). This document is fed into *connexion*. The combined Swagger document can also be output as either yaml or json.

### 4.1 connexion

connexion is a Python program written by Zalando SE to combine OpenAPI and Flask [4]. Effectively, connexion takes the place of Swagger Codegen. It accepts a Swagger specification and creates a Flask server with pre-generated routes. Typically, with connexion, the user would have to specify the controllers manually or allow the connexion *resty resolver* to automatically route the requests. However, due to the modularity of *cmenv*, we will create our own router. See the Routing section for more information.

## 5 API STRUCTURE

Because *cmenv* is a composable system, each part of the API should be independent. Therefore, we use a modular system of self-contained APIs, which are then combined into a single server. This allows for a "plugin" (and plug out) system where new API modules can easily be created and added to the program.

Directory layout:

```
.
|-- swagger.yml # Swagger specification
|-- requirements.txt # python requirements
|-- packages.txt # Ubuntu package requirements
|-- controllers # dpython controllers for API
```

```
|-- ...
```

- **swagger.yml** This file contains the Swagger specification, for this API only (paths, definitions, etc.). The top-level Swagger specification (Swagger version, API title) are handled by the main program.
- **requirements.txt** This file is a newline-separated list of python package requirements, to be installed via *pip3 install* (currently all packages use Python 3).
- **pakackages.txt** This is a newline-separated list of Ubuntu package requirements, to be installed via *apt-get install*.
- **controllers** This directory is where the API controllers are placed, with name corresponding to the path. Each file in this directory is a Python file which acts as a controller, defined by the path names from swagger.yml. See Routing section for more information.

Another reason for following such a structure is to properly manage dependencies. Each API has its own dependencies, both in the operating system and in the programming language (Python). With a modular setup, individual dependencies can be assigned and handled by the program. Therefore, no master list of dependencies is required, and individual modules can be installed or uninstalled along with their dependencies.

## 6 ROUTING

To fit with the modularized system of APIs, a custom router, or resolver, is used. The router needs to be able to find each individual API, and call the controllers located within the API. First, a one-to-one mapping of modules to paths is created, to ensure that the controller can refer to the original module. Then, the method is placed as the endpoint. In keeping with the connexion package standard, a GET request to the base path is referred to as "search".

For example, if in the *store* module, we have the path "/key" and method "GET", the request is routed to apis.store.controllers.key.search. This leads to the *store* module, the "controller" subdirectory, the "key.py" file, and the "search" function.

Example using *store* module:

```
paths:
  /key:
    get:
      # route: apis.store.controllers.store.search

  '/key/{key}':
    get:
      # route: apis.store.controllers.store.get
    delete:
      # route: apis.store.controllers.store.delete

  '/key/{key}/{value}':
    put:
      # route: apis.store.controllers.store.put
```

## 7 APIS

The purpose of *cmenv* is to provide an arbitrary number of services. However, a few services come packaged. These services, which are outlined here, are crucial to maintaining the operating environment.

### 7.1 Key-Value Store

A Key-value store is implemented, using TinyDB. TinyDB is a simple, lightweight local file store written in Python. Essentially, it acts like a local version of MongoDB. Each item is stored as a json document, and all documents for a "db" are combined into a json file. These files are read in and out of Python and easily dumped or loaded as dictionaries. This solution provides an efficient, queryable file store with no dependencies on the operating system.[3]

### 7.2 Services

A services library is necessary to manage the API modules. The library handles not only running services, but the management of services as well. This will be useful for changing the initial configuration parameters after the container has been created. The API can talk to the program and turn off and on whichever services are requested (requiring a restart).

## 8 FUTURE IMPROVEMENTS

This project is a work-in-progress. Many more changes and adjustments will need to be made for full implementation. In this discussion we discuss the future of the project.

### 8.1 Core APIs

More core APIs need to be added for maintenance and functionality. The following core APIs will be added in the future:

- **key management** This API will manage SSH/public keys.
- **accounting** This API will enable accounting of services and resources. Also included will be a logging system.
- **data services** This API will manage a file and object store. There will also be virtual files and virtual objects. This service might also benefit from TinyDB.

### 8.2 Service Monitor

A service monitor would be an important asset for the combined APIs. Such a monitor could run in a browser and have the capability to start/stop services and generate URLs or parameters. An HTML template could be created using Jinja and Flask to communicate with any running API.

### 8.3 Remote Setup

A remote setup option is possible, without requiring a local directory, Dockerfile, or configuration file. However, This setup would be a more standardized setup, with all services running. The user would be required to manually configure the setup through the *services* module.

## 9 CONCLUSION

*cmenv* is a modern solution to a modern problem. We are able to deploy a container of modularized and combined REST APIs easily and efficiently. We use connexion and OpenAPI specifications to create a combined specification of all services running on a Flask server. We create a custom router to route HTTP requests to the modules. There is still more work to do on *cmenv*, which has been outlined in the paper. However, we believe that *cmenv* has a solid

foundation moving forward to simplifying distributed and cloud computing problems.

## REFERENCES


[1] Docker. 2016. The Evolution Of The Modern Software Supply Chain. (2016). https://www.docker.com/survey-2016

[2] OpenAPI Initiative. 2016. OpenAPI. (2016). https://www.openapis.org/

[3] Markus Siemens. 2016. Welcome to TinyDB! (2016). http://tinydb.readthedocs.io/en/latest/

[4] Zalando SE. 2015. Welcome to Connexionfis documentation! (2015). http://connexion.readthedocs.io/en/latest/

**CONTENTS**

# Report: Big Blockchains

Lokesh Dubey
Indiana University
3209 E 10th St
Bloomington, IN 47408, USA
ldubey@indiana.edu

Gregor von Laszewski
Indiana University
Smith Research Center
Bloomington, IN 47408, USA
laszewski@gmail.com

## ABSTRACT

Blockchain and specifically cryptocurrencies have been trending high and have been buzzwords in recent past. In particular, Blockchain and its popularity have been mostly tied with cryptocurrencies and all the related entities that comes into picture with any financial transaction, traditional or contemporary. Most of this attention remains channelized towards cryptocurrency and focuses primarily on the challenges of implementations and ways to overcome the limitations that may be subjected to cryptocurrency. In this study we do a detailed introspection on how Blockchain technologies are being used not just with cryptocurrencies but, also, how it is being applied into various other solutions where we treat Blockchain as open, distributed, peer to peer transacting, database. As, Blockchain is simply agnostic to any currency and can be applied to multiple other solutions which are not even closely related to a financial system. Further, we discuss that implementation of non financial systems can result into very large number of transactions and blocks to be created than it could ever be the case in any cryptocurrencies. And that would throw many more implementation and operational challenges at this new technology along with the very large amount of data being generated in form of Big Blockchain, Big Data in Blockchain.

## KEYWORDS

hid-sp18-702, Volume: 9, Chapter: TBD, Status: 0.
Blockchain, Transaction Management, Decentralization, Middleman, Digital Autonomous Organizations

## 1 CURRENT ECONOMICS AND TRANSACTION MANAGEMENT IN SUPPLY CHAIN

There have been extensive technological strides in most of what human lifestyle interacts with on day to day basis. One can pick up any product or entity around them and can always relate to the historical advancements that have been done on it. It has been human nature always to keep improving on the lifestyle and technological advancements play a vital role in it. Most of these advancements in one form of another have been about Internet of information. From century old abacus, to an electronic calculator, to high performance computers is just one example of it [41]. In everything around us, one can find, that there have been smarter solutions developed and the basic principle of any system has been altered to make the devices smarter to make the efficient use of time and resources. Most of the scenarios eventually find solutions in information technology in unison with electronic advancements. And as the name suggests information technology is all about information and digitizing every single data that is around us and using it to its fullest extent so that one can increase the efficiency of the process and also, in the meanwhile, make the systems smarter by artificial intelligence. However, all of these advancements, for instance, self driving cars, smart grocery shops, or artificial intelligence, have all been the result of Internet of information [41]. Everything that is being done is mostly in form of making copies of the physical information that we have at hand and making an impression of it on internet. It has always been about digitizing the information and passing it around.

One particular system, however, is extremely vital for the humans but has not progressed a lot in principle [35]. Transaction management and accounting of any form, even if digitized, is still using the old methodologies and there is very little advancements done on it is principle. For example. We can indeed pay for our Uber (a Smartphone-Enabled Ride Hailing Service Alternative to Taxi Cabs) car ride directly to the driver but eventually its all about transfer of a certain asset between two parties [21]. In principle it indeed is just an alternative of giving cash to the cab driver vs paying it via smart phone app. And that includes a lot of intermediaries who are doing the job of transferring the asset, in digitized form, to all the entities involved and then depositing appropriate cash in the drivers bank account. Even though it appears that this system works seamlessly while enforcing a cashless economy if we look at it from a principle that this is based upon, it remains the same. Eventually we are still just transferring an asset, which is digitized, from one party to another. Comparing this to the old system where you pay the cab driver in cash one just has the comfort of using his phone to call a cab and pay for it without worrying about cash. But eventually one is just transferring cash. In fact to make this more beneficial one has introduced a lot of intermediaries and entities in this complete transaction which adds too much time for this process to be taken care of [41]. When there are so many moving parts to a transaction it will deter its performance. As an end result the Uber driver who could have gotten the payment right away will get his payment delayed or may be at the end of the month. Here are some use cases that illustrate the problems with traditional transaction management in case of supply chain and in the financial paradigm.

### 1.1 Brexit

Brexit is a term for the potential or hypothetical departure of the United Kingdom from the European Union. Brexit will result into extremely complex trade predicament which would be put in place after United Kingdom's exit from European Union [7]. In a very small amount of time borders and customs will have to be setup between these geographies in a very short amount of time. The traditional financial system, no matter, how much use of technology it uses would either fail or would be extremely inefficient to provide any timely transactions at such large level. Merely because of the

fact that an establishment which was so old and working needs to be changed starting from a certain day where there are lot of intermediate parties involved and many of them would be created viz. regulatory bodies, logistical entities.United Kingdom's Brexit team has however suggested a completely novice technology based solution in form Blockchain to address this issue [7].

There have been multiple and mixed mentions of Blockchain in finding resolutions to these complex problems in the UK. There are many organizations which are encouraging the use of Blockchain and Hyperledger technology a lot to find solutions to transform global shipping sector. Hyperledger is an umbrella project of open source blockchains and related tools. Biggest proponents of these technologies have been IBM and Maersk [40]. IBM has started a joint venture with Maersk and according to them they can get the infrastructure ready if not complete but at least ready for this transition in a period of six months. As Blockchain provides a safest and quickest way to handle these supply chain models which keeping track of commodities and assets without any misuse, with all the possible right protections of tangible and intellectual goods these technologies can prove to be really helpful.

There are however some difficulties and problems which have been identified with applying Blockchain technology solution to these issues as well. These flows which have to be applied for the custom border transaction management and the smart contracts to be developed for these technologies might be far difficult to implement than any typical supply management problem. Even though the joint venture of IBM and Maersk have committed a certain deadline to achieve this but it does not seem like a reality. On top of the complexity of the problem at hand the will to take these risks and investing infrastructure and resources on these problems to solve something which already has a tested solutions, but of course an inefficient solution, would be difficult. It has to be British and Irish governments both who have to agree on going ahead with this implementation and have to take the risk of going ahead with this new buzzword technology which may or may not work. It is simply not possible to provide any kind of accountability for the solutions which cab be applied using Blockchain because simply there is not much information available to even visualize what is being tried. Secondly, the technology itself is novice and has only been applied to simple Hello world solutions and cryptocurrencies only like Bitcoin, Ethereum. However, those transactions are fairly straightforward as there is still a lot of skepticism about that as well. Which it the reason why not a lot of people have been adopting these technologies.

Lot of governments have been in fact banning the circulation and usage of cryptocurrencies because they are simply not ready for it [32]. There are multiple factors that apply to the readiness however. Some can be attributed to the ignorance and typical lack of knowledge of the governments, some are because they actually understand that these currencies can take away a lot of control over these pseudo currency on the internet which does have an affect on real world.

## 1.2 Information is duplicated, not commodity

Internet of information (things), has revolutionized the impact of technological advancements on human race. The impact of this information on humanity can be imagined sheerly by the fact that now oil is not considered as the most valuable resource, but, its the data. However, most of this information is just replicas [18]. Any data we can think of that is on the internet is a duplicate or a copy of something in real world. All the information that we have on internet is either a blatant copy of something in real world. For example, textual information extracted from physical documents and is saved as files in internet, or a song which was sung by an artist and is uploaded as an mp3 file. Or on the other hand it could be the information that is derived from this real world information and has not significance of its own. For example, Facebook has around 2.2 Billion active users monthly. Who are sharing a lot of information in form of photos which are theirs, conversing with friends which are their real thoughts, and liking, disliking posts on Facebook which is an information which is derived by their action on this different information posts [39]. So as far as its information it can be digitized, made multiple copies of, and can be shared across internet. This, however, does not apply to commodity.

On the other hand there can be some commodities which can be considered as commodity and are also digital. Some examples of such commodities are intellectual property of any user. This can actually be considered in various different perspectives. There can be some organizations who are basically providing nothing but some tutorials which are not really created in real world but are actually the write ups coming from many experts or experience people who have actually worked on those products and know what needs to be done with them. Lets consider Quora (a question-and-answer site where questions are asked, answered, edited, and organized by its community of users), it is a social networking site however it has a very different objective than any other major sites like Facebook, Twitter. Quora is very similar to an old site Yahoo Answers but that site is literally to get answers for any random question that one can imaging and ask. Quora on the other hand is a site where a lot people collaborate, socialize with people intellectually. In most cases we consider it like an online book club but with millions of users to it. There are many users on these site who write up really nice articles on this site which are essentially a commodity. These writeup are even published and cited by many publications, news papers.

But in this case a well the matter which is being generated online is completely open to all the users of quora because there is no content on Quora that is hidden. Its just an intellectual property of a Quora user and if the wishes to she should be able to charge royalty for any usage of such property and most important should have control over any illegitimate and abuse of the property by online users knowingly or even unknowingly.

Unknowingly, because internet is nothing but the users essentially jumping from one url to another without even knowing what is about to appear on their screen [20]. The same hyperlink basically may take you to a random site which is allowing a lot of ads to make revenue. However those ads are nothing but url to some other sites which most likely may be taking you to sites which are offering movies and songs which are freely available or are simply pirated copies. In these kinds of scenarios it makes it extremely difficult to figure out who was actually consuming the content knowingly and was actually looking for it. Or the person was simply driven into this mesh of hyperlinks because of which the user is here.

There are indeed a lot of ways to tackle these problems but the most any government of the copyright protection boards can do is shutdown the website. They cannot really charge them or hold them accountable for it because they were simply providing links to their ad providers and did not intend to do that. These kind of loop holes in the existing system makes it extremely difficult to keep the commodity, which indeed exists on the internet in a virtual form, secure.

## 2 ASSET

Commodity, on the other hand, cannot be really digitized and eventually cannot be shared. For the sake of simplicity lets just consider our commodity of concern for sometime as currency. This particular theory, whereas, can be applied on any commodity as well. So there have been a lot of technological advancements done in case of banking in all this time as well and we have moved from the age old barter system, to having a currency which basically means the government of one's country owes the bearer of the currency the amount of money mentioned on the currency. From direct handling and exchanging of cash there were more advancements made in the case of banking where the currency was indeed pseudo digital [9]. Yes, for a certain period of time it felt like the currency is digital and you can exchange it and purchase real world things with it, but it was not really digital. In this case as well, just like information, the currency was indeed a replica of a real world currency sitting in internet. On one hand indeed the transaction that was carried out was digital but at the end of the someone else did the physical transfer of money in any scenario. We can consider the same example as described before where one can pay the uber driver with an app and do not use cash at all. But indeed there is physical money that exists in the real world which is exchanged at a different level, in this case federal banks.

Now, lets take another look at this and flip the currency with a commodity. Let us take an example of Youtube, a free video sharing website that makes it easy to upload, watch online videos. Youtube creator space and advertisement model. In current model there are lot of intermediaries involved in the complete payment model [29]. For example there are advertisers, youtube organization, viewers, youtube creators, subtitle providers and many more. Here the commodity is basically a youtube video in form of a song or DIY (Do it yourself). which is uploaded on Youtube. Now before the artists get paid there are a lot of unknowns and a lot of hops that the money takes to eventually reach to the actual creator of the content. Moreover with increasing competition from fellow youtube content creators even though we have digitized everything and it has been made easier for any youtube user to have very wide and quick outreach to a lot of viewers in form of youtube platform but the artists are getting a lot less paid than the traditional copyright model [27, 41].

As we can see in most of these scenarios there are a lot of flaws in the system which makes it extremely inefficient. We can see that it is indeed a natural progression of research to make life easier and make the systems more efficient. But in most of these cases everything is digitized from real word to information on internet. But in almost all of the cases the underlining fundamental principle remain the same. We do see some benefits of it because its easily accessible over internet and has a much wider reach than it had every before but still there are a lot of inefficiencies in the system which needs to be addressed.

## 3 CENTRALIZED INTERMEDIARIES

The biggest problem with the traditional transaction management systems or financial institutions operating over internet are the middlemen [42]. Here a middleman is some who exists in practically every transaction that occurs in a supply chain model or any financial transaction that happens in banking sector, stocks & bonds or it can simply be a tap a consumer does with his Visa card on Walmart. In fact, the middleman existed right from the days of barter system where if the commodity you need is with the person who is not interested in your commodity can be solved simply by involving a middleman who charges a certain amount and exchanges the commodities for you and stores what is in excess with him to be utilized for a similar transaction later. In today's world however this can be considered as entities like Banks, National Security Agency (NSA), Google which in most cases are centralized. As described before in our previous example of the Uber cab driver payment we listed numerous intermediaries down ranging from the app provider, banking institutions, regulatory authorities. Let us take one more example of remittance of currency to another country. In today's highly developed and technologically advance system as well it takes at least 5-6 approximately to send money from one country to another [16]. There are many intermediaries in this scenarios like the sender, sender's bank, sender's remittance bank, federal bank, authority from sender's country, federal authority from receiver's country, receiver's remittance bank. And even though highly digitize these all intermediaries are still a bottleneck in making these transactions efficient and faster, because of the sheer fact that they bring any transaction to a centralized ecosystem and it can be validated, rejected, updated only from that centralized system.

Blockchain as a concept and the related technologies are primarily taking out the middleman out of the picture, inherently [28]. As the blockchain concept in itself is based on the fact that its peer to peer transaction, which can be regulated by regulatory authorities but it would not be governed, approved or moved by the middleman. As blockchain technologies and the principle behind it in form of a distributed ledger basically takes away any kind of need of a central authority or an approver to complete the transaction. One of the postulates of these kinds of establishments later would be that because this is a peer to peer transaction the participants can decide the amount they need to charge or can actually get to agreement faster, with mutual consent, and most importantly whenever they want to.

Middleman in most cases is not really the person who is even in the transaction. A transaction, if we juxtapose it with Barter system, is more or less necessary just to involve two or three parties who really wish to exchange a particular commodity with another. One commodity in most of the cases would be financial asset like currency. But in most case there are two person involved in any transaction where one is the provider and has either excess of a certain commodity, or grows that commodity and makes a living out of it by selling it, or is planning to get rid of that commodity for

some reason. And the other person involved in these transactions is the one who is actually in need of this commodity. But where does the middleman come into this picture.

Here middleman is basically a finder, minder and grinder. There are many issues when these commodities are to be exchanged. First of all there has to be a way to connect these two participants who are trying to exchange commodity but there is no mutual channel between them so that they can share that information. Second the commodity which is being sold may too much or too less for the other participant who needs it. In whcih this same transactions becomes a little more complicated with two participants, who are to be found first, will provide the commodity to just one participant. This is a fairly simple and very basic problem statement in that a middleman helps a lot by taking the risk and finding the participants for a transaction, making sure the appropriate security is maintained and both participants are feeling positive about transactions and are rest assured of any fraudulent transaction.

In any of these scenarios we can find that any transaction related to any commodity when involves a lot of middlemen the system becomes inefficient irrespective of how digital or best the solution is because at the end of the day the commodity has to exchange this multiple hands, even though digitized. And this takes a lot of time and resources. What this showcases is that even though most of the information and commodity digitization has improved the speed of these transactions to a certain extent but in principle they are all following the same traditional real world methodologies and are centralized bottlenecks.

A research paper that was published in 2008 focusses only on one certain aspect of currency called Bitcoin [26]. However, Bitcoin is merely one form of commodity for which there is a Blockchain which is active and people own and transaction in that Blockchain to carry out transaction of their Bitcoins. The principle on which Bitcoin is based is something which is extremely novice and provides solution to all the limitations of the traditional transaction management systems mentioned above and is called Blockchain. Which provides a way to devise decentralized and distributed database that allows peer to peer transactions by removing the middleman, is agnostic to any international boundaries and can be applied to any kind of commodity, asset, intellectual property and not just money.

## 4 BLOCKCHAIN

Blockchain is a distributed database in form of a ledger which is updated with distributed transactions that are secured by cryptography and works based on the consensus of all the parties involved in it [4]. A Blockchain is simply a immutable ledger of various transactions that are stored in that ledger and any transaction can only be added to the ledger after its validated by all the parties involved and is secured by cryptography to avoid any foul play. These ledger in itself consists of various blocks which are a set of transactions which are combined together in form a Block and are appended in an already existing chain of Blocks. To process these transactions and to add any block to existing block chain there is a certain amount of computational work has to be put in to figure out a certain cryptographic key which can validate that next block is the correct block to be appended to the chain. Blockchains are generally open to all on the channel to see and process transactions for all users known as miners.

Rather than delving into the details of what Blockchain is and to remain consistent with scope of this study the focus would here after would be more on the benefits of Blockchain, their applications and how they open new avenues for Big data. At every transformation that Blockchain can bring we will look at each aspect it from both perspectives, monetary and non-monetary both. As, this has been the premise of this study right from start that Blockchain is not just about cryptocurrency or something monetary but it can be applied to any commodity, use case where there are transactions [4].

In this further exploration we will look in various aspects and issues in real world that blockchain technology has more or less uncovered. Some of the major industries and domains which this technology can really help with our the global financial establishment, a genuine true global world economy, while making sure that the security is not compromised. And providing appropriates hooks for all kinds of security auditing so that there is always no question or hesitation in any adopters to be skeptical about the technology and to only focus and invest energy in getting the best out of this technology.

This particular study, right from the start, have been insinuating a lot on not considering Blockchain as a cryptocurrency driven technology. In fact there is a high possibility that Bitcoin like cryptocurrencies which are highly driven by the market are extremely sensitive commodity to be handled by blockchain [2]. However, the sheer concept of distributed database which immutable in itself makes a very good case of, if not cryptocurrencies, blockchain technologies in themselves have a very good future. Many organizations and big supply chain management companies have already started garnering benefits of the available solutions around blockchain. However, there are not at all chances that cryptocurrencies will not survive. Its just the general human tendency to be skeptical about anything related to finance. But still cryptocurrencies have a very bright future as they are and will complement any service that is not centralized and not to rely on any legacy on payment systems.

In the interest of keeping this study scoped and because there is already a lot of work that is mostly done on Bitcoin and cryptocurrencies in the subsequent topics as well we will focus mostly on the real world, non cryptocurrency applications, limitations, challenges of Blockchain [45].

## 5 GLOBAL ECONOMY AND PROSPERITY

As we have established that the Banking sector has been digitized at a very aggressive rate nearly everything in a bank these days is either available through internet and provide easy access to multiple services of a bank over a computer or a mobile phone. As glittery and ambitious it may sound the underlining systems are still following the same old traditional approach to handle any kinds of transactions. And anything in banking security eventually can be related to the identity of the transactions. A bank will require a lot of identifications before anyone can get a bank account [11]. This is one of the major flaws in the current banking and financial systems. Anyone who wishes to be a part of the global economy should have an identity. Even though the lender and the receiver both have no business whatsoever with providing their identities, their address

proofs or even their names to the bank. For a simple transaction like transferring money from one bank account to another, the intermediary here, the bank demands both of the parties to identify themselves. Which simply gets extremely complex if one wishes to work on an asset transaction beyond the geographical boundaries of a country or a union.

This is simply not possible and it does not provides a growth-centric environment which facilitates and promotes prosperity in the world. There are still a large population in this world which does not even have a bank account. More than 15 countries have only 15% or less people have a bank account. This does not mean that this population does not require money or is simply living off a barter system. Vast majority in this remaining 85% of population simply cannot afford to have identification because of procedural hassles, corruption in the countries or they simply think its not of a benefit to them [23]. In addition to this there are majorities who are simply do not trust banking systems with their less but hard earned money. However, in most scenarios while in a transaction the identity should not matter just like an economy with cash. When you have cash and if you want to exchange it for some commodity the seller does not demand an identification. Of course there are various caveats to this theory but for those kinds of problem statements as well controlled audits can be arranged to avoid any misuse.

In contrast, if we take a commodity in consideration after all the technological advancements and global push towards building a prosperous and better world, we still see famines and malnutrition being an epidemic in most of the under developed countries. Even though there are organizations like United Nations, Unicef which after being non profit organizations dedicated for the betterment and uplifting of world's economy there are still countries where large set of population dies of hunger and does not have reliable source of clean water, food and clothes [3]. This is primarily because of a broken supply chain system which has a lot of intermediaries, and however with a good intention, when this model constitutes of many countries it makes it extremely complex, inefficient which causes delays [43].

Blockchain can provide various ingenious solutions to these problems. On banking perspective in blockchain a simple peer to peer economy can be established where an identity is only exposed to an extent that its needed [45]. One does not need to have a government authorized document to identify herself. Anyone with a small device that connects to the blockchain can carry out transactions directly, in real time, even with least money and denominations possible. With Blockchain a free and open economy can be raised and established where people who never had a bank account can make transactions across the world without ever having to even show their identification. All that identifies them is their private key for their account.

In juxtaposition any supply chain model for any of the relief that is being sent from across the world can be driven and governed by a central blockchain channel where smart contract govern what relief was committed, sent and keep track of its location in real time [10]. Multiple auditors on the channel can ensure the timely delivery of the good and services which can make the system void if any of the intermediaries are causing delays in which case the transactions can be voided at anytime. This can avoid delays completely and drives the complete system based on performance and inherent accountability where one only gets paid when work is done in a timely and efficient manner.

## 6 SHADES OF IDENTITIES

Identity theft and misuse of personally identifiable information has been one of the biggest threats in recent past. With the emerging social media and Internet of Things it is next to impossible to be discrete about information being shared on the internet [31]. It is not only just about the social media but there are millions of apps which are on one hand very useful and saves a lot of resources and energy but on the other hand makes every user extremely vulnerable towards their usage [38]. Internet however has become one of the most easiest target for stealing personally identifiable information and a lot of information which can prove to be costly to the users [22].

Most of these identities are already pretty much exposed in the internet world and are being misused all the time. These identities are used and misused as seen fit by multiple users of the internet. Even for the sites like Facebook and Twitter which are considered social networking sites the data is always at risk. One would knowingly put all his personal information on his social media account to share with his friends and family but still that information is uploaded on the internet and has no control over its security and it can be in fact misused and have been misused very easily. On one hand with GDPR (General Data Protection Regulation)[13] European unions are trying to get rid of any PII[44] (Personally Identifiable Information) information to get out on any other country's infrastructure even on cloud but on the other hand with these social medias and mass utilized sites on the internet there is no control over this information being shared with others.

There Is indeed a fundamental flaw in the systems that we use today where we have to to share our actual identity in most scenarios to be a part of any product that is on internet. For example in financial and banking institutions we should be able to only share the information that pertains to and is needed by the banking firm. It does not have to be completely information where you share your full name, your email address, your address and other details. Similarly if someone is utilizing let us say cable from the cable provider. In which case the person does not have to share anything else with the cable provider other than the address and may be a way to identify himself. A very novice way to handle this completely by Blockchain is that it provides control to each user to have complete discretion over her identity and the user can choose and decide to divest that information on that particular channel or not, or in other words is that information really needed to be shared [36]. In these implementations one can really own and control the identity, trust the system not the other user on channel as the trust arrives inherently by mutual approvals.

## 7 OWNERSHIP AND ROYALTY

With the explosion of information and rise of social media there have been a complete paradigm shift in ways the different businesses and domains play out. Social media explosion and information sharing capabilities have provided a wide outreach for very small creators who could never get to share their skills or benefit from them because of the competition and limited resources. With

platforms like Twitter, Facbook, Instagram, Youtube now small creators and artists whose work could never see the day light and garner appreciation merely because of limited outreach, are provided very large number of audience from their homes and hand held devices [1]. However, all of this comes with a price. For example, a pottery artist who could simply create his art and sell it locally in a shop would earn much more profit than he could ever do that on internet or Instagram by sharing his art to millions of followers and devise a merchandize model on top of it to sell the product out. It is understood that it is indeed difficult for anyone to sustain a business in real world in a shop vs no cost internet but the comparable earning of the creator is too low simply because of all the middlemen. Instagram, Youtube themselves have their revenue models, to devise a merchandize model one will have to hire more external vendors who can operate on the sites incurs a lot of cost and the artist gets his final share after paying off all the middlemen. Let alone the time it takes for the final price to arrive in creators bank account which takes a very long time [24].

With Blockchain implementation in these different domains a straight forward peer to peer to transaction mechanism which gets settled in real time. For example a song uploaded on Spotify, Youtube, Soundcloud will basically be tied down on the % composition of the revenue model and the artist gets paid in real time based on the number of views or listens on spotify. And it is the artist who decides the price of single online playback of the song, or the usage of the song in some other productions which may require royalty to be paid to the artists. With blockchain implementation all of these transactions can be settled in real time with the help of smart contracts and trust less, handshaking transaction mechanism.

## 8 PROTECTION OF RIGHTS AND INTELLECTUAL PROPERTY

After discussing identity theft and solutions which Blockchain provides there are some other concerns in security as well on the information sharing platforms of social media. As very very large amount of information is being uploaded to the internet, where around 8 exabyte of data is being created only on phones [34]. On the internet virtually there is no restriction on the copyrights and if some one wishes to upload their copyrighted content or intellectual property. Any content that is uploaded on the internet can easily be duplicated and pirated with an extreme ease even today [19]. There are technologies working really hard to make this happen but it has always been a back and forth between tightening the security with various technical advancements and the internet users looking to hack the systems, finding loopholes and exploiting them are gaining equal amount of expertise and technical help. There has never been a robust way to protect the rights on any intellectual property or content on the internet. It can be the movies, songs, copyrighted documents and much more.

The biggest hurdle in enforcing this is in fact the geographical limitations of jurisdiction across countries which either does not exist or takes a lot diplomatic deliberations to achieve [19]. And that consumes a lot of time to enforce anything which in the this age of exponentially growing internet which is bolstered by Cloud and as a service products becomes mute. The as a Service concept, where companies offer services to help other companies become more efficient, offers a path to reduced costs and streamlined workflows. For example, the sites which are pirating the data and are distributing freely over internet just to get more advertisers never really get blocked because of similar limitations [30]. Even if they are identified and shutdown they can easily relocate themselves to any other data center of any Infra/Platform as a Service provider.

Blockchain can prove to be the best solution in this case where any intellectual property is considered an asset and is treated as an asset where any user of the internet trying to access this asset will have to authorize themselves and pay, in case it requires payment, before they can even see the data [6]. As the data security would not really be driven by the application that is hosting the data, in fact the applications can be created without any authentication mechanism in the first place and an asset based authentication and authorized can be applied using Blockchain channels.

## 9 HALCYON WAYS OF BUSINESS

Most important aspect that traditionally business minds use to follow was to keep everything local. Old business model which many big businesses followed was to reduce cost of raw material and cost of transportation of raw material, build everything locally. For example a car manufacturer would like to produce the raw goods required to make the car locally to avoid transportation and higher margins charged by other providers. This however have changed recently. Companies now, considering the high demand and supply, try to pay another companies to manufacture the complete goods and just ship them as their labels [33]. However, in these business models as well there are a lot issues where a suppliers reliability on availability, sustenance of quality and cost can still be an issue. The major issues in these cases are again the intermediaries. Because a provider from whom a business is getting their finished could very well be following the same business model and is primarily dependent on another provider. This chain can be really hurtful for any business where a single point of failure can cause huge delays and hurt he reputation in the market.

Blockchain solutions can provide the goodness of the both the old solutions and the contemporary ones. It is indeed not a good idea to start manufacturing bricks as well when an eventual goal is to build houses. However blockchain solutions can provide a new twist to the chain of providers which are now bound by the smart contracts of blockchain channels and live biddings can be associated with contracts to avoid any delays from any of these providers.

## 10 DISASTER RECOVERY AN OVERKILL

This is a staple benefit of anything decentralized but let us consider this particular scenario in detail. There have been a lot of companies which are running major platforms which provide disaster recovery tooling for mission critical infrastructures which cannot afford to lose any of their data, intellectual property or even the service availability. These organizations can vary from being financial institutions which are holding records of and money itself for billions of people and are handling all the transactions, there are major services which are running cloud services on which services there many applications which are running their services as a platform. However, one of the biggest problem and fear of these systems

is that if they are static and in house it is a responsibility of the organization itself to maintain and implement disaster recovery processes for their infrastructure. If its basically on Cloud then it all depends on the DOUs and the terms and conditions that one agrees to before going on cloud so there is no sense of security there. Cloud can provide easy backup and recovery processes because of virtualization. But the kind of catastrophe we are discussing here is worst case scenarios, natural calamities where everything in that town, city, state of a country is lost.

There have many natural calamities in which complete infrastructure of towns, and multiple cities is completely lost to the nature and its extremely difficult and tiring of some countries to even get the basic amenities like power, water supply running, let alone someone looking into recovering he the data of a data center. There are, however, many disaster recovery solutions available which are providing these services to replicate the data from one data center remotely to a totally different data center. Replicating the data locally enough with multiple exclusive pods so that they are running on completely separate infrastructure. In these kinds of scenarios the disaster recovery services charge hefty amount of money as they would like to invest more and more based on what category of recovery one would want. One being where a simple recovery from a lost hard drive is needed and highest level of recovery where if the complete site or datacenter is lost one should be able to recovery everything from a different site.

Blockchain technology can provide a centralize database which is inherently in principle distributed. In fact these are the kind of database which are replicated across the world in there default implementation itself and do not really have to be thought out specifically for disaster recovery. When there is any disaster the continuity of the business stays as is because even though some of the nodes on the channel are lost but there are still always the rest of the nodes which have complete data on them. It provides a $24 \times 7$ support through out the year without applying any complex technologies like replicating the data on a different continent or in a different city. Also there are smart contracts which are capable of storing the transaction related code and will trigger on provided conditions to alert, roll back transactions, procedures, network lockdown and backup procedures [5].

Disasters can be of some other forms as well like if there are malicious attacks done to the internet facing infrastructure and if there are Denial of Service (DOS)[14] attacks which can render the targeted systems down. In these scenarios as well the distributed setup of blockchain channels can really help keeping the system up and running. Even though there is a long way before blockchain technologies can be polished more for these kinds of totally unrelated looking applications but with a very high interest of major technology entrepreneurs and businesses all that has been shared as a proof of concept and research articles it would not be long before which blockchain can be used as reliable, secure, highly available and peer-peer fault tolerant infrastructure.

## 11 BIG BLOCKCHAIN

We have explored numerous possibilities that can overcome the limitation of todays information sharing world and how blockchain technologies can be applied to these contemporary solutions which showcases a futuristic path towards a truly shared economy over internet, built on mutual trust, ensured by cryptography while protecting the rights to intellectual properties and assets of billions of the users all over the world. However, as it appears to be the case, introduction of Blockchain technology to solve all these problems also poses new challenges on the adopters of these new technologies which we have to be prepared for [37]. These challenges do not pertain to the challenges of enabling more features and capabilities of blockchain. But on the other hand these are the challenges which will have to be addressed before the blockchains got beyond our control. Let us introspect some of these challenges, the causality and status of solutions to address these challenges, if any.

## 12 AUDITING CHALLENGES IN BIG BLOCKCHAINS

On one hand it can be safely said that Blockchain technology's substantial benefit is decentralization. In spite of that it opens numerous avenues of improvement on the current world economics and supply chain model which poses their own challenges which needs to be well thought of before hand. Protecting rights of a user's intellectual property on a blockchain based channel seems like a quick and viable solution but if we increase the cardinality of this system to billions of users on the internet it can very swiftly turn into a system which, even though secure, would still require a lot of auditing and safety fall back mechanisms [37]. Which are devised in a way that the issues are quickly and easily identified before they can go out of hand. For instance it is simple enough to say that a particular song was uploaded on the Spotify, Youtube, Soundcloud by a creator and it was made available to be listened to, downloaded by all the users on the internet in a peer to peer transaction mechanism. However, the sheer amount of data that can be generated just for this one kind of transaction could be huge and the length of the chain may start to grow very aggressively. Having a control over that much amount of data, for appropriate auditing, to make sure the creator is getting paid her fair share in real time, there are no delays in processing of the transaction appropriate BigData solutions have to applied on Blockchains too.

Blockchain as distributed it may be and the mutual trust it is based on and is backed by cryptography, at the end of the day its a chain. And to follow any kind of trail for any asset or commodity, or users for unforeseen reasons might not be simple enough once the adoption of blockchain has matured enough to be an integral part of the world's economy and financial institution. From protecting rights, to protecting any illegitimate use or wrong doings, to preventing delays based on smart contracts. All has to be settled and has to be auditable at one point in time. In all fairness the transactions on the blockchain will eventually have to be corrected, and would require transaction malleability [8]. Digital Autonomous Organizations would come into existence and would pose another challenge for auditing as these organizations do not really come under any jurisdiction or law. Simlar, challenges can also be posed by Long-term and short term blockchain forks where the chain has been forked into multiple chains.

On one hand it is being considered as one of the salient points of blockchain where it does not come under any specific countries jurisdiction and can seamlessly integrate and allow peer-peer

transactions over internet across the world. But these solutions do pose audit challenges for a particular country as it cannot go beyond its geographic limits to enforce its policies. Same situation would be exacerbated with cross chain transactions where there are transactions which are carried out between two separate self sustained chains.

# 13 CHALLENGES WITH BLOCKCHAIN AND CRYPTOGRAPHY

There are already a numerous challenges which are posed against this fledgling technology at a very preliminary level. After adoption of blockchains, as discussed above, at a very large scale and creating an impression in almost all of the global economy may pose its own challenges beyond auditing. There are some basic challenges even today around the technology readiness of our industry to be able sustain this mammoth task. On the oher hand the basic principle behind blockchain for investing compute and providing power to computer is raising a lot of red flags already from the conversationalists. Also, beyond auditing, governments of any nations would not easily digest this fact there are pseudo economies, businesses running under Digital autonomous organizations which has no real world presence and no one in real world can be held accountable for the misdoings of such rogue organizations [25].

On the other hand there is a an issue pertains and affects the development and enthusiasm towards blockchain i.e. losing jobs to blockchain [17]. Not very different from the scenario when computers arrive, there was harsh criticism towards computers too from a range of communities concerned with people losing jobs to computer. Security as well has been one of the major concerns in the blockchain paradigm [12]. However, when the idea of blockchain was published and the way it was devised it is nearly impossible for someone to breach any of the block chain and make changes to it which can sustain [26]. As the whole model is based completely on the cryptography and it requires a certain amount of compute spent on it, it is nearly impossible for someone to process a fake transaction and add the block to the chain. Because even though the block might get added it may never get trusted by all the other channel users. Basically the malicious user trying to hack the system will have to put in enough compute so that he can race against all the compute that is being used by all the miners in the world for that particular blockchain. And not just that to make the chain bigger enough to trust the same user will have to keep increasing his same chain with multiple block addition before anyone else in the channel does it.

Whilst we explored the nearly endless possibilities in which blockchain technologies can be leveraged and can prove to be able to provide smart solutions and applications to a wide spectrum of information technology world. There are still a lot of challenges ahead for this fledgling with a bright future technology. Ranging from the fact that there is a still a lot volatility in acceptance of the technology by all just like it was with Cloud computing when it came to energy consumed to do the computation work to make the work be validated. There are many challenges which various technology houses who are endlessly focused on building smarter solution to the nearly gone out of hand internet of things and make the world economy for global, secure and controllable.

## 13.1 Technical advancements

Technology not being ready is one of the major concerns in most of the cases. There Is a lot of enthusiasm towards blockchain and the technology powerhouses, businesses, entrepreneurs are continuously working on expanding the capabilities of blockchain but with the tech world in a state where right now everyone understands that it has bright future but it will still take sometime to be mature enough to be widely acceptable. Primary reason for this is that there are a lot of efforts being made in this direction but the focus has been fairly distracted. Ranging from getting the cost lower to implement it, making it truly decentralized, and to make really energy efficient. There have not been a single chain yet that implements all of these together. Any chain that is fast and is truly decentralized would require a really high cost of implementation because of the appropriate infrastructure needs.

## 13.2 Energy consumption

With blockchain technology and the underlining principle of cryptography and putting in cpu work to validate a certain block on the block, at least in these scenarios is considered a waste of power. The world is already having discussions on various sources of energy that while being produced and even consumed are causing global warming and climate change on earth. In that context introduction something like blockchain which needs a lot of compute essentially to make the channel secure is a difficult imposition to be widely accepted. But there are a lot of initiatives that are being carried out right now from companies like IBM, Intel which are continuously working building green blockchain models and implementations which are energy efficient.

## 13.3 Governments

There are many governments of various nations who see a lot of benefits in the emerging technologies and Government of the USA, China are trying to adopt the technology and are trying to get on board while its still growing so that the infrastructure needs are satisfied in an early stage and they are prepared for it. The primary focus of this study has been mostly on non cryptocurrency application of blockchain and the government views on that aspect of it has been fairly positive. It is the cryptocurrencies and the self sustainable pseudo currency which can be exchanged across the boundaries of the countries has been one of the biggest areas where government may want stifle excessive growth in that direction.

## 13.4 Jobs

Blockchain technology's underlining fundamental is to remove the bottlenecks and make the communication peer to peer. Which highly drives in a direction to remove the middlemen and all the intermediaries involved to an automated balanced systems of Smart Contracts, Transactions to do the auditing, approvals and any additional processes. One of the challenges which this, and most of it is due to lack of education as well, is that adoption of blockchain would cause a lot of job loss. This is not a new aspect of growth in the information technology industry. When the internet was booming and even before that when computers where getting more and more popular in the industry almost all were worried about the job loss by automation. But eventually there is a broader answer

to this problem and it is that yes there might some jobs which will reduced to a smart contract code which does, let us say an approval, but in most likelihood this is only going to be shift of job to something of more importance while building the planet smarter with a truly shared economy. However, it has to be understood that to keep up with technological advancements everyone has to keep themselves updated so that the day something that one does is automated new avenues will open and one should remain updated and ready to capitalize on such opportunities. There have been various studies being done to figure out the impact of this new technology on the jobs but it has mostly been superficial and qualitative, not quantitative [17].

### 13.5 Too autonomous

One of the other challenges of blockchain technology is pertaining to Digital Autonomous Organizations and they going rogue and building their old self sustaining economy once the marriage of artificial intelligence happens with blockchain. There have always been a lot of discussion around automation, robotics and artificial intelligence taking over the world. Which has mostly been a discussion and there has been no legitimate study on how much truth it is to it. It is indeed not possible to predict something like that because that implies that we can predict how our technology is shaping up. It is indeed true that technology is mostly driven with a common cause and intent of the drivers of the technological world like technical institutions, research institutions, businesses. And it highly depends on their interests and a balance between their intent behind shaping some technology. It could be for the common good of humanity but the same can apply to someone with a malicious intent too.

However, every now and then after all the security measures, whenever there is any cryptographic attack which goes viral the prices of cryptocurrencies suddenly start crashing. There Is a certain truth to it but indeed in principle the technology is safe [15].

## 14 A TRULY SHARED ECONOMY ACROSS THE WORLD

Blockchain development have been always driven with a good intent to overcome the shortcomings of the world's current status in form of misuse and distrust because of aggressive growth that drives a lot of collaborative with multiple parties involved which ends up being a bottleneck and does not provide the right value of the asset to the producer but in fact gets a lot of intermediaries who make the system complex and stifle the growth from the end producer and consumer perspective. With blockchain it would be possible to get the whole world on a single shared economy not only from the monetary aspect of it but from a supply chain perspective as well where the geographical borders and authorities do not drive the common intent of the world towards progress. Here are some insights on how the world might look after it.

Blockchain like technology and continuous endorsements and advancements in it will cut down the monopoly of certain big organizations who are now part of this recursive loop where they keep pushing their agendas and ideas with their financial capital over the industry. Blockchain promises that it will change this completely and will render the intermediaries, middlemen and all those bottlenecks to simple auditing, smart contract automation. The technology will provide a women with no access or enough money to own a bank account or even a national identification, her own place in shared economy where she can easily transact with a person across the world without any long stretching processes. It will be able to provide that musician who has been uploading music to the internet but does not really get the right value of his product because of unauthorized use of his intellectual property. The musician will be able to transact directly with each listener and charge accordingly rather than a generic rate applied and governed by sites like Youtube, Spotify.

And most importantly this will provide a shared economy across the globe in which without having a middleman any small or large business will get the benefit of purchasing the raw material directly from the producer and saving a lot of resources which make this complete transaction as profitable as it can be. Which in turn boosts a collective business revenue.

## 15 CONCLUSION

Keeping the non-cryptocurrency applications of Blockchain technology in this study we found that there is a huge endorsement from the technological entrepreneurs, businesses and governments on adopting and to continue advancement of blockchain technologies. It was identified that the intent and underlining principle of Blockchain to eliminate bottlenecks, intermediaries, middlemen is mutually shared by a numerous of organizations. We identified that the technology is moving towards highly ambitious goal of transforming the global economy to a shared and prosperous economy, protecting the rights of the intellectual property, making sure that the creators get their right compensation, to own and monetize data. We discussed a few heterogenous applications of blockchain which showcase how beneficial and generally application this technology is. We discussed in detail how the middlemen are hurting the collective growth of economy and how blockchain and its related solutions and advancements can help with it. For example, how blockchain can be really helpful in a quick solution to the Brexit problem of customs transaction management with a new geographical border being assumed between United Kingdom and Ireland. We also discussed with numerous examples how blockchain solutions can help with securing identities, ownership of creator on his intellectual property, and in providing the right value of creators product. We also delved in detail on how against the common perception its not just the blockchain application to BigData world that is important. But for technological advancements of Blockchain and for it to be able to sustain the kind of applications that are being built based on Blockchain as an underlining solution may require a lot of progress in Big data in Blockchain as well to be able sustain the aggressive growth of Blockchain in future without compromising with any of the underlining principles.

## ACKNOWLEDGMENTS

The author would like to thank Dr. Gregor von Laszewski for his support and suggestions to write this extended report.

# A APPENDIX DESCRIPTIONS AND ACRONYMS

**Unicef.** The United Nations International Children's Fund is a United Nations program headquartered in New York City that provides humanitarian and developmental assistance to children and mothers in developing countries.

**NSA.** The National Security Agency is a national-level intelligence agency of the United States Department of Defense, under the authority of the Director of National Intelligence.

**Twitter.** Twitter is an online news and social networking service on which users post and interact with messages known as tweets.

**Facebook.** Facebook is a social networking site that makes it easy for you to connect and share with your family and friends online.

**Instagram.** Instagram is a photo and video-sharing social networking service.

**Spotify.** Spotify is a music, podcast, and video streaming service.

**SoundCloud** SoundCloud is an online audio distribution platform.

**Digital Autonomous Organizations.** Digital Autonomous Organizations is a decentralized autonomous organization, sometimes labeled a decentralized autonomous corporation, is an organization that is run through rules encoded as computer programs called smart contracts.

# REFERENCES


[1] Saleem Alhabash and Mengyan Ma. 2017. A Tale of Four Platforms: Motivations and Uses of Facebook, Twitter, Instagram, and Snapchat Among College Students? *Social Media + Society* 3, 1 (2017), 2056305117691544. https://doi.org/10.1177/2056305117691544 arXiv:https://doi.org/10.1177/2056305117691544

[2] Iyke Aru. 2018. Can Blockchain Technology Survive Without Cryptocurrencies? (2018). https://www.ccn.com/can-blockchain-technology-survive-without-cryptocurrencies/ accessed 2018.

[3] BBC. 2017. Why are there still famines? (2017). http://www.bbc.com/news/world-africa-39039255 accessed 2018.

[4] Roman Beck, Michel Avital, Matti Rossi, and Jason Bennett Thatcher. 2017. Blockchain Technology in Business and Information Systems Research. *Business & Information Systems Engineering* 59, 6 (01 Dec 2017), 381–384. https://doi.org/10.1007/s12599-017-0505-1

[5] Elmer Berico. 2017. Blockchain for Business Continuity and Disaster Recovery. (2017). https://www.bcinthecloud.com/2017/10/blockchain-for-business-continuity-and-disaster-recovery/ accessed 2018.

[6] Bitcoinist. 2018. IPCHAIN DATABASE USING BLOCKCHAIN TO PROTECT INTELLECTUAL PROPERTY. (2018). http://bitcoinist.com/ipchain-database-using-blockchain-protect-intellectual-property/ accessed 2018.

[7] Nicolas Botton. 2018. The implications of blockchain for trade and Brexit. (2018). https://esharp.eu/debates/the-uk-and-europe/the-implications-of-blockchain-for-trade-and-brexit accessed 2018.

[8] Daniel Broby and Greig Paul. 2017. The Financial Auditing of Distributed Ledgers, Blockchain and Cryptocurrencies. (2017). https://pure.strath.ac.uk/portal/files/67044797/Broby_Paul_JFT_2017_The_financial_auditing_of_distributed_ledgers_blockchain.pdf accessed 2018.

[9] Zhuming Chen, Yushan Li, Yawen Wu, and Junjun Luo. 2017. The transition from traditional banking to mobile internet finance: an organizational innovation perspective - a comparative study of Citibank and ICBC. *Financial Innovation* 3, 1 (24 Jul 2017), 12. https://doi.org/10.1186/s40854-017-0062-0

[10] K. Christidis and M. Devetsikiotis. 2016. Blockchains and Smart Contracts for the Internet of Things. *IEEE Access* 4 (2016), 2292–2303. https://doi.org/10.1109/ACCESS.2016.2566339

[11] Luisanna Cocco, Andrea Pinna, and Michele Marchesi. 2017. Banking on Blockchain: Costs Savings Thanks to the Blockchain Technology. *Future Internet* 9, 3, Article 25 (2017), 20 pages. https://doi.org/10.3390/fi9030025

[12] Mauro Conti, Sandeep Kumar E, Chhagan Lal, and Sushmita Ruj. 2017. A Survey on Security and Privacy Issues of Bitcoin. *CoRR* abs/1706.00916 (2017), 36. arXiv:1706.00916 http://arxiv.org/abs/1706.00916

[13] Joe Curtis. 2018. What is GDPR? Everything you need to know before the 2018 deadline. (2018). http://www.itpro.co.uk/it-legislation/27814/what-is-gdpr-everything-you-need-to-know accessed 2018.

[14] Cyberpedia. 2018. WHAT IS A DENIAL OF SERVICE ATTACK (DoS)? (2018). https://www.paloaltonetworks.com/cyberpedia/what-is-a-denial-of-service-attack-dos accessed 2018.

[15] David Dawkins. 2018. Bitcoin price LIVE: Bitcoin drops $2000 as yet ANOTHER hack attack rocks cryptocurrencies. (2018). https://www.express.co.uk/finance/city/905277/bitcoin-price-live-updates-ethereum-ripple-blockchain-hackers-steal accessed 2018.

[16] Jose de Luna Martinez. 2005. *Workers ' Remittances To Developing Countries : A Survey With Central Banks On Selected Public Policy Issues.* The World Bank, onlineresources@worldbank.org. https://doi.org/10.1596/1813-9450-3638 arXiv:https://elibrary.worldbank.org/doi/pdf/10.1596/1813-9450-3638

[17] Michael del Castillo. 2017. Threat or Opportunity? Blythe Masters Talks Blockchain Jobs Impact. (2017). https://www.coindesk.com/threat-opportunity-blythe-masters-addresses-blockchain-jobs-impact/ accessed 2018.

[18] The Economist. 2017. The worldfis most valuable resource is no longer oil, but data. (2017). https://www.economist.com/news/leaders/21721656-data-economy-demands-new-approach-antitrust-rules-worlds-most-valuable-resource accessed 2018.

[19] Anita B. Frohlich. 2009. Copyright Infringement in the Internet Agefi!?Primetime for Harmonized Conflict-of-Laws Rules? *Berkeley Technology Law Journal* 24, 2 (2009), 851–896. http://www.jstor.org/stable/24118564

[20] Jane C. Ginsburg and Luke Ali Budiardjo. 2018. Liability for Providing Hyperlinks to Copyright-Infringing Content: International and Comparative Law Perspectives. (2018). https://lawandarts.org/wp-content/uploads/sites/14/2018/03/41.2_Ginsburg-Budiardjo_FINAL_3-11-18-PE2.pdf accessed 2018.

[21] Jonathan Hall and Alan Krueger. 2015. An Analysis of the Labor Market for Uberfis Driver-Partners in the United States. (2015). http://phillipsandco.com/files/4015/0333/6847/Uber_Driver-Partners_Hall_Kreuger_2015.pdf accessed 2018.

[22] Ali Hedayati. 2018. An analysis of identity theft: Motives, related frauds, techniques and prevention. *Academic Journals* 4, 1 (04 2018).

[23] Camilla Hodgson. 2017. The world's 2 billion unbanked, in 6 charts. (2017). http://www.businessinsider.com/the-worlds-unbanked-population-in-6-charts-2017-8 accessed 2018.

[24] Margaret Holland. 2016. How YouTube Developed into a Successful Platform for User-Generated Content. (2016). https://www.elon.edu/u/academics/communications/journal/wp-content/uploads/sites/153/2017/06/06_Margaret_Holland.pdf accessed 2018.

[25] MyungSan Jun. 2018. Blockchain government - a next form of infrastructure for the twenty-first century. *Journal of Open Innovation: Technology, Market, and Complexity* 4, 1 (13 Feb 2018), 7. https://doi.org/10.1186/s40852-018-0086-3

[26] Helienne Lindvall. 2009. Bitcoin: A peer-to-peer electronic cash system. (2009). http://bitcoin.org/bitcoin.pdf accessed 2018.

[27] Helienne Lindvall. 2013. How record labels are learning to make money from YouTube. (2013). https://www.theguardian.com/media/2013/jan/04/record-labels-making-money-youtube accessed 2018.

[28] Lloyd Marino. 2016. Blockchain fi!? The End of the Middleman. (2016). https://medium.com/@LloydMarino/blockchain-the-end-of-the-middleman-37d97a67d7f accessed 2018.

[29] Bryan Mueller. 2013. Participatory culture on YouTube: A case study of the multichannel network Machinima. (2013). http://www.lse.ac.uk/media-and-communications/assets/documents/research/msc-dissertations/2013/104-Mueller.pdf accessed 2018.

[30] Ian Phau, Min Teah, and Johan Liang. 2016. Investigating the Factors Influencing Digital Movie Piracy. *Journal of Promotion Management* 22, 5 (2016), 637–664. https://doi.org/10.1080/10496491.2016.1185491 arXiv:https://doi.org/10.1080/10496491.2016.1185491

[31] Reuters. 2018. Cambridge Analytica CEO claims influence on U.S. election, Facebook questioned. (2018). https://www.reuters.com/article/us-facebook-cambridge-analytica/cambridge-analytica-ceo-claims-influence-on-u-s-election-facebook-questioned-idUSKBN1GW1SG accessed 2018.

[32] Kate Rooney. 2018. India's central bank bans financial firms from dealing with cryptocurrency. (2018). https://www.cnbc.com/2018/04/05/indias-central-bank-bans-financial-firms-from-dealing-with-cryptocurrency.html accessed 2018.

[33] Ewan Roy. 2015. 5 methods to build a more efficient procurement strategy. (2015). http://www.tradeready.ca/2015/fittskills-refresher/5-methods-build-efficient-procurement-strategy/ accessed 2018.

[34] Jeff Schultz. 2017. How Much Data is Created on the Internet Each Day? (2017). https://blog.microfocus.com/how-much-data-is-created-on-the-internet-each-day/ accessed 2018.

[35] Lionel Shriver. 2018. Why cryptocurrency is the answer. (2018). https://www.spectator.co.uk/2018/01/why-cryptocurrencies-are-the-answer/ accessed 2018.

[36] Alex Simons. 2018. Decentralized Digital Identities and Blockchain fi?! The Future as We See It. (2018). https://cloudblogs.microsoft.com/enterprisemobility/2018/02/12/decentralized-digital-identities-and-blockchain-the-future-as-we-

see-it/ accessed 2018.
[37] A Michael Smith. 2018. The blockchain challenge nobody is talking about. (2018). http://usblogs.pwc.com/emerging-technology/the-blockchain-challenge/ accessed 2018.
[38] statista. 2017. Number of apps available in leading app stores as of March 2017. (2017). https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/ accessed 2018.
[39] Statista. 2018. Number of monthly active Facebook users worldwide as of 4th quarter 2017. (2018). https://www.statista.com/statistics/264810/number-of-monthly-active-facebook-users-worldwide/ accessed 2018.
[40] Stratfor. 2018. Could Blockchain Solve A Brexit Sticking Point? (2018). https://www.forbes.com/sites/stratfor/2018/02/28/could-blockchain-solve-a-brexit-sticking-point/#e42d79e5f921 accessed 2018.
[41] Don Tapscott and Alex Tapscott. 2016. *Blockchain Revolution: How the Technology Behind Bitcoin Is Changing Money, Business, and the World.* Brilliance Audio, Amazon.com Legal Department 2021 7th Avenue Seattle, WA 98121 USA.
[42] Rui Torres de Oliveira. 2017. Institutions, Middleman, and Blockchains fi?! Shuffle and Re-Start. *SSRN* 10, 2139 (27 Aug 2017), 15. https://doi.org/10.2139/ssrn.3027633
[43] L N Van Wassenhove. 2006. Humanitarian aid logistics: supply chain management in high gear. *Journal of the Operational Research Society* 57, 5 (01 May 2006), 475–489. https://doi.org/10.1057/palgrave.jors.2602125
[44] Cory Warren. 2017. What Is Personally Identifiable Information (PII)? (2017). https://www.lifelock.com/education/what-is-personally-identifiable-information/ accessed 2018.
[45] Jesse Yli-Huumo, Deokyoon Ko, Sujin Choi, Sooyong Park, and Kari Smolander. 2016. Where Is Current Research on Blockchain Technology?fi!?A Systematic Review. *PLOS ONE* 11, 10 (10 2016), 1–27. https://doi.org/10.1371/journal.pone.0163477

# Deploying CouchDB Cluster

Ribka Rufael
School of Informatics and Computing
Bloomington, IN 47408, USA
rrufael@iu.edu

## ABSTRACT

This paper focuses on deployment of CouchDB Cluster using Ansible playbook on Ubuntu16.04 OS virtual machines on Chameleon Cloud by using Ansible script which is invoked by bash script. The deployment process provides the option of installing CouchDB version 2.1.1 in single node or cluster configuration. The deployment process also involves creation of database in CouchDB with different number of shards and replication. Part of this paper also focuses on performing benchmarking tests and analysis to asses the performance of CouchDB write, read and mapreduce tasks by variying the number of shards and replicas.

## KEYWORDS

hid-sp18-703, Volume: 9, Chapter: Ansible, Status: 100.
CouchDB, Ansible, Chameleon Cloud

## 1 INTRODUCTION

CouchDB [1] is a NoSQL database management system under Apache. Data is stored as documents in CouchDB. In this project CouchDB is deployed and configured on two Chameleon Cloud virtual machines using Ansible playbook which is invoked through bash script. The two Chameleon Cloud virtual machines have Ubuntu16.04 OS installed. The bash script, `couchdbinstall.sh`, by taking three command line parameters from user is able to install and configure CouchDB in single or cluster configuration. This bash script also creates database with different number of shards and replicas.

Benchmarking tests were performed to find the impact of number of shards and replicas on time taken for bulk documents load into CouchDB, find documents operation and mapreduce operation on documents in CouchDB. The bash script `DatasetBulkLoadReadCouchDB.sh` is used to run the benchmarking tests. The dataset which was used for benchmarking tests on CouchDB is the wine quality dataset from UCI Machine Learning Repository [12]. The graphs used for the analysis of the benchmarking process were plotted using `plotBenchmark.py` python script which utilizes pandas and matplotlib libraries.

## 2 ANSIBLE

Ansible is a tool that help automate software deployments, system configuration and continues deployment of applications. Automation tasks are defined in Ansible Playbooks using YAML language. For transportation, Ansible uses OpenSSH [11].

Ansible was used to install CouchDB, configure CouchDB in single node or cluster configuration and to create database in CouchDB with different number of replicas and shards configurations on Chameleon Cloud virtual machines.

## 3 CHAMELEON CLOUD

The Chameleon project provides open and large scale platforms for reaserchers. Chameleon project gets its funding from National Science Foundation (NSF). Reaserchers will be able to investigate and come up with solutions for problems in the areas of Software as a Service, Platforms as a Service and vitual technologies. Both Bare metal and OpenStack Virtual environments are available for researchers to use on Chameleon project [9].

OpenStack virtual machines were used for deloyment and benchmarking of CouchDB. Launching of new virtual machine instances, assigning IP addresses to instances, assigning security groups and other virtual management tasks were done through the OpenStack web user interface [10].

## 4 PYTHON

Python programing language was used in two parts of this project. One of the Python scripts was used for preparing the test dataset with proper JSON format used in CouchDB benchmarking process. Also Python was used to develop the script which is used in the analysis of the benchmark process outputs.

## 5 COUCHDB

Apache CouchDB is an open source and NoSQL database system. In CouchDB data is stored in documents. Documents in CouchDB have unique names and contain metadata. The content of the document in CouchDB is in semi structured JSON format. The fields in the document can be different data types and there is no size limit. Some of the supported data types for the content of documents in CouchDB are strings, numbers, arrays, boolean and JSON objects [1].

There is no locking mechanism in CouchDB when adding, updating or deleting documents. If two applications are making an update to the same document in CouchDB database, upon save the application has to resolve the conflict before merging the new changes. Also while adding, updating or deleting documents in couchDB, if the process fails before finishing then nothing gets saved in the document. The update process has to finish successfully in order to save the changes to the document. Read, write, update and delete operations to CouchDB documents are done through RESTful APIs [1].

User can interact with CouchDB through Fauxton which is an administration user interface [1]. User can also interact with CouchDB using comand line tool called curl. Curl is used to make call to the RESTful APIs through the command line [7].

Multi-Version Concurrency Control (MVCC) is used as concurrency model in CouchDB. CouchDB uses views to present structured data to users and applications. Views are implemented in

**Table 1: Resource Specification**

| Instance | Image | Size | RAM | VCPUs | Disk |
|---|---|---|---|---|---|
| Node1 | Ubuntu16.04-20180205 | m1.medium | 4GB | 2 | 40GB |
| Node2 | Ubuntu16.04-20180205 | m1.medium | 4GB | 2 | 40GB |

Javascript. Views in CouchDB use mapreduce model and they are stored under the design documents [1].

### 5.1 Version

The version of CouchDB used for the deployment process is version 2.1.1 and this is the current version at the time of writing this report.

### 5.2 Architecture

CouchDB 2.1.1 provides two setup configurations. These setup configurations are single node and clustered configurations. As the name suggests, CouchDB in single node configuration runs on a single node. Whereas in clustered configuration CouchDB runs on multiple nodes or servers [1, 8].

### 5.3 Cluster Shards and Replicas

Shards are partions of database table which are split by rows [13]. The number of copies of each document in CouchDB database is called replica [4]. When creating databases in CouchDB cluster, user can provide the number of shards and replicas. If the user does not provide the number of shards and replicas, CouchDB sets the default values of 8 shards and 3 replicas. Having the number of replicas greater than one increases the failure resistance of the cluster. In scenarios where we have multiple replicas and one of the replicas fail then user can still perform read and write operations on documents without any disruption [4].

### 5.4 Security

The ports that are used by CouchDB cluster are 5984 and 5986. Port 4369 is used by Erlang to identify the nodes in CouchDB cluster. Also ports 9100-9200 for Erlang on different virtual machines to communicate to each other. These ports should be opened to accept TCP protocol for all the servers or virtual machines that are in the cluster [6].

In the deployment of CouchDB, security group rules were created in Chameleon Cloud which opened inbound TCP protocol for ports 5984, 5986 and 9100-9200.

## 6 DEPLOYMENT

The deployment of CouchDB was done through an automated Ansible playbook which is invoked by bash script.

### 6.1 Resource

Table 1 provides the specification for the Chameleon Cloud Virtual Machines used in this project.

Before starting the deployment of CouchDB, user have to perform the following preparation steps.

(1) Start two Instances in Chameleon Cloud with the specification in Table 1 1
(2) Allocate floating IPs to the instance created
(3) Add security rules as mentioned under Security section of this report to the two instances
(4) User has to manually insert the IP addresses by modifying inventory.txt file which is found under project-code /couchdbansible directory. There are two hosts defined under inventory.txt. One of the IP addresses goes under `[couchdb_Coordination_host]` section of inventory.txt and the second IP address goes under `[couchdb_hosts]`.

### 6.2 couchdbinstall.sh

The bash script `couchdbinstall.sh`, takes three command line inputs from the user. First parameter tells the script whether to install CouchDB in single node or cluster configuration. The second input represents the number of replicas. The third input is the number of shards. An example command which deploys CouchDB in cluster configuration and creates a database with 3 replicas and 8 shards looks like:

```
./couchdbinstall.sh true 3 8
```

The Ansible script that deploys CouchDB and it's dependencies are run within `couchdbinstall.sh`. This script also creates CSV file which contains information about cluster setup, number of replicas, number of shards and the time it took for couchDB install.

### 6.3 Ansible Scripts

All Ansible scripts reside under `/project-code/couchdbansible` directory.

(1) `playbook.yml`- defines the hosts on which CouchDB will be installed and the role name. This file along with `inventory.txt` file are needed when runing Ansible playbook.
(2) `main.yml`- resides under `/roles/couchdb/tasks` directory and this where all the installation tasks are defined. Some of the tasks defined are installation of Python pandas library since the virtual machines in Chameleon did not have pandas installed, installation of CouchDB 2.1.1 and all it's dependencies and database creation after the installation of CouchDB.
(3) `vmargs.j2`- under `/roles/couchdb/templates` directory, template file that defines the CouchDB node names with the correct IP addresses.

### 6.4 Deploy Time

The time taken for deployment of CouchDB on two Chameleon VMs was recorded for different scenarios. As it can be seen in table 2, CouchDB was installed in single node configuration/cluster configuration and different number of replicas and shards. The first scenario which is basic installation of CouchDB in single node configuration and no replicas and shards took 54 seconds amount of time to finish. The sceond scenario, installation of CouchDB in cluser configuration with two nodes configuration, 3 replicas and 8 shards took 52 seconds. The third scenario, installation of CouchDB in cluser configuration with two nodes configuration, 4 replicas and 12 shards took 53 seconds.

[2]

**Table 2: CouchDB Deploy Time**

| | Cluster_Setup | replica_val | shard_val | Install time in seconds |
|---|---|---|---|---|
| 1 | False | 1 | 1 | 54 |
| 2 | True | 3 | 8 | 52 |
| 4 | True | 4 | 12 | 53 |

# 7 BENCHMARK RESULTS

The results of benchmarking process that was performed to assess the impact of variying the number of shards and replicas on performance of CouchDB write, read and mapreduce tasks will be discussed in this section.

## 7.1 Dataset

The dataset that was used for the benchmarking process is the white vinho verde varient of Portuguese wine sample from UCI Machine Learning Repository. The size of the dataset used `winequality-white.csv` is 258KB. The dataset was in CSV file format and contains twelve different attributes related to wine such as fixed acidity,pH, alcohol and quality [12].

Since CouchDB bulk load process only accepts file in proper JSON format, `csv_to_json.py` python script was used to download the white wine dataset (`winequality-white.csv`) directly from UCI Machine Learning Repository and convert the CSV file into JSON format by utilizing pandas and urllib Python libraries. After converting into CouchDB compatible JSON format the dataset size was 1.1MB.

## 7.2 DatasetBulkLoadReadCouchDB.sh

This is the script when executed, loads the JSON wine sample dataset into CouchDB, performs simple read operation on the wine sample dataset in CouchDB and mapreduce on the wine sample dataset in CouchDB. It also measures the time taken for each operation.

(1) The first thing this script does is, it copies `csv_to_json.py` Python script to the remote VM and executes this script inorder to generate the JSON wine dataset.
(2) CouchDB has `/{db}/_bulk_docs` endpoint that is used to create multiple documents with singe POST request [2]. `DatasetBulkLoadReadCouchDB.sh` loads JSON wine dataset into *test* database in CouchDB using this POST call through curl command. Then measures the time taken to finish the bulk documents load operation.
(3) `DatasetBulkLoadReadCouchDB.sh` performs find document operation to find documents from wine dataset which have quality greater than 3. Then measures the time taken to finish this find operation. The find operation is accomplished by making POST call to `/{db}/_find` endpoint which returns documents that meet query criteria [3].
(4) We wrote mapreduce function mapreducefun.js, which returns total number of rows from wine dataset which have quality greater than 3. This mapreduce function was uploaded to CouchDB design document to create view and GET call to `/db/_design/design-doc/_view/view-name` was made [5]. DatasetBulkLoadReadCouchDB.sh executes this mapreduce function on wine dataset in CouchDB and measures the time taken to finish execution of mapreduce.
(5) The final step in `DatasetBulkLoadReadCouchDB.sh` is to gather all the time durations from these mentioned operations and save into CSV file under /CouchDBBenchmark directory.

## 7.3 Benchmark Result Analysis

There were two aims of the benchmarking process. One of the aims of the benchmarking process was to find the impact of number of shards on bulk documents load into CouchDB, find documents operation and map reduce on documents in CouchDB. The second aim of the process was to find the impact of number of replicas on bulk documents load into CouchDB, find documents operation and map reduce on documents in CouchDB.

Tests involving different combination of number of replicas and shards were performed and the results were captured in CSV files. All these CSV files were combined into one CSV file named `CouchDBfinal.csv` under `/project-code/CouchDBBenchmark` directory by running `CombineBenchmark.sh` bash script. Finally, we developed `plotBenchmark.py` python script that takes combined CSV files and plots the graphs which will be discussed in the sections to follow. For the purpose of reproducing the graphs in this project in the future, CSV file used `CouchDBfinal.csv` is saved under `project-artifact directory`.

To asses the impact of number of shards, number of replicas was set to a constant value of 3 and tests were performed. The number of shards used were 1,2,4,6,10 and 12. To asses the impact of replicas, number of shards was set to a constant value of 1 and tests were performed. The number of replicas used were 1,2,3,4 and 6.

*7.3.1 Number of Shards Impact on Bulk Load.* Figure 1 depicts the the impact of number of shards on bulk load documents to couchDB time in seconds. As it can be seen in the graph, it took 3 seconds to bulk load documents in couchDB when the number of shards was one. For number of shards starting from two onwards, the bulk load time decreased to a value of 2 seconds and remained constant. It took less time for bulk load documents when the number of shards increased from one to two.

*7.3.2 Number of Shards Impact on Document Find.* Figure 2 shows the impact of number of shards on find documents on CouchDB time in seconds. There was not any performace improvement of the time it took to find documents in CouchDB by changing the value of the number of shards. As the it can be seen in 2 the time for find documents remained at 1 second regardless of changing the the number of shards.

*7.3.3 Number of Shards Impact on MapReduce.* Figure 3 shows the impact of number of shards on mapreduce process on CouchDB time in seconds. When the number of shards was one, it took 4 seconds to finish the mapreduce operation. It took 3 seconds to finish the mapreduce operation when the values of shards were between two and six. There was a performance improvement for time taken for performing mapreduce operations for shards two, four and six. The time duration started to go up after increasing the number of shards to eight and above. There is no performance

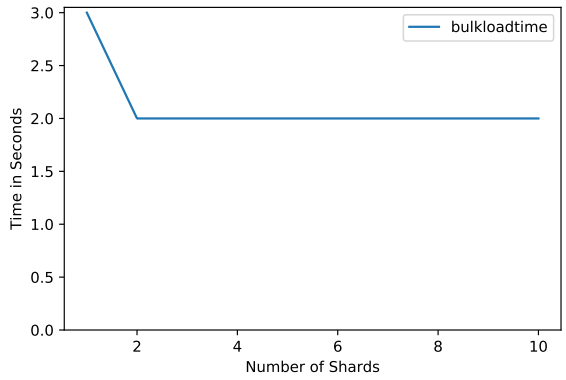


**Figure 1: Number of Shards vs Bulk Load Time in seconds**

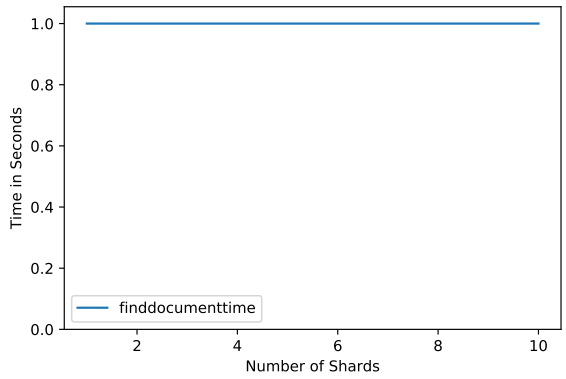


**Figure 2: Number of Shards vs Find Document Time in seconds**

improvement on the time for mapreduce operations by increasing the number of shards greater than six.

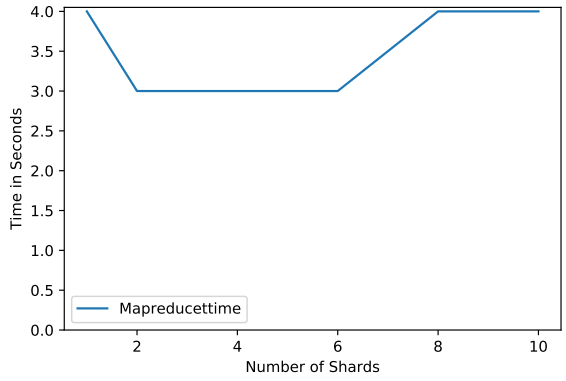


**Figure 3: Number of Shards vs Mapreduce Time in seconds**

*7.3.4 Number of Replicas Impact on Bulk Load.* Figure 4 depicts the the impact of number of replicas on bulk load documents to couchDB time in seconds. As it can be seen in the graph, it took 3 seconds to bulk load documents in couchDB when the number of replicas was one. There was a decrease in time taken when the number of replicas was two. Then as the value of replicas increased to three and above the time it took for documents bulk load increased to 3 seconds and remained the same. The increase of time as the number of replicas increases is expected because the documents have to be copied the same number of times as the number of replicas.

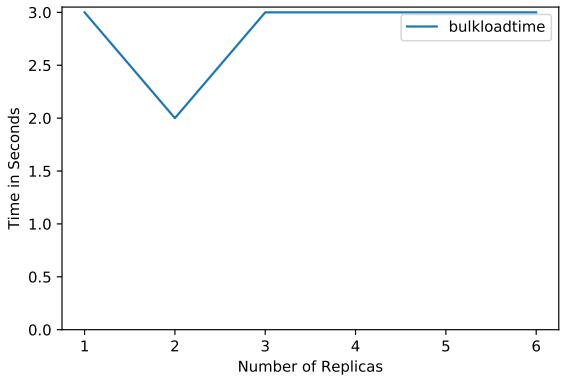


**Figure 4: Number of Replicas vs Bulk Load Time in seconds**

*7.3.5 Number of Replicas Impact on Document Find.* Figure 5 shows the impact of number of replicas on find documents on CouchDB time in seconds. There was not any performace improvement of the time it took to find documents in CouchDB by changing the value of the number of replicas. As the it can be seen in 5 the time for find documents remained at 1 second regardless of changing the the number of replicas.

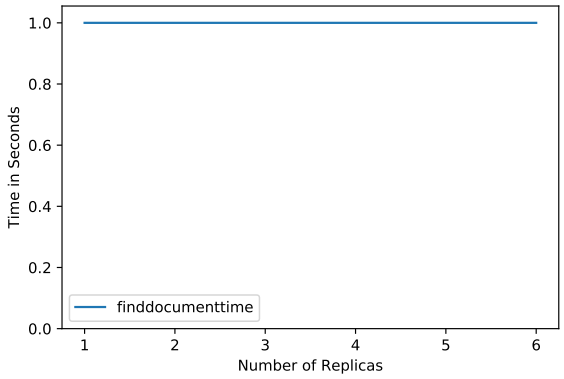


**Figure 5: Number of Replicas vs Find Document Time in seconds**

*7.3.6 Number of Replicas Impact on MapReduce.* Figure 6 shows the impact of number of replicas on mapreduce process on CouchDB time in seconds. The time it took for mapreduce operation was 3 seconds for one, two and six replicas. For replicas three and four mapreduce operation took 4 seconds. The variations in time could be related to network speeds at the time of executing the tests as these API requests are made through network.

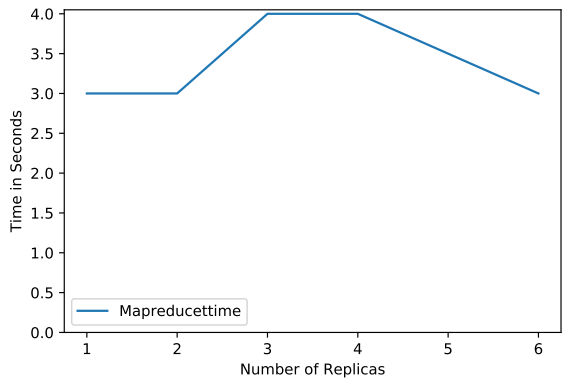


**Figure 6: Number of Replicas vs Mapreduce Time in seconds**

## 8 CONCLUSION

We have succesfully deployed and configured CouchDB on two virtual machines on Chameleon cloud using Ansible playbook which is invoked through bash script. The bash script also allowed CouchDB to be configured in single node or cluster configuration and created database with different number of shards and replicas. Although for the purpose of this paper only two virtual machines on Chameleon cloud were used, the Ansible script will be able to support installation of CouchDB to more than two virtual machines by providing more host IP addresses under inventory.txt for Ansible.

Analysis of benchmarking tests showed that there was not any performace improvement of the time it took to find documents in CouchDB by changing the value of the number of shards or replicas. For number shards between two to six, there was a performance improvement for time taken for performing mapreduce operations. In general, mapreduce took more time than find documents operation for different number shards and replicas, which could be the result of mapreduce using JavaScript query server for map and reduce operations.

## ACKNOWLEDGEMENTS

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

## REFERENCES

[1] Apache Software Foundation. 2017. 1.1. Technical Overview - Apache CouchDB 2.1 Documentation. Web Page. (March 2017). http://docs.couchdb.org/en/latest/intro/overview.html accessed: 2017-03-11.
[2] Apache Software Foundation. 2018. 10.3.2. /db/_all_docs - Apache CouchDB 2.1 Documentation. Web Page. (April 2018). http://docs.couchdb.org/en/2.1.1/api/database/bulk-api.html accessed: 2018-04-19.
[3] Apache Software Foundation. 2018. 10.3.4. /db/_find - Apache CouchDB 2.1 Documentation. Web Page. (April 2018). http://docs.couchdb.org/en/2.1.1/api/database/find.html accessed: 2018-04-19.
[4] Apache Software Foundation. 2018. 11.2. Theory - Apache CouchDB 2.1 Documentation. Web Page. (April 2018). http://docs.couchdb.org/en/2.1.1/cluster/theory.html accessed: 2018-04-21.
[5] Apache Software Foundation. 2018. 11.5.4. /db/_design/design-doc/_view/view-name - Apache CouchDB 2.1 Documentation. Web Page. (April 2018). http://docs.couchdb.org/en/master/api/ddoc/views.html accessed: 2018-04-19.
[6] Apache Software Foundation. 2018. 12.1. Setup - Apache CouchDB 2.1 Documentation. Web Page. (April 2018). http://docs.couchdb.org/en/latest/cluster/setup.html#cluster-setup-wizard accessed: 2018-04-14.
[7] Apache Software Foundation. 2018. 1.7. cURL: Your Command Line Friend - Apache CouchDB 2.1 Documentation. Web Page. (April 2018). http://docs.couchdb.org/en/2.1.1/intro/curl.html accessed: 2018-04-21.
[8] Apache Software Foundation. 2018. Apache CouchDB. Web Page. (April 2018). https://couchdb.apache.org accessed: 2018-04-14.
[9] Chameleon Cloud. 2018. About Chameleon. Web Page. (April 2018). https://www.chameleoncloud.org/about/chameleon/ accessed: 2018-04-16.
[10] Chameleon Cloud. 2018. KVM-Chameleon Cloud Documentation. Web Page. (April 2018). https://chameleoncloud.readthedocs.io/en/latest/technical/kvm.html accessed: 2018-04-16.
[11] Red Hat, Inc. 2018. Overview-How Ansible Works. Web Page. (April 2018). https://www.ansible.com/overview/how-ansible-works accessed: 2018-04-16.
[12] UCI Machine Learning Repository. 2018. Wine Quality Dataset. Web Page. (March 2018). https://archive.ics.uci.edu/ml/datasets/Wine+Quality accessed: 2018-03-28.
[13] Wikipedia contributors. 2018. Shard (database architecture) — Wikipedia, The Free Encyclopedia. Web Page. (April 2018). https://en.wikipedia.org/w/index.php?title=Shard_(database_architecture)&oldid=827064066 accessed: 2018-04-21.

## A PROJECT CODE EXECUTION

All the codes for this project are under project-code directory. There are three directories under project-code.

- CouchDBBenchmark which contains CSV files from deployment and benchmarking processes.
- couchdbansible which contains Ansible scripts
- logs which contains console output logs

Before starting the deployment of CouchDB, user have to perform the following preparation steps.

(1) Start two Instances in Chameleon Cloud with the specification in Table 1 1
(2) Allocate floating IPs to the instance created
(3) Add security rules as mentioned under Security section of this report to the two instances
(4) User has to manually insert the IP addresses by modifying inventory.txt file which is found under project-code /couchdbansible directory. There are two hosts defined under inventory.txt. One of the IP addresses goes under `[couchdb_Coordination_host]` section of inventory.txt and the second IP address goes under `[couchdb_hosts]`.

### A.1 Deploy CouchDB

To deploy CouchDB, on command prompt cd to `project-code/` directory and run `couchdbinstall.sh` as providing three parameters.

(1) First parameter takes true or false to indicate whether to install CouchDB in single node or cluster
(2) Second parameter is an integer number for the value of replica
(3) Third parameter is an integer number for the value of shard

An example command

```
./couchdbinstall.sh true 3 8
```

## A.2 Benchmark Tests

To run benchmark tests, on command prompt cd to `project-code/` directory and run

`./BulkLoadReadCouchDB.sh`

This script performs different benchmark tests and saves all the time durations into CSV file name starting with `benchmark_` under `CouchDBBenchmark` directory.

## A.3 Combining CSV files

To combine all CSV files from benchmarking tests into one file named `CouchDBfinal.csv` run

`./CombineBenchmark.sh`

A copy of this file is also saved under `project-artifact` if needed in the future to reproduce the graphs in this project

## A.4 Plotting graphs for Benchmark Tests

To Plot the graphs for benchmarking analysis run the `plotBenchmark.py` script from command line as follows

`python plotBenchmark.py`

# Diagnosis of Coronary Artery Disease Using Big Data Analysis

Hady Sylla
Indiana University
711 N.Park Ave
Bloomington, Indiana 47408
hsylla@iu.edu

## ABSTRACT

Coronary artery disease (CAD) is the most prevalent heart disease. Worldwide, it is among the primary causes of heart failure and mortality [2]. Therefore, its early diagnosis is essential. Specialists have invented different techniques for diagnosing CAD. They conduct most of these methods using the Irvine Dataset (University of California), which have limitations of features and missing values, thus being unreliable. The present study employs big data techniques and the latest dataset, with no missing values, and with features such as the Q wave, ST depression, functional class, T inversion, and dyspnea. This data has been compiled from the Shaheed Rajaei Cardiovascular, Medical and Research Center, between 2011 and 2012, and involves 303 patients. This study uses Nave Bayes, SMO, and a proposed ensemble algorithm, using these features to conduct the analyses. Tenfold cross-validation on the dataset has been used to design accurate algorithms with an ensemble algorithm, achieving 88.5% accuracy. This study extracts the relevant features and rules regarding CAD, which previous studies have not discussed.

## KEYWORDS

hid-sp18-706, Volume: 9, Chapter: Application, Status: 100.
Coronary disease, big data, Random Forest

## 1 INTRODUCTION

Coronary artery disease (CAD) comes about when atherosclerotic plaques develop in the coronary arteries, giving rise to a narrowing of the coronary luminal, and sometimes occlusion, which leads to heart failure or sudden cardiac failure.In Western nations, CAD is among the killer diseases.It is essential for people to understand the pathophysiology of CAD, how to control its development, to identify any effective innovation in the cardiovascular risk elements, how to diagnose and treat it in its early stages, and its reversible stages. Experts have considered big data mining technique as the 'gold standard' for diagnosing CAD, and it is commonly used. However, big data mining is an expensive and invasive procedure that requires a high level of technological expertise and the latest technology. Thus, medical experts cannot use it to screen or treat many patients.

For this reason, many health facilities use other noninvasive tools to diagnose CAD.The most popular of these methods are stress echocardiography (ECHO), electrocardiogram (ECG), and scintigraphy or SPECT. Additionally, clinical setups currently use coronary magnetic resonance angiography (CMRA) MSCT or EBCT. The objective of this study is to show how to use big data mining technique to diagnose Coronary Artery Disease and why this is the best way to control heart attacks and deaths. The review discusses the features in the big data mining tool, how they derive the results, and presents a discussion of the results.

Additionally, this paper provides resources for future studies and offers recommendations to improve the examination of CAD.

Data mining is a technique for finding hidden information in a database. Nowadays, several fields use data mining; it has different applications, such as scientific discoveries, entrepreneurship, and fraud detection. Data mining calculations work on databases in which each information archive has several attributes. One of these attributes is the class label, which indicates the category of the data. The data mining algorithms people use in problem solving are association rule mining, classification, regression analysis, and clustering. For classification algorithms, there is required a learning phase on a labeled data set, which requires the researcher to determine the missing test record class label. In the learning phase, a scientist constructs a classification model to predict the data label class label using the values of its features. A specialist categorizes heart disease into cardiovascular and cardiomyopathy disorders. Coronary Artery Disease (CAD) is a significant cardiomyopathy disease subgroup, which causes disability, serious illness, and even death, as the disease affects the supply of oxygen and blood to the heart muscles.

The early symptoms of heart disease comprise pain in the centre of the chest or a sense of numbness, and palpitation. Other signs are fainting fits or dizzy spells, and dyspnea on exertion. Due to the important nature of heart failures, it is essential to find out its causes. Currently, the primary goal of many scientific pursuits is making an accurate diagnosis of cardiac disease. Researchers have collected a great deal of information while examining CAD patients, and by processing the data, they can find out how the main features of cardiac disorders are related (e.g., the amount of cholesterol, blood pressure, etc.) and the probability of the occurrence of such a disease. The next chapter discusses several research articles that discuss this topic and relate it to the findings of the present study.

## 2 CONTRIBUTION OF BIG DATA TO CORONARY ARTERY DISEASE

This section provides selected examples of the potential of big data arising from the variety, volume and value of the data people have obtained. Additionally, it shows how big data contribute to scientific advances in cardiovascular medicine from the discovery of underlying disease mechanisms, disease taxonomy, of treatment relevant sub-types of the disease which underpin drug development, and precision medicine.

## 2.1 Discovery in genetic and big data

Lee, Noh and Ryu [6] point out that it is challenging to provide deep mechanistic insight into large-scale big data mining resources given the limited availability of genetic information in sufficient depth. Bespoke, recallable investigator-led studies, such as East London Genes & Health (ELGH) and the NIHR BioResource enable the coupling of big data mining with extreme genotypes and facilitate their in-depth study using bespoke experimental protocols. Complete gene knockouts inform people about the gene process. With the latest survey of 3,222 British Pakistani-heritage exome sequenced persons with high parental relatedness, there have been discovered 1,111 rare-variant homozygous most likely function loss (rhLOF) genotypes to predict the disruption (knockout) of 781 genes [10].

Rajkumar [7] stated that linking big data to the rhLOF genotypes is not associated with a doctor or prescription consultation rate, and there are no illness-related phenotypes in 33 out of 42 individuals with rhLOF genotypes in Mendelian recessive complexity genes. Phasing the genome sequencing of a healthy PRDM9 knockout woman, her controls, and baby, found meiotic recombination sites localized from the hotspots with PRDM9 dependency, depicting PRDM9 redundancy in human beings. There have been genomic approaches to validating case definitions: across thousands of hospital ICD codes, there have been reproduced associations from genome-wide association studies obtained one phenotype at a time.

## 2.2 Discovery in larger scale epidemiology

Big health records data can contribute to the discovery of new associations which would be hard to generate from traditional consented cohorts without record linkage. This involves a resolution across a range of risk factor levels and a range of different initial presentations of cardiovascular disease. Various associations have been discovered in a cohort of more than one million adults initially free from diagnosed cardiovascular disease using national structured linked electronic health records from the CALIBER resource, in which EHR phenotyping algorithms are created, validated and shared using a robust methodology [10]. Sideman [9] argued that the primary necessity for precision medicine is estimating the patient's state. The experts mainly analyze the correlations of illness trajectories with a focus on a few complications or by the use of large-scale techniques without taking time into consideration. Researchers have performed a discovery-driven study of illness progression patterns using data from an electronic health registry covering the entire population of Denmark. Using the whole illness spectrum, they created 14.9fiftyears of records data for 6,200,000 patients for 1,171 critical trajectories. Those experts identified primary diagnoses, such as chronic obstructive pulmonary disease and gout obstructive pulmonary disease, as the cause of the progression of CAD across many of these paths. Thus, it is of added significance to diagnose CAD earlier. Such a data-driven approach analysis is useful for preventing and predicting future complications in individual patients [9].

## 2.3 Discovery with deep phenotypic data

Most cardiovascular diseases (including acute myocardial infarction) have syndromic descriptions and labels, which may span multiple underlying pathological disease processes. One approach to discovering the mechanistically relevant disease types is to phenomenal the disease [8]. For example, in heart failure with preserved ejection fraction, machine learning in 46 continuous clinical, laboratory, electrocardiographic, and echocardiographic findings have determined mutually exclusive groups related to the subsequent outcomes. The cardiac atlas project (of healthy and diseased hearts) is an example of a large-scale collaboration on feature extraction in imaging, using data sharing in standard formats Digital Imaging and Communications in Medicine (DICOM) of the pixel and non-pixel data. Personalization using physiological simulations, for example, for cardiac resynchronization therapy, has been proposed. Unstructured free-text data in big data mining adds a further way to estimate disease co-occurrence and patient stratification, which people map to the biological frameworks of systems [8].

## 2.4 Discovering and validating drug targets

Regarding a discussion by [7], big data-DNA resources may play an increasingly important role in drug discovery, genomic drug target validation, marker validation, and drug repurposing. For example, [7] demonstrates a strategy in which human mutations that inactivate a gene encoding a drug target can mimic the action of an inhibitory drug, here ezetimibe, and thus can be used to infer potential effects of that drug. Ezetimibe is known to affect the marker (LDL cholesterol) but, until recently, not the disease (myocardial infarction). Among the most significant sources of cases of MI and controls in this study was a DNA resource integrated into a health system with rich big data. The discovery of PCSK9 as a drug target to lower cholesterol, which the EHR-DNA resources make in principle, illustrates the importance of rare variants in the identification of pathways relevant to the whole population. Mendelian randomization studies are essential in evaluating whether markers, such as heart rate and HDL cholesterol, are causal for the disease of interest. Such genetic studies have questioned the role of heart rate and HDL cholesterol in the etiology of heart attack [11].

## 2.5 Cost effectiveness of innovation

The Institute of Medicine (U.S.), Institute of Medicine (U.S.), and Institute of Medicine (U.S.) (2011) shows that big data provide new opportunities for understanding the cost-effectiveness of existing and new interventions. Because of the ability to assess baseline risks in unselected general populations (usually a higher risk than those reported in trials), such 'real world evidence' is increasingly required by payers and regulators. As more data sources are linked, greater granularity of the care data (e.g., 67 different types of primary care 'consultation') may provide more accurate and more complete resource use data. For example, cost-effectiveness decision models can be developed before trials, reporting an estimate of the willingness to pay and pricing of a drug according to different trial benefits (relative risk reductions) applied to patients at various strata of risk [3].

## 2.6 Public health

Agrawal [1] assert that there are significant gaps in our ability to prevent the onset of and prolong life in, many of the most common cardiovascular diseases in the 21st century including atrial fibrillation, heart failure, and peripheral arterial disease. There are also

gaps in our ability to measure disease and model the impact of interventions in populations. Clinicians diagnose more specific entities than 'heart attack', 'CHD' or 'CVD,' yet conventional consented cohorts have lacked the statistical size or the phenotypic resolution to measure clinically relevant sub-types of disease [4]. Big data can study the conditions that clinicians diagnose to provide scalable, population-based, updatable measurements of new disease burden vital for the evaluation of alternative strategies of prevention. For example, significant data can be used to estimate the incidence and survival of the treatment-relevant sub-types of MI (ST elevation and non-ST elevation) [5].

## 3 ANALYSIS

To characterize the algorithms, a researcher requires a learning stage on a labeled dataset collection, which makes it possible to settle on the missing test record class [2]. In the learning stage, a researcher builds an order model to predict the data label class label using the values of its features. A specialist categorizes heart diseases as cardiovascular and cardiomyopathy disorders [2]. Coronary Artery Disease (CAD) is a noteworthy cardiomyopathy inconvenience subgroup which causes inability, genuine sickness, and even death, as the disease affects the supply of oxygen and blood to the heart muscles. The early indications of coronary illness involve torment in the focal point of the chest or a feeling of deadness, and palpitation. Different signs include blacking out fits or spells of feeling mixed up, and dyspnea on the effort.

Due to the vital importance of heart failure, the primary goal of many scientific pursuits is to make an accurate diagnosis of cardiac diseases. Analysts have gathered a lot of data while looking at CAD patients, and by preparing the information, they have uncovered how the main features highlighting cardiovascular issues are related (e.g., the measure of cholesterol, circulatory strain, and so on) and the probabilities of the occurrence of these diseases.

### 3.1 Dataset

¡A dataset constructed from the information collected from 303 random visitors (216 patients) to the Shaheed Rajaei Cardiovascular, Medical and Research Center was used to evaluate the effects of different demographic, clinical, and ECG features on the diagnosis of coronary artery disease. This dataset is publicly available and does not contain any patient identifiers. Demographics, such as age and sex, are also included in the dataset.

### 3.2 Method

I will use Random Forest machine learning for predicting the features to diagnose coronary artery disease. Random Forests (RF) is a prominent tree-based ensemble machine learning tool that is exceptionally data adaptive [3]. I will primarily be using the SciPy stack, especially pandas, matplotlib, seaborn, and sklearn. I use the `describe()` method, which gives me the basic statistics for every column in the dataset. This method will allow me to generate descriptive statistics that summarize the central tendency, dispersion, and shape of a dataset.

For a better visualization of the data, I will generate a histogram. Data visualization facilitates the interpretation and breaks down big data in a manner that is easily understandable.

**Table 1: Dastaset Description**

| Age | Weight | Length | Sex | BMI | DM | HTN | Current Smoker |
|---|---|---|---|---|---|---|---|
| 0 | 53 | 90 | 175 | Male | 29.387755 | 0 | 1 |
| 1 | 67 | 70 | 157 | Female | 28.398718 | 0 | 1 |
| 2 | 54 | 54 | 164 | Male | 20.077335 | 0 | 0 |
| 3 | 66 | 67 | 158 | Female | 26.838648 | 0 | 1 |
| 4 | 50 | 87 | 153 | Female | 37.165193 | 0 | 1 |

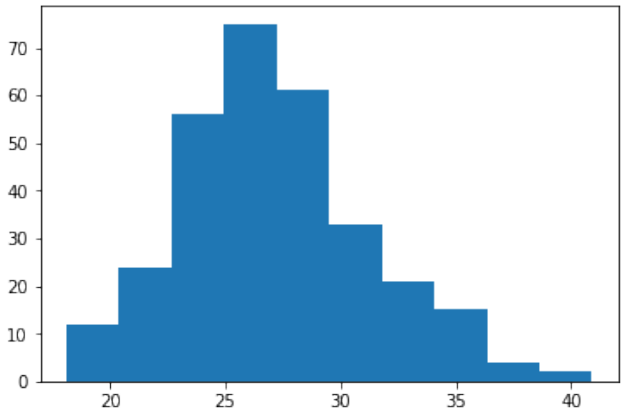


**Figure 1: Bar Plot**

```
cols = df.columns num_cols = df._get_numeric_data().columns cat_cols=list(set(cols) -
set(num_cols)) ##BBB, VHD have multiple levels rest are binary ###Cath is
predictor var cat_cols.remove('VHD') cat_cols.remove('Cath') cat_cols.remove('BBB')
cat_cols
```

```
['DLP',
 'CRF',
 'Obesity',
 'Poor R Progression',
 'Exertional CP',
 'Airway disease',
 'LowTH Ang',
 'Sex',
 'Nonanginal',
 'Diastolic Murmur',
 'Dyspnea',
 'Thyroid Disease',
 'Lung rales',
 'CVA',
 'CHF',
 'Weak Peripheral Pulse',
 'Atypical',
 'LVH',
 'Systolic Murmur']
```

**Figure 2: Encoding**

### 3.3 Data Preparation

Since this dataset includes string data, I will use encoding to transform the categorical features to a format that works better with classification by taking all categorical features that have only two levels and label-encode them to get binary features.

Then I will apply one hot encoding to our multiple level features, that will allow the representation of categorical data to be more expressive.

### 3.4 Measure of Variable Importance Ranking

An essential element of RF is that it gives a quickly calculable inner measure of the significance of a variable (VIMP) that can be used to

```
Index(['Age', 'Weight', 'Length', 'Sex','BMI','DM','HTN','Current Smoker',
       'EX-Smoker', 'FH', 'Obesity', 'CRF', 'CVA', 'Airway disease',
       'Thyroid Disease', 'CHF', 'DLP', 'BP', 'PR', 'Edema',
       'Weak Peripheral Pulse', 'Lung rales', 'Systolic Murmur',
       'Diastolic Murmur','Typical Chest Pain','Dyspnea','Function Class',
       'Atypical', 'Nonanginal', 'Exertional CP', 'LowTH Ang', 'Q Wave',
       'St Elevation', 'St Depression', 'Tinversion', 'LVH',
       'Poor R Progression', 'FBS', 'CR', 'TG', 'LDL', 'HDL', 'BUN','ESR',
       'HB','K','Na','WBC', 'Lymph', 'Neut','PLT', 'EF-TTE','Region RWMA',
       'Cath', 'BBB=LBBB', 'BBB=N', 'BBB=RBBB', 'VHD=Moderate', 'VHD=N',
       'VHD=Severe', 'VHD=mild'],
      dtype='object')
```

**Figure 3: Hot Encoding**

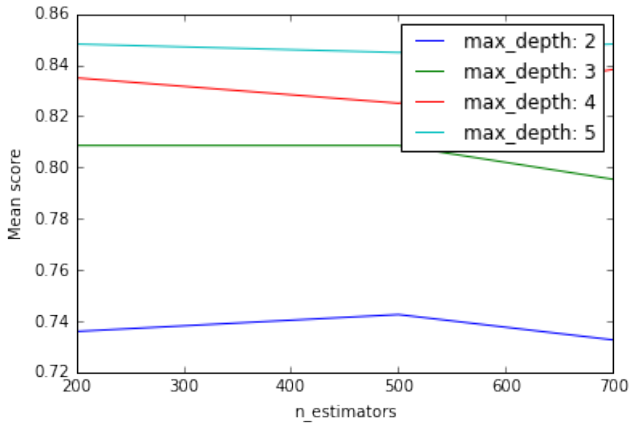


**Figure 4: Scores Attribute**

rank the factors. This is particularly valuable for high-dimensional genomic information. In spite of the fact that there are numerous fruitful applications using change significance, a drawback is that it is a ranking-based approach. Ranking is significantly more troublesome than the variable determination issue, which just tries to choose a group of factors that when consolidated are prescient, without forcing a positional structure. In any case, in view of the many-sided quality in organic frameworks, positional quality records in light of RF or RSF which consider relationship and communication impacts are still an immense change from univariate positioned quality records in light of the $t$-test's or Cox corresponding peril occurring when utilizing one variable at any given moment. Be that as it may, caution is required when deciphering any straight positioning since it is as a rule likely that various arrangements of feebly prescient highlights are mutually prescient. This seems, by all accounts, to be an uncertain issue of positioning, and further investigation is required. I will specify which dataset and variable I want to predict. I like to keep a parameter file where I specify data sources and such. This lets me create generic analytics code that is easy to reuse. After I have specified what dataset I want to study, I split the training and test datasets. Then I will scale the data, which makes most classifiers run better.

By checking the model's accuracy on the test set, as shown in the figure below, the accuracy score on the test by Random Forest attained 0.8852.

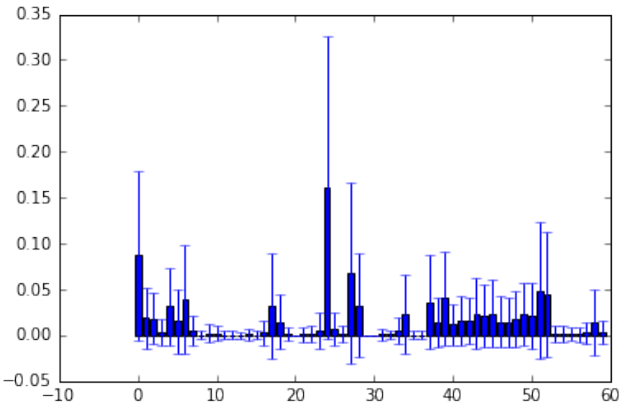


**Figure 5: Feature importance with their standard deviations**

**Table 2: Feature Importance Table**

| Importance | Std | |
|---|---|---|
| Age | 0.086779 | 0.092552 |
| Weight | 0.018672 | 0.032956 |
| Length | 0.018183 | 0.027928 |
| Sex | 0.003414 | 0.013603 |
| BMI | 0.031609 | 0.042215 |
| DM | 0.015427 | 0.034442 |
| HTN | 0.038735 | 0.059296 |
| Current Smoker | 0.005287 | 0.015965 |
| Ex-Smoker | 0.000587 | 0.004463 |
| FH | 0.002472 | 0.010216 |
| Obesity | 0.002333 | 0.008439 |
| CRF | 0.000534 | 0.004805 |
| CVA | 0.000469 | 0.004108 |
| Airway disease | 0.000474 | 0.003385 |
| Thyroid Disease | 0.001086 | 0.006045 |
| DLP | 0.000407 | 0.004046 |
| BP | 0.003296 | 0.013382 |
| PR | 0.031988 | 0.056958 |
| Edema | 0.014277 | 0.029568 |
| Weak Peripheral Pulse | 0.001758 | 0.007496 |
| Lung rales | 0 | 0 |
| Systolic Murmur | 0.001332 | 0.007931 |
| Diastolic Murmur | 0.001853 | 0.008916 |
| Typical Chest Pain | 0.00495 | 0.020264 |
| Dyspnea | 0.161378 | 0.165277 |
| Function Class | 0.006237 | 0.01781 |
| Atypical | 0.001798 | 0.008412 |
| Nonanginal | 0.067681 | 0.098911 |
| Exertional CP | 0.032299 | 0.056285 |
| LowTH Ang | 0 | 0 |
| Q Wave | 0 | 0 |
| St Elevation | 0.000976 | 0.005923 |
| St Depression | 0.005336 | 0.014798 |

```
clf = RandomForestClassifier(n_estimators= clf.best_params_['n_estimators'],m
ax_depth=clf.best_params_['max_depth'])
clf.fit(X_train, y_train)
y_pred=clf.predict(X_test)
print(accuracy_score(y_test,y_pred))
print()
print()
print('The accuracy score on test by Random Forest:{}'.format(accuracy_score(
y_test,y_pred)))
0.885245901639


The accuracy score on test by Random Forest:0.8852459016393442
```

**Figure 6: Accuracy Test**

## 4 CONCLUSION

This research uses a new dataset, comprising 38 features, together with data mining methods, to get useful results about this area of research. Afterward, the study chooses 16 features using an element selection algorithm and some common classification algorithms, and applies the proposed ensemble algorithm to the dataset. The research obtains the highest accuracy (88.51%) when both the element selection algorithm and the ensemble algorithm are used. Additionally, the study uses association rule mining methods to extract rules of high confidence from the dataset. Coronary Artery Disease (CAD) is among the primary causes of death in the world. Thus people must put measures in place to predict, treat, and control this disease. The use of big data mining techniques will help achieve this and, therefore, reduce the mortality rate of people with this disease. Additionally, not only should medical experts know the use of this technique, but the entire community, because heart failures happen at unexpected times.

The objectives of future studies should include adding other features, such as echo and lab echo data, to investigate the effect of these elements on CAD diagnosis and get a higher accuracy in predicting the progression of this disease. Furthermore, future works should use more algorithms and data mining techniques to improve the findings. Lastly, studies should extend the dataset with more sick people who can help in finding interesting results which may not be apparent for the patients in the dataset used here. Another aim of future research should be simplifying the use of big data mining methods to make them user friendly; this will make many medical facilities adopt the idea. Moreover, future studies should research other tools that can improve the big data mining technique's process, thus making it less costly and require less technological know-how.

## 5 RECOMMENDATIONS FOR FUTURE STUDIES

Many nations in the world have big data mining systems that research can use. Hence, to enable the research potential of these information sources, these countries have to centralize the resources that govern the research dataset. Some states have initiatives towards achieving this, although few tackle all the elements of this procedure. For example, the CALIBER platform, which is in the United Kingdom, joins a repository of big data mining phenotypes with curated registered linkages combining disease registry (Myocardial Ischaemia Nation Audit Project). Additionally, it joins primary care (Clinical Practice Research Datalink), death registry (Office of National Statistics), and hospital discharge (Hospital Episode Statistics) data for over two million adults with ten million individualsfi follow-up years. However, this resource does not offer tools for bidirectional relations with big data mining information sources. The United States National Human Genome Research Institute-funded consortium, the eMERGE Network, combines big data mining with a phenotype repository from multiple secondary healthcare systems, including text and imaging, joined to genotypic data for all candidates. Lastly, the Clinical Record Interactive Search system, which is at the Maudsley NHS Foundation Trust, NIHR Mental Health Biomedical Research Center and Dementia, and South London's Dementia Unit also help researchers. The Clinical Record Interactive Search system allows scholars to analyze secondary data, comprising clinical notes and other texts, through simple tools that help to identify patients meet specific criteria, and help develop text-mining algorithms.

Regarding this, the national big data mining portals can join the strengths of these projects by including: (i) Standards-driven types of equipment that enable reviewers to create interventional and observational studies; (ii) A countrywide catalog of contemporary big data mining sources which is metadata standards curated; (iii) Big data mining phenotype algorithms with an interactive thesaurus. By accomplishing this, the national catalog can support metadata integration and harvest from manually curated and external sources by researchers within a reproducible and standardized framework, and also offer a guide to the data content and access. In this way, users will be able to find sources of information that provide data both across and within disease areas.

The creation of big data mining datasets and algorithms should be implemented in a way that supports modification and reuse by other users, as well as the academic citation and credit that is appropriate. Creating this form of resource enables fostering an 'open source' approach to big data mining research in which researchers learn and collaborate with one another. It will eventually produce an advance in big data mining, more than any other research group in isolation can achieve.

## ACKNOWLEDGMENTS

The authors would like to thank Professor Gregor von Laszewski and his team for assistance in providing me the tools and knowledge to complete this project.

## REFERENCES

[1] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. 1993. Mining Association Rules Between Sets of Items in Large Databases. *SIGMOD Rec.* 22, 2 (June 1993), 207–216. https://doi.org/10.1145/170036.170072

[2] Roohallah Alizadehsani, Jafar Habibi, Mohammad Javad Hosseini, Reihane Boghrati, Asma Ghandeharioun, Behdad Bahadorian, and Zahra Alizadeh Sani. 2012. Diagnosis of coronary artery disease using data mining techniques based on symptoms and ecg features. *European Journal of Scientific Research* 82, 4 (2012), 542–553.

[3] Leo Breiman. 2001. Random forests. *Machine learning* 45, 1 (2001), 5–32.

[4] Chi-Ming Chu, Wu-Chien Chien, Ching-Huang Lai, Hans Bernd Bludau, Huei-Jane Tschai, Lu Pai, Shih-Ming Hsieh, Nian Fong Chu, Angus Klar, Reinhold Haux, and et al. 2009. A Bayesian expert system for clinical detecting coronary artery disease. *J Med Sci* 29, 4 (2009), 187–194.

[5] Thomas G Dietterich and et al. 2000. Ensemble methods in machine learning. *Multiple Classifier Systems* 1857 (2000), 1–15.
[6] Huikyong Lee, Jee-Yeon Noh, Yumin Oh, Youngdoo Kim, Jae-Woong Chang, Chul-Woong Chung, Soon-Tae Lee, Manho Kim, Hoon Ryu, and Yong-Keun Jung. 2011. IRE1 plays an essential role in ER stress-mediated aggregation of mutant huntingtin via the inhibition of autophagy flux. *Human Molecular Genetics* 21, 1 (2011), 101–114.
[7] Asha Rajkumar and G Sophia Reena. 2010. Diagnosis of heart disease using datamining algorithm. *Global Journal of Computer Science and Technology* 10, 10 (2010), 38–43.
[8] Yasamin Sahaf. 2011. *Comparing sensor modalities for activity recognition.* Ph.D. Dissertation. Washington State University.
[9] Samuel Sideman and Rafael Beyar. 1985. Simulation and Imaging of the Cardiac System. *State of the Heart* 4 (1985), 112–118.
[10] Zhiqiang Wang and Wendy E Hoy. 2005. Is the Framingham coronary heart disease absolute risk function applicable to Aboriginal people? *Medical Journal of Australia* 182, 2 (2005), 66–69.
[11] Harindra C Wijeysundera, Maria Koh, David A Alter, Peter C Austin, Cynthia A Jackevicius, Jack V Tu, and Dennis T Ko. 2017. Association of high-density lipoprotein cholesterol with non-fatal cardiac and non-cardiac events: a CANHEART substudy. *Open Heart* 4 (2017), 2–5. https://doi.org/10.1136/openhrt-2017-000731 arXiv:http://openheart.bmj.com/content/4/2/e000731.full.pdf

# A CODE REFERENCE

All code and files for this project can be found in the Github repository: https://github.com/bigdata-i523/hid339/blob/master/project/code

# Twitter sentiment analysis of the Affordable Care Act in 2018

Michael Smith
Indiana University
School of Informatics and Computing
Bloomington, Indiana 47408
mls35@iu.edu

## ABSTRACT

Gaining public sentiment on a topic of interest can be accomplished with modern big data technologies and the cloud empowered by programming languages such as python. This project encompasses social media mining of twitter, analysis of data through apache spark, and testing on a multimachine cluster. Benchmarks on the cloud infrastructure Digital Ocean was performed in 1 to 3 node clusters and various machine specifications were tested. The main spark program was a python script which processed over 15000 tweets associated with the affordable care act. Each tweet was processed through natural language processing techniques and its negative or positive sentiment was identified. Visualizations of the processed data were produced to show outcomes. An automated deployment of this analysis was produced through the configuration tool ansible.

## KEYWORDS

hid-sp18-707, Volume: 9, Chapter: Twitter, Status: 100.
Apache Spark, Sentiment Analysis, Ansible, Python

## 1 INTRODUCTION

The current political climate of the United States is divided on many important issues. There is a disconnect between the motivations of the politicians of today and what is deemed important to the American people. The affordable care act (ACA) also known as Obamacare has been a target of the GOP, however it is uncertain if that sentiment is shared by most Americans. With a law that affects millions of Americans it is often difficult to gauge how the American people feel about the current health care law. It is the goal of the project utilize big data technologies to gauge how Obamacare is viewed through social media.

## 2 PROJECT OVERVIEW

The project is designated into four phases as seen in figure 1. Interface with twitter to acquire data for future analysis, the goal of this to accumulate a substantial number of tweets that provide affordable care act sentiment. The second phase involves the development of an ansible playbook that will setup a three node cluster of virtual machines with all the environment components necessary to deploy spark, fetch files, and run python scripts all in an automated process. The third phase in the project entails development of a spark program that will run successfully on a cluster and analyze the twitter data which will provide an overview of the sentiment. The final phase involves testing the deployment on a cloud infrastructure with single and three node benchmarks, various node specifications such as the amount of cores and allocated memory will also be benchmarked.

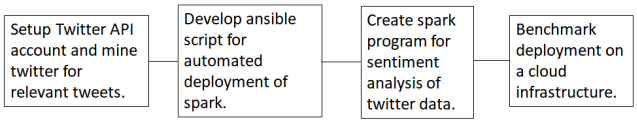


**Figure 1: Project Overview**

## 3 DIGITALOCEAN

The benchmarking the deployment on a cloud infrastructure is one of the primary objectives in this project. The cloud service DigitalOcean, Inc was utilized for software deployment [12]. While they are an American company, they possess datacenters in New York, San Francisco, Amsterdam, Singapore, London, Frankfurt, Toronto and Bangalore. They offer a range of virtual machines with RAM allocation from 1 to 192 GB, vCPU from 1 to 32 cores, and SSD storage from 25 GB to 3840 GB. Their service allows the resizing of existing virtual machines to different specifications allowing ease in comparing performance. For all virtual machines the OS ubuntu 16.04.4 was utilized.

## 4 TWITTER

The social media platform Twitter is a service utilized by millions of users who interact through micro-blogging in 280 characters or less [15]. While the messages are brief the breadth of information available is an ideal resource to perform a sentiment analysis of a given topic. Twitter offers an API that allows developers to establish an account to collect tweets of interest [17]. It is subdivided into two components: a representational state transfer REST and streaming API. The REST component allows the ability to query a twitter users account or modify an account with extra permissions. The streaming API delivers tweets based on selected search terms and will deliver data in real time. Data is returned to the user in a JavaScript Object Notation format or JSON. Some notable aspects of the JSON schema include user id, text of the tweet, user location, and timestamp. The streaming twitter API was selected to gain real time data regarding the topic of interest.

### 4.1 Twitter mining

In order to acquire unique tweets, a python script was developed using the twython library which is a python wrapper for the twitter API [9]. The code for this script was adopted from Joel Grus [3]. The script fetchtweets.py will first authenticate with twitter through unique consumer keys and access tokens. Secondly, will listen for tweets that contain keywords and capture data objects and store them locally in a JSON format. The keywords utilized for selecting tweets contained any of the following phrases: obamacare,

ACA, or various combinations of affordable care act. An additional requirement for data aquisition relied on setting language key value to be english. The script was triggered to shut off after 1000 tweets and was repeatedly ran over a period of two weeks. Once the dataset contained over 15,000 tweets the data acquisition phase of the project was concluded.

## 5 AUTOMATED DEPLOYMENT

Deployment of an apache spark program can be a tedious process and sensitive to potential errors depending on the computing environment of the user. This creates a difficult hurdle for collaboration and repeatability. In order to automate the setup and create a reliable application, the technology ansible was utilized for system configuration [1].

### 5.1 Ansible

Ansible is an open source software used for automated provisioning of computing environments. This tool is beneficial in assurance that each node is setup with the appropriate dependencies and environment for a consistent execution. Ansible executes a series of commands from a file in the YAML format which is referred to as a playbook [5]. A configuration will be checked to see if it is already present on the target VM, if not the play will be installed otherwise the command is skipped. The YAML format is human readable and utilizes indentation similar to python. In order to successfully run a playbook on a cluster of nodes or a single node. It is vital that a secure connection be made, most commonly through a secure shell key also known as an SSH key. SSH ensures security between nodes so that configuration is only occurring where it is allowed, moreover the convenience of a secure connection without a password. For the purpose of this project, an ansible playbook was developed to install all dependencies and programs necessary to automate a spark cluster configuration.

### 5.2 Ansible playbook design

Deployment on Digital Ocean required a manual input of ip addresses and username within the groups of either Master or Node section in an inventory.txt. The ansible playbook calls the inventory file along with the main.yml to initiate deployment. The main features of the main.yml are detailed in the process of configuration below.

- **Python**: In order for ansible to execute its yaml file which is written in python, it needs to be installed on the target vm. The command getfacts will receive information about the vm including whether or not python has been installed. If it is not present this play will execute an installation of python minimal. Code for this play was utilized from discussion in the following github [2].
- **Variable Assignment**: The ip address of the master node is utilized in multiple plays during deployment. To prevent the requirement of the user to manually set this value within the playbook. All ip addresesses of hosts were gathered and the variable ENV was created and assigned only the master ip address through the set facts command.
- **Java**: Apache spark runs on top of Java, thus all nodes require its installation. Selecting the correct Java version is important as versions of java below 8 will not run on Spark 2.2.0 or higher. Therefore java 8 was installed which is sufficient for spark 2.3.0. Code utilized from the github repository dsugden [16].
- **Apt**: Apt packages are then installed including python-pip for library installations, python-dev, and also python-tk. A follow up script will then verify that python-pip is upgraded to the latest version.
- **Spark**: Spark 2.3.0 is then downloaded and extracted using the geturl command.
- **Python packages**: Python library installations via Python-pip added the necessary dependencies for the spark program are installed including: NumPy, TextBlob, Matplotlib, and WordCloud.
- **wget**: The twitter.json dataset and main.py script is then fetched through wget with ansible shell commands. Conditional statements within the shell script will first verify the presence of files to prevent replicated files, this adds convenience to repeated playbook runs.
- **start-master.sh**: Host specific plays on the master node begin spark in standalone mode.
- **start-slave.sh**: Slaves are then started and allocated the proper ip address configurations to connect with the master.
- **spark-submit**: The final command in the playbook initiates the spark-submit command to run main.py.

## 6 APACHE SPARK

As the size of data has grown far beyond the processing capacity of a single machine, a need developed to process large data sets in a timely and efficient manner. It has become commonplace for data processing to occur in a cluster computing environment which is referred to as high performance computing or HPC. A well known technology Apache Hadoop written in Java is capable of batch processng data over a cluster of machines with a programming model known as MapReduce [4]. Unfortunately Hadoop possess some disadvantages such as the inability to perform streaming processing, it uses only Java for writting applications and processes data in a disk based manner which leads to a cost performance with big data [14]. Apache Spark was created at UC Berkeley AMP Lab and was created in such a way that helped overcome some of the shortcomings of Hadoop. Spark applications can be written in several languages such as R, Java, Scala, and Python which adds flexibility for the user to write spark programs in the language of their choosing. Instead of disk based processing Spark processes data in-memory which will make data easier to access and increase overall speed. The core functionality in spark lies within the RDD or resilient distributed dataset.

### 6.1 RDD

An RDD is a collection of objects with unique features that make it advantageous for big data applications. They are immutable which means that when performing a computation on an RDD, it is not modified but a new RDD is created as the computed output [7]. RDDs can be split into partitions for distribution amongst a cluster of machines with different tasks acting on it. RDDs are considered

to be resilient or fault tolerant which affords spark the ability to reprocess a partition if a worker node fails in the cluster. There are two ways that an RDD is acted up which is an action or a transformation. Spark is referred to as a lazy evaluator, which means it will apply a transformation until an action is called. This methodology does not waste resources and only does enough to satisfy the action when called. When an action is completed nothing is saved, this can create issues when invoking multiple actions on the same RDD. In order to increase effiency Spark allows caching of an RDD or allow it to persist across a cluster in memory or on disk. All of these features of RDD enhance the cluster computing functionality of Spark.

### 6.2 Spark Architecture

The main spark program is run in the driver which creates RDDs, performs necessary actions on the data and initiates a sparkcontext. The sparkcontext will connect to a cluster manager which can be a Mesos, YARN, or sparks built-in standalone cluster manager, these will assist resource management on the cluster [14]. The standalone client mode was selected for this project, its driver program runs on the master node and it will schedule jobs based on a first in and first out order FIFO. It is the responsibility of the driver to split the applications into sets of tasks to send to the cluster manager which will launch executors on worker nodes for processing. The processed result is sent back to the driver through the cluster manager.

### 6.3 Core components of Spark

Spark can be categorized into several primary components that serve different functions within spark such as Spark Core, Spark SQL, Spark Streaming, MLlib and GraphX [7].

- **Spark Core**: Spark core is the primary framework with APIs in Scala, Java, Python and R.
- **Spark SQL**: Spark SQL enables the usage of SQL queries on data frames within a spark program.
- **Spark streaming**: Enables the batch processing through the utilization of repeatable code through sources such as HDFS, Flume and even twitter. While the dataset is from twitter, spark streaming is not utilized for this project so that accurate controlled bench marking on varied cluster allocations can occur.
- **MLlib**: MLlib is a library enabling the application of various machine learning algorithms on a dataset for training and testing.
- **GraphX**: A library for graph computation. The latest version of Spark (version 2.3.0) was utilized for deployment across all nodes.
- **SparkR**: SparkR provides a frontend for R to interface with Spark.

## 7 TECHNOLOGIES

Spark and ansible are the two primary technologies utilized in this project but other technologies were use for specific uses throughout the project such as python and several libraries and modules described below.

### 7.1 Python

Python is an object oriented programming language, it is considered a high level programming language because it has to be processed by an interpreter prior to running [13]. It is more human readable than other programming languages and is a valuable tool for processing data. Writing in python can be ran in an interactive mode useful for testing code or and an entire python file can be processed at once in script mode. The core python script of this project main.py was ran on the spark cluster.

### 7.2 TextBlob

Textblob is a python library for processing data consisting of text [8]. In addition to sentiment analysis this library also has uses in tagging parts of speech, translating languages and classification through Naiive Bayes and Decision Trees. The sentiment analysis functionality of this library will be the primary tool used for this project.

### 7.3 Matplotlib

Matplotlib is a python library used for data visualization [6]. It utilizes NumPy to plot 2D arrays in python, it offers numerous options regarding the plot type, use of color, and other functionalities.

### 7.4 Numpy

Numpy is a python library for applying mathematical functions on top of arrays of varying dimensions [11]. It is more efficient in processing data than pythons standard list feature making it superior option for handling larger datasets.

### 7.5 Wordcloud

Wordcloud is a python library created by Andreas Mueller that produces interesting visualizations of data from a text file [10]. All words are extracted from a document and given weight based on the number of occurrences of the word. The project image then precomputes the number of places for rectangles in the image and randomly samples words for placement. Bigger words will stand out more due to their high frequency leading to interesting insights into the text data.

### 7.6 Re

The regular expression module for python allows processing strings by locating unique sequence of characters in order to add, modify or delete the string. This tool facilitates modifying strings at scale in an efficient manner.

## 8 SPARK PROGRAM DESIGN

In order to process the twitter.json dataset the main.py invoked a number steps with some key functionalities. Spark allows user defined functions UDF to develop custom actions on RDDs which was utilized in this program. A number of UDF's were generated in order to effectively process RDDs with the integration of python libraries. The UDFs process the data by converting all text data from unicode to ascii. This prevents future errors when writing the various outputs to textfiles. Another UDF encompassed processing text with regular expression which removes all usernames, hashtags and hyperlinks of tweets. Finally leveraging textblob within

a UDF allowed the sentiment analysis of the cleaned tweet and generates a spark dataframe containing the cleaned tweet with its corresponding polarity, this was collected and written to an output file called twittersentimentlist.txt. Development of the wordcloud afforded some unexpected issues when dealing with twitter data. A significant amount of tweets in the dataset were from retweets which inflated the weighting of word frequencys in those tweets. To overcome this hurdle, all retweeted text could be filtered by utilizing the distinct operation in Spark. This removes all text that is identical and will filter it down to simply one instance of a given retweet. This RDD was collected into a text file and passed to the Wordcloud function to generate an image which is saved as acatextcloud.pdf. The final analysis involved plotting a histogram of all polarity values that were calculated, those values were selected and stored written to a list and text file called polaritylist.txt. The list was plotted with matplotlibs histogram function to give insight into the twitter sentiment, the plot was saved as polarityhistogram.txt. An overview.txt file was also generated which gives a breakdown of the number of tweets, retweets, the most popular tweet, and a mean of all calculated polarities in the dataset.

## 9 RESULTS

What occurred in the spark program is referred to as natural language processing (NLP) which is defined in the simplest of terms as enabling a computer to gain meaning from human language. This can be used in a number of different ways including sentiment analysis. The polarity scoring in sentiment analysis is on a range from -1 to +1 which is a negative to a postiive sentiment respectively. Observing the final output of the distribution of scores found in the histogram in Figure 2, it is clear that sentiment regarding ACA is not as negative as Washington makes it out to be.

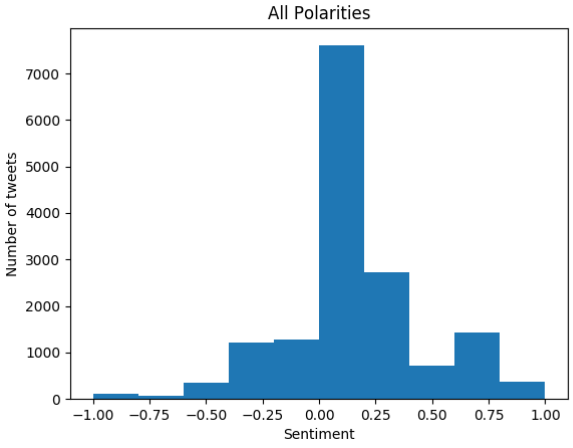


**Figure 2: Result of sentiment analysis of all tweets.**

While the distribution sentiment is seen to reach both ends of the spectrum, the majority of sentiment lies predominantly in the slightly positive range. When calculating the average sentiment value of all polarities a value of 0.13 was observed showing consistency with the histogram visualization.

## 10 WORLDCLOUD

Here is a word cloud at Figure 3

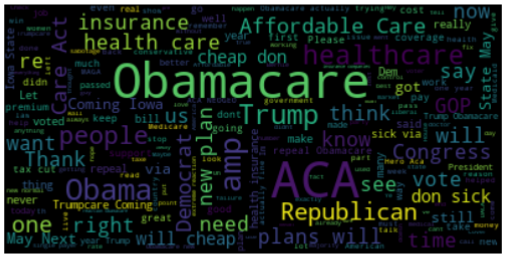


**Figure 3: Wordcloud output of Analysis.**

## 11 BENCHMARKS

The program was tested on 1 node and 3 node clusters through DigitalOcean service. The programmed was clocked using pythons time library to accurately assess completion time. In addition to different nodes, the flavor of the nodes was modified to see how improved computing would affect benchmarks. Three variants of each node was assessed with the following specs, Table 1 outlines the different flavors tested, CPU was tested on 1, 2, and 4 vCPUs with an increased memory of 1, 2 and 4 GB respectively. Each configuration was tested five times and the average was reported. The single node cluster saw benchmark average times of 90.18, 71.19, and 29.75 seconds on the small, medium and high flavors respectively shown in Figure 4.

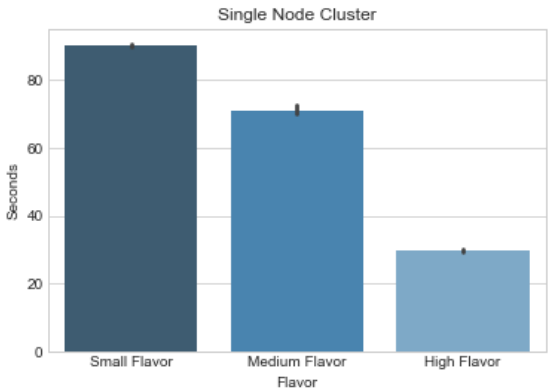


**Figure 4: Benchmarks of all flavors on a single node cluster.**

The three node custer had average times of 489.77, 277.86 and 210.05 seconds on small, medium and high flavors as seen in Figure 5. The relatively small size of the dataset 112 Mb will not see a benefit in performance moving to a cluster which explains why the benchmarks decreased in efficiency when moving to three nodes. The limiting factor in speed became the network between nodes. In order to limit this effect, all nodes where setup on the same datacenter located in New York, however a decrease in performance was still apparent.

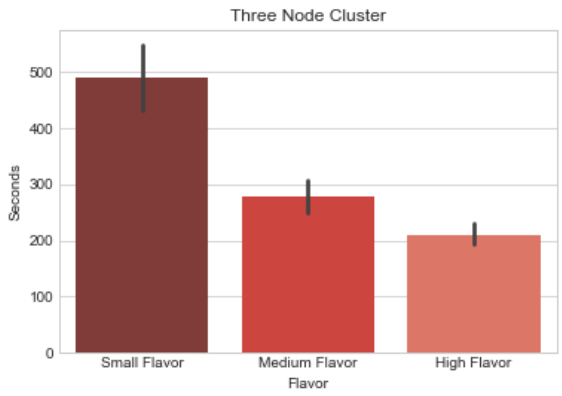


**Figure 5: Benchmarks of all flavors on a three node cluster.**

**Table 1: Specification of each flavor tested**

| Specifications | Small Flavor | Medium Flavor | High Flavor |
|---|---|---|---|
| vCPU | 1 Core | 2 Cores | 4 Cores |
| RAM | 1 GB | 2 GB | 4 GB |

## 12 CONCLUSION

Thie project details a successful deployment of apache spark on a cloud infrastructure. The deployment was automated with ansible configuration management tool. A spark program was designed that can process tweets orginally fetched through the Twitter API. Through natural language processing the textual data was cleaned and polarities were calculated for over 15,000 tweets. The polarities were visualized in a histogram and a wordcloud to gain insight into public impressions. A general sentiment of the affordable care act was gained from the results showing a distribution of sentiment that was net positive. The program was benchmarked in single and three node clusters with multiple flavor specifications in the digital ocean infrastructure. While improved flavors improved benchmarks, single node performance proved to be superior to the three node counterpart. A likely reason for this lies in the fact that the small size of the dataset was sufficiently ran in a single node cluster. Some limitations in this methodology is the reliance on the accuracy of textblob. Sample size is also relatively small, taking a snippet of tweets from a two week period in 2018 is not enough to capture sentiment in an effective manner. Mining over longer periods of time would give better insights as sentiments are subject to change over time. For future work, it would be beneficial for optimization of code in the spark program to improve cluster performance and development of a spark streaming component for real-time processing of tweets. Moreover testing sentiment with machine learning algorithms and benchmarking accuracy against textblob would be a goal for future work.

## 13 AUTHOR BIOGRAPHIES

Michael Smith is a senior quality control analytical chemist at Creosalus Inc in Louisville, Kentucky. Michael possesses a MS in pharmaceutical sciences and a BS in Biology from the University of Kentucky. He will obtain his MS in Data Science program from Indiana University in May 2018. His current interests are python programming, data analytics, and spending time with his wife and children.

## ACKNOWLEDGMENTS

The author would like to thank Dr. Gregor von Laszewski for his support and suggestions to write this paper. Thank you to all of the Teaching Assistants who promptly answered my questions. Having little experience in programming and even less working in the linux environment made this course a very challenging but equally rewarding experience.

## REFERENCES

[1] Ansible. 2018. Ansible. Webpage. (April 2018). http://docs.ansible.com/ansible/latest/index.html
[2] William De Groot. 2016. ansible bootstrap ubuntu. Webpage. (Nov. 2016). https://gist.github.com/gwillem/4ba393dceb55e5ae276a87300f6b8e6f
[3] Joel Grus. 2015. Data Science from Scratch. Webpage. (Oct. 2015). https://github.com/joelgrus/data-science-from-scratch/blob/master/code-python3/getting_data.py
[4] Hadoop. 2018. Hadoop. Webpage. (2018). http://hadoop.apache.org/
[5] Lorin Hochstein. 2015. *Ansible Up and Running*. O'Reilly.
[6] John Hunter and Michael Droettboom. 2018. Matplotlib. Webpage. (March 2018). https://pypi.org/project/matplotlib/2.2.2/
[7] Holden Karau and Rachel Warren. 2017. *High Performance Spark*. O'Reilly.
[8] Steven Loria. 2018. textblob. Webpage. (Jan. 2018). http://textblob.readthedocs.io/en/dev/
[9] Ryan McGrath. 2017. Twython. Webpage. (Aug. 2017). https://pypi.org/project/twython/
[10] Andreas Mueller. 2018. wordcloud. Webpage. (Jan. 2018). https://github.com/amueller/word_cloud
[11] Numpy. 2018. Numpy. Webpage. (2018). http://www.numpy.org/
[12] Digital Ocean. [n. d.]. Digitalocean.com. Webpage. ([n. d.]). https://www.digitalocean.com
[13] Python. 2018. Python.org. Webpage. (2018). https://www.python.org/
[14] Sandy Ryza, Uri Laserson, Sean Owen, and Josh Wills. 2015. *Advanced Analytics with Spark*. O'Reilly.
[15] Statista. [n. d.]. Number of monthly active Twitter users in the United States. ([n. d.]). https://www.statista.com/statistics/274564/monthly-active-twitter-users-in-the-united-states/
[16] Dave Sugden. 2015. vagrant ansible ubuntu oracle java8. Webpage. (April 2015). https://github.com/dsugden/vagrant-ansible-ubuntu-oracle-java8/blob/master/ansible/roles/java8-oracle/tasks/main.yml
[17] Twitter. 2018. Twitter. Webpage. (2018). https://developer.twitter.com/en.html