# Introduction

University of Michigan scans the IP space daily with a variety of tools. It publishes snapshots daily over Alex Top Million website and known X.509 certificates. The dataset provides the technical specification of each server (by ip address) such as the spec for each open port and OS. The size of the dataset is around 900GB of JSON. A big data solution is required to process such volume. This report will present a cloud based big data analysis system and proposing a big data solution to address the key challenges proposed.

The report will be divided into 2 section:

1. To analyze the sample dataset, we implemented a big data analysis platform. In this section of the report, we will present an overview of the implementation as well as demonstrate some of the key concerns behind each design decisions.
2. In the second section, we proposed a big data system for handling this dataset. It will be responsible for generating regular report and integrate with realtime events. We will present a high-level solution architecture for the system and explain our strategy of handling each proposed challenge.

Execution summary and code snapshot for data analysis platform is going to be included in appendix.

# Big data analysis system

The Censys dataset [1] is around 900GB and the given sample dataset is about 900MB. Given the fact, the average memory size of a personal computer is around 8GB. It is rather slow to analyze the sample dataset and nearly impractical for processing the full dataset on a single machine. (It is only possible when a programmer implemented a solution which handles memory effectively, however, even that the program will still take days to execute a single query due to the nature of data). Thus, we implemented a cloud based cluster to analyze given dataset.

## Solution Overview

Figure 1 illustrates the architecture overview of this data analysis system. The implementation is based / deployed on AWS.

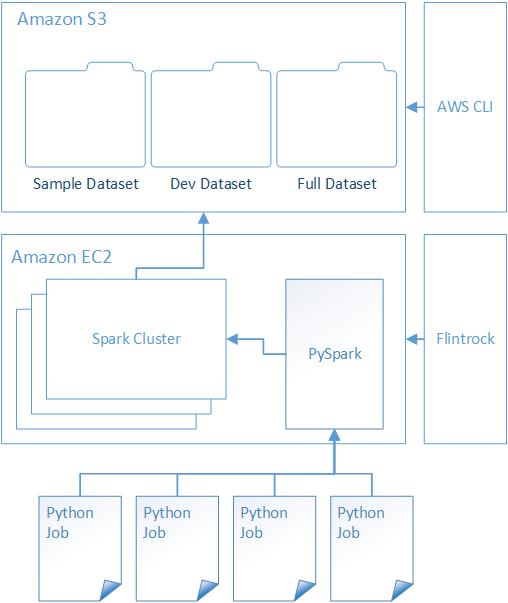


Figure 1 Solution Overview

**Data Repository** (Amazon S3)

We choose Amazon S3 as data repository. There are a few reasons behind this choice.

* Network – the analysis component of the solution is based on AWS. To load the data on to cluster with lowest latency, S3 is the most reasonable solutions. On the other hand, source data is provided on S3. It is a lot faster to transfer huge dataset within AWS’s internal network.
* Easy to use – S3 provides both GUI and CLI interface to manage it. There is no need to setup another cluster in compare with Hadoop. The overall task required for this solution is very limited.
* Pricing – S3 storage is relatively cheaper compare to other services.

**Data Analysis Cluster** (Flintrock + Spark + EC2)

We use spark cluster to analyze the given dataset. There are a few technologies behind this.

* Flintrock - It is a command-line tool for launching Apache Spark clusters [2]. It provides a fast way for us to manage spark cluster on EC2.
* EC2 – elastic clouds provided a scalable and economical platform for developers to launch experiment / production cluster.
* Spark Cluster – Distributed cluster to process large volume data via MapReduce.
* PySpark - The Spark Python API (PySpark) exposes the Spark programming model to Python [3].

There are a few motivations behind this design.

1. **Performance** – as mentioned previously, the given dataset is nearly impossible / impractical to be processed on a single computer. It needs a distributed system to increase the performance.
2. **Dedication** – Spark is designed to handling large scale MapReduce problem for big data in compare with MPI (Message Passing Interface) [4] and Globus [5].
3. **Scalability** – Developer can easily add a new / remove slave node to boost the performance of the cluster on EC2. For example, the development cluster for this exercise only contains 5 slave’s nodes while the experiment cluster contains 19 (overall 20 working nodes plus master).
4. **Easy to use** – with the help of tools such as Flintrock, launching / destroying analysis cluster becomes relatively easy.
5. **Economical** – Developer can destroy the whole cluster if no longer used. It is a lot cheaper than buying physical computing equipment.

**Job / Query** (Python)

We use Python/Pyspark to interact with Spark cluster. There are a lot of other options such as Scala/Java/C#. We choose Python due to its popularity and simplicity.

**Cluster Spec**

Here is the key spec of cluster that has been used for experiment. We attached detail configuration into appendix.

|  |  |
| --- | --- |
| Name | oxclo-sc |
| Operation System | ami-30041c53 # Amazon Linux, ap-southeast-2 |
| Region | ap-southeast-2 # Sydney |
| Instance Type | m3.medium #1 Core – 4 GB memory |
| No of Slaves | 19 |
| Spark | 2.2.0 |

## Challenge outline (Q&A)

1 – **What big data framework / algorithm did we choose?**

As mentioned in previous section, the system adapt spark as its foundation of processing given dataset. MapReduce is the algorithm backbone of the implementation.

2 – **Which cloud infrastructure?**

We choose to use AWS to hosting our solution due to following reasons:

* **Simplicity** – Launching spark cluster on AWS is relatively easy with the help of developer tools such Flintrocks.
* **Performance** – AWS provides decent infrastructure for data transferring, CPU utilization.
* **Economical** – Developer can switch off the instance when no longer needed.

3 – **How we are going to scale this?**

To answer this question, we need to analyze with the core technologies that has been chosen.

* S3 Bucket – S3 is the on-demand storage services provided by AWS. It scales up naturally if developer is willing to pay.
* Spark + EC2 – Spark cluster solves the scalability issue by enabling adding/removing slave nodes into cluster. The cluster becomes extremely scalable with the help of EC2. Developer can add / remove a new slave instance to the cluster with very minimum amount of work.

4 – **How are we going to efficiently process the data?**

MapReduce is decent algorithm which solves the performance issue in a distribute system. Spark compiles the code into a distributed execution plan and coordinates the execution automatically. In our example, the lambda calculations for each task are shared across cluster. Thus, in theory the more slave nodes that we add into cluster the better performance we will achieve.

5 – **What language did we use to process data?**

We choose Python because of its simplicity in handling data compare to traditional procedure oriented language such as Java / C#. We also choose python because it has better support in data analysis and AI system. For example, there are more options for python when dealing with Machine Learning comparing to R or Scala.

# Big Data System

We created a cloud based data analytics engine in previous chapter. In this chapter, we would extend the platform so it can provide continuous analytics. The new system will need to support following features.

1. Capable of consuming the stream of real-time updated data.
2. Identify changes in the system,