5-2020

DETECTING APPLICATION ANOMALIES: MACHINE LEARNING APPROACH

Lakshmipriya Thaduri
lakshmithaduri@gmail.com

Detecting Application Anomalies: Machine Learning Approach

by

LakshmipriyaThaduri

A Starred Paper

Submitted to the Graduate Faculty of

St. cloud State University

in Partial Fulfillment of the Requirements

for the Degree of

Master of Science

In Information Assurance

May, 2020

Starred Paper Committee:
Abdullah Abu Hussein
Lynn Collen
Balasubramanian Kasi

Abstract

In the modern era, the world has completely relied on software technology. As software applications became highly demanded, security concerns have arrived. Application security has become one of the chief concerns where companies have to protect their systems from vulnerabilities. Various other securities include mobile or end-point security, operating system security, and network security. All these security categories are intended to protect their users and clients from malicious intents and hackers. Application security became a prime requirement. Security risks of the applications are enveloped and lead to a direct threat to the available business. All the application vulnerabilities take the advantage to compromise the software application security. Once a flaw has been found, and private data access is determined, the attacker will have the capability to exploit the software application vulnerability to facilitate cyber crimes. The confidentiality of the data, availability, and integrity of resources are targeted by cybercrimes(“What is Application Security?” 2019). Overall, more than 13% of the reviewed sites were compromised with the web application security vulnerabilities, and they are not completely extinct even with the traditional security methodologies(*Application Security Vulnerability*, 2014). In order to resolve these numerous common security issues, few of the detection, remediation, and prevention techniques are to be used, which includes defensive programming, sophisticated input validation, dynamic checks, and static source code analysis. In this paper, the runtime environment framework has been introduced. This research study extracted a few publications. All the publications considered various approaches to resolve the issue. In this research paper framework,machine learning is utilized for training and predicting the output. Firstly, a sample java code is executed in various CPU cores, and the generated output files are collected. These output files are then used to train machine learning. Machine learning results are then compared with actual output for the decision statement.

Table of Contents

List of Figures

Figure **Page**

Chapter I - Introduction

Introduction

This project focuses on detecting the vulnerabilities in the existing software application code by considering the metrics of the three distinct Linux machines, train machine learning, and predict the result in order for decision statement.

Problem Statement

The objective of this research project is to detect software application vulnerability using machine learning. As my interest inclined towards the development field, and as a developer, it is beneficial to have exposure on how to detect and able to find a solution to resolve the vulnerability issues using machine learning.

Nature and Significance of the Problem

In our daily lives, software applications play a vital role. Irrespective of the location, whether at the workplace or home, the usage of software for various purposes, which includes communicating with people, staying advanced with the activities happening around the globe, which serves as entertainment, doing work, and much more. Besides, they are also real threats in terms of security. Though the security has tightened its privileges, hackers still hit upon new customs to bypass security resistance. In addition, there is a high probability of performance issues when resources go down. So in order to track the organization's response to certain

challenges, the solution to detect various described problems are monitored and hence possible solutions are been constructed to address these issues.

In the current trend, applications make huge money. According to the researchStatistics, claims that the worth of applications market range $189B by 2020. Back in 2017, the availability of Google Play Store is 2.8 million. Whereas in the Apple app store, it has crossed 2.2 million. Other than smartphones, tablets, pods, and other devices prolong to advance, and the apps count has increased; at the same time, thousands of websites and apps are createdon a daily basis. Excluding the benefits, the apps also pose challenges. In specific, the electronically connected devices with the software application installed have become an objective for hackers. They actively look for new and utilize old techniques to steal, modify, and delete private and business data. Research Akamai says, in 2017, count of attacks on applications grew by over 60%, and about 75% of security risks were application breaches by Alert Logic.

Hence in order to protect the data from risks and breaches, application security has to be considered primarily. In consideration of application creation and release, developers must continue to monitor, detect, secure, and prevent vulnerabilities. So, there should be effective methods utilized to detect the bugs in software applications. In this paper, machine learning is used to train, construct an algorithm based on the given data as input, and predict. With machine learning, the systems are trained with data, identify patterns, and show results to make decisions with the least human involvement.

Objective of the Research

- The main objective is to detect application anomalies using machine learning.
- Train and predict the output from machine learning.
- In an attempt to catch the post-deployment phase anomalies.

Project Questions/Hypothesis

1. Will machine learning help to detect software application anomalies?
2. How can Machine learning be trained? How can it be predicted?
3. Does machine acts intend?

Definition of Terms

- *Machine Learning:* Machine learning is a way of analyzing data that automatically analyzes the model building. In other words, machine learning is defined as a science of training computers to perform by itself without being overly programmed. It is a branch of artificial intelligence. In this field, the systems are trained with data, identify patterns and show results to make decisions with least human involvement(“Machine Learning | Coursera,” 2019).
- *Java:*Java is one of the computer programming languages. Java is fast, secure, and reliable. The code written in Java is platform-independent. The code needs to be compiled once and executed many times irrespective of OS. The code, when compiled, is converted into binary code, which is the combination of 0’s and 1’s. In detail, the.java file is compiled to produce a .class file, which is basically a compiled code. In this research study, various OS platforms

have been used. A small piece of java code will be executed in distinct OS's then compared in order to test the anomalies.

- *IP Fragmentation*: IP is an internet protocol, and fragmentation is a process of breaking down packets into small chunks of data (fragments), in such a way that the resulting piece is allowed to pass through a link with smaller MTU (maximum transmission unit) than the original packet size.
- *Vulnerabilities:*It is a weakness of the system; it is performed by an attacker by accessing unauthorized data in the system or gaining sensitive information or any unauthorized action on a computer.
- *Anomalies:*Anomalies are basically problems. Poor planning or un-organization (un-normalized) of data in the database is the main cause ofanomalies.
- *Artificial Intelligence:*It is a contrast of natural intelligence. Artificial Intelligence is the intelligence of machines; sometimes, it is referred to as machine intelligence. Research says that without human involvement, the decisions are taken by machines, and the environment is perceived by any device that maximizes the probability of achieving the goals("Artificial intelligence," 2019).
- *Deployment phase:*The final stage of SDLC is the deployment phase. The product which has been developed is now ready for real-time use in the production environment. The product once it is deployed and successful, all end users are allowed to utilize the benefits of the product(*Deployment Phase in SDLC - Video & Lesson Transcript | Study.com*, n.d.).

- *Breaches:*A security breach is known as a security violation. It is incident fallouts to the activities which include unauthorized access of services, data, networks, applications, or any

Summary

So far, the main objective of the research project, along with the purpose and importance of application security, will be described in this chapter. In the following lessons, the reader will get more idea ofhow distinct researchersapproach in order to resolve the issue. In addition, the following contents will include a literature review on the problem and a literature review of the solution.

Chapter II - Background and Literature Review

Introduction:

This chapter describes the information about background research related to the problem, introduction, challenges, and causes of vulnerabilities.

Background literature related to the problem:

Introduction:

As we know, software application vulnerability remainsa serious issue. Numerous companies, organizations, and end-users faced the software vulnerabilities issue. For a few years, it is observed that there were several vulnerability attacks reported which occurred with high distressing effects on users. With this, the need to focus on software vulnerability detection implementation tools and techniques has raised. Due to the necessity of software security detection tools, many software developers have invented various tools andmethods which detect the vulnerabilities in the system and also report the issues which cause a threat to system and user data. (Amankwah, Kudjo, &Antwi, 2017) In 2003, the CERT/CC (Computer emergency response team Coordination Center) reported that there were about 6.66 US dollars economic loss caused by the intrusion attacks. And still, the value of the numbers has increased with the time passage. The real scenario relevant to the economic loss is, in 2007, the total vulnerabilities are 7236, and by the end of half year in 2008, the total vulnerabilities in the system incremented to 4110 (Aboud, 2009).

The term vulnerability has been described in a broad sense as it is an activity that violates any security policy. The violation activities can be occurred due to any errors in the software code or might be due to the weak security rules. In theory, all systems have anomalies, but the vulnerability effect relies on the damage they cause to the system.

Many authors did tremendous research to know and define vulnerabilities. According to MITRE's definition, a vulnerability is a state in which an attacker is allowed to execute commands, access data which has specified restrictions, pretend as another entity, to conduct DOS (Denial of Service)(*Software vulnerabilities*, n.d.).

Though there is no standard definition for software vulnerability, researcher studies earlier have given various definitions. Software Vulnerability is defined as "fault that can be viciously used to harm the security of software systems" by Kauang et al.
Author Krsul(Victor Krsul, 2011)defines software vulnerability as "a defect that allows an attacker to violate an explicit or implicit security policy to achieve some impact".

In another research article, define the terminology as "software vulnerability as a flaw, weakness or even an error in the system that can be exploited by an attacker in order to alter the normal behavior of the system" (Jimenez, Mammar & Cavalli, 2010). Schultz et al.(Jr. Schultz, Brown, & Longstaff, 1990) say software vulnerability as ''a defect, which enables an attacker to bypass security measure". Finally, OIS (Organization of Internet Safety) defines security vulnerability as "a flaw within a software system that can cause it to work contrary to its documented design and could be exploited to cause the system to violate its documented security policy". By examining all these above-defined statements by various authors clearly indicates

that the main cause for information security breaches is due to software errors. The report generated in 2010 by software application security researchers and specialists is evident that organizations of international cybersecurity say about 25 highly malicious software errors led to cyber-crime. These software errors were classified into three categories as described below;

1. Software Error based on insecure interaction among components
2. Software Error based on unsafe resource management
3. Software Error based on Porous Defenses.

The cyber attacks on organizations such as Google, SMEs, home users, governmental organizations, banks, and universities were all affected by the above software errors based on defined categories.Thus faults are the main cause of software vulnerability. These vulnerabilities are defined based on the weakness, fault, defects, errors, and failures which arise in software.Apart from these, there are few other most common causes of software vulnerability. Analyzing the probable causes can trim down the vulnerabilities in software applications.Krsul et al. (Krsul et al., 1998) did some research over the past few decades in the investigation and presented a few common effects of vulnerabilities. The common attacks include IP Fragmentation and Buffer overflow. Buffer overflow takes place when a program copies some data from an object into the other object, during the process program does not check whether the destination object has enough space to contain the source object's content. A buffer overflow occurred in 2001 caused vulnerability in Microsoft IIS Web Server, reported by e-Eye Digital security (Shaneck, 2003). IP Fragmentation – IP is an internet protocol, and fragmentation is a process of breaking down a packet into small chunks of data (fragments), in such a way that the resulting piece is allowed to pass through a link with smaller MTU (maximum transmission unit)

than the original packet size. Later these fragments are reassembled by the host, which receives the data. The vulnerabilities which occur during the process/design of the protocol and IP fragmentation are known as teardrop (“IP fragmentation,” 2018).

In 2015, the ICS-CERT (The Industrial Control system Cyber emergency response team) had reported the major causes of vulnerabilities affected by the organizations [11].

1. Insufficient Entropy: This type of vulnerability occurs by random guess by the attacker. So when the attacker randomly guesses numbers generated by the system to gain access, which is not authorized to a system.
2. Using cryptographic weak ping: This usually occurs in the cryptographic context, non-cryptographic PING is used. By this, the cryptography is exposed to certain sorts of attacks.
3. Spoofing with authentication bypass: Due to the improper implementation authentication scheme, there will be a possibility of a spoofing attack.
4. And also due to improper check for exceptional conditions or even unusual conditions.

Based on the report generated by research experts in 2010 on 25 extremely dangerous software errors are caused due to the below software vulnerabilities identified.

A. Software Error based on insecure interaction among components

- Uploading a dangerous file that does not have restrictions.
- Redirecting URL to the site which is not trusted.
- Utilization of special elements in an SQL command which are not properly neutralized.

- During web page generation, improper neutralization of input.
- Utilization of special elements in an OS command which are not properly neutralized.
- Cross-site request forgery

B. Software Error based on unsafe resource management

- Including functionality from a control sphere that is not trusted.
- Using a probable highly dangerous function.
- To a restricted directory, there is no proper boundary of a path.
- Wrap around or integer overflow.
- Calculation errors of buffer size.
- Creating a buffer copy without checking the size of an input.
- Format string, which is not controlled.

C. Software Error based on Porous Defenses.

- Assignments of unauthorized permissions for critical resources.
- Missing validation for significant function.
- Authorization errors
- Utilizing one-way hash without salt.
- Misplaced encryption of sensitive data.
- Utilization of cryptographic algorithm which is not working.
- Unnecessary privilege executions.
- Unrestricted access to excessive unauthorized attempts.

- While security decisions, dependence on untrusted inputs.
- Utilization of hardcoded identifications and credentials.

In addition, there are additional eight vulnerability causes reported by the National vulnerability database as follows,

1. Exceptional Condition Error Handling
2. Input Validation Error
3. Environmental Error
4. Configuration Error
5. Race Condition Error
6. Access Validation Error
7. Design Error
8. Others: nonstandard errors

So far, the probable causes for Software vulnerabilities have been discussed in the paper, and now in order to resolve the issues, there is a need to detect vulnerabilities in software. Researchers came up with various vulnerability detection methods in order to prevent anomalies in the software application. Detecting vulnerability is like finding 50% of the solution. When we are able to detect a problem in a system, finding a solution will become easier.

Vulnerability Detection Methods:

In detail, the analysis of tools and techniques utilized to detect vulnerabilities in software applications have been described in this section of the paper. These tools help to detect the

system gaps, which can be capitalized by the hacker. With the attack, the security of the system or where the system platform runs will get compromised.

I. Fuzzing:

It is one of the vulnerabilities detection methods. The random or invalid inputs are entered in the software application, and the unexpected output behaviors, errors identified, and expected vulnerabilities have been captured. These methods are important because software applications hold some level of vulnerabilities that have to be detected. Data generation is key to fuzzing. In this technique, significant tests are conducted in order to break down the source code and to opt suitable tool to supervise the procedure. However, in order to detect vulnerabilities, currently, developers analyze executable codes rather than source code. Fuzzed data generation can be executed in two ways: white box and black box fuzzing. In black-box fuzzing, there is no requirement of application details. It can be generated by random modification of correct data. This method of fuzzing is known as Black Box Fuzzing.Whereas for White Box Fuzzing, complete knowledge of software application codeand also the behavior is assumed for generating tests. Gray Box Fuzzing is a third type fuzzing, which is the combination of bothwhite and black box fuzzing. Gray box take the benefits of both the fuzzing tests. The minimum behavior target knowledge has been utilized in Gray box fuzzing. According to the fuzzing key – data generation methods are categorized as generation-based, random, direction-based, and mutation-based fuzzing.Among the above-listed fuzzing techniques, random fuzzing is the simplest technique. In this technique, a stream of random data is sent as an input for testing. The input data can be sent either as network

packets, command lines, or events. This fuzzing is useful when a program reacts to huge or unacceptable input data. Severe vulnerabilities can be detected by this random fuzzing, whereasmodern fuzzing has a detailed understanding of an input.

The testing tool in mutation-based will have format knowledge about the program input. The algorithm improves the efficiency of mutation-based fuzzing, which acts as a key. Program inputs are generated according to the specifications in Generation based fuzzing. While testing, generation based attains more coverage compared to random based fuzzing.

Program control flow has been utilized in direction based fuzzing in order to direct the testing flow. This is also known as test case generation fuzzing. SAGE is one of the types of direction based fuzzing. Firstly, the initial and valid inputs IN0 are given to the program P, the symbol execution engine monitors path and processes which is in the form of logical formulae; Secondly, during the execution of the path which is negated will be encountered then a new constraint will be solved and a new input IN1 will be created (varied from input IN0). Finally, the new process input IN1 is allowed to follow the same three previous procedures.Apart from the above listed fuzzing methods, there are many other fuzzing tools invented. The other few research tools on fuzzing are Peach, Sulley, SPIKE, and others.

II. Web Application Scanners

It is a type of scanner which examines web applications automatically for security vulnerabilities. As web security is monitored through public networks, it is difficult to handle. As web security takes the requests through HTTP (hypertext transfer protocol), it makes the processing complicated. The testing in web application security is carried out in two ways for vulnerability detection: white box and black box testing. White box testing is the process of analyzing the source code manually with the utilization of tools such as Pixy, FORTIFY, or Ounce. Because of the complexity of coding, it is not an easy task with the manual process. And sometimes, with this complexity and manual procedure, the vulnerabilities might not be detected effectively.

In the black-box testing process, in order to detect vulnerabilities, the scanner uses the fuzzing technique. It is also known as penetration testing. Penetration testing is famously known as ethical hacking/pen-testing. It is a process of testing a web application, network, or system to detect vulnerabilities that an attacker can make use of. It is an automated process of software application testing, or it can also be executed manually(“What is pen test (penetration testing)?” 2018).

The scanners in the web application are mostly applied in the development stage of testing. This is also capable of doing below functionalities,

- Low false positives ratio will be generated
- Detect vulnerabilities in web applications

- It generates an output (causes of vulnerabilities), a report which is to be carried out in order to protect the system from vulnerabilities.

In addition to the above-described scanners, there are few commercial scanners which detect vulnerabilities in web application. The scanners include WebKing, Appscan, and WebInspectNTOspider(Fong & Okun, 2007).

III. Brick

This is an integer-based vulnerability scanner that detects at run time. It is one of the effective approaches which results in less false positives and false negatives. This process comprises into three stages as follows,

1. Convert the binary code into a dynamic binary instrumented framework (intermediate representation) on Valgrind(Nethercote & Seward, 2007).
2. Capture statements relevant to integers at run time, and also, it records the required data.
3. Identify and spot out the vulnerabilities with a set checking format.

IV. CRED:C Range Error Detector

It is also one of the vulnerability detectors but is not capable of detecting Dynamic Buffer Overrun applications. It is unable to such programs because of its power lack to guard against buffer overrun attacks, breaking existed code, and also due to production of high overhead. It has been proved that this is the only tool to protect

against 20 distinct buffer overflow attacks. CRED is an effective tool for detecting known vulnerabilities in programs that are attacked with buffer overrun(Wilander & Kamkar, 2003).

V. Static Analysis Techniques

As we know, the importance and usage of software applications have grown tremendously high. Unfortunately, the security issues in software applications lead to gaps and weaknesses for attacks. The report generated regarding the web application security statistics states that over 60% of assessed websites are vulnerable. Each application is affected by a minimum of 6 unsolved flaws (Gupta & Sharma, 2012). The report generated in 2013 proved that Common Vulnerabilities and Exposures ("CVE - Home," 2007) and Open Web Application Security Project specifythat the attacks: SQLi (SQL injection), XSS (cross-site scripting) are the most two severe attacks in top ten attacks occurred in web-based applications in a system.

Static analysis is one of the vulnerabilities detecting technique. It is the most defensive as well as preventive technique. The chief goal of this technique is to recognize the defects in the source code prior to the first execution in the user's environment.This technique assists in identifying vulnerabilities early enough in its case, financial damage. This approach is useful in performing the below-described activities:

- Be pertaining to any particular algorithm or set of rules which are known as inference.

- Assess the input code.
- Generates program vulnerabilities list.

Buffer overflow is also an effective attack famous in web applications. The static analysis technique is one of the effective techniques which detects the errors prior to the program execution. The errors such as Buffer overflow. Numerous static analysis techniques have been invented by researchers to detect Buffer overflow vulnerabilities (Dor, Rodeh & Sagiv, n.d.), (Hackett, Das, Wang & Yang, 2006). The various distinct approaches are classified as:

- Analysis of sensitivity
- Soundness
- Language
- Interference technique
- Analysis granularity

Literature Related to the Methodology:

Below are the articles and research work methodologies of various publications. The distinct researcher's approaches on how to solve the vulnerability issues have been described in this section.

1. An article named "Automatic detection and correction of web application vulnerabilities using data mining to predict false positives" written by authors: Ibéria Medeiros, Nuno F. Neves, and Miguel Correiain 2014. This article is about the detection of anomalies and

correction of them in a web application in order to predict false positives utilizing data mining. As we all know, security has become very important in the internet field. The main problem arises with the developers who are not proficient insecure coding and leave the built applications with anomalies.In order to solve this problem, the best approach believed by the authors in this paper is to use static analysis to detect bugs. But unfortunately, these tools produce false positives results, which the intern makes the job complicated in finding bugs in the application. So the authors in this paper found a solution on how to detect vulnerabilities with fewer false positives using hybrid methods. In order to achieve that, the initial step in this paper utilizes taint analysis to detect candidate anomalies and then utilize the data mining method to reduce false positives. Authors in this paper came up with two opposite approaches: one is about humans program in regard to vulnerabilities, and the other is obtaining automatic knowledge from machine learning for data mining. With these approaches, more specific detection can be implemented, which corrects and fix the source code automatically. This approach is implemented through the WAP tool, and with the huge set of PHP source applications, code evaluation has been performed(Medeiros et al., 2014)**.**

2. Article “Vulnerability detection with deep learning” written by Fang Wu, Jiqiang Liu, Jiqiang Wang, and Wei Wang in 2017. As the protection of software systems against vulnerabilities has become a very important issue. So, in this paper, the authors came up with the methodology to detect vulnerabilities. In this paper, in order for vulnerability detection, three deep learning methods were proposed. The three deep learning models

for vulnerability detection, namely, long short term memory (LSTM), convolution neural network (CNN), and convolution neural network – long short term memory (CNN-LSTM). Nine thousand eight hundred seventy-two sequences of method calls, which represent the features of the binary code patterns for the execution, were gathered in order to check the performance of their methodology approach. After applying these deep learning models to predict the anomalies, the results obtained show that it is accurate to 83.6%. This approach is more effective than traditional methods like multi-layer perception (MLP)(Wu, Wang, Liu & Wang, 2017).

3. "Discovering software vulnerabilities using data-flow analysis and machine learning," written by Arjen Hommerson, JorritKronjee, and Harald P. E. Vranken. In their approach, vulnerability detection, specifically SQL injection (SQLi) and Cross-Site Scripting (XSS), is done in PHP applications. Authors came up with a novel method for static type analysis, which combines machine learning with data-flow analysis. The vulnerable PHP code, along with the solved solution versions, is collected from the assembled dataset of the National Vulnerability Database and the SAMATE project. Data-flow techniques, which include reaching constants, taint analysis, reaching definitions analysis, were applied in order to extract the features from the samples provided by the code. Additionally, these features were utilized in their methodology to train the machine learning with a variety of probabilistic classifiers.

Once the machine learning is trained, they constructed a tool named WIRECAML for the effectiveness of their approach. Then results obtained are then allowed to compare their tool with the other tools for the detection of PHP code vulnerability detection. Results show better performance of their tool in the detection of SQLi and XXS anomalies. With the experiment performed on other open source applications, previously unknown vulnerabilities were also detected in a photo-sharing web application (Kronjee et al., 2018).

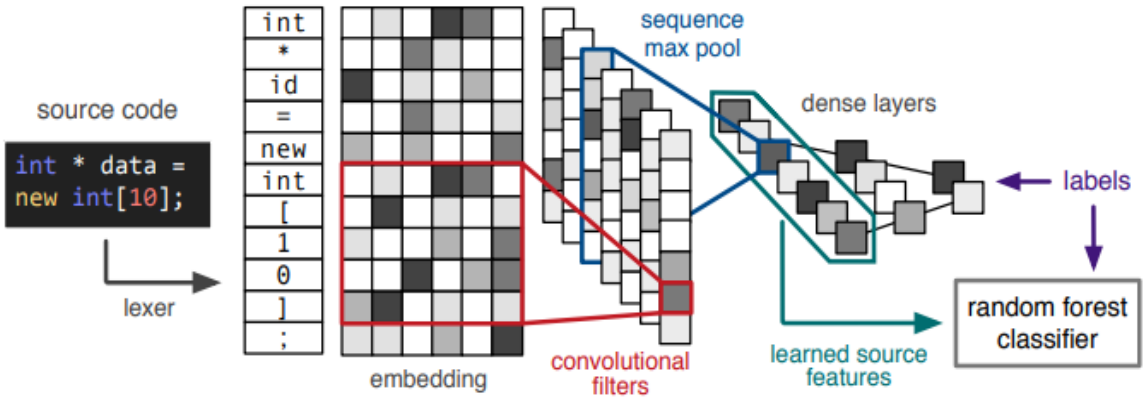
4. (Russell, Kim, Hamilton, Lazovich, Harer, Ozdemir, Ellingwood & McConley, 2018)In the article “Automated Vulnerability Detection in source code Using Deep Representation Learning” written by authors - Rebecca L. Russell, Louis Kim, Lei H. Hamilton, TomoLazovich, Jacob A. Harer, OnurOzdemir, Paul M. Ellingwood and Marc W. McConley. As numerous software anomalies were detected, reported, or discovered openly or secretly in proprietary code. Such type of anomalies poses severe risk exploit and leads to the various issues such as DOS (denial of service), information leaks, system compromise, etc., Authors in this papers research methodology, utilize benefits of C and C++ open-source code which is available and capable to detect huge-scale function-level vulnerability detection system. This methodology used millions of open-source functions in order to substitute existing labeled vulnerability datasets. Those data-sets are then marked with carefully-selected outputs of three distinct static analyzers that point toward potential exploits. The data-setswhich are marked were available at https://osf.io/d45bw/. These data-sets were utilized to develop tools that are capable of detecting rapid and

scalable vulnerabilities. The tool is based on deep learning featured, which can interpret the leaked source code. Researchers evaluated their tool from both data-sets NIST SATE IV benchmark and also the real software packages. This research demonstration illustrates that deep feature learning with respect to source code is a more effective and trustable detection approach for software applications.

The below figure demonstrates the approach of neural representation learning with respect to the source code.

Figure 1

Convolutional neural representation-learning approach to source code classification (Russell et al., 2018)

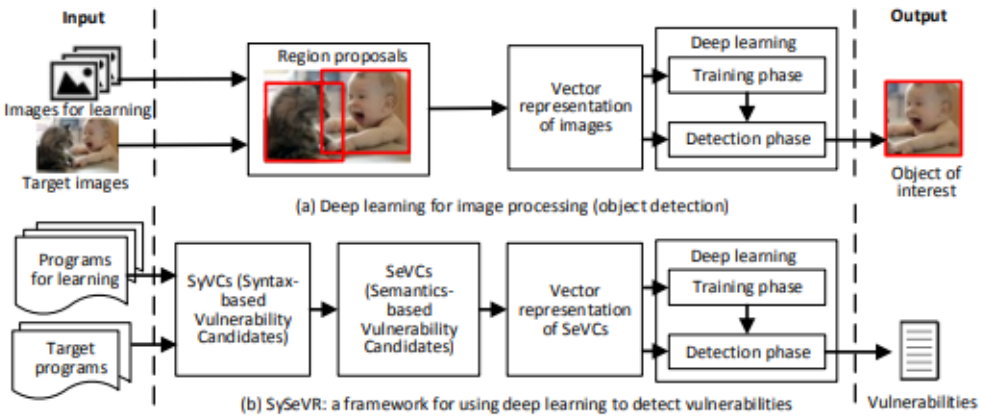


5. (Li, Zou, Xu, Jin, Zhu, Y & Chen, n.d.)Authors - Zhen Li, Deqing Zou, Shouhuai Xu, Hai Jin, Senior Member, IEEE, Yawei Zhu, and Zhaoxuan Chen, worked on a research article named “A Framework for Using Deep Learning to detect software vulnerabilities”. In this paper, the importance of vulnerability detection, which has to be handled as

apparently various vulnerabilities reported on a daily basis, was described. Consequently, the purpose of automating anomaly detection machines such as machine learning was discussed. As it is known that deep learning usage is very attractive for vulnerability detection as human involvement is very less to manually define the features.Although there is a tremendous success history behind the deep learning in some domains, still the vulnerability detection is undetermined. Soauthors in this article focus on how to fill the void. Hence they first proposed the systematic framework for the detection of vulnerabilities using deep learning.This framework focuses mainly on obtaining representations of the program, which contain syntax and semantic content relevant to vulnerabilities. This can be obtained by dubbing Syntax, Semantic, and Vector representations (SySeVR, 2018) based. With the help of this framework, authors able to detect 15 unreported vulnerabilities in the National Vulnerability Database.

Figure 2

A framework for using deep learning to detect vulnerabilities (Li et al., n.d.)



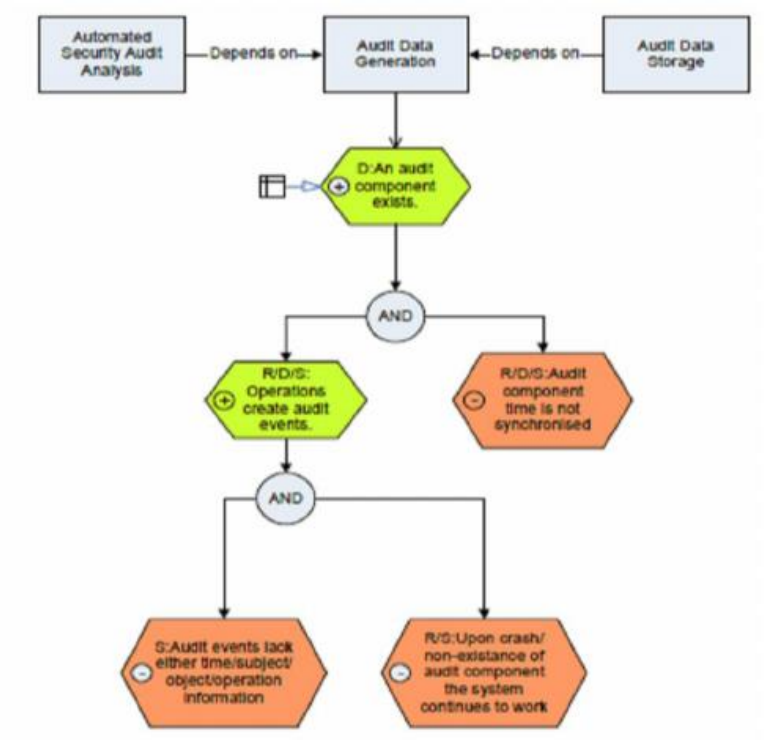
The experiments were conducted with four software products, and the results demonstrate the usefulness of the architecture. Among 15 detected but unreported vulnerabilities, seven are unknown but have been reported to the vendors, whereas the remaining eight have been "silently" patched.

6. (Jimenez et al., 2010)In the article named "Software vulnerabilities, Prevention and Detection Methods: A Review 1," authored by Willy Jimenez, AmelMammar, and Ana Cavalli. The importance of software applications in the current society and about their complexity development in different programming languages have been described. The main purpose of vulnerabilities and code errors cause has been described by the authors. Usually, the programmer's mistakes would become the major cause of generating software vulnerabilities. This will be the best approach for the attacker to attain privileges

to access private data in the system. This states that the vulnerabilities are the possible doorway for the attacker to access the system. Though it is clear that the vulnerabilities in the current date still a mounting tendency in the software applications, still regardless, the demand for software applications never got down. So, in order to detect and catch the vulnerabilities in the software production field, there is a need for tools which can help developers to detect the vulnerabilities in the code. Consequently, this research on anomaly detection presents an outline of vulnerabilities and the respective methods for detecting them. The below figure shows their methodology of the research on vulnerability detection.

Figure 3

Security Goal Indicated Tree (Jimenez et al., 2010)



7. (Daymont, 2017) In the article “Software vulnerabilities detection system and methods”, the author invented reveals a scheme and technique of detecting software anomalies in a computer program. One of the invention methods compiled software has been used for every single instruction. Basically, this invented compiled software was used to examine both the properties of data and control flow of the target program.In this article's methodology, a comprehensive instruction model has been utilized for every instruction

of the compiled code. This code is complemented by a graph. The graph is a control flow model that contains all potential instruction flow paths. Essentially, the data flow models are used to save the data flow record of unsafe data during the program execution. During this process, the system analyzes the data flow pattern and generates results corresponding to each execution, which calls unsafemethods/functions. And thus, the retrieved results are aggregated along with the related debug information, recommendations, and all other correlated instructions information that are triggered have been added in a security report.

Figure 4

Data flow model to identify vulnerabilities (Daymont, 2017)

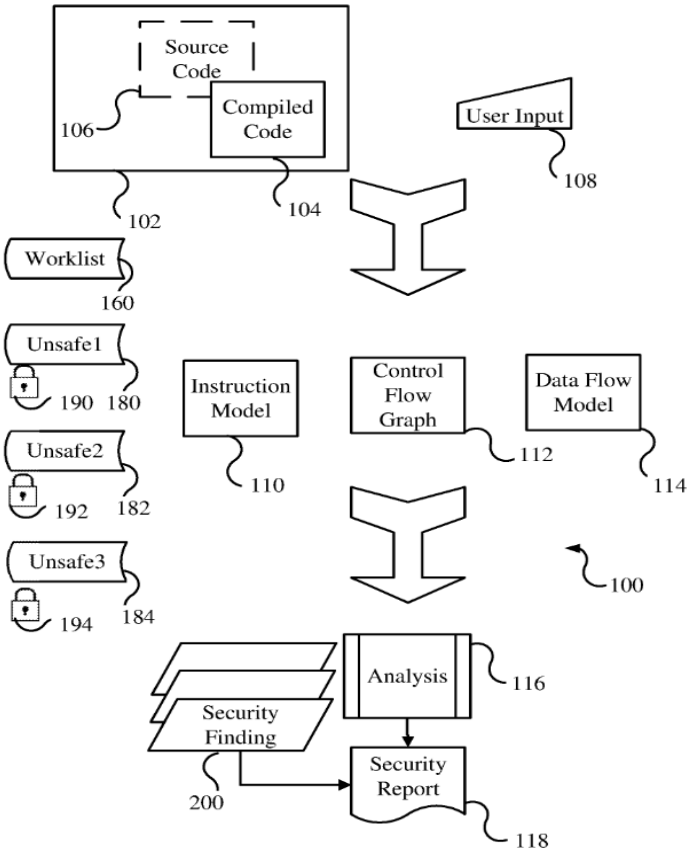
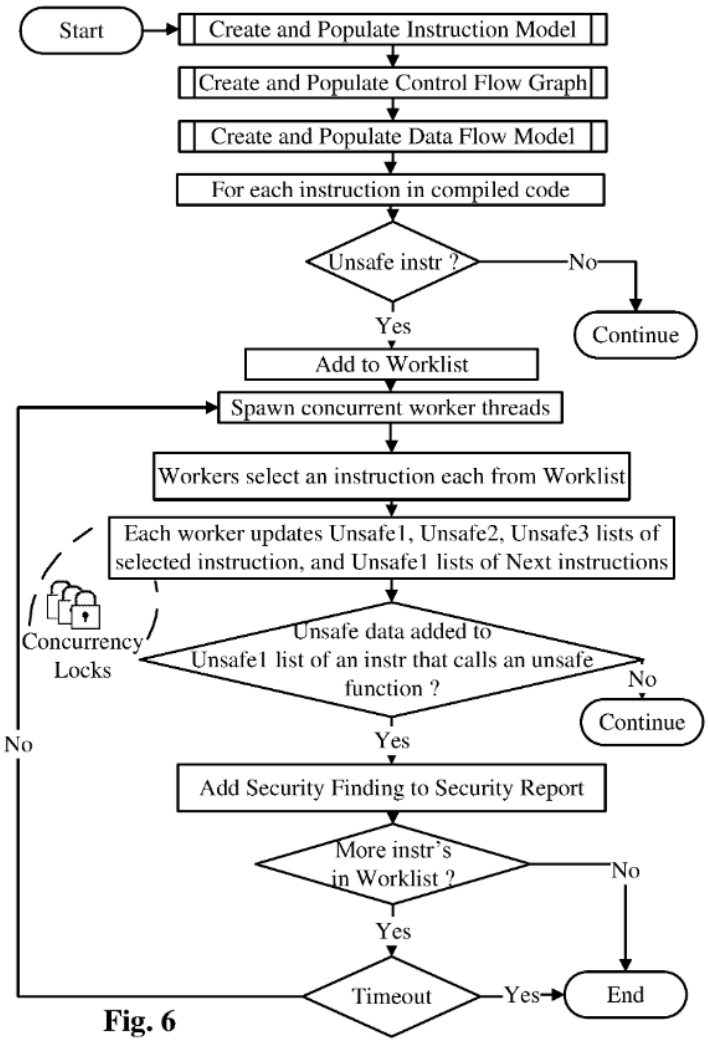


Figure 5

Data flow model to identify vulnerabilities (United States Patent No. US9715593B2, 2017)



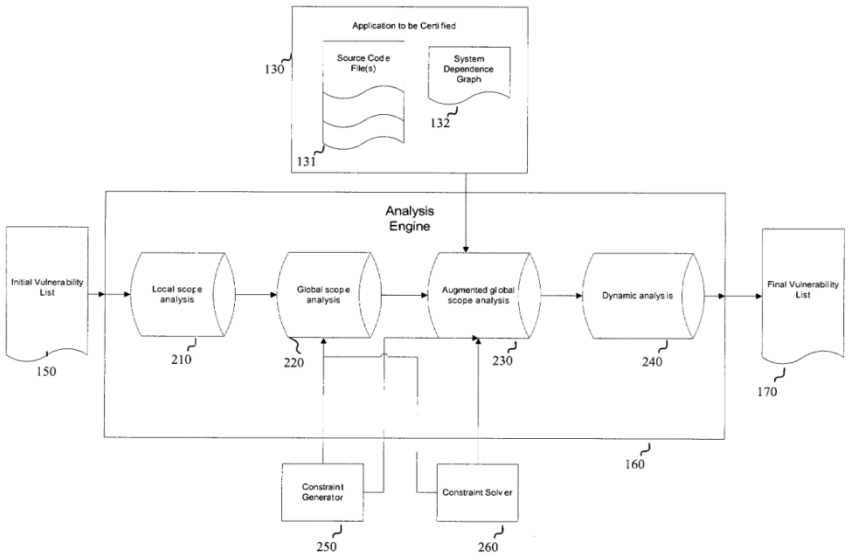
8. Weber, Shah, & Ren, 2008) An article named "Systems and methods for detecting software security vulnerabilities", describes the embodiment of the current innovation

related to software application methods and systems for static analysis.According to the personification, this framework includes a scanner which has already been programmed and is coupled to the analysis engine. The program scanner is used to identify the numerous software program patterns of vulnerabilities. It also generates a list of vulnerabilities in the output file. In order to test the vulnerability potentiality, the analysis engine is configured to apply additional rules to determine the vulnerable resistance.

Figure 6 describes the framework of the work.

Figure 6

Analysis engine to determine whether the potential vulnerability is vulnerable (United States Patent No. US7392545B1, 2008)



9. (Letychevskyi, 2018)The author – Oleksandr Letychevskyi published an article name "Algebraic methods for detection of Vulnerabilities in software systems" presented an

approach for detecting vulnerabilities in a program with an algebraic algorithm. The vulnerabilities in this research framework are found by the sequence of processor instructions as an input. In a particular methodology, formulas have been used as a logic language. These formulae are presented as logic and are achieved by transforming the code into an algebraic specification. Along with the algebraic specifications, symbolic models were utilized in order to detect vulnerability cases which are represented in the form of logic language. But with this algebraic approach, the usage of this framework in order to solve and prove the systems integration along with the Algebraic Programming system will be anticipated.

10. (Tevis & Hamilton, 2004)In a few published articles, authors did great research about the vulnerable attacks on software applications. Their profound research helped them to find the major reasons, and the necessity to build software has become a dominant goal in the field of software development. Accordingly, researchers in this software field found that the users can also be the reason for exploiting the software by their malicious inputs, and as a result, researchers found a way to fix these issues. In addition to these solutions, in order to partially mechanize the tasks which perform a security analysis of a program, researchers also constructed various source codes that automatically check software applications. Even though researchers came up with immense advances, still the core issue of how to secure the software applications from vulnerabilities still exists. All in all, the author’s solution to this problem could transform from imperative to functional

programming techniques. This solution may be the key approach to get rid of software vulnerabilities altogether.

Chapter III - Methodology

Introduction:

This chapter describes the approach and the plan to address the issues discussed in chapter 2.

Design of the Study:

As we know, security for software applications has become a real threat. This framework implementation is to achieve the following goals,

- Firstly, be able to see if the system acts intended or not
- Enable users to monitor if program execution is consuming more resources in terms of memory, CPU, network bandwidth, disk usage, IO requests, etc.
- And with the known resource information such as CPU, RAM, we can predict program performance on any machine which helps in problem identification.

The architecture of the Framework:

Figure 7

Data Collection and Preparation

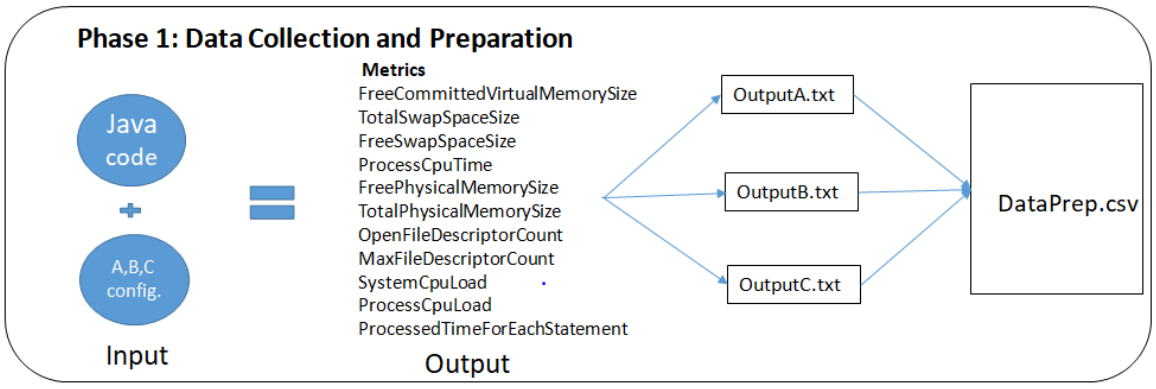


Figure 8

Data Training

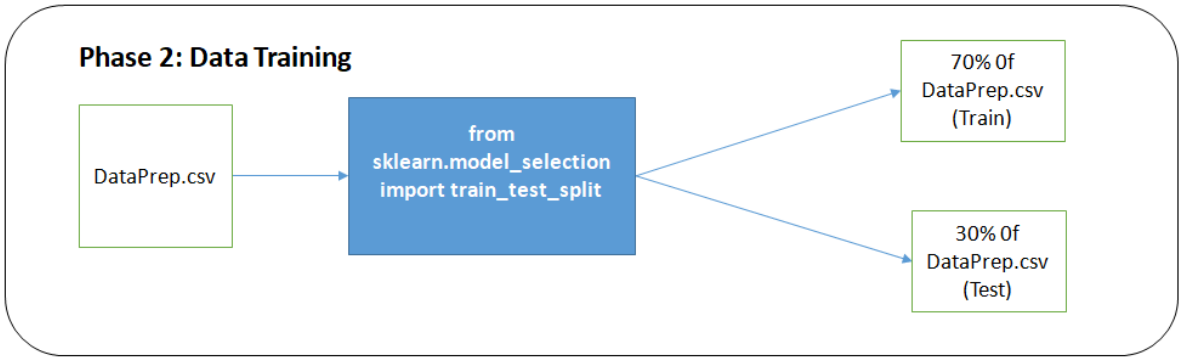


Figure 9

Data Prediction

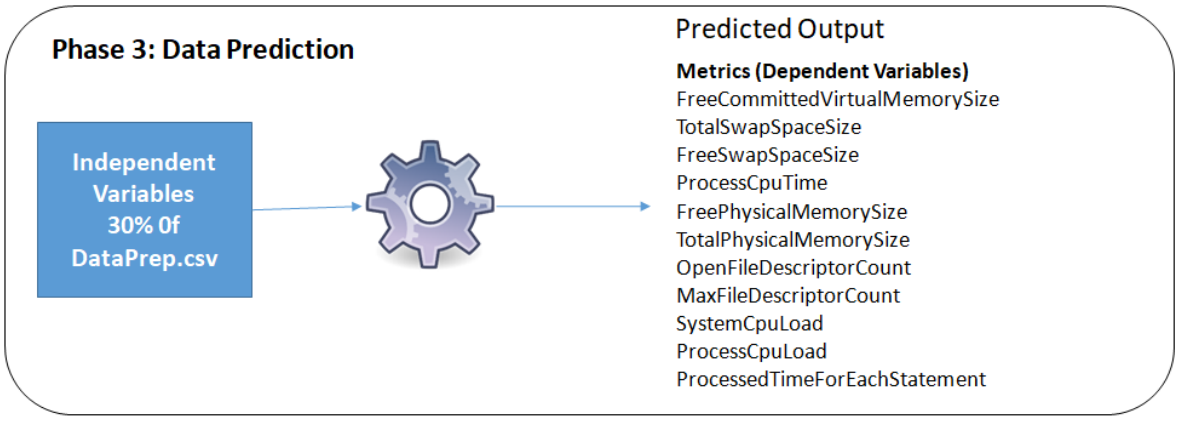
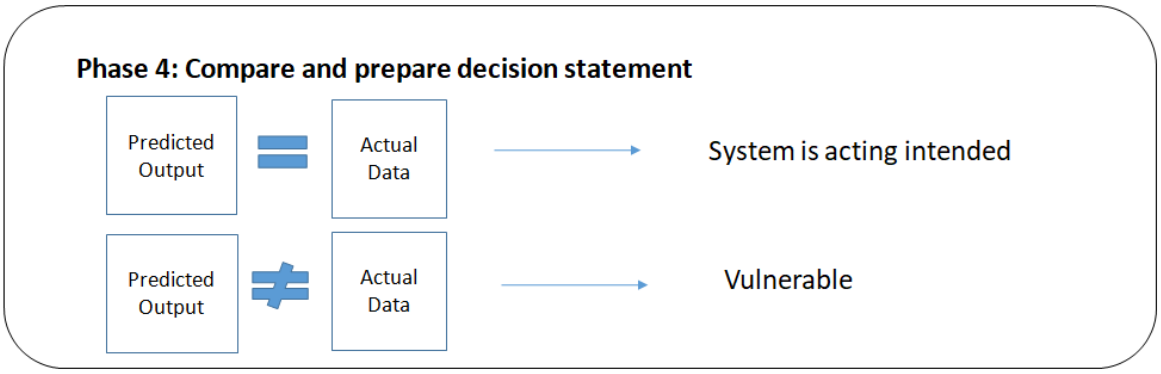


Figure 10

Compare and prepare decision statement



The construction of this framework comprises of 4 phases as follow:

1. Data Collection and Preparation
2. Data Training
3. Data Prediction
4. Compare and Prepare Decision Statement

Phase 1: Data Collection and Preparation

In the first phase - Data Collection and Preparation, a piece of java code is executed in 3 different configurations let's say A (vCPU 1, Memory 1GiB); B (vCPU 4, Memory 16GiB); C (vCPU 16, Memory 32GiB). The outputs generated in these respective machines are saved as OutputA.txt, OutputB.txt, and OutputC.txt, respectively. The output file has the eightmetrics captured while programming execution. It captures the CPU performance and memory

consumption by the specific machine for each java statement. Below are the 11 metrics collected in .txt files,

Metrics

- FreeCommittedVirtualMemorySize
- TotalSwapSpaceSize
- FreeSwapSpaceSize
- ProcessCpuTime
- FreePhysicalMemorySize
- TotalPhysicalMemorySize
- OpenFileDescriptorCount
- MaxFileDescriptorCount
- SystemCpuLoad
- ProcessCpuLoad
- ProcessedTimeForEachStatement

Phase 2: Data Training:

In this second phase - All three or more collected output.txt files in phase1 are appended together and saved in CSV format. This CSV file is given as an input to the Machine Learning, and it is trained. An algorithm is generated by machine learning according to the data given for training. Among the given feed, only 70% of the data is given as feed for training. And the remaining 30% data's independent values and the dependent values are expected as output.

Phase 3:Data Prediction:

In the third phase – Data in CSV file, among which only 70% of the data is given as feed for training and the remaining 30% data's independent columns, and the dependent columns are expected as output. This is known as predicted data by Machine Learning Algorithm.

Phase 4: Compare and Prepare Decision Statement:

In this last phase of the framework – Compare and Prepare Decision Statement, both predicted data, the one which is generated by the ML algorithm, and the actual result set are allowed to compare. If the result predicted by machine learning is similar or reaches its threshold point, then that particular machine is acting intended; if not, the machine is vulnerable.

Summary:

The research plan, which comprises 4 phases, have been constructed and described in this chapter. The following chapter demonstrates the implementation steps to address this issue.

Chapter IV - IMPLEMENTATION OF FRAMEWORK

Introduction:

This chapter shows the implementation steps, an approach to address the anomaly detection in any system.

Demonstration steps:

1. Login to 3 distinct instances
2. Collect data of 11 metrics for Java code
3. Data Prep
4. Train and Predict Data in SageMaker Notebook instance
5. Plot a graph between Actual and Predicted results

Implementation Process:

Phase 1: Data Collection and Preparation

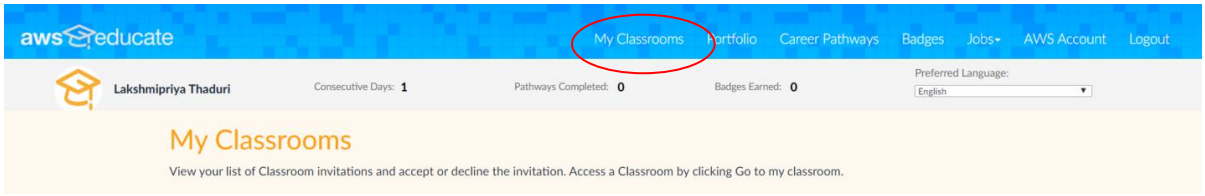
Steps for connecting to Linux Instance from Windows Using PuTTY. (*Connecting to Your Linux Instance from Windows Using PuTTY - Amazon Elastic Compute Cloud*, n.d.)

Make sure EC2 instance is running before attempt connecting through PuTTY. In order to start the instance, complete the following steps.

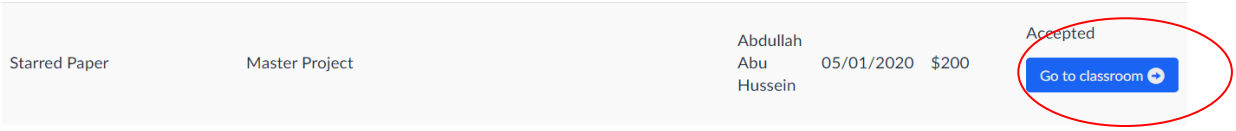
1. Login to AWS console
2. Launch an Instance
3. Generate private key
4. Connect to the Instance through PuTTY

➢ Login to AWS console,

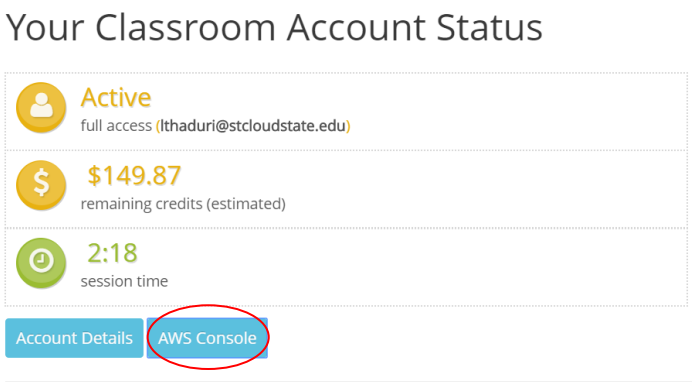
- Go to “My Classrooms”



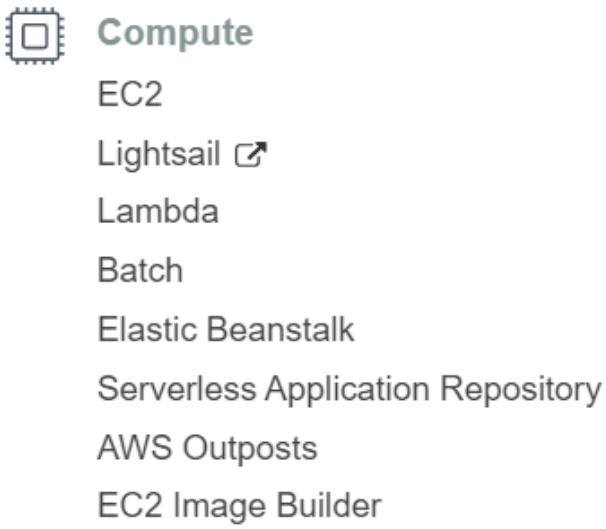
- Scroll down to the desired course and click on “Go to classroom.”



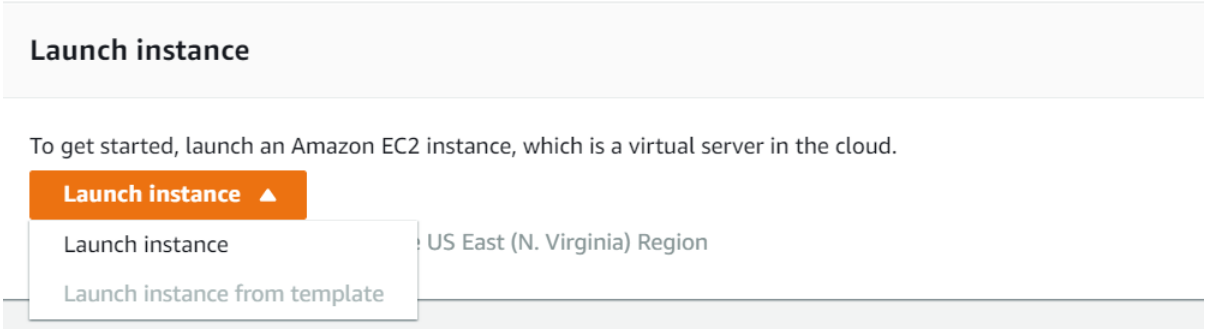
- Select AWS Console under “Your Classroom Account Status”



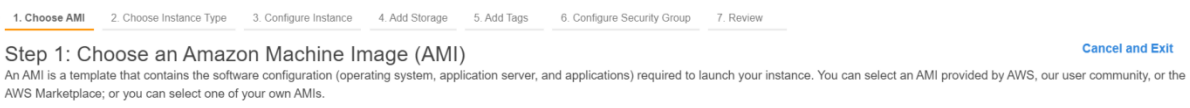
- Scroll down services, under compute – EC2 services



- Launch an Instance:

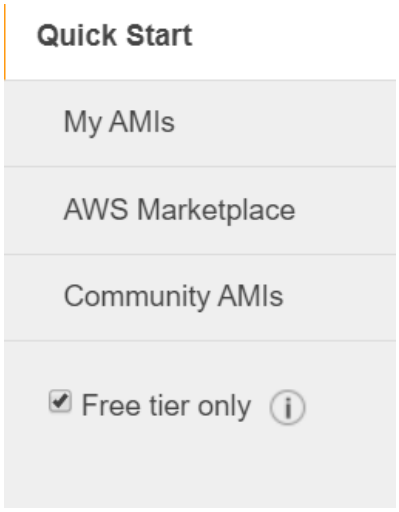


- Click on the dropdown arrow – Launch instance -> Launch instance, as shown in the above screenshot.
- Below are the steps to be completed to launch an instance

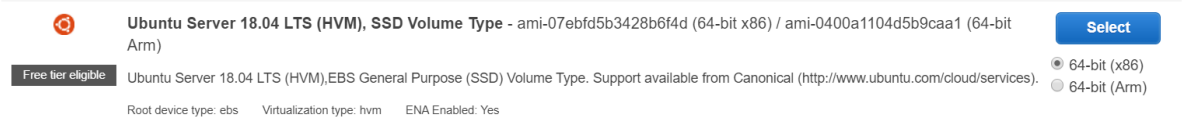


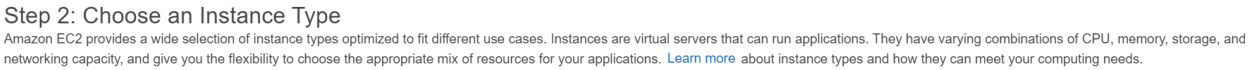
- Step 1: Choose an Amazon Machine Image (AMI)

- Check “Free tier only.”

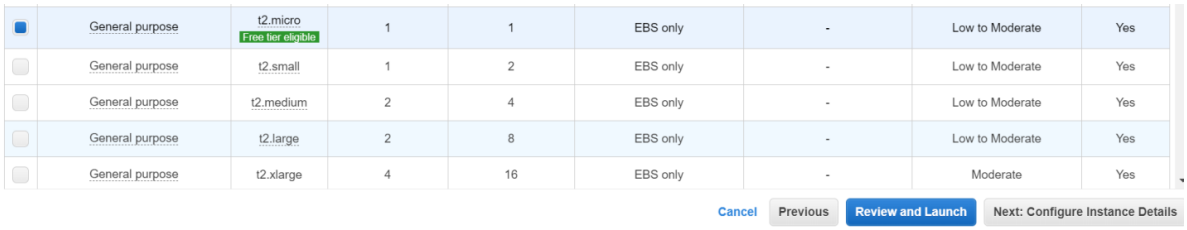


- Select an instance with desired configurations

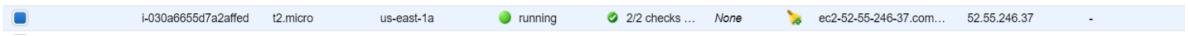




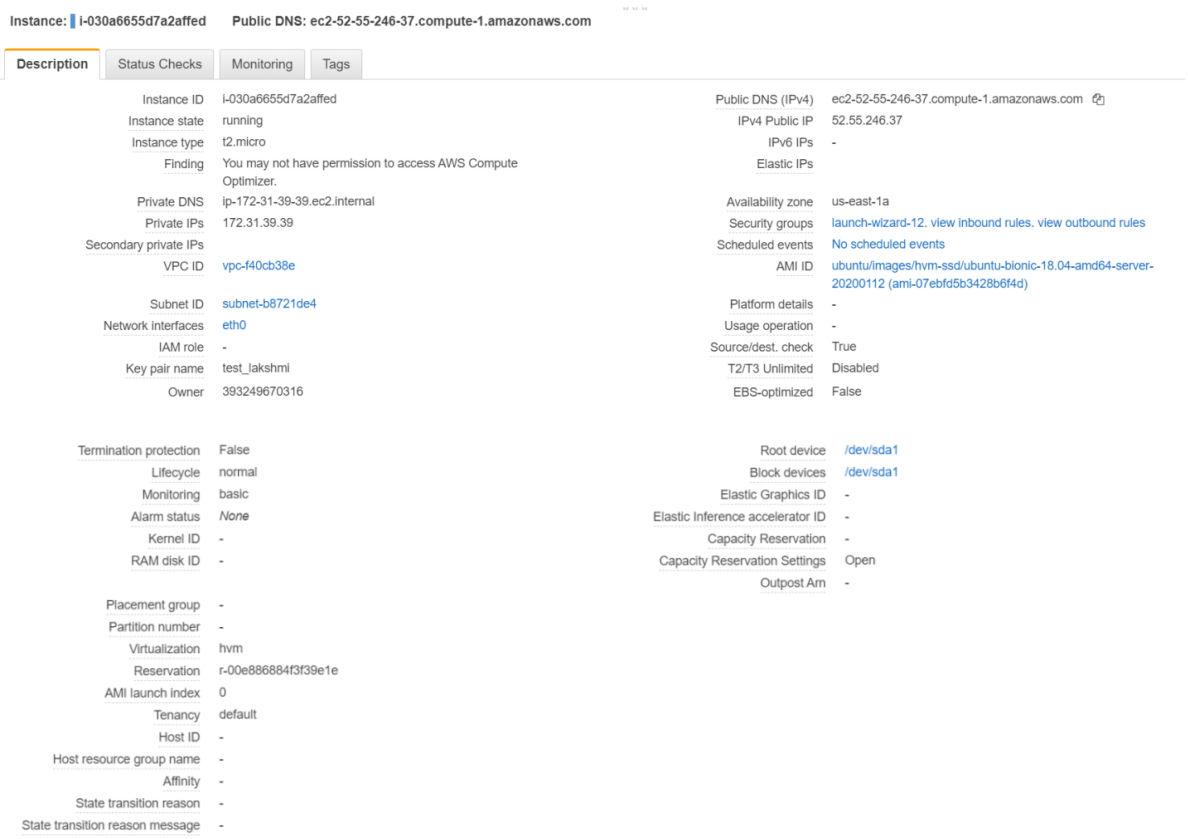
- Step 2: Choose an instance type -> Review and Launch



- Select an Instance to run



- Below are the properties and respective configurations

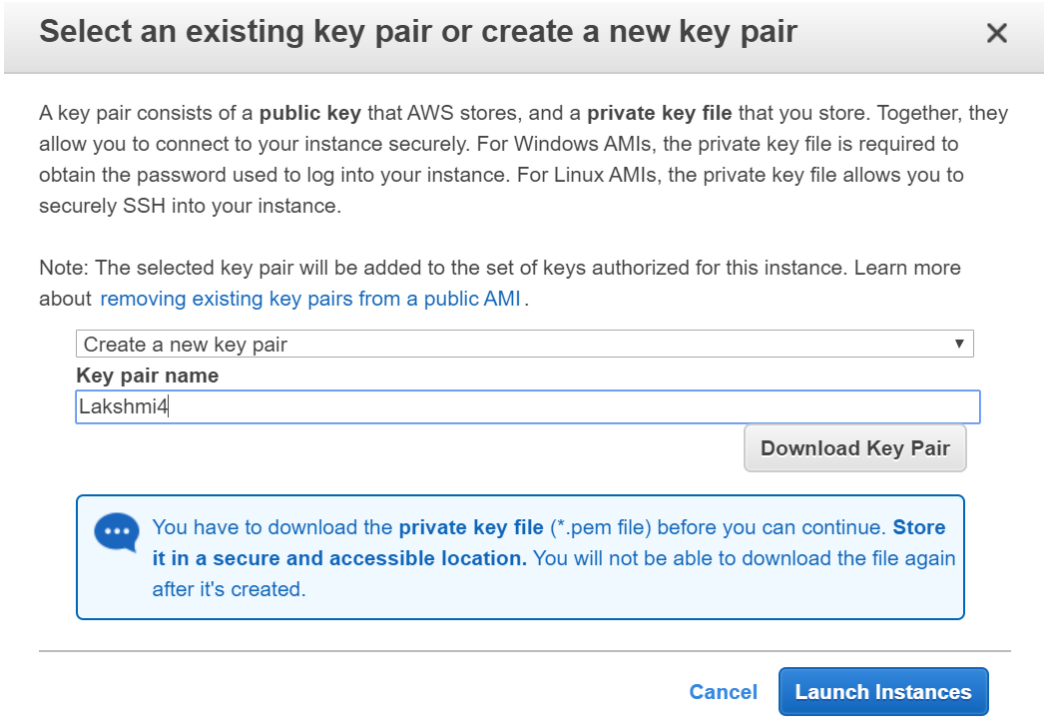


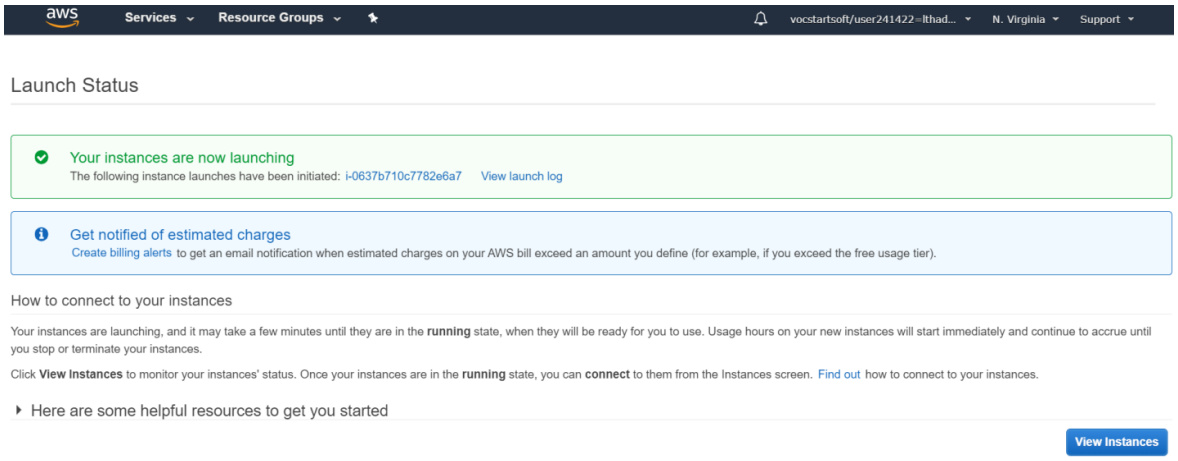
As the instance is running, now try connecting through putty. In order to connect through putty, complete the below steps.

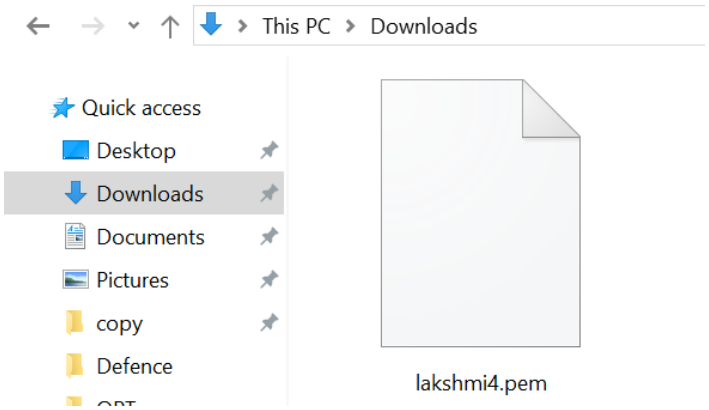
- Make sure PuTTY is installed in your local machine.
- If not, download the latest version and install PuTTY.
- Also, Install PuTTYgen to convert the private key.

- Once you launch an instance in AWS services, download .pem file on to your local by creating a key pair, as shown below.

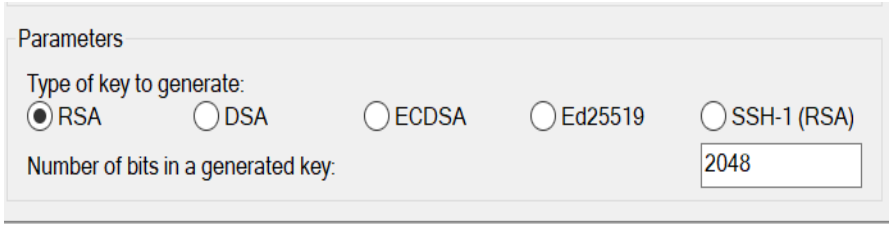
➢ Generate private key:



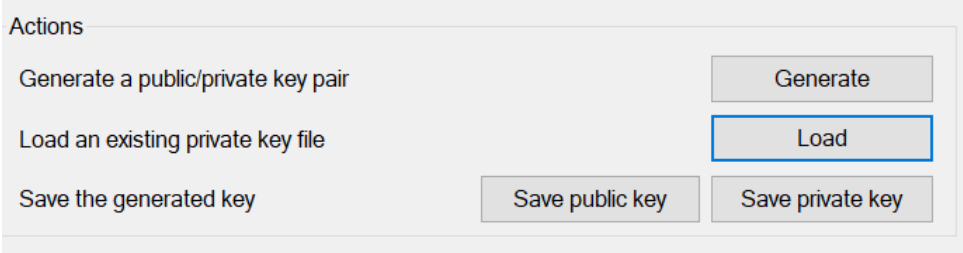




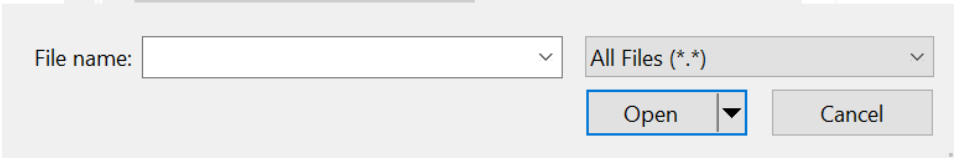
- Once you have PEM key downloaded in Downloads, start converting .pem to .ppk using PuTTYgen.
- Steps to get .ppk file. Do the following steps.
- Open PuTTYgen application, which is installed in your machine.
- In Type of key to generate: select RSA radio button and for Number of bits in a generated key: 2048 bits



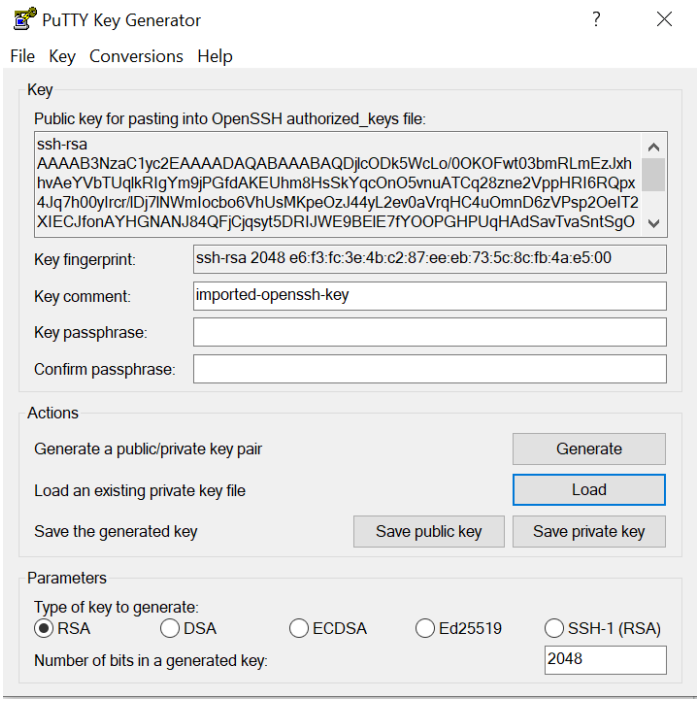
- In Actions, Load an existing private key file



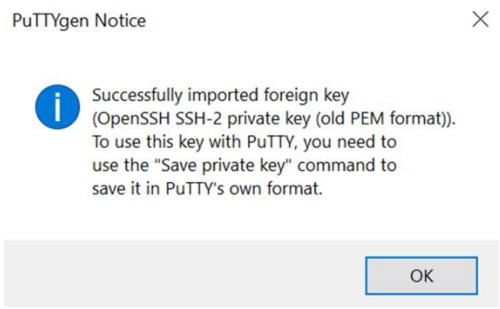
- Open the file – browse the .pem key, which is in the Downloads folder.



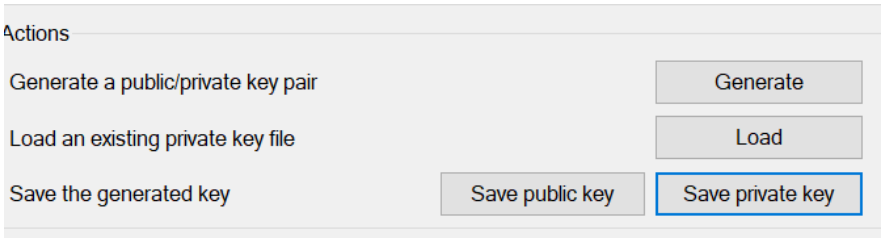
- Select - Load at Load an existing private key file.



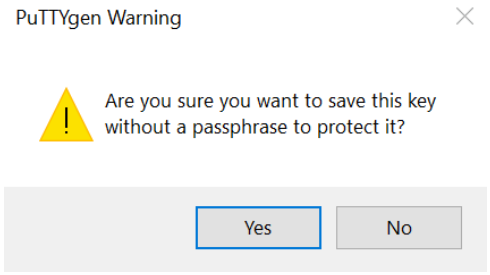
- Select Ok when the PuTTygen Notice box is populated.



- Choose to Save private key button under Save generated key option.

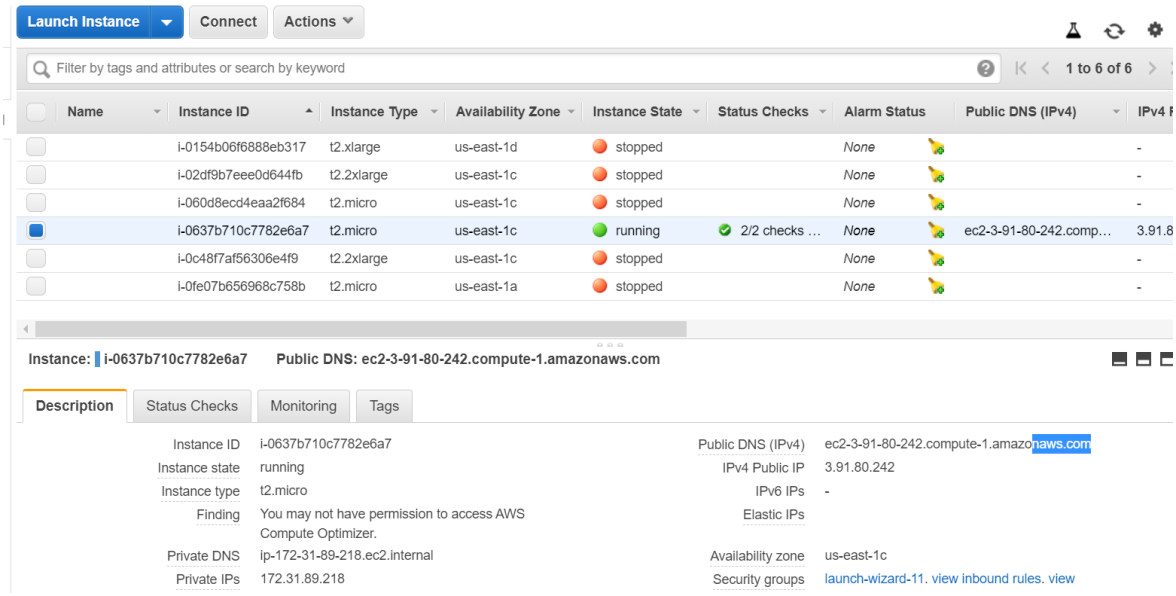


- Accept the PuTTYgen Warning message.

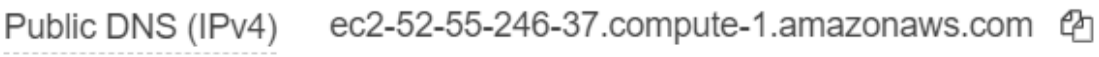


➢ Connect to the instance through PuTTY:

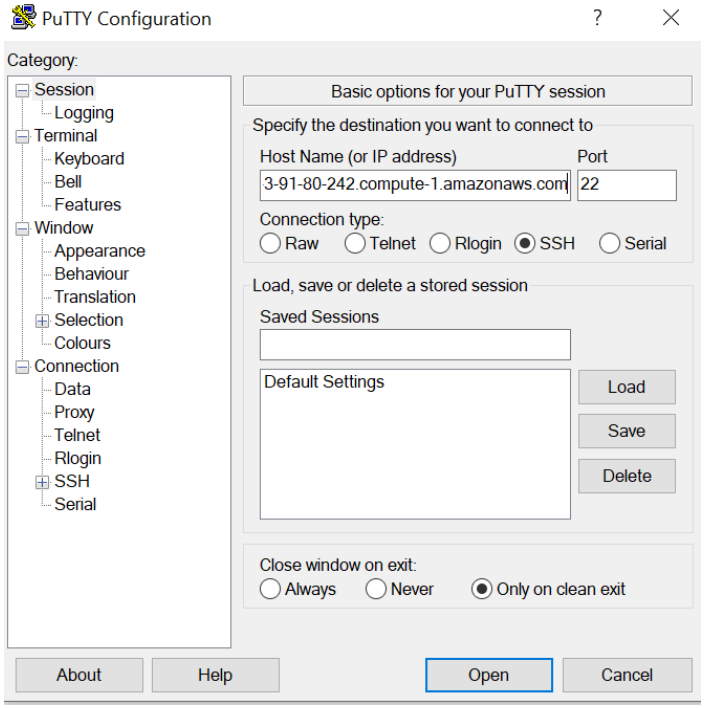
- Make sure instance is running



- copy Public DNS (IPv4) from Description



- Paste above-copied IPv4 in the Host Name section, Port 22, and connection type is SSH.



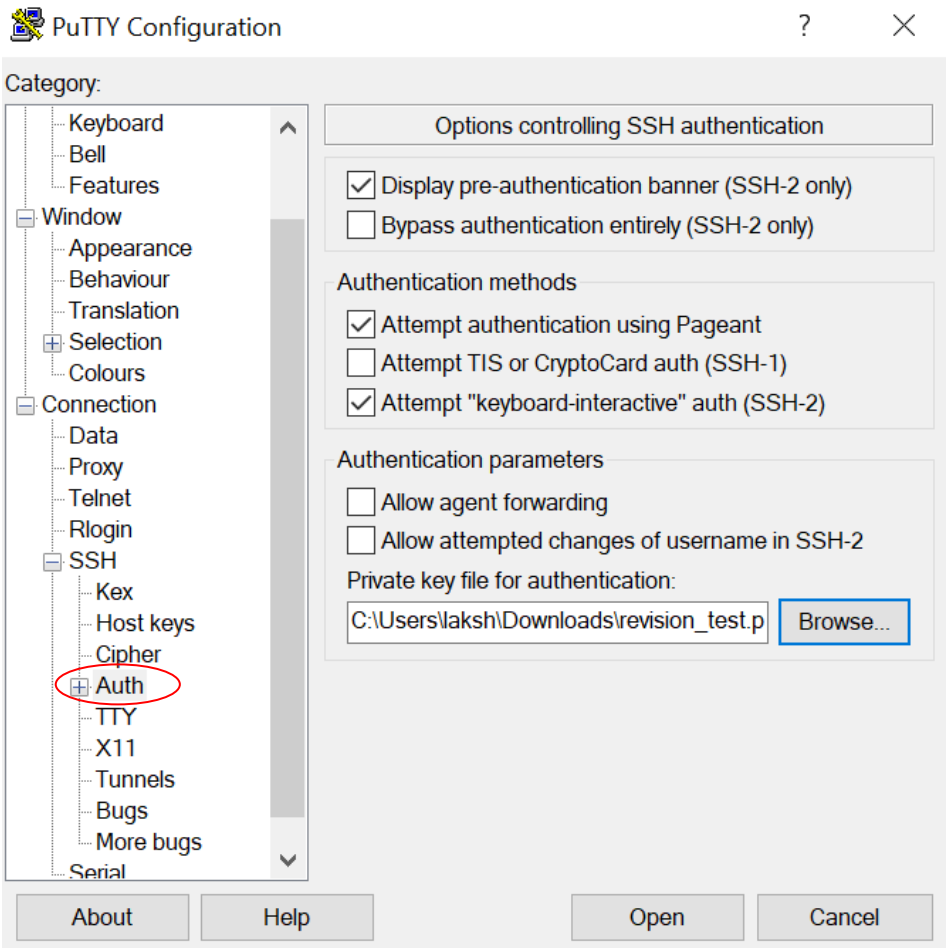
Alternatively, if the chosen instance has an IPv6 address, the hostname can be user_name@ipv6_address.

For user_name, be sure to specify according to the AMI as listed below,

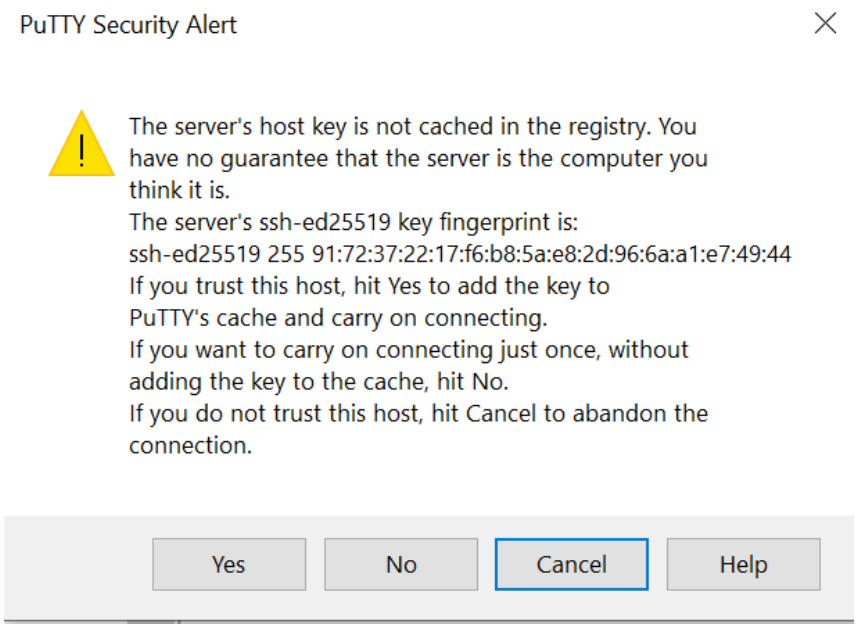
1. Amazon Linux 2 or the Amazon Linux AMI: ec2-user
2. CentOS: centos
3. Debian: admin or root
4. Fedora: ec2-user or fedora
5. RHEL: ec2-user or root
6. SUSE: ec2-user or root
7. Ubuntu: ubuntu

8. If any issues connecting instance with the given ec2-user and root, check with the AMI provider.

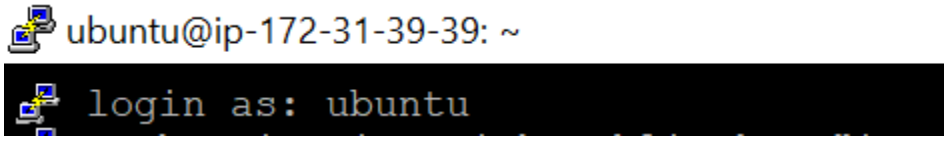
On the left section under Category -> SSH -> Auth. Browse, open converted .ppk file from Downloads.



- Accept PuTTY Security Alert, as shown below.



- Ubuntu shell will be opened, now login with “ubuntu” user.



- Once you successfully login to the instance, make sure java is installed. To verify, follow the below commands.

```
ubuntu@ip-172-31-39-39:~$ java

Command 'java' not found, but can be installed with:

sudo apt install default-jre
sudo apt install openjdk-11-jre-headless
sudo apt install openjdk-8-jre-headless
```

```
ubuntu@ip-172-31-39-39:~$ sudo add-apt-repository ppa:openjdk-r/ppa
```

```
ubuntu@ip-172-31-39-39:~$ sudo apt-get update
```

```
ubuntu@ip-172-31-39-39:~$ sudo apt-get install openjdk-8-jre
```

As java is installed successfully, check version as shown below

```
ubuntu@ip-172-31-39-39:~$ java -version
openjdk version "1.8.0_242"
OpenJDK Runtime Environment (build 1.8.0_242-8u242-b08-0ubuntu3~18.04-b08)
OpenJDK 64-Bit Server VM (build 25.242-b08, mixed mode)
```

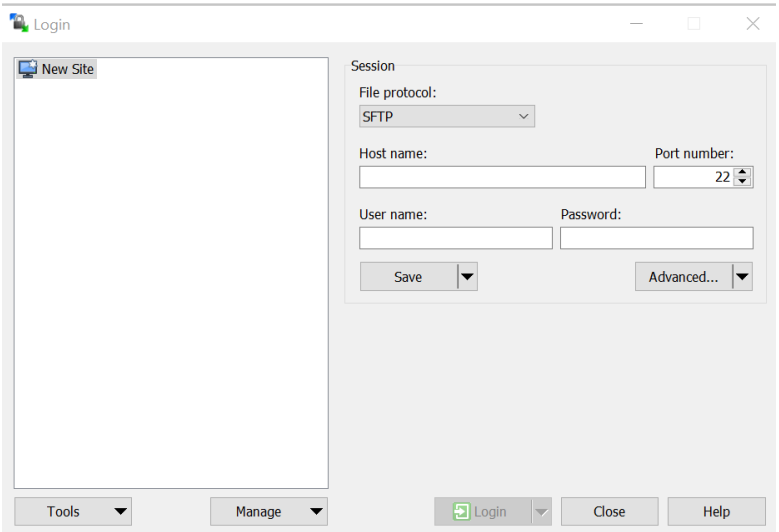
```
ubuntu@ip-172-31-39-39:~$ javac

Command 'javac' not found, but can be installed with:

sudo apt install default-jdk
sudo apt install openjdk-11-jdk-headless
sudo apt install ecj
sudo apt install openjdk-8-jdk-headless
```

```
ubuntu@ip-172-31-39-39:~$ sudo apt install openjdk-8-jdk-headless
```

Now we have java 8 version installed. In order to run a java program, we need to copy the java program into our ec2 instance using WinSCP.

Below steps will show how to copy files from local to ec2 instance.

- Start, WinSCP, open application. Before make sure this application is download and installed in your local. If not, get the latest application installed.

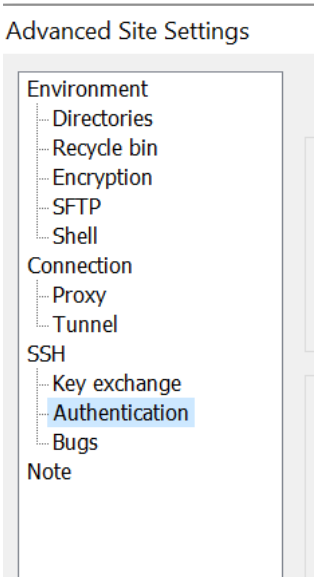


- Fill below details:

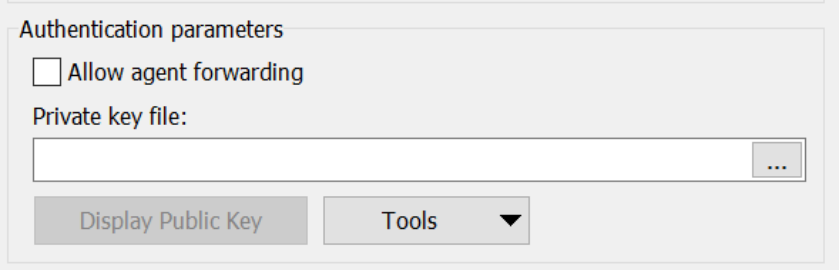
hostname: ec2-52-55-246-37.compute-1.amazonaws.com

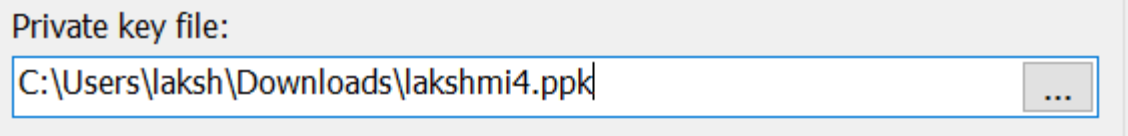
User name: ubuntu

For password: Click on the Advanced button. On your left, select authentication under SSH.

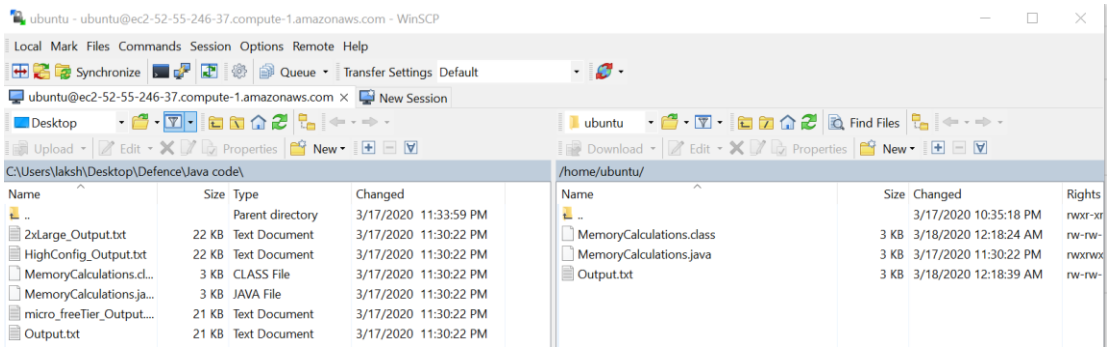


In authentication parameters, browse the private key file and OK

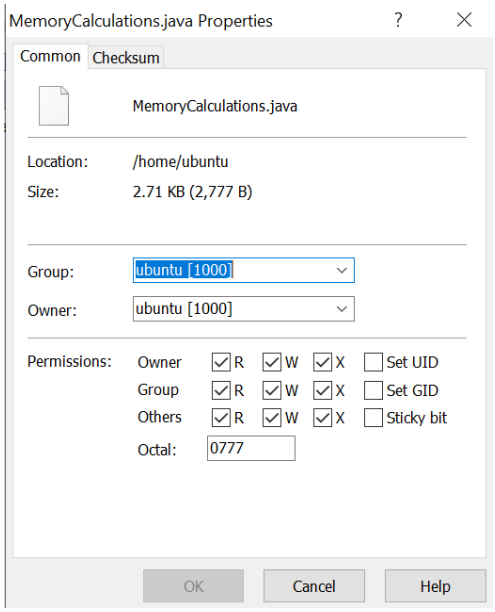




Below the left window is a local directory, and the right is ubuntu. Copy respective files from the left window and paste it in the right window.



Once files copied, right-click on the respective .java file, go to properties and grant permission “777” which means read, write and execute permissions to owner, group, and others



Now we have .java file in ubuntu instance root folder with all required permissions.

Java code:

Lets drive into java code, andthe name of the file is – MemoryCalculations.java

```
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.PrintStream;
import java.lang.management.ManagementFactory;
import java.lang.management.OperatingSystemMXBean;
import java.lang.reflect.Method;
import java.lang.reflect.Modifier;

public class MemoryCalculations {

public static void main(String[] args) throws IOException {
boolean append = true;
booleanautoFlush = false;
PrintStream out = new PrintStream(new FileOutputStream("/home/ubuntu/Output.txt",
append), autoFlush);
System.setOut(out);
System.out.println("**********Console Output**********");
```

```
long startTime1 = System.currentTimeMillis();
printUsage();
long stopTime1 = System.nanoTime();
long elapsedTime1 = stopTime1 - startTime1;
System.out.println("Time calculated in millisecond : " + elapsedTime1);
System.out.println("-------------");
long startTime2 = System.currentTimeMillis();
int x = 10;
int y = 10;
printUsage();
long stopTime2 = System.nanoTime();
long elapsedTime2 = stopTime2 - startTime2;
System.out.println("Time calculated in millisecond : " + elapsedTime2);
System.out.println("-------------");
long startTime3 = System.currentTimeMillis();
int z = x + y;
printUsage();
long stopTime3 = System.nanoTime();
long elapsedTime3 = stopTime3 - startTime3;
System.out.println("Time calculated in millisecond : " + elapsedTime3);
System.out.println("-------------");
long startTime4 = System.currentTimeMillis();
enhancedLoop();
printUsage();
long stopTime4 = System.nanoTime();
long elapsedTime4 = stopTime4 - startTime4;
System.out.println("Time calculated in millisecond : " + elapsedTime4);
System.out.println("-------------");
long startTime5 = System.currentTimeMillis();
printUsage();
long stopTime5 = System.nanoTime();
long elapsedTime5 = stopTime5 - startTime5;
System.out.println("Time calculated in millisecond : " + elapsedTime5);
System.out.println("-------------");

}
public static String printUsage() {
OperatingSystemMXBeanoperatingSystemMXBean                                    =
ManagementFactory.getOperatingSystemMXBean();
for (Method method :operatingSystemMXBean.getClass().getDeclaredMethods()) {
method.setAccessible(true);
if (method.getName().startsWith("get") &&Modifier.isPublic(method.getModifiers())) {
Object value;
```

```
try {
value = method.invoke(operatingSystemMXBean);
} catch (Exception e) {
value = e;
} // try
System.out.println(method.getName() + " = " + value);
} // if
} // for

return "";
}
public static void enhancedLoop() {
System.out.println("Using regular for loop.");
for (int i = 0; i< 50; i++) {
System.out.println(i);

}
}
}
```

Code Description:

1. First, all necessary packages were imported.
2. Get into the public class, which has 3 methods - main(), printUsage() and enhancedLoop().
3. The first method main() throws IOException. This method has mainly divided into sixparts:
    - Part1 (L14 - L18): Created an object “out” from PrintStream class in order to save executed console Output to the Output.txt file in the given directory.
    - Part2 (L20 – L25) : Simply callingprintUsage() method to see the performance metrics. Performance is calculated by doing elapsed time is the time difference between after and before the execution of code.

In Simple way, elapsed time = stopTime - startTime

- Part3 (L27 – L33): Assign two integer values to two different variables x and y.
- Part4 (L35 – L41): Perform the addition of x and y, and then assign addition value to variable z.
- Part5 (L43 – L49) : Calling enhancedLoop().
- Part6 (L51 – L56) : No additional execution of code, just calling performance calculation method printUsage().

4. Second method – printUsage() will result all various performance calculation metrics of the resources as listed below,

   getCommittedVirtualMemorySize

   getTotalSwapSpaceSize

   getFreeSwapSpaceSize

   getProcessCpuTime

   getFreePhysicalMemorySize

   getTotalPhysicalMemorySize

   getOpenFileDescriptorCount

   getMaxFileDescriptorCount

   getSystemCpuLoad

   getProcessCpuLoad

   Time calculated in millisecond

5. Third method – enhancedLoop(). This is an enhanced forloop which prints 1 to 49 integers.

As .java file is in place and ready to execute in the ubuntu instance, now execute above .java file as shown below,

Compile java code with javac MemoryCalculations.java

```
ubuntu@ip-172-31-39-39:~$ javac MemoryCalculations.java
```

```
ubuntu@ip-172-31-39-39:~$ ll
total 44
drwxr-xr-x 5 ubuntu ubuntu 4096 Mar 18 05:18 ./
drwxr-xr-x 3 root   root   4096 Mar 18 03:35 ../
-rw-r--r-- 1 ubuntu ubuntu  220 Apr  4  2018 .bash_logout
-rw-r--r-- 1 ubuntu ubuntu 3771 Apr  4  2018 .bashrc
drwx------ 2 ubuntu ubuntu 4096 Mar 18 03:43 .cache/
drwx------ 3 ubuntu ubuntu 4096 Mar 18 03:43 .gnupg/
-rw-r--r-- 1 ubuntu ubuntu  807 Apr  4  2018 .profile
drwx------ 2 ubuntu ubuntu 4096 Mar 18 03:35 .ssh/
-rw-r--r-- 1 ubuntu ubuntu    0 Mar 18 03:51 .sudo_as_admin_successful
-rw------- 1 ubuntu ubuntu  878 Mar 18 05:15 .viminfo
-rw-rw-r-- 1 ubuntu ubuntu 2712 Mar 18 05:18 MemoryCalculations.class
-rwxrwxrwx 1 ubuntu ubuntu 2777 Mar 18 04:30 MemoryCalculations.java*
```

Once compiled, now execute .class file as java MemoryCalculations

```
ubuntu@ip-172-31-39-39:~$ java MemoryCalculations
```

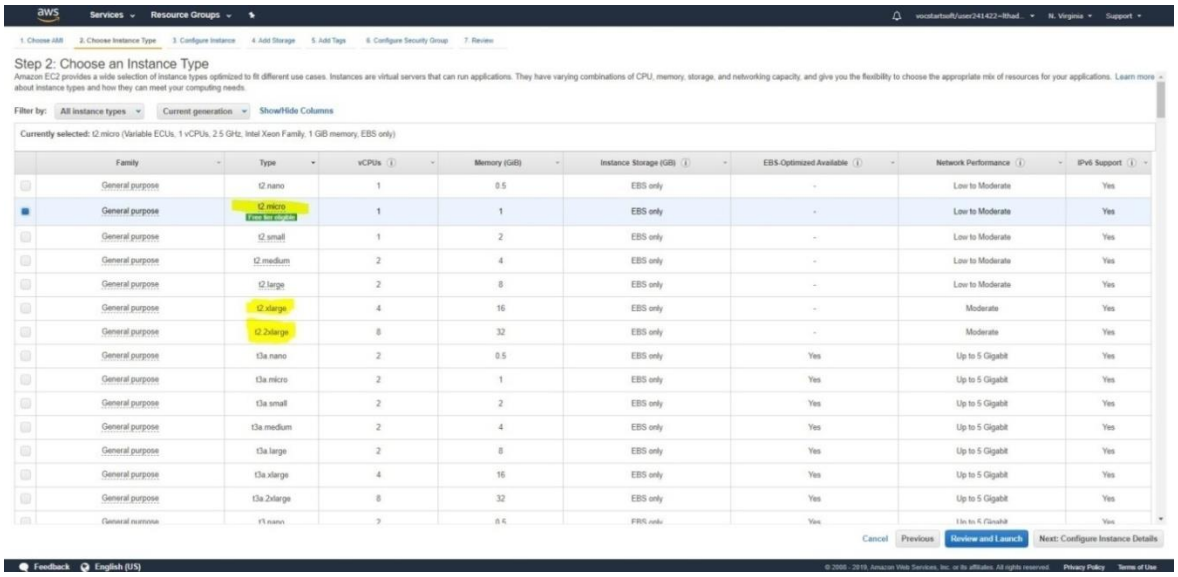
After code is executed, we see the Output.txt file generated. This .txt has the console output of the executed file, as shown below.

```
ubuntu@ip-172-31-39-39:~$ ll
total 48
drwxr-xr-x 5 ubuntu ubuntu 4096 Mar 18 05:18 ./
drwxr-xr-x 3 root   root   4096 Mar 18 03:35 ../
-rw-r--r-- 1 ubuntu ubuntu  220 Apr  4  2018 .bash_logout
-rw-r--r-- 1 ubuntu ubuntu 3771 Apr  4  2018 .bashrc
drwx------ 2 ubuntu ubuntu 4096 Mar 18 03:43 .cache/
drwx------ 3 ubuntu ubuntu 4096 Mar 18 03:43 .gnupg/
-rw-r--r-- 1 ubuntu ubuntu  807 Apr  4  2018 .profile
drwx------ 2 ubuntu ubuntu 4096 Mar 18 03:35 .ssh/
-rw-r--r-- 1 ubuntu ubuntu    0 Mar 18 03:51 .sudo_as_admin_successful
-rw------- 1 ubuntu ubuntu  878 Mar 18 05:15 .viminfo
-rw-rw-r-- 1 ubuntu ubuntu 2712 Mar 18 05:18 MemoryCalculations.class
-rwxrwxrwx 1 ubuntu ubuntu 2777 Mar 18 04:30 MemoryCalculations.java*
-rw-rw-r-- 1 ubuntu ubuntu 2119 Mar 18 05:18 Output.txt
```

Similarly, repeate same process for other two instances. Figure 11 shows all the three instances with respective configurations.

Figure 11

Three Ubuntu Instances



Three instance configuration details are described below,

1. t2.micro:
    - Instance type: Ubuntu
    - vCPUs: 1
    - Memory (GB) : 1
2. t2.xlarge:
    - Instance type: Ubuntu
    - vCPUs: 4
    - Memory (GB) : 16
3. t2.2xlarge:
    - Instance type: Ubuntu
    - vCPUs: 8
    - Memory (GB) : 32

The above-described java code has been executed in the above listed three instances. As shown, three output files generated in three different machines are micro_freetier.txt, Highconfig_Output.txt, and 2xLarge_Output.txt files, respectively. These .txt files details described below,

1. micro_freetier.txt - Contains Run1 to Run10 with 11 performance metrics for fivelines of Java code.
2. HighConfig_Output.txt - Contains Run1 to Run10 with 11 performance metrics for five lines of Java code.

3. 2xLarge_Output.txt - Contains Run1 to Run10 with 11 performance metrics for five lines of Java code.

The data collected in these files are used to train machine learning and expect to generate an algorithm out of it. So, in order to train Machine Learning, these .txt files are modified from horizontal records to vertical retrieving records and saved in .xlsx format as described below,

1. micro_freetier.xlsx - Contains columnar based values of micro_freetier.txt

2. HighConfig_Output.xlsx - Contains columnar based values of HighConfig_Output.txt

3. 2xLarge_Output.xlsx - Contains columnar based values of 2xLarge_Output.txt

All data from the above.txt files are gathered together and prepared table and saved in DataPrep.xlsx and this file format is converted to .csv as Notebook instance accepts .csv file formats. Columns which have similar values are eliminated. After similar value columns elimination, DataPrep.xlsx contains below columns,

1. CPU
2. Memory
3. Int_CommittedVirtualMemorySize
4. Int_ProcessCpuTime
5. Int_FreePhysicalMemorySize
6. Int_TotalPhysicalMemorySize

7. Int_SystemCpuLoad
8. Int_ProcessCpuLoad
9. Int_Time
10. Var_CommittedVirtualMemorySize
11. Var_ProcessCpuTime
12. Var_FreePhysicalMemorySize
13. Var_TotalPhysicalMemorySize
14. Var_SystemCpuLoad
15. Var_ProcessCpuLoad
16. Var_Time
17. Sum_CommittedVirtualMemorySize
18. Sum_ProcessCpuTime
19. Sum_FreePhysicalMemorySize
20. Sum_TotalPhysicalMemorySize
21. Sum_SystemCpuLoad
22. Sum_Time
23. loop_CommittedVirtualMemorySize
24. loop_ProcessCpuTime
25. loop_FreePhysicalMemorySize
26. loop_TotalPhysicalMemorySize
27. loop_SystemCpuLoad
28. loop_ProcessCpuLoad

29. loop_Time
30. final_CommittedVirtualMemorySize
31. final_ProcessCpuTime
32. final_FreePhysicalMemorySize
33. final_TotalPhysicalMemorySize
34. final_Time

Access AWS SageMaker – Notebook Instance

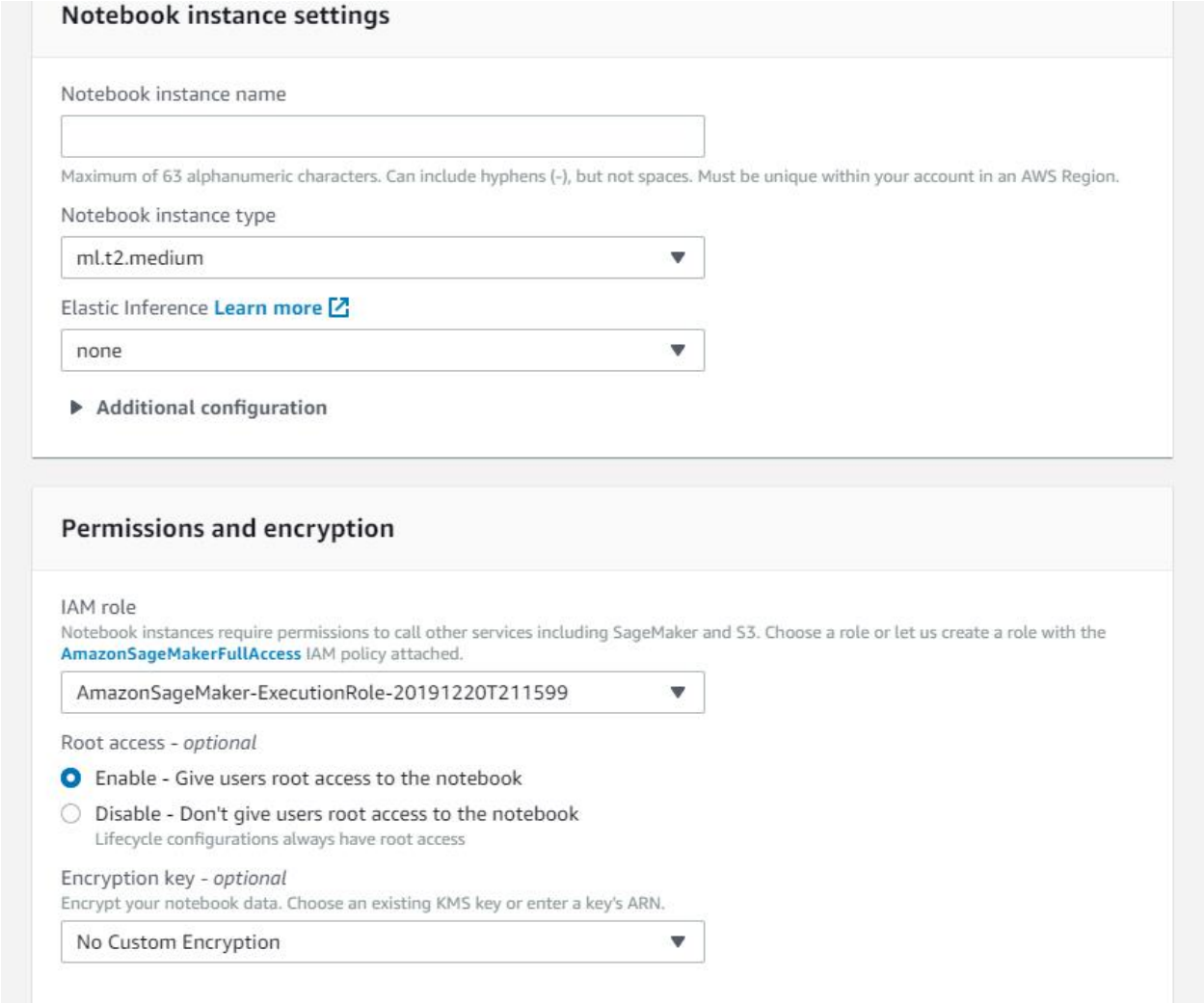
Amazon SageMaker is one of the Machine Learning Services, where data can be feed as input for the ML Algorithm to train and predict. Create a Notebook Instance in the SageMaker where the set of commands can be executed to test and train data.

Below are the instructions which are followed to create Notebook Instance:

Once you access Amazon SageMaker, scroll down to Notebook services, and select Notebook Instance. Once you get into the page, create new Instance with the details shown in Figure 12.

Figure 12

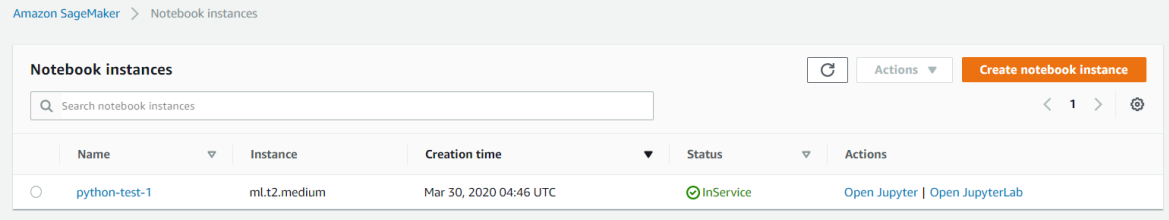
Notebook Instance Settings, Permission and Encryption details



Once Instance is created, under Actions -> select Open Jupyter and wait until the status shows InService in Figure 13.

Figure 13

Notebook instance status



Once Jupyter Notebook is opened, upload DataPrep.csv file into the instance as shown in Figure 14.

Figure 14

Upload option in Notebook Instance

Also, create a conda_python3 shell to execute commands. For that, select dropdown button New > conda_python3 as shown in Figure 15.

Figure 15

Notebook type – conda_python3

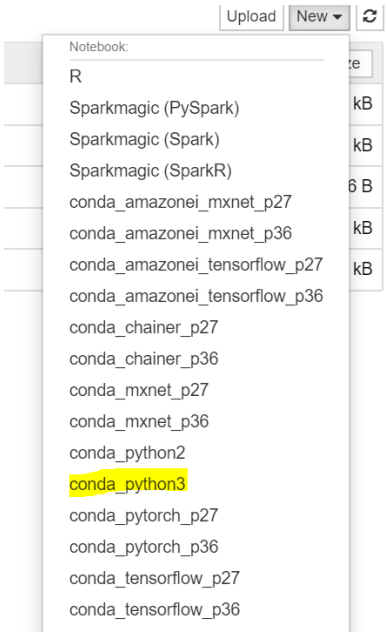


Figure 16

File uploaded in Jupyter Python nNotebook – PredictData_Instance

Few testing sheets were created to test the code. The actual results were captured in PredictData_Instance.ipynb. DataPrep.csv is the file uploaded that contains data for training and prediction.

Phase 2 – Data Training

Import “pandas” package to read DataPrep.csv and save the data in the panda's data frame “df” As shown in Figure 17.

df.head () prints the contents of the data frame.(*Machine Learning Tutorial Python—7: Training and Testing Data—YouTube*, 2018)

Figure 17

Read CSV file into a dataframe

```
In [1]:  import pandas as pd
         df = pd.read_csv("DataPrep.csv")
         df.head()
```

Out[1]:

	CPU	Memory	Int_CommittedVirtualMemorySize	Int_ProcessCpuTime	Int_FreePhysicalMemorySize	Int_TotalPhysicalMemorySize	Int_SystemCpuLoad	Int
0	1	1	2208010240	60000000	143585280	1038839808	0.003497	
1	1	1	2208010240	60000000	142438400	1038839808	0.003322	
2	1	1	2208010240	60000000	142774272	1038839808	0.003302	
3	1	1	2208010240	60000000	143343616	1038839808	0.003305	
4	1	1	2208010240	50000000	145678336	1038839808	0.003309	

5 rows × 34 columns

Import matplotlib.pyplot, this package will plot graphs across X and Y-axis variables assigned are shown in Figure 18. (*Py/train_test_split.ipynb at master · codebasics/py · GitHub*, 2018)

Figure 18

Import matplotlib

```
In [2]:  import matplotlib.pyplot as plt
         %matplotlib inline
```

The graph is plotted with the below variables CPU on X-axis and the contents of data frame on Y-axis which is shown in Figure 19.

Figure 19

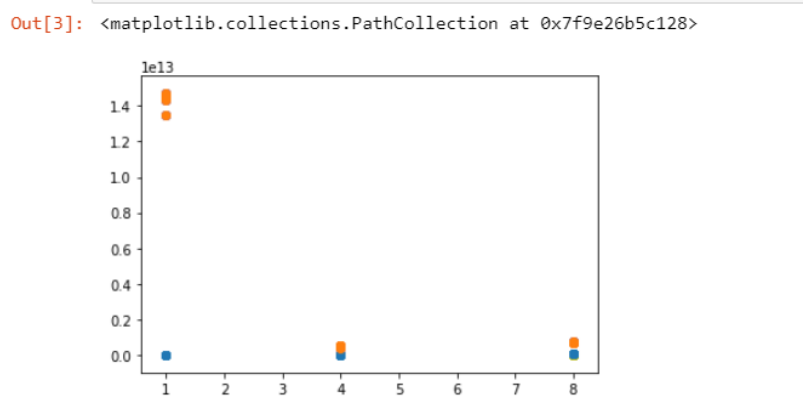
Dataframe for Graph plot between Dependent and Independent columns I

```
In [3]:  plt.scatter(df['CPU'],df['Int_CommittedVirtualMemorySize'])
         plt.scatter(df['CPU'],df['Int_ProcessCpuTime'])
         plt.scatter(df['CPU'],df['Int_FreePhysicalMemorySize'])
         plt.scatter(df['CPU'],df['Int_TotalPhysicalMemorySize'])
         plt.scatter(df['CPU'],df['Int_SystemCpuLoad'])
         plt.scatter(df['CPU'],df['Int_ProcessCpuLoad'])
         plt.scatter(df['CPU'],df['Int_Time'])

         plt.scatter(df['CPU'],df['Var_CommittedVirtualMemorySize'])
         plt.scatter(df['CPU'],df['Var_ProcessCpuTime'])
         plt.scatter(df['CPU'],df['Var_FreePhysicalMemorySize'])
         plt.scatter(df['CPU'],df['Var_TotalPhysicalMemorySize'])
         plt.scatter(df['CPU'],df['Var_SystemCpuLoad'])
         plt.scatter(df['CPU'],df['Var_ProcessCpuLoad'])
         plt.scatter(df['CPU'],df['Var_Time'])

         plt.scatter(df['CPU'],df['Sum_CommittedVirtualMemorySize'])
         plt.scatter(df['CPU'],df['Sum_ProcessCpuTime'])
         plt.scatter(df['CPU'],df['Sum_FreePhysicalMemorySize'])
         plt.scatter(df['CPU'],df['Sum_TotalPhysicalMemorySize'])
         plt.scatter(df['CPU'],df['Sum_SystemCpuLoad'])
         plt.scatter(df['CPU'],df['Sum_Time'])

         plt.scatter(df['CPU'],df['loop_CommittedVirtualMemorySize'])
         plt.scatter(df['CPU'],df['loop_ProcessCpuTime'])
         plt.scatter(df['CPU'],df['loop_FreePhysicalMemorySize'])
         plt.scatter(df['CPU'],df['loop_TotalPhysicalMemorySize'])
         plt.scatter(df['CPU'],df['loop_SystemCpuLoad'])
         plt.scatter(df['CPU'],df['loop_ProcessCpuLoad'])
         plt.scatter(df['CPU'],df['loop_Time'])

         plt.scatter(df['CPU'],df['final_CommittedVirtualMemorySize'])
         plt.scatter(df['CPU'],df['final_ProcessCpuTime'])
         plt.scatter(df['CPU'],df['final_FreePhysicalMemorySize'])
         plt.scatter(df['CPU'],df['final_TotalPhysicalMemorySize'])
         plt.scatter(df['CPU'],df['final_Time'])
```

Below graph is been plotted between CPU and one of the dependent columns as shown in Figure 20.

Figure 20

Graph plot between Dependent and Independent columns I



Similarly, the additional graph is plotted between one of the independent (Memory) and the dependent columns as shown in Figure 21.

Figure 21

Dataframe for Graph plot between Dependent and Independent columns II

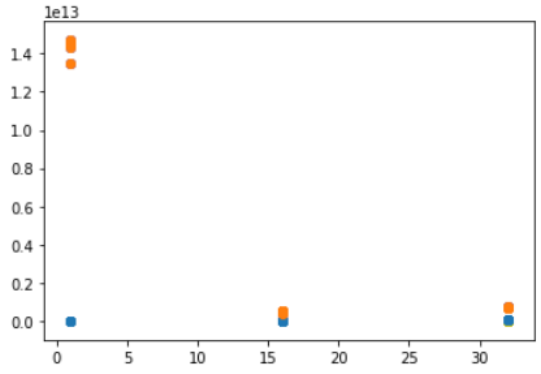
```
In [4]:
plt.scatter(df['Memory'],df['Int_CommittedVirtualMemorySize'])
plt.scatter(df['Memory'],df['Int_ProcessCpuTime'])
plt.scatter(df['Memory'],df['Int_FreePhysicalMemorySize'])
plt.scatter(df['Memory'],df['Int_TotalPhysicalMemorySize'])
plt.scatter(df['Memory'],df['Int_SystemCpuLoad'])
plt.scatter(df['Memory'],df['Int_ProcessCpuLoad'])
plt.scatter(df['Memory'],df['Int_Time'])

plt.scatter(df['Memory'],df['Var_CommittedVirtualMemorySize'])
plt.scatter(df['Memory'],df['Var_ProcessCpuTime'])
plt.scatter(df['Memory'],df['Var_FreePhysicalMemorySize'])
plt.scatter(df['Memory'],df['Var_TotalPhysicalMemorySize'])
plt.scatter(df['Memory'],df['Var_SystemCpuLoad'])
plt.scatter(df['Memory'],df['Var_ProcessCpuLoad'])
plt.scatter(df['Memory'],df['Var_Time'])

plt.scatter(df['Memory'],df['Sum_CommittedVirtualMemorySize'])
plt.scatter(df['Memory'],df['Sum_ProcessCpuTime'])
plt.scatter(df['Memory'],df['Sum_FreePhysicalMemorySize'])
plt.scatter(df['Memory'],df['Sum_TotalPhysicalMemorySize'])
plt.scatter(df['Memory'],df['Sum_SystemCpuLoad'])
plt.scatter(df['Memory'],df['Sum_Time'])

plt.scatter(df['Memory'],df['loop_CommittedVirtualMemorySize'])
plt.scatter(df['Memory'],df['loop_ProcessCpuTime'])
plt.scatter(df['Memory'],df['loop_FreePhysicalMemorySize'])
plt.scatter(df['Memory'],df['loop_TotalPhysicalMemorySize'])
plt.scatter(df['Memory'],df['loop_SystemCpuLoad'])
plt.scatter(df['Memory'],df['loop_ProcessCpuLoad'])
plt.scatter(df['Memory'],df['loop_Time'])

plt.scatter(df['Memory'],df['final_CommittedVirtualMemorySize'])
plt.scatter(df['Memory'],df['final_ProcessCpuTime'])
plt.scatter(df['Memory'],df['final_FreePhysicalMemorySize'])
plt.scatter(df['Memory'],df['final_TotalPhysicalMemorySize'])
plt.scatter(df['Memory'],df['final_Time'])
```

Graph is been plotted between Memory and one of the dependent columns as shown in Figure 22.

Figure 22

Graph plot between Dependent and Independent columns II

```
Out[4]: <matplotlib.collections.PathCollection at 0x7f9e26abd4a8>
```



CPU and Memory are two independent variables that are saved like an array and assigned to independent_var as derived in Figure 23.

Figure 23

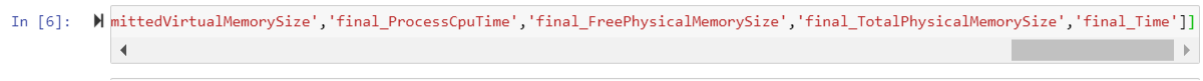
Independent variable assignment

```
In [5]: independent_var = df[['CPU','Memory']]
```

All other 32 columns are dependent variables that are saved like an array and assigned to dependent_var as derived in Figure 24.

Figure 24

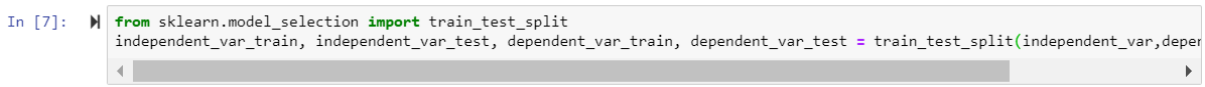
Dependent variable assignment



In Figure 25, 30% of data is taken out for testing machine learning for prediction whereas 70% of data is to feed data for training.

Figure 25

Independent variable assignment



The random data picked up by the ML algorithm for training is shown in Figure 26 and Figure 27:

Figure 26

Independent train variable

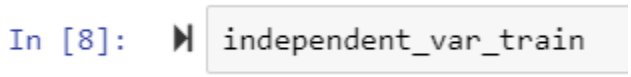


Figure 27

Independent train variable data

Out[8]:

	CPU	Memory
2	1	1
28	8	32
1	1	1
29	8	32
27	8	32
7	1	1
18	4	16
5	1	1
23	8	32
17	4	16
25	8	32
3	1	1
20	8	32
9	1	1
24	8	32
11	4	16
16	4	16
13	4	16
22	8	32
19	4	16
6	1	1

Figure 28 is the 30% of the data that is given for ML to test. We are not feeding this data as to not have any clue for the prediction. Figure 28 shows is the actual output predicting from ML output.

Figure 28

Independent test variable with data

```
In [9]: independent_var_test
```

Out[9]:

	CPU	Memory
26	8	32
4	1	1
15	4	16
14	4	16
0	1	1
8	1	1
10	4	16
21	8	32
12	4	16

Figure 29 represents dependent_var_train is the dataset used to train ML

Figure 29

Dependent train variable

```
In [10]: dependent_var_train
```

Out[10]:

	Int_CommittedVirtualMemorySize	Int_ProcessCpuTime	Int_FreePhysicalMemorySize	Int_TotalPhysicalMemorySize	Int_SystemCpuLoad	Int_ProcessCpuLo
2	2208010240	60000000	142774272	1038839808	0.003302	0.0000
28	11468390400	60000000	33294565376	33737129984	0.001429	0.0000
1	2208010240	60000000	142438400	1038839808	0.003322	0.0000
29	11468390400	60000000	33295867904	33737129984	0.001426	0.0000
27	11468390400	60000000	33294299136	33737129984	0.001432	0.0000
7	2208010240	50000000	142397440	1038839808	0.003307	0.0000
18	6732087296	60000000	15030919168	16825704448	0.011551	0.0000
5	2208010240	60000000	142438400	1038839808	0.003299	0.0000
23	11468390400	60000000	33295187968	33737129984	0.001437	0.0000
17	6732087296	60000000	15029895168	16825704448	0.011572	0.0000
25	11468390400	60000000	33294458880	33737129984	0.001432	0.0000
3	2208010240	60000000	143343616	1038839808	0.003305	0.0000
20	11468390400	60000000	33294262272	33737129984	0.001490	0.0000
9	2208010240	50000000	142295040	1038839808	0.003291	0.0000
24	11468390400	60000000	33295400960	33737129984	0.001434	0.0000
11	6732087296	60000000	15029694464	16825704448	0.011676	0.0000
16	6732087296	60000000	15030718464	16825704448	0.011588	0.0000
13	6732087296	60000000	15030075392	16825704448	0.011629	0.0000
22	11468390400	60000000	33294704640	33737129984	0.001437	0.0000
19	6732087296	60000000	15029383168	16825704448	0.011535	0.0000
6	2208010240	60000000	142184448	1038839808	0.003303	0.0000

21 rows × 32 columns

dependent_var _test here refers to the expected output needs to be predicted by the ML algorithm for the input given from dependent_var _test as represented in Figure30.

Figure 30

Dependent test variable

```
In [11]: dependent_var_test
```

Out[11]:

	Int_CommittedVirtualMemorySize	Int_ProcessCpuTime	Int_FreePhysicalMemorySize	Int_TotalPhysicalMemorySize	Int_SystemCpuLoad	Int_ProcessCpuLo
26	11468390400	60000000	33295233024	33737129984	0.001434	0.0000
4	2208010240	50000000	145678336	1038839808	0.003309	0.0000
15	6732087296	60000000	15029145600	16825704448	0.011603	0.0000
14	6732087296	50000000	15032500224	16825704448	0.011611	0.0000
0	2208010240	60000000	143585280	1038839808	0.003497	0.0000
8	2208010240	50000000	143708160	1038839808	0.003311	0.0000
10	6732087296	60000000	15030235136	16825704448	0.012339	0.0000
21	11468390400	60000000	33293635584	33737129984	0.001438	0.0000
12	6732087296	60000000	15028113408	16825704448	0.011650	0.0000

9 rows × 32 columns

Linear regression is a linear approach to modeling the relationship between a scalar response (or dependent variable) and one or more explanatory variables (or independent variables). This package is imported as shoen in Figure 31.(*Linear regression—Wikipedia*, n.d.)

Figure 31

LinearRegression Import

```
In [12]: from sklearn.linear_model import LinearRegression
         clf = LinearRegression()
         clf.fit(independent_var_train, dependent_var_train)

Out[12]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
                          normalize=False)
```

Phase 3 - Data Prediction:

With the regression variable, the prediction is performed to get independent values when dependent values are fed as input in Figure 32, Figure33, Figure 34.

Figure 32

Prediction of Independent test variable I

```
In [13]: clf.predict(independent_var_test)

Out[13]: array([[ 1.14683904e+10,  6.00000000e+07,  3.32948434e+10,
                  3.37371300e+10,  1.43967338e-03,  9.12938750e-06,
                  6.92631674e+11,  1.14683904e+10,  6.37500000e+07,
                  3.32947958e+10,  3.37371300e+10,  9.37500000e-02,
                  2.50000000e-01,  6.92631043e+11,  1.14683904e+10,
                  6.62500000e+07,  3.32926331e+10,  3.37371300e+10,
                  1.11022302e-16,  6.92630522e+11,  1.14683904e+10,
                  6.87500000e+07,  3.32926331e+10,  3.37371300e+10,
                  1.87500000e-01,  1.25000000e-01,  6.92629073e+11,
                  1.14683904e+10,  6.87500000e+07,  3.32926331e+10,
                  3.37371300e+10,  6.92628603e+11],
                [ 2.20801024e+09,  5.71428571e+07,  1.42553088e+08,
                  1.03883981e+09,  3.30408800e-03,  3.55820000e-06,
                  1.45275861e+13,  2.20801024e+09,  6.00000000e+07,
                  1.42553088e+08,  1.03883981e+09,  5.71428571e-01,
                  3.05311332e-16,  1.45275875e+13,  2.20801024e+09,
                  6.00000000e+07,  1.42498670e+08,  1.03883981e+09,
                  2.85714286e-01,  1.45275884e+13,  2.20801024e+09,
                  6.57142857e+07,  1.42498670e+08,  1.03883981e+09,
                  1.42857143e-01,  1.52655666e-16,  1.45275908e+13,
                  2.20801024e+09,  6.71428571e+07,  1.42498670e+08,
                  1.03883981e+09,  1.45275936e+13],
                [ 6.73208730e+09,  6.00000000e+07,  1.50301143e+10,
                  1.68257044e+10,  1.15919003e-02,  7.29775000e-06,
                  5.06965182e+11,  6.73208730e+09,  6.16666667e+07,
                  1.50299116e+10,  1.68257044e+10,  4.16666667e-01,
                 -1.11022302e-16,  5.06965783e+11,  6.73208730e+09,
                  6.33333333e+07,  1.50277482e+10,  1.68257044e+10,
                  1.11022302e-16,  5.06966299e+11,  6.73208730e+09,
                  6.33333333e+07,  1.50277482e+10,  1.68257044e+10,
                  3.33333333e-01, -5.55111512e-17,  5.06967763e+11,
                  6.73208730e+09,  6.50000000e+07,  1.50277482e+10,
                  1.68257044e+10,  5.06968228e+11],
```

Figure 33

Prediction of Independent test variable II

```
  [ 6.73208730e+09,  6.00000000e+07,  1.50301143e+10,
    1.68257044e+10,  1.15919003e-02,  7.29775000e-06,
    5.06965182e+11,  6.73208730e+09,  6.16666667e+07,
    1.50299116e+10,  1.68257044e+10,  4.16666667e-01,
   -1.11022302e-16,  5.06965783e+11,  6.73208730e+09,
    6.33333333e+07,  1.50277482e+10,  1.68257044e+10,
    1.11022302e-16,  5.06966299e+11,  6.73208730e+09,
    6.33333333e+07,  1.50277482e+10,  1.68257044e+10,
    3.33333333e-01, -5.55111512e-17,  5.06967763e+11,
    6.73208730e+09,  6.50000000e+07,  1.50277482e+10,
    1.68257044e+10,  5.06968228e+11],
  [ 2.20801024e+09,  5.71428571e+07,  1.42553088e+08,
    1.03883981e+09,  3.30408800e-03,  3.55820000e-06,
    1.45275861e+13,  2.20801024e+09,  6.00000000e+07,
    1.42553088e+08,  1.03883981e+09,  5.71428571e-01,
    3.05311332e-16,  1.45275875e+13,  2.20801024e+09,
    6.00000000e+07,  1.42498670e+08,  1.03883981e+09,
    2.85714286e-01,  1.45275884e+13,  2.20801024e+09,
    6.57142857e+07,  1.42498670e+08,  1.03883981e+09,
    1.42857143e-01,  1.52655666e-16,  1.45275908e+13,
    2.20801024e+09,  6.71428571e+07,  1.42498670e+08,
    1.03883981e+09,  1.45275936e+13],
  [ 2.20801024e+09,  5.71428571e+07,  1.42553088e+08,
    1.03883981e+09,  3.30408800e-03,  3.55820000e-06,
    1.45275861e+13,  2.20801024e+09,  6.00000000e+07,
    1.42553088e+08,  1.03883981e+09,  5.71428571e-01,
    3.05311332e-16,  1.45275875e+13,  2.20801024e+09,
    6.00000000e+07,  1.42498670e+08,  1.03883981e+09,
    2.85714286e-01,  1.45275884e+13,  2.20801024e+09,
    6.57142857e+07,  1.42498670e+08,  1.03883981e+09,
    1.42857143e-01,  1.52655666e-16,  1.45275908e+13,
    2.20801024e+09,  6.71428571e+07,  1.42498670e+08,
    1.03883981e+09,  1.45275936e+13],
```

Figure 34

Prediction of Independent test variable III

```
  [ 6.73208730e+09,  6.00000000e+07,  1.50301143e+10,
    1.68257044e+10,  1.15919003e-02,  7.29775000e-06,
    5.06965182e+11,  6.73208730e+09,  6.16666667e+07,
    1.50299116e+10,  1.68257044e+10,  4.16666667e-01,
   -1.11022302e-16,  5.06965783e+11,  6.73208730e+09,
    6.33333333e+07,  1.50277482e+10,  1.68257044e+10,
    1.11022302e-16,  5.06966299e+11,  6.73208730e+09,
    6.33333333e+07,  1.50277482e+10,  1.68257044e+10,
    3.33333333e-01, -5.55111512e-17,  5.06967763e+11,
    6.73208730e+09,  6.50000000e+07,  1.50277482e+10,
    1.68257044e+10,  5.06968228e+11],
  [ 1.14683904e+10,  6.00000000e+07,  3.32948434e+10,
    3.37371300e+10,  1.43967338e-03,  9.12938750e-06,
    6.92631674e+11,  1.14683904e+10,  6.37500000e+07,
    3.32947958e+10,  3.37371300e+10,  9.37500000e-02,
    2.50000000e-01,  6.92631043e+11,  1.14683904e+10,
    6.62500000e+07,  3.32926331e+10,  3.37371300e+10,
    1.11022302e-16,  6.92630522e+11,  1.14683904e+10,
    6.87500000e+07,  3.32926331e+10,  3.37371300e+10,
    1.87500000e-01,  1.25000000e-01,  6.92629073e+11,
    1.14683904e+10,  6.87500000e+07,  3.32926331e+10,
    3.37371300e+10,  6.92628603e+11],
  [ 6.73208730e+09,  6.00000000e+07,  1.50301143e+10,
    1.68257044e+10,  1.15919003e-02,  7.29775000e-06,
    5.06965182e+11,  6.73208730e+09,  6.16666667e+07,
    1.50299116e+10,  1.68257044e+10,  4.16666667e-01,
   -1.11022302e-16,  5.06965783e+11,  6.73208730e+09,
    6.33333333e+07,  1.50277482e+10,  1.68257044e+10,
    1.11022302e-16,  5.06966299e+11,  6.73208730e+09,
    6.33333333e+07,  1.50277482e+10,  1.68257044e+10,
    3.33333333e-01, -5.55111512e-17,  5.06967763e+11,
    6.73208730e+09,  6.50000000e+07,  1.50277482e+10,
    1.68257044e+10,  5.06968228e+11]])
```

Figure 35 shows actual dataset of dependent test variable.

```
In [15]:  dependent_var_test_predict = clf.predict(independent_var_test)
```

Figure 35

Prediction of Dependent test variable

```
In [14]:  dependent_var_test
```

Out[14]:

	Int_CommittedVirtualMemorySize	Int_ProcessCpuTime	Int_FreePhysicalMemorySize	Int_TotalPhysicalMemorySize	Int_SystemCpuLoad	Int_ProcessCpuLo
26	11468390400	60000000	3329233024	33737129984	0.001434	0.0000
4	2208010240	50000000	145678336	1038839808	0.003309	0.0000
15	6732087296	60000000	15029145600	16825704448	0.011603	0.0000
14	6732087296	50000000	15032500224	16825704448	0.011611	0.0000
0	2208010240	60000000	143585280	1038839808	0.003497	0.0000
8	2208010240	50000000	143708160	1038839808	0.003311	0.0000
10	6732087296	60000000	15030235136	16825704448	0.012339	0.0000
21	11468390400	60000000	3329635584	33737129984	0.001438	0.0000
12	6732087296	60000000	15028113408	16825704448	0.011650	0.0000

9 rows × 32 columns

dependent_var _test_predict is predicted dataset variable formatting the data type to be suitable with the actual data type.

Figure 36 shows the Numpy package import is to change the datatype format to float.

Figure 36

Numpy package import

```
In [16]:  import numpy as np
          np.set_printoptions(formatter={'float_kind':'{:f}'.format})
```

With the regression variable, the prediction is performed to get independent values when dependent values are fed as input in Figure 37, Figure38, Figure 39.

Figure 37

Prediction of Dependent test variable I

```
In [17]: dependent_var_test_predict

Out[17]: array([[11468390400.000000, 60000000.000000, 33294843392.000000,
                 33737129984.000000, 0.001440, 0.000009, 692631674054.098633,
                 11468390400.000000, 63750000.000000, 33294795775.999992,
                 33737129984.000000, 0.093750, 0.250000, 692631043259.231445,
                 11468390400.000000, 66250000.000000, 33292633087.999996,
                 33737129984.000000, 0.000000, 692630522006.495117,
                 11468390400.000000, 68750000.000000, 33292633087.999996,
                 33737129984.000000, 0.187500, 0.125000, 692629072803.094727,
                 11468390400.000000, 68750000.000000, 33292633087.999996,
                 33737129984.000000, 692628603387.866211],
                [2208010239.999999, 57142857.142857, 142553088.000002,
                 1038839808.000005, 0.003304, 0.000004, 14527586131415.859375,
                 2208010239.999999, 60000000.000000, 142553088.000003,
                 1038839808.000005, 0.571429, 0.000000, 14527587458637.722656,
                 2208010239.999999, 60000000.000000, 142498669.714288,
                 1038839808.000005, 0.285714, 14527588421344.867188,
                 2208010239.999999, 65714285.714286, 142498669.714288,
                 1038839808.000005, 0.142857, 0.000000, 14527590845919.292969,
                 2208010239.999999, 67142857.142857, 142498669.714288,
                 1038839808.000005, 14527593639241.031250],
                [6732087296.000000, 60000000.000000, 15030114304.000000,
                 16825704448.000002, 0.011592, 0.000007, 506965181906.489258,
                 6732087296.000000, 61666666.666667, 15029911551.999998,
                 16825704448.000002, 0.416667, -0.000000, 506965783388.497070,
                 6732087296.000000, 63333333.333333, 15027748181.333332,
                 16825704448.000002, 0.000000, 506966298861.338867,
                 6732087296.000000, 63333333.333333, 15027748181.333332,
                 16825704448.000002, 0.333333, -0.000000, 506967762730.657227,
                 6732087296.000000, 65000000.000000, 15027748181.333332,
                 16825704448.000002, 506968227537.678711],
```

Figure 38

Prediction of Dependent test variable II

```
 16825704448.000002, 506968227537.678711],
[6732087296.000000, 60000000.000000, 15030114304.000000,
 16825704448.000002, 0.011592, 0.000007, 506965181906.489258,
 6732087296.000000, 61666666.666667, 15029911551.999998,
 16825704448.000002, 0.416667, -0.000000, 506965783388.497070,
 6732087296.000000, 63333333.333333, 15027748181.333332,
 16825704448.000002, 0.000000, 506966298861.338867,
 6732087296.000000, 63333333.333333, 15027748181.333332,
 16825704448.000002, 0.333333, -0.000000, 506967762730.657227,
 6732087296.000000, 65000000.000000, 15027748181.333332,
 16825704448.000002, 506968227537.678711],
[2208010239.999999, 57142857.142857, 142553088.000002,
 1038839808.000005, 0.003304, 0.000004, 14527586131415.859375,
 2208010239.999999, 60000000.000000, 142553088.000003,
 1038839808.000005, 0.571429, 0.000000, 14527587458637.722656,
 2208010239.999999, 60000000.000000, 142498669.714288,
 1038839808.000005, 0.285714, 14527588421344.867188,
 2208010239.999999, 65714285.714286, 142498669.714288,
 1038839808.000005, 0.142857, 0.000000, 14527590845919.292969,
 2208010239.999999, 67142857.142857, 142498669.714288,
 1038839808.000005, 14527593639241.031250],
[2208010239.999999, 57142857.142857, 142553088.000002,
 1038839808.000005, 0.003304, 0.000004, 14527586131415.859375,
 2208010239.999999, 60000000.000000, 142553088.000003,
 1038839808.000005, 0.571429, 0.000000, 14527587458637.722656,
 2208010239.999999, 60000000.000000, 142498669.714288,
 1038839808.000005, 0.285714, 14527588421344.867188,
 2208010239.999999, 65714285.714286, 142498669.714288,
 1038839808.000005, 0.142857, 0.000000, 14527590845919.292969,
 2208010239.999999, 67142857.142857, 142498669.714288,
 1038839808.000005, 14527593639241.031250],
```

Figure 39

Prediction of Dependent test variable III

```
 [6732087296.000000, 60000000.000000, 15030114304.000000,
  16825704448.000002, 0.011592, 0.000007, 506965181906.489258,
  6732087296.000000, 61666666.666667, 15029911551.999998,
  16825704448.000002, 0.416667, -0.000000, 506965783388.497070,
  6732087296.000000, 63333333.333333, 15027748181.333332,
  16825704448.000002, 0.000000, 506966298861.338867,
  6732087296.000000, 63333333.333333, 15027748181.333332,
  16825704448.000002, 0.333333, -0.000000, 506967762730.657227,
  6732087296.000000, 65000000.000000, 15027748181.333332,
  16825704448.000002, 506968227537.678711],
 [11468390400.000000, 60000000.000000, 33294843392.000000,
  33737129984.000000, 0.001440, 0.000009, 692631674054.098633,
  11468390400.000000, 63750000.000000, 33294795775.999992,
  33737129984.000000, 0.093750, 0.250000, 692631043259.231445,
  11468390400.000000, 66250000.000000, 33292633087.999996,
  33737129984.000000, 0.000000, 692630522006.495117,
  11468390400.000000, 68750000.000000, 33292633087.999996,
  33737129984.000000, 0.187500, 0.125000, 692629072803.094727,
  11468390400.000000, 68750000.000000, 33292633087.999996,
  33737129984.000000, 692628603387.866211],
 [6732087296.000000, 60000000.000000, 15030114304.000000,
  16825704448.000002, 0.011592, 0.000007, 506965181906.489258,
  6732087296.000000, 61666666.666667, 15029911551.999998,
  16825704448.000002, 0.416667, -0.000000, 506965783388.497070,
  6732087296.000000, 63333333.333333, 15027748181.333332,
  16825704448.000002, 0.000000, 506966298861.338867,
  6732087296.000000, 63333333.333333, 15027748181.333332,
  16825704448.000002, 0.333333, -0.000000, 506967762730.657227,
  6732087296.000000, 65000000.000000, 15027748181.333332,
  16825704448.000002, 506968227537.678711]])
```

Output match of datasets between actual and predicted is 99.72%. this can be seen in Figure 40.

Figure 40

ActualVs Predicted score

```
In [18]: clf.score(independent_var_test, dependent_var_test)
Out[18]: 0.9972016951887358
```

Phase 4 - Comparison and Decision statement:

Below graphs are been plotted between Actual and Predicted values.

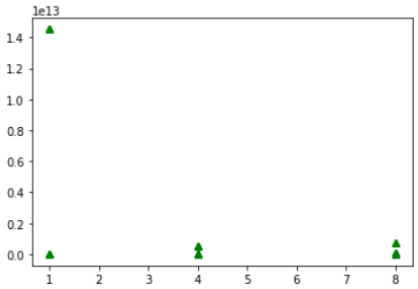
Figure 41 shows the graph plotted between CPU and Dependent variables.

Predicted results:

Figure 41

Predicted Graph plot between CPU and Dependent variables

```
In [19]: newdata = np.squeeze(dependent_var_test_predict)
plt.plot(independent_var_test['CPU'],newdata[:,0],'g^',independent_var_test['CPU'],newdata[:,1],'g^',independent_var_test['CP
plt.show()
```

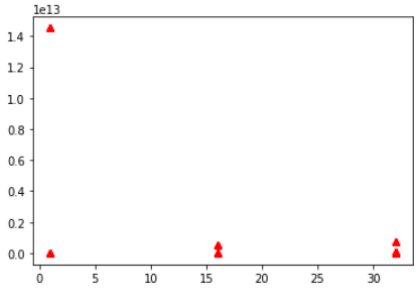


Graph is been plotted between Memory and Dependent variables as shown in Figure 42.

Figure 42

Predicted Graph plot between Memory and Dependent variables

```
In [20]:
newdata = np.squeeze(dependent_var_test_predict)
plt.plot(independent_var_test['Memory'],newdata[:,0],'r^',independent_var_test['Memory'],newdata[:,1],'r^',independent_var_te
plt.show()
```



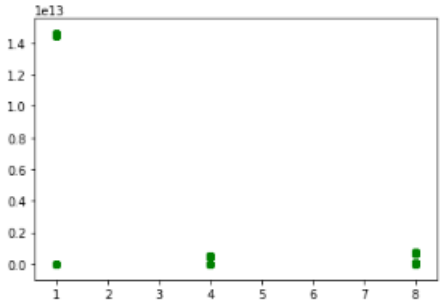
Actual results:

Figure 43

Actual Graph plot between CPU and Dependent variables

```
plt.scatter(independent_var_test['CPU'],dependent_var_test['Int_CommittedVirtualMemorySize'],color=['green'])
plt.scatter(independent_var_test['CPU'],dependent_var_test['Int_ProcessCpuTime'],color=['green'])
plt.scatter(independent_var_test['CPU'],dependent_var_test['Int_FreePhysicalMemorySize'],color=['green'])
plt.scatter(independent_var_test['CPU'],dependent_var_test['Int_TotalPhysicalMemorySize'],color=['green'])
plt.scatter(independent_var_test['CPU'],dependent_var_test['Int_SystemCpuLoad'],color=['green'])
plt.scatter(independent_var_test['CPU'],dependent_var_test['Int_ProcessCpuLoad'],color=['green'])
plt.scatter(independent_var_test['CPU'],dependent_var_test['Int_Time'],color=['green'])

plt.scatter(independent_var_test['CPU'],dependent_var_test['Var_CommittedVirtualMemorySize'],color=['green'])
plt.scatter(independent_var_test['CPU'],dependent_var_test['Var_ProcessCpuTime'],color=['green'])
plt.scatter(independent_var_test['CPU'],dependent_var_test['Var_FreePhysicalMemorySize'],color=['green'])
plt.scatter(independent_var_test['CPU'],dependent_var_test['Var_TotalPhysicalMemorySize'],color=['green'])
plt.scatter(independent_var_test['CPU'],dependent_var_test['Var_SystemCpuLoad'],color=['green'])
plt.scatter(independent_var_test['CPU'],dependent_var_test['Var_ProcessCpuLoad'],color=['green'])
plt.scatter(independent_var_test['CPU'],dependent_var_test['Var_Time'],color=['green'])

plt.scatter(independent_var_test['CPU'],dependent_var_test['Sum_CommittedVirtualMemorySize'],color=['green'])
plt.scatter(independent_var_test['CPU'],dependent_var_test['Sum_ProcessCpuTime'],color=['green'])
plt.scatter(independent_var_test['CPU'],dependent_var_test['Sum_FreePhysicalMemorySize'],color=['green'])
plt.scatter(independent_var_test['CPU'],dependent_var_test['Sum_TotalPhysicalMemorySize'],color=['green'])
plt.scatter(independent_var_test['CPU'],dependent_var_test['Sum_SystemCpuLoad'],color=['green'])
plt.scatter(independent_var_test['CPU'],dependent_var_test['Sum_Time'],color=['green'])

plt.scatter(independent_var_test['CPU'],dependent_var_test['loop_CommittedVirtualMemorySize'],color=['green'])
plt.scatter(independent_var_test['CPU'],dependent_var_test['loop_ProcessCpuTime'],color=['green'])
plt.scatter(independent_var_test['CPU'],dependent_var_test['loop_FreePhysicalMemorySize'],color=['green'])
plt.scatter(independent_var_test['CPU'],dependent_var_test['loop_TotalPhysicalMemorySize'],color=['green'])
plt.scatter(independent_var_test['CPU'],dependent_var_test['loop_SystemCpuLoad'],color=['green'])
plt.scatter(independent_var_test['CPU'],dependent_var_test['loop_ProcessCpuLoad'],color=['green'])
plt.scatter(independent_var_test['CPU'],dependent_var_test['loop_Time'],color=['green'])

plt.scatter(independent_var_test['CPU'],dependent_var_test['final_CommittedVirtualMemorySize'],color=['green'])
plt.scatter(independent_var_test['CPU'],dependent_var_test['final_ProcessCpuTime'],color=['green'])
plt.scatter(independent_var_test['CPU'],dependent_var_test['final_FreePhysicalMemorySize'],color=['green'])
plt.scatter(independent_var_test['CPU'],dependent_var_test['final_TotalPhysicalMemorySize'],color=['green'])
plt.scatter(independent_var_test['CPU'],dependent_var_test['final_Time'],color=['green'])
```

25]: <matplotlib.collections.PathCollection at 0x7f32d6560f28>



Actual results of CPU Vs Dependent variables are shown in Figure 43.

Figure 44

Actual Graph plot between Memory and Dependent variables

```
plt.scatter(independent_var_test['Memory'],dependent_var_test['Int_CommittedVirtualMemorySize'],color=['red'])
plt.scatter(independent_var_test['Memory'],dependent_var_test['Int_ProcessCpuTime'],color=['red'])
plt.scatter(independent_var_test['Memory'],dependent_var_test['Int_FreePhysicalMemorySize'],color=['red'])
plt.scatter(independent_var_test['Memory'],dependent_var_test['Int_TotalPhysicalMemorySize'],color=['red'])
plt.scatter(independent_var_test['Memory'],dependent_var_test['Int_SystemCpuLoad'],color=['red'])
plt.scatter(independent_var_test['Memory'],dependent_var_test['Int_ProcessCpuLoad'],color=['red'])
plt.scatter(independent_var_test['Memory'],dependent_var_test['Int_Time'],color=['red'])

plt.scatter(independent_var_test['Memory'],dependent_var_test['Var_CommittedVirtualMemorySize'],color=['red'])
plt.scatter(independent_var_test['Memory'],dependent_var_test['Var_ProcessCpuTime'],color=['red'])
plt.scatter(independent_var_test['Memory'],dependent_var_test['Var_FreePhysicalMemorySize'],color=['red'])
plt.scatter(independent_var_test['Memory'],dependent_var_test['Var_TotalPhysicalMemorySize'],color=['red'])
plt.scatter(independent_var_test['Memory'],dependent_var_test['Var_SystemCpuLoad'],color=['red'])
plt.scatter(independent_var_test['Memory'],dependent_var_test['Var_ProcessCpuLoad'],color=['red'])
plt.scatter(independent_var_test['Memory'],dependent_var_test['Var_Time'],color=['red'])

plt.scatter(independent_var_test['Memory'],dependent_var_test['Sum_CommittedVirtualMemorySize'],color=['red'])
plt.scatter(independent_var_test['Memory'],dependent_var_test['Sum_ProcessCpuTime'],color=['red'])
plt.scatter(independent_var_test['Memory'],dependent_var_test['Sum_FreePhysicalMemorySize'],color=['red'])
plt.scatter(independent_var_test['Memory'],dependent_var_test['Sum_TotalPhysicalMemorySize'],color=['red'])
plt.scatter(independent_var_test['Memory'],dependent_var_test['Sum_SystemCpuLoad'],color=['red'])
plt.scatter(independent_var_test['Memory'],dependent_var_test['Sum_Time'],color=['red'])

plt.scatter(independent_var_test['Memory'],dependent_var_test['loop_CommittedVirtualMemorySize'],color=['red'])
plt.scatter(independent_var_test['Memory'],dependent_var_test['loop_ProcessCpuTime'],color=['red'])
plt.scatter(independent_var_test['Memory'],dependent_var_test['loop_FreePhysicalMemorySize'],color=['red'])
plt.scatter(independent_var_test['Memory'],dependent_var_test['loop_TotalPhysicalMemorySize'],color=['red'])
plt.scatter(independent_var_test['Memory'],dependent_var_test['loop_SystemCpuLoad'],color=['red'])
plt.scatter(independent_var_test['Memory'],dependent_var_test['loop_ProcessCpuLoad'],color=['red'])
plt.scatter(independent_var_test['Memory'],dependent_var_test['loop_Time'],color=['red'])

plt.scatter(independent_var_test['Memory'],dependent_var_test['final_CommittedVirtualMemorySize'],color=['red'])
plt.scatter(independent_var_test['Memory'],dependent_var_test['final_ProcessCpuTime'],color=['red'])
plt.scatter(independent_var_test['Memory'],dependent_var_test['final_FreePhysicalMemorySize'],color=['red'])
plt.scatter(independent_var_test['Memory'],dependent_var_test['final_TotalPhysicalMemorySize'],color=['red'])
plt.scatter(independent_var_test['Memory'],dependent_var_test['final_Time'],color=['red'])
```

```
<matplotlib.collections.PathCollection at 0x7f32d6431cf8>
```

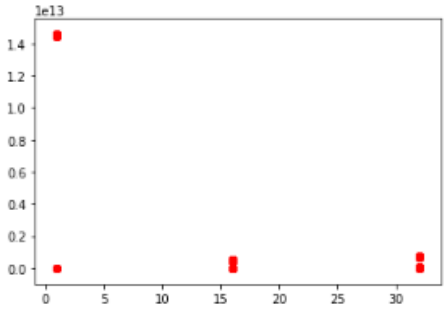
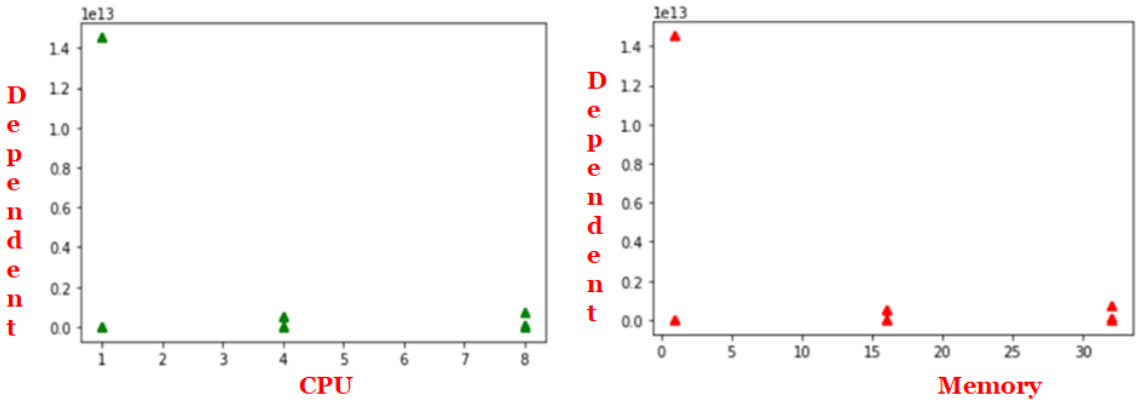


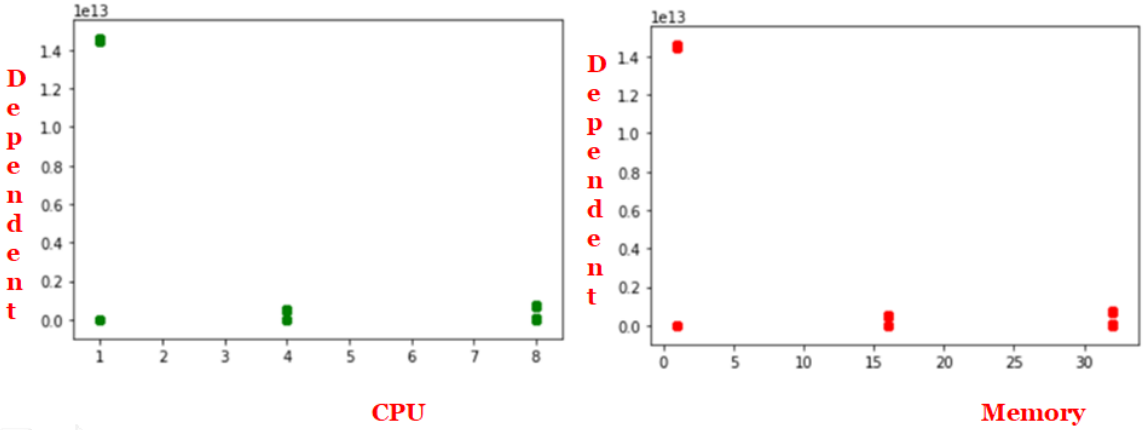
Figure 44 showns the graph which is been plotted between Memory and Dependent variables.

Graph Plot between Predicted vs. Actual

Predicted results



Actual results



Result Analysis:

As we notice, actual and predicted results are within the threshold; by this, we can say System is acting intended. In the graph plots, the score of the actual and prediction results are 99% accurate.

With this result analysis, if the predicted values by ML are not tallied, then we can predict the existence of anomaly in one or the other form. By this, developers can beneficial from the ML predictictions towards the post deployment failures ahead of time. Also, with known system configurations' like CPU and Memory of any machine's performance metrics can be achieved with this framework.

In addition, users are enabled to see if program execution consumes more than required resources in terms of CPU, memory, network bandwidth or disk IO requests.

Chapter V - Future work

This framework enables us to identify if any vulnerability exists in the system. Also enables users to identify program performance if the system key configurations are known. In addition, this frameworks helps enable users to see the required resources are been utilized by the program or not with the known configuration details such as CPU, memory, network bandwidth, etc. Though the identification of anomalies are been successful, still the specification of anomaly type and the solution towards the issue could not be achieved with this framework. The research work performed in this paper in order to resolve the issue is limited.

Further findings of the issue type and the possible solutions to resolve certain anomaly will still be continued. Future work of this research extension will first include identifying the type of anomaly in the system when the metrics prediction do not meet the threshold values. Secondly, work on possible solutions in order to address the issue.

Chapter VI - Conclusion

As software applications hold high demand, application security should be considered as a chief goal. With the help of this machine learning approach, application anomalies can be detected with minimal human involvement. Main goal of this framework is to identify anomalies in Software applications by calculating performance metrics with the systems key configurations. With the trained data to the ML, systems performance resources can be captured. After the demonstration of the project, the predicted results are achieved with an accuracy of about 99%. Systems performance metrics were successfully able to be achieved with the known resourcs like CPU, Memory, network bandwidth..

References

Aboud, S. (2009, June 1). *Protection of e-commerce Using Hybrid Tools*.

Application Security Vulnerability: Code Flaws, Insecure Code. (2014, February 2). Veracode. https://www.veracode.com/security/application-vulnerability

Artificial intelligence. (2019). In *Wikipedia*. https://en.wikipedia.org/w/index.php?title=Artificial_intelligence&oldid=891515233

Beal, V. (n.d.). *What endpoint security? Webopedia Definition.* Retrieved April 8, 2019, from https://www.webopedia.com/TERM/E/endpoint_security.html

Connecting to Your Linux Instance from Windows Using PuTTY - Amazon Elastic Compute Cloud. (n.d.). Retrieved April 21, 2020, from https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/putty.html

Cross-site scripting. (2019). In *Wikipedia*. https://en.wikipedia.org/w/index.php?title=Cross-site_scripting&oldid=889687114

CVE - Home. (2007). Retrieved April 9, 2019, from https://cve.mitre.org/cve/

Cybercrime – what it is and how to defend against it | Avast. (n.d.). Retrieved April 8, 2019, from https://www.avast.com/c-cybercrime?hsSkipCache=true&hs_ungate__cos_renderer_combine_all_css_disable=true

Data-flow analysis. (2019). In *Wikipedia*. https://en.wikipedia.org/w/index.php?title=Data-flow_analysis&oldid=884254867

Daymont, J. M. (2017). *Software vulnerabilities detection system and methods* (United States Patent No. US9715593B2). https://patents.google.com/patent/US9715593/en

Denial-of-service attack. (2019). In *Wikipedia*. https://en.wikipedia.org/w/index.php?title=Denial-of-service_attack&oldid=890075244

Deployment Phase in SDLC - Video & Lesson Transcript | Study.com. (n.d.). Retrieved April 1, 2019, from https://study.com/academy/lesson/deployment-phase-in-sdlc.html

Dor, N., Rodeh, M., & Sagiv, M. (n.d.). *Semantics, Program analysis.*

Fong, E., & Okun, V. (2007). Web Application Scanners: Definitions and Functions. *Proceedings of the 40th Annual Hawaii International Conference on System Sciences*, 280b–. https://doi.org/10.1109/HICSS.2007.611

Gupta, S., & Sharma, L. (2012). Exploitation of Cross-Site Scripting (XSS) Vulnerability on Real World Web Applications and its Defense. *International Journal of Computer Applications* (0975 – 8887), 60, 28–33. https://doi.org/10.5120/9762-3594

Hackett, B., Das, M., Wang, D., & Yang, Z. (2006). Modular checking for buffer overflows in the large. *Proceeding of the 28th International Conference on Software Engineering - ICSE '06*, 232. https://doi.org/10.1145/1134285.1134319

Input Validation. (n.d.). Retrieved April 8, 2019, from https://www.whitehatsec.com/glossary/content/input-validation

IP fragmentation. (2018). In *Wikipedia*. https://en.wikipedia.org/w/index.php?title=IP_fragmentation&oldid=874793315

Jimenez, W., Mammar, A., & Cavalli, A. (2010). *Software Vulnerabilities, Prevention and Detection Methods: A Review 1*.

Kronjee, J., Hommersom, A., & Vranken, H. (2018). *Discovering software vulnerabilities using data-flow analysis and machine learning*. 1–10. https://doi.org/10.1145/3230833.3230856

Krsul, I., Spafford, E., & Tripunitara, M. (1998). *An Analysis of Some Software Vulnerabilities.*

Letychevskyi, O. (2018). Algebraic methods for detection of vulnerabilities in software systems. *2018 IEEE 9th International Conference on Dependable Systems, Services and Technologies (DESSERT)*, 198–201. https://doi.org/10.1109/DESSERT.2018.8409127

Li, Z., Zou, D., Xu, S., Jin, H., Zhu, Y., & Chen, Z. (2018). SySeVR: *A Framework for Using Deep Learning to Detect Software Vulnerabilities.* 13.

Linear regression—Wikipedia. (n.d.). Retrieved April 21, 2020, from https://en.wikipedia.org/wiki/Linear_regression

Machine Learning | Coursera. (2019). Retrieved April 8, 2019, from https://www.coursera.org/learn/machine-learning

Machine Learning Tutorial Python—7: Training and Testing Data—YouTube. (2018). Retrieved April 21, 2020, from https://www.youtube.com/watch?v=fwY9Qv96DJY&t=10s

Machine Learning: What it is and why it matters. (n.d.). Retrieved April 8, 2019, from https://www.sas.com/en_us/insights/analytics/machine-learning.html

Medeiros, I., Neves, N. F., & Correia, M. (2014). Automatic Detection and Correction of Web Application Vulnerabilities Using Data Mining to Predict False Positives. *Proceedings of the 23rd International Conference on World Wide Web*, 63–74. https://doi.org/10.1145/2566486.2568024

Nethercote, N., & Seward, J. (2007). *Valgrind: A Framework for Heavyweight Dynamic Binary Instrumentation*. 12.

Programming Concepts: Static vs. Dynamic Type Checking | Aaron Krauss. (2015). Retrieved April 8, 2019, from https://thesocietea.org/2015/11/programming-concepts-static-vs-dynamic-type-checking/

Py/train_test_split.ipynb at master · codebasics/py · GitHub. (2018). Retrieved April 21, 2020, from https://github.com/codebasics/py/blob/master/ML/6_train_test_split/train_test_split.ipynb

Russell, R., Kim, L., Hamilton, L., Lazovich, T., Harer, J., Ozdemir, O., Ellingwood, P., & McConley, M. (2018). Automated Vulnerability Detection in Source Code Using Deep Representation Learning. *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA), 757–762.* https://doi.org/10.1109/ICMLA.2018.00120

Security pattern. (2019). In *Wikipedia*. https://en.wikipedia.org/w/index.php?title=Security_pattern&oldid=888650717

Shaneck, M. (2003). *An Overview of Buffer Overflow Vulnerabilities and Internet Worms. 8.*

Software vulnerabilities. (n.d.). Retrieved April 9, 2019, from https://encyclopedia.kaspersky.com/knowledge/software-vulnerabilities/

Tevis, J.-E. J., & Hamilton, J. A. (2004). Methods for the Prevention, Detection and Removal of Software Security Vulnerabilities. *Proceedings of the 42Nd Annual Southeast Regional Conference*, 197–202. https://doi.org/10.1145/986537.986583

tutorialspoint.com. (n.d.). *Operating System Security*. Www.Tutorialspoint.Com. Retrieved April 8, 2019, from https://www.tutorialspoint.com/operating_system/os_security.htm

Victor Krsul, I. (2011). Software Vulnerability Analysis. *ETD Collection for Purdue University.*

Weber, M. D., Shah, V. R., & Ren, C. (2008). *Systems and methods for detecting software security vulnerabilities* (United States Patent No. US7392545B1). https://patents.google.com/patent/US7392545/en

What is a Security Breach? - Definition from Techopedia. (2017). Techopedia.Com. Retrieved April 2, 2019, from https://www.techopedia.com/definition/29060/security-breach

What is Application Security? - Definition from Techopedia. (2019). Techopedia.Com. Retrieved April 1, 2019, from https://www.techopedia.com/definition/13567/application-security

What Is Network Security? (n.d.). Cisco. Retrieved April 8, 2019, from https://www.cisco.com/c/en/us/products/security/what-is-network-security.html

What is pen test (penetration testing)? - Definition from WhatIs.com. (2018). SearchSecurity. Retrieved April 9, 2019, from https://searchsecurity.techtarget.com/definition/penetration-testing

What is SQL Injection (SQLi) and How to Prevent It. (n.d.). Acunetix. Retrieved April 8, 2019, from https://www.acunetix.com/websitesecurity/sql-injection/

What Is Static Analysis (Static Code Analysis)? | Perforce. (2020). Retrieved April 8, 2019, from https://www.perforce.com/blog/qac/what-static-code-analysis

Wilander, J., & Kamkar, M. (2003). *A Comparison of Publicly Available Tools for Dynamic Buffer Overflow Prevention£.* 14.

Wu, F., Wang, J., Liu, J., & Wang, W. (2017). Vulnerability detection with deep learning. *2017 3rd IEEE International Conference on Computer and Communications (ICCC)*, 1298–1302. https://doi.org/10.1109/CompComm.2017.8322752

Appendix A

Below is the data of three VM instances (Ubuntu) 11 metrices 10 runs of each instance. All the data is been gathered for feeding as an input to the ML for training 70% and prediction 30% (random data).

CPU	Memory	Int_CommittedVirtualMemorySize	Int_ProcessCpuTime
1	1	2208010240	60000000
1	1	2208010240	60000000
1	1	2208010240	60000000
1	1	2208010240	60000000
1	1	2208010240	50000000
1	1	2208010240	60000000
1	1	2208010240	60000000
1	1	2208010240	50000000
1	1	2208010240	50000000
1	1	2208010240	50000000
4	16	6732087296	60000000
4	16	6732087296	60000000
4	16	6732087296	60000000
4	16	6732087296	60000000
4	16	6732087296	50000000
4	16	6732087296	60000000
4	16	6732087296	60000000
4	16	6732087296	60000000
4	16	6732087296	60000000
4	16	6732087296	60000000
8	32	11468390400	60000000
8	32	11468390400	60000000
8	32	11468390400	60000000
8	32	11468390400	60000000
8	32	11468390400	60000000
8	32	11468390400	60000000
8	32	11468390400	60000000
8	32	11468390400	60000000
8	32	11468390400	60000000
8	32	11468390400	60000000

Int_FreePhysicalMemorySize	Int_TotalPhysicalMemorySize	Int_SystemCpuLoad
143585280	1038839808	0.003497052
142438400	1038839808	0.003321757
142774272	1038839808	0.003302004
143343616	1038839808	0.003305015
145678336	1038839808	0.003308768
142438400	1038839808	0.003299414
142184448	1038839808	0.003302786
142397440	1038839808	0.003306789
143708160	1038839808	0.003310753
142295040	1038839808	0.003290851
15030235136	16825704448	0.012338837
15029694464	16825704448	0.01167598
15028113408	16825704448	0.011650133
15030075392	16825704448	0.011629428
15032500224	16825704448	0.011610898
15029145600	16825704448	0.011602982
15030718464	16825704448	0.011587859
15029895168	16825704448	0.011572471
15030919168	16825704448	0.011551093
15029383168	16825704448	0.011534571
33294262272	33737129984	0.001489881
33293635584	33737129984	0.001438163
33294704640	33737129984	0.001437085
33295187968	33737129984	0.00143661
33295400960	33737129984	0.001434442
33294458880	33737129984	0.001432256
33295233024	33737129984	0.001433931
33294299136	33737129984	0.001431869
33294565376	33737129984	0.001428878
33295867904	33737129984	0.001426366

Int_ProcessCpuLoad	Int_Time	Var_CommittedVirtualMemorySize	Var_ProcessCpuTime
0.00000399	1.352E+13	2208010240	60000000
3.7754E-06	1.435E+13	2208010240	60000000
0.000003748	1.447E+13	2208010240	60000000
3.7458E-06	1.448E+13	2208010240	60000000
3.7451E-06	1.448E+13	2208010240	60000000
3.7275E-06	1.456E+13	2208010240	60000000
3.7263E-06	1.456E+13	2208010240	60000000
3.1044E-06	1.457E+13	2208010240	60000000
0.00000372	1.457E+13	2208010240	60000000
0.00000308	1.47E+13	2208010240	60000000
7.8061E-06	3.73E+11	6732087296	60000000
7.3805E-06	4.838E+11	6732087296	60000000
7.3564E-06	4.905E+11	6732087296	60000000
7.3372E-06	4.958E+11	6732087296	60000000
7.3201E-06	5.006E+11	6732087296	60000000
7.3074E-06	5.042E+11	6732087296	60000000
7.2918E-06	5.086E+11	6732087296	60000000
7.2767E-06	5.128E+11	6732087296	70000000
0.000007258	5.181E+11	6732087296	60000000
7.2423E-06	5.226E+11	6732087296	60000000
1.09664E-05	7.529E+11	11468390400	70000000
8.9979E-06	7.173E+11	11468390400	60000000
8.9168E-06	7.097E+11	11468390400	60000000
1.03141E-05	7.025E+11	11468390400	70000000
8.7555E-06	6.942E+11	11468390400	60000000
8.6803E-06	6.868E+11	11468390400	60000000
8.6208E-06	6.808E+11	11468390400	60000000
0.00000854	6.726E+11	11468390400	70000000
8.4716E-06	6.655E+11	11468390400	60000000
8.3904E-06	6.569E+11	11468390400	60000000

Var_FreePhysicalMemorySize	Var_TotalPhysicalMemorySize	Var_SystemCpuLoad
143585280	1038839808	1
142438400	1038839808	0
142774272	1038839808	0
143343616	1038839808	1
145678336	1038839808	1
142438400	1038839808	0
142184448	1038839808	1
142397440	1038839808	1
143708160	1038839808	1
142295040	1038839808	1
15029981184	16825704448	0
15029239808	16825704448	1
15028113408	16825704448	0
15029821440	16825704448	0
15032246272	16825704448	0.5
15029145600	16825704448	0
15030718464	16825704448	0
15029641216	16825704448	1
15030919168	16825704448	0
15029129216	16825704448	0.5
33293881344	33737129984	0
33293635584	33737129984	0
33294704640	33737129984	0
33295187968	33737129984	0.25
33295400960	33737129984	0
33294458880	33737129984	0.5
33294852096	33737129984	0
33294299136	33737129984	0
33294565376	33737129984	0
33295867904	33737129984	0

Var_ProcessCpuLoad	Var_Time	Sum_CommittedVirtualMemorySize
0	1.3517E+13	2208010240
0	1.4354E+13	2208010240
0	1.447E+13	2208010240
0	1.448E+13	2208010240
0	1.4483E+13	2208010240
0	1.4559E+13	2208010240
0	1.4564E+13	2208010240
0	1.4568E+13	2208010240
0	1.4573E+13	2208010240
0	1.4698E+13	2208010240
0	3.7301E+11	6732087296
0	4.8383E+11	6732087296
0	4.9048E+11	6732087296
0	4.9583E+11	6732087296
0	5.006E+11	6732087296
0	5.0415E+11	6732087296
0	5.0856E+11	6732087296
0	5.1281E+11	6732087296
0	5.1814E+11	6732087296
0	5.2262E+11	6732087296
0	7.5292E+11	11468390400
0	7.1728E+11	11468390400
1	7.097E+11	11468390400
0	7.0245E+11	11468390400
0	6.942E+11	11468390400
0	6.8678E+11	11468390400
0	6.8082E+11	11468390400
1	6.7258E+11	11468390400
0	6.6549E+11	11468390400
0	6.5692E+11	11468390400

Sum_ProcessCpuTime	Sum_FreePhysicalMemorySize	Sum_TotalPhysicalMemorySize
60000000	143458304	1038839808
60000000	142438400	1038839808
60000000	142774272	1038839808
60000000	143216640	1038839808
60000000	145551360	1038839808
60000000	142311424	1038839808
60000000	142184448	1038839808
60000000	142270464	1038839808
60000000	143708160	1038839808
60000000	142295040	1038839808
60000000	15027757056	16825704448
60000000	15027142656	16825704448
60000000	15025905664	16825704448
60000000	15027724288	16825704448
70000000	15030022144	16825704448
60000000	15027048448	16825704448
60000000	15028621312	16825704448
70000000	15027400704	16825704448
70000000	15028695040	16825704448
60000000	15026905088	16825704448
70000000	33291800576	33737129984
60000000	33291411456	33737129984
70000000	33292607488	33737129984
70000000	33293090816	33737129984
70000000	33293303808	33737129984
60000000	33292234752	33737129984
60000000	33292754944	33737129984
70000000	33292042240	33737129984
60000000	33292341248	33737129984
60000000	33293643776	33737129984

Sum_SystemCpuLoad	Sum_Time	loop_CommittedVirtualMemorySize
0	1.3517E+13	2208010240
1	1.4354E+13	2208010240
0	1.447E+13	2208010240
0	1.448E+13	2208010240
0	1.4483E+13	2208010240
1	1.4559E+13	2208010240
0	1.4564E+13	2208010240
0	1.4568E+13	2208010240
0	1.4573E+13	2208010240
0	1.4698E+13	2208010240
0	3.7301E+11	6732087296
0	4.8383E+11	6732087296
0	4.9048E+11	6732087296
0	4.9583E+11	6732087296
0	5.0061E+11	6732087296
0	5.0415E+11	6732087296
0	5.0856E+11	6732087296
0	5.1281E+11	6732087296
0	5.1814E+11	6732087296
0	5.2262E+11	6732087296
0	7.5292E+11	11468390400
0	7.1728E+11	11468390400
0	7.0969E+11	11468390400
0	7.0245E+11	11468390400
0	6.942E+11	11468390400
0	6.8678E+11	11468390400
0	6.8082E+11	11468390400
0	6.7258E+11	11468390400
0	6.6549E+11	11468390400
0	6.5692E+11	11468390400

loop_ProcessCpuTime	loop_FreePhysicalMemorySize	loop_TotalPhysicalMemorySize
60000000	143458304	1038839808
70000000	142438400	1038839808
60000000	142774272	1038839808
70000000	143216640	1038839808
60000000	145551360	1038839808
60000000	142311424	1038839808
70000000	142184448	1038839808
60000000	142270464	1038839808
60000000	143708160	1038839808
70000000	142295040	1038839808
70000000	15027757056	16825704448
60000000	15027142656	16825704448
60000000	15025905664	16825704448
60000000	15027724288	16825704448
70000000	15030022144	16825704448
70000000	15027048448	16825704448
60000000	15028621312	16825704448
70000000	15027400704	16825704448
70000000	15028695040	16825704448
60000000	15026905088	16825704448
70000000	33291800576	33737129984
70000000	33291411456	33737129984
70000000	33292607488	33737129984
70000000	33293090816	33737129984
70000000	33293303808	33737129984
70000000	33292234752	33737129984
60000000	33292754944	33737129984
70000000	33292042240	33737129984
70000000	33292341248	33737129984
60000000	33293643776	33737129984

loop_SystemCpuLoad	loop_ProcessCpuLoad	loop_Time
0	0	1.3517E+13
0	0	1.43541E+13
1	0	1.44704E+13
0	0	1.448E+13
0	0	1.4483E+13
0	0	1.45588E+13
0	0	1.45636E+13
0	0	1.45683E+13
0	0	1.45731E+13
0	0	1.4698E+13
0	1	3.73012E+11
0	0	4.83828E+11
0.5	0	4.90482E+11
0	0	4.95835E+11
0	0	5.00606E+11
0	0	5.04156E+11
1	0	5.08562E+11
0	0	5.12812E+11
1	0	5.18146E+11
0	0	5.22624E+11
0	0	7.52923E+11
0	0	7.17278E+11
0	0	7.09693E+11
1	0	7.02452E+11
0	0	6.94199E+11
0	0.5	6.86782E+11
0	0	6.80815E+11
0	0	6.72578E+11
0.5	0.5	6.65488E+11
0	0	6.56918E+11

final_CommittedVirtualMemorySize	final_ProcessCpuTime	final_FreePhysicalMemorySize
2208010240	60000000	143458304
2208010240	70000000	142438400
2208010240	60000000	142774272
2208010240	70000000	143216640
2208010240	60000000	145551360
2208010240	70000000	142311424
2208010240	70000000	142184448
2208010240	60000000	142270464
2208010240	60000000	143708160
2208010240	70000000	142295040
6732087296	70000000	15027757056
6732087296	60000000	15027142656
6732087296	60000000	15025905664
6732087296	70000000	15027724288
6732087296	70000000	15030022144
6732087296	70000000	15027048448
6732087296	60000000	15028621312
6732087296	70000000	15027400704
6732087296	70000000	15028695040
6732087296	60000000	15026905088
11468390400	70000000	33291800576
11468390400	70000000	33291411456
11468390400	70000000	33292607488
11468390400	70000000	33293090816
11468390400	70000000	33293303808
11468390400	70000000	33292234752
11468390400	70000000	33292754944
11468390400	70000000	33292042240
11468390400	70000000	33292341248
11468390400	60000000	33293643776

final_TotalPhysicalMemorySize	final_Time
1038839808	1.3517E+13
1038839808	1.4354E+13
1038839808	1.447E+13
1038839808	1.448E+13
1038839808	1.4483E+13
1038839808	1.4559E+13
1038839808	1.4564E+13
1038839808	1.4568E+13
1038839808	1.4573E+13
1038839808	1.4698E+13
16825704448	3.7301E+11
16825704448	4.8383E+11
16825704448	4.9048E+11
16825704448	4.9584E+11
16825704448	5.0061E+11
16825704448	5.0416E+11
16825704448	5.0856E+11
16825704448	5.1281E+11
16825704448	5.1815E+11
16825704448	5.2262E+11
33737129984	7.5292E+11
33737129984	7.1728E+11
33737129984	7.0969E+11
33737129984	7.0245E+11
33737129984	6.942E+11
33737129984	6.8678E+11
33737129984	6.8081E+11
33737129984	6.7258E+11
33737129984	6.6549E+11
33737129984	6.5692E+11