



cGraph: Graph Based Extensible Cyber Threat Intelligence Platform

W. W. Daluwatta
Index No : 16000226

L. R. S. De Silva
Index No : 16000252

S. N. Kariyawasam
Index No : 16000684

Supervisor: Dr. Kasun de Zoysa

March 2021

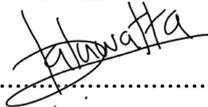
Submitted in partial fulfillment of the requirements of the
B.Sc(Hons) in Software Engineering 4th Year Project (SCS4123)



Declaration

I certify that this dissertation does not incorporate, without acknowledgment, any material previously submitted for a degree or diploma in any university and to the best of my knowledge and belief, it does not contain any material previously published or written by another person or myself except where due reference is made in the text. I also hereby give consent for my dissertation, if accepted, be made available for photocopying and for interlibrary loans, and for the title and abstract to be made available to outside organizations.

Candidate Name: W. W. Daluwatta


.....

Signature of Candidate

Date: 26.03.2021

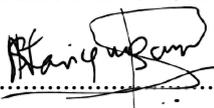
Candidate Name: L. R. S. De Silva


.....

Signature of Candidate

Date: 26.03.2021

Candidate Name: S. N. Kariyawasam


.....

Signature of Candidate

Date: 26.03.2021

This is to certify that this dissertation is based on the work of Mr. W. W. Daluwatta, Mr. L. R. S. De Silva and Ms. S. N. Kariyawasam under my supervision. The dissertation has been prepared according to the format stipulated and is of the acceptable standard.

Supervisor's Name: Dr. Kasun de Zoysa

.....

Signature of Supervisor

Date:

Abstract

Cyber security, also referred to as computer security, is the field of study where computer systems and networks are protected from malicious digital attacks such as spam attacks, phishing attacks, malware attacks, and ransomware attacks from perpetrators. The number of cyber threats and malicious attacks continue to rise each year globally at a rapid pace and thus there is a need for implementing proper and effective cyber security measures and recognizing possible attacks or malicious internet resources early on.

To address this issue, we propose "cGraph: Graph Based Extensible Cyber Threat Intelligence Platform" which is a scalable big data processing and storing system that uses intelligence derived from state-of-the-art graph inference algorithms utilizing vast amounts of passive DNS network traces and limited amount of external threat intelligence.

Acknowledgement

The authors of the thesis would like to take this opportunity to express our gratitude to the people who have lent their support throughout the project.

We would like to offer our gratitude for Dr. Kasun de Zoysa for supporting us throughout the project by giving advice and guidance to make this project a success.

Our special thanks are extended to Dr. Mohamed Nabeel from the Qatar Computing Research Institute (QCRI) for giving the idea for this research project and for providing us with valuable knowledge related to the research area.

Contents

Declaration	i
Abstract	ii
Acknowledgement	iii
Contents	vii
List of Figures	xi
List of Tables	xii
List of Acronyms	xiii
1 Introduction	1
1.1 Motivation	1
1.2 Goal	2
1.3 Scope of the Project	2
1.4 Justification as product-based project	3
2 Background Study	4
2.1 Literature study of the algorithm	4
2.1.1 What is maliciousness?	5
2.1.2 The malicious domain detection system	5
2.1.3 Graph inference	6
2.1.4 Knowledge graph	6
2.1.5 Gathering data on known malicious resources	7
2.1.6 Belief propagation	8

2.1.7	Formal definition about belief propagation	8
2.1.8	Inference-based classification	10
2.2	Current software engineering solutions	12
2.3	Competitive Analysis	13
2.3.1	AT&T Cybersecurity	13
2.3.2	Anomali Threat Intelligence Platform	14
2.3.3	Cisco Umbrella Investigate	14
2.3.4	VirusTotal Graph	15
2.4	Significance of the proposed solution	15
3	Functional and Non-functional Requirements	17
3.1	Functional Requirements	17
3.1.1	End user’s functional requirements	17
3.1.2	User Stories	18
3.2	Quality Attribute Scenarios	18
3.2.1	Availability Scenarios	18
3.2.2	Modifiability Scenarios	19
3.2.3	Performance Scenarios	19
3.2.4	Scalability scenarios	23
4	Analysis and Design	24
4.1	Sources of data	24
4.1.1	Passive DNS data from Farsight Security	25
4.1.2	Intelligence data from VirusTotal	29
4.1.3	Geolocation Data from MaxMind	32
4.1.4	Data from Alexa Rank	32
4.2	System Design	33
4.2.1	Data preprocessing	33
4.2.2	Data storing	34
4.2.3	System architecture design	40
4.2.4	Design of the inference system	43
4.2.5	Technology stack	46
4.3	Engineering Challenges and Solutions	46

4.3.1	Profiling VT records	47
4.3.2	Modeling PDNS data to graph	54
4.3.3	Bootstrapping and daily insertions (Populating databases)	56
4.3.4	MaxMind Data Storage	56
4.3.5	Alexa Data Storage	56
4.3.6	Designing a scalable architecture	57
5	Implementation	59
5.1	Implementation of the infrastructure and database clusters	59
5.1.1	Implementing the Kubernetes cluster	59
5.1.2	Implementing the ArangoDB cluster	60
5.1.3	Implementing the GreenplumDB cluster	60
5.2	Data processing pipelines using Spark	60
5.2.1	Spark jobs for processing PDNS data	60
5.2.2	Spark jobs for processing VT data	64
5.2.3	Spark jobs for processing Alexa Rank data	64
5.2.4	Bootstrapping pipelines and daily pipelines	65
5.3	Middleware	65
5.3.1	IAM Microservice	66
5.3.2	IAM Authentication	68
5.3.3	Search microservice	68
5.3.4	Graph microservice	69
5.3.5	Inference microservice	72
5.3.6	Detail microservice	75
5.3.7	Monitoring and matrices	75
5.3.8	Deployment strategies	76
5.4	End consumer products	78
5.4.1	Web Application	78
5.4.2	Chrome Extension	83
5.4.3	Developer API	84
6	Results and Analysis	86
6.1	Evaluation of the inference process	86

6.1.1	Ground truth creation for the evaluation process	86
6.1.2	ROC and AUC	87
6.1.3	Evaluation of the inference process against different malicious- ness levels indicated by VirusTotal	89
6.2	Demonstration	89
7	Evaluation and Testing	95
7.1	Spark data processing pipelines	95
7.2	Database Clusters	96
7.2.1	Evaluation of data insertion in ArangoDB	96
7.2.2	Evaluation of the indexing performance of the Greenplum DB cluster	97
7.3	Middleware	99
7.3.1	Performance	99
7.3.2	Availability	100
7.4	Web application	101
7.4.1	Front-end testing and end-to-end evaluation	101
7.5	Cost evaluation	102
8	Conclusion	104
8.1	Limitations	105
8.2	Future Work	105
	References	108
A	Contribution	112
B	SQL Schema for Alexa data	115
C	Product Documentation	117
D	System View	118
E	Container images	122
F	External References	123

List of Figures

2.1	Computing the maliciousness score of a node by inference	9
4.1	An instance of a heterogeneous knowledge graph (Apex as start node) .	29
4.2	Scalability comparison	34
4.3	Any Direction	35
4.4	Inbound	35
4.5	Outbound	35
4.6	Execution time vs depth under 10 000 node limit	36
4.7	High level diagram of infrastructure orchestration with Kubernetes . . .	37
4.8	Namespaces inside Kubernetes	37
4.9	Ant Performance of Force Layouts AntD vs D3	38
4.10	Graphin: Performance of Layouts	39
4.11	Force Layout	39
4.12	Random Layout	39
4.13	Infrastructure point of view	41
4.14	Data point of view	42
4.15	Concepts and Relationships point of view	43
4.16	Subgraph with a depth of 2	45
4.17	Labeling the depth two subgraph before Inference	46
4.18	VT profiling	47
4.19	ER diagram for the Normalized design	49
4.20	Sharding using APEX-ID (Hash distribution)	51
4.21	Sharding using FQDN-ID (Hash distribution)	52
4.22	Sharding using FQDN-ID (Random distribution)	53
4.23	Timestamping graph edges	55
4.24	Graph sharding	55

4.25	MaxMind's subnet distribution	57
4.26	Resource intensive architecture design	58
5.1	Spark data pipeline with 3 phases	61
5.2	Converting Type A record to graph nodes and edges	62
5.3	Converting Type NS, MX, SOA, and CNAME records to graph nodes and edges	64
5.4	Spark jobs for processing VT data	64
5.5	Alexa cleansing and transformation process	65
5.6	Microservices Architecture	66
5.7	Identity and access management endpoints	66
5.8	Flowchart for Authentication	67
5.9	Search endpoints	68
5.10	Component diagram for search	69
5.11	Graph endpoints	69
5.12	/graph endpoints	70
5.13	Component diagram of graph	71
5.14	/timeline endpoints	71
5.15	Component diagram of timeline	72
5.16	Belief propagation process	72
5.17	Programming languages in implementaion	73
5.18	Inference endpoints	73
5.19	Inference microservice and caching layer	73
5.20	pluggable architecture	74
5.21	Details endpoints	75
5.22	Spark Monitoring	76
5.23	Kubernetes Monitoring	76
5.24	API monitoring	77
5.25	Github Actions Page	77
5.26	Struggle with large braph	81
5.27	Benign indication	84
5.28	Malicious indication	84
5.29	Token Manager (Public API key generator)	84

5.30	Public API endpoints	85
6.1	ROC curve	88
6.2	Confusion Matrix	88
6.3	Evaluation of the inference process against different maliciousness levels indicated by VirusTotal	89
6.4	Search results for the keyword "Instagram"	90
6.5	Advanced filtering of search results	90
6.6	Catching malicious acts from search results	91
6.7	Sorted search results	91
6.8	Visiting the graph view	92
6.9	View historical DNS information	92
6.10	View historical DNS information (Summarized View)	93
6.11	Initial graph view	93
6.12	Inference graph	94
7.1	Evaluation of the Write speed of the ArangoDB graph database cluster	96
7.2	Partitions Vs Number of Records	97
7.3	Partitions Vs Index Creation Time	98
7.4	Cumulative measure of index creation time in VirusTotal profile database	99
7.5	Increasing the Number of users with time	100
7.6	Increasing the Number of requests with time	101
7.7	Average response time	101
7.8	Cypress dashboard (Summary)	102
7.9	Cypress dashboard (Test case)	102
7.10	Initial cost	103
7.11	Incremental cost	103
C.1	Documentation - MkDocs	117
D.1	SignIn	118
D.2	SignUp	118
D.3	Forgot Password	119
D.4	Search View	119

D.5	Search Results	120
D.6	Search Results	121
E.1	Container images - DcokerHub#1	122
E.2	Container images - DcokerHub#2	122

List of Tables

3.1	Availability Scenario 1	19
3.2	Modifiability Scenario 1	19
3.3	Performance Scenario 1	20
3.4	Performance Scenario 2	20
3.5	Performance Scenario 3	20
3.6	Performance Scenario 4	21
3.7	Performance Scenario 5	21
3.8	Performance Scenario 6	22
3.9	Performance Scenario 7	22
3.10	Performance Scenario 8	22
3.11	Scalability Scenario 1	23
4.1	Structure and Volume of data feeds (daily)	25
4.2	Overview of Farsight data	26
4.3	Statistics of PDNS data	28
4.4	Analysis of volume and number of records of VT data feed	30
4.5	Top Apex Domains with Most FQDNs	30
4.6	Infrastructure Components	41
4.7	Overview of the Technology Stack	46
6.1	TP rate and FP rate against different threshold values on inference results	87
6.2	Evaluation matrix for the inference process	88
7.1	Measuring the performance of the Spark big data pipeline	96
7.2	Performance evaluation using JMeter	100

List of Acronyms

AUC	Area under the ROC Curve
CDN	Content delivery network
CM	Confusion Matrix
CNAME	Canonical Name Record
FQDN	Fully Qualified Domain Names
IP	Internet Protocol
MX	Mail Server
NS	Name Server
PDNS	Passive Domain Name System
QCRI	Qatar Computing Research Institute
RDD	Resilient Distributed Datasets
ROC	Receiver operating characteristic
SOA	Start of Authority Record
VM	Virtual Machine
VT	VirusTotal
SPA	Single-Page Application
MPA	Multiple-Page Application

Chapter 1

Introduction

cGraph: Graph Based Extensible Cyber Threat Intelligence Platform is a web based platform that allows any party including cyber-security researchers, security operations teams and general public to visualize and analyze Internet resources such as IP addresses, domain names, TLS certificates and registration records related to cyber attacks including spam attacks, phishing attacks, malware attacks, ransomware attacks etc, using the intelligence derived from state-of-the-art graph inference algorithms utilizing vast amounts of passive DNS network traces and a limited amount of external threat intelligence.

This chapter gives an introduction to the study under the following sections. Section 1.1 describes the motivation for undertaking this project and the goal and scope of the project is given in sections 1.2 and 1.3 respectively. The project is justified to be a product-based software engineering project in section 1.4.

1.1 Motivation

Internet domains are the launch pad for many cyber attacks we observe nowadays. One effective way to reduce the damage caused by such attacks is to identify the domains involved early and take actions such as blocking, taking down or sinkholing. Further, these malicious domains have underlying associations with other Internet resources that help to uncover malicious infrastructures. We observe that existing cyber threat intelligence systems fail to take advantage of such associations to detect stealthy malicious domains nor do they capture such associations in a user-friendly graphical form that allows users

to easily explore related Internet resources. Motivated by these gaps in the industry, we propose to build a proactive cyber threat intelligence platform that can uncover malicious domains early and visualize associated Internet resources in a graph format for easy analysis.

In this project, we are adopting predictive threat intelligence algorithms and models published in prior academic research done by Qatar Computing Research Institute (QCRI) [1]. Key goals in this project is to engineer these algorithms to work in a big data environment and make them available for the public to use.

Prior academic research has been conducted on controlled environments with cleaned input data, limited amount of data, no real-time considerations, and short study periods. There is a considerable gap between software engineering view and research view. Throughout this project we work to fill that gap and adopt these algorithms in a real time system.

We build a heterogeneous knowledge graph of the Internet (derived from passive DNS data) for each time epoch. Due to the large volume and the dynamic nature of that data, we observe quite aggressive volatile behaviour on nodes and edges of the graph. Despite the above challenges, in accordance with the engineering goals we have laid out we build a system that can handle very large dynamic graphs and run threat intelligence algorithms. We utilize open source tools and software engineering techniques to overcome these problems and meet the functional and non-functional requirements while maintaining the correctness of published research algorithms.

1.2 Goal

Implement a scalable, production ready cyber threat intelligence web platform which connects research findings with software engineering aspects.

1.3 Scope of the Project

Threat intelligence data will be gathered from multiple external sources. Since this aggregated data will add up to 10 petabytes, a scalable and dynamic backend needs to be developed to model and manipulate this big data environment. This will be followed by

a Middleware that will give the ability to query the persistent storage (database clusters) within seconds according to the user's needs.

Additionally, an interactive web-based front-end will be built in order to facilitate Text-based as well as Graph-based search of domains for end users.

These components will be implemented while taking the necessary functional and non-functional requirements into consideration and a fully working web platform will be released within the time period of a year.

1.4 Justification as product-based project

Our project involves the application of software engineering techniques to develop a domain threat intelligence big data platform incorporating the latest results from research publications with a plugin-based architecture that addresses real-world gaps in the cybersecurity community. These SE techniques will include analyzing the product idea, writing the specification, design, and development, thus making our project a product-based project.

According to the Software Engineering White Paper published by Networked European Software and Services Initiative [2], despite the abundance of storage at relatively low cost, the storage and querying of data at a large scale continue to remain challenging. With our project we will be looking into technologies out there that can (partially) solve these problems, how to use them and improve them to do better, and look into previous research efforts that we can implement to address some of the weaknesses in the existing systems and libraries, thus making our product innovative.

Considering these factors, the development of cGraph: Graph Based Extensible Cyber Threat Intelligence Platform is justified as a Product-based Software Engineering project.

Chapter 2

Background Study

This chapter discusses current methods and techniques of large scale graph data manipulation mentioned in literature and existing competitive systems. A literature study of the inference algorithm is given in section 2.1. Section 2.2 describes current software engineering solutions and a competitive analysis of the existing systems is given in section 2.3. The significance and competitive advantage of the proposed solution over other systems is explored in section 2.4.

2.1 Literature study of the algorithm

The Internet is a massive infrastructure which provides various services and facilitates consumer engagements. Malicious attacks against internet users have become a daily occurrence and thus, early prediction of malicious resources has become a tedious and complicated task. For an example, the amount of daily scans happening through Virus-Total (VT) is approximately 5-7 million. But the amount of Passive DNS (PDNS) traffic that Farsight, a state of the art PDNS service, captures is more than 600 million. Thus, it is clear that the amount of unseen internet resources is much larger than the seen ones. Numerous researches have been conducted to predict and classify malicious resources and security researchers mainly focus on two paradigms to carry out this task which has been explored by Khalil et al in their work in “Killing Two Birds with One Stone: Malicious Domain Detection with High Accuracy and Coverage” [1]. They have identified two approaches to classification and prediction systems; classification-based and inference-based.

The main focus of our system was to build a big data graph infrastructure to adopt the research outcomes of Kahlil et al research work and utilize graph inference-based techniques for prediction of malicious resources.

2.1.1 What is maliciousness?

In terms of Internet security, maliciousness can be said to be the trait of harming Internet users, such as if a person or organization tries to establish an internet accessible resource (website, file sharing services, email etc.) in order to compromise access of a victim (a user or another organization). There are a number of malicious attacks that can be carried out such as DDOS attacks, phishing attacks, malware attacks, botnet attacks, ransomware attacks etc. The root of these attacks can be an Internet resource such as IP, name server, mail server etc. and if the resource we are considering can be used to cause harm to an internet user, they can be classified as malicious.

Maliciousness of a website measures the likelihood of the website being malicious. The higher the maliciousness of a website, the more damage it may cause to the visitors of the website. A malicious website may be either a malware website or a phishing website. A malware website contains codes that install malicious software into visitors' computers whereas a phishing website pretends to be a legitimate website so that attackers can steal account credentials, cause financial losses, and/or steal private information.

2.1.2 The malicious domain detection system

The entire Internet is bound by two commonly used versions of the Internet Protocol (IP): the 32 bit long IPv4 addressing and the 128 bit long IPv6 addressing. From these, IPv4 is more mature compared to IPv6 since it has been around for a long time. Thus, the current impact of IPv4 is high and all domains, name servers, mail servers use this version. Currently, around 4 billion IPv4 addresses are in use and out of this only 3 billion addresses are available for public use; the rest is reserved for private network use [3]. Timely monitoring of changes in this massive infrastructure is the key to uncovering malicious resources.

Khalil et al in their work has identified two main approaches to uncover malicious resources; classification-based and inference-based.

Classification-based

These models are built by using classical statistical and machine learning models such as random forest, support vector machines, and deep neural networks. The key ingredient behind this model is to have a well-defined ground truth and adversarial proof feature set. Defining an adversarial proof feature set and gathering a well-defined ground truth [4] are two key issues with this approach. Other challenges are determining the re-train period, establishing well-defined ground truth for each new training cycle and building the model in a way that it is robust to noise in data.

Inference-based

Inference-based algorithms mainly run message propagation on top of graphs and uncover knowledge from the knowledge that is already available, which is different from the ground truth based supervised learning or deep learning model. It uses label propagation algorithms to propagate the labels from known nodes to unknown nodes in the graph and uncover the state. An example inference algorithm is belief propagation [5] which is a well-known label propagation algorithm.

2.1.3 Graph inference

According to Khalil et al's work, building up an inference-based classification system is a threefold approach.

1. Constructing the knowledge graph with the use of co-relation rules.
2. Labeling the graph with known malicious and benign resources which will be used as the initial belief or messages for the message passing algorithms.
3. Running proven message passing algorithms such as Belief Propagation on top of the graph and tuning the set of hyperparameters to the relevant context.

2.1.4 Knowledge graph

A graph is a data structure comprising of two basic elements; nodes and edges. Nodes are the representatives of the entities of a graph ~~build~~ while edges symbolize relationships between those entities. Edges can be directed or undirected. Graphs with directed edges

are referred to as directed graphs and graphs with undirected edges are referred to as undirected graphs. Furthermore, nodes and edges in a graph may have types and weights. A graph with a single type of node is referred to as a homogeneous graph and a graph with multiple types of nodes is referred to as a heterogeneous graph.

Our graph is built using DNS data which consists of different types of nodes such as IPs, Apex, FQDNs, SOA, CNAME, Name Servers, and Mail Servers and its edges are bidirected. Thus, the knowledge graph can be referred to as a heterogenous bidirectional graph. Since DNS is highly dynamic, its data is extracted daily from the DNS feed, and the graph is constructed daily. A set of day granular graphs were used to run the inference algorithm for extracted windows and extracted sub graphs specifically needed by the user.

2.1.5 Gathering data on known malicious resources

Initial belief or initial information is a key fact in inference. In order to label the nodes in the knowledge graph, it is necessary to understand the information ecosystems, what the credibility of the source is, how far one can use that information to label the knowledge graph, and what improvements that one can do to enhance the credibility of the information.

For this purpose, we can use domain or IP blacklists as an indicator to label the knowledge graph as well as the state of the art reputation systems like VirusTotal, Alexa Rank and Google Safe Browsing. However, all these sources have both advantages and drawbacks.

When considering the domain or IP blacklists, they contain identities of prior known offenders. These are usually used to aid more information to defense such as indicators of compromise signals etc. But blacklists can be a victim of adversities and lead to misinformation propagation [6].

Likewise, the state of the art reputation systems such as VirusTotal also have some limitations. If one anti-virus engine detects a scanned domain or IP to be unsafe, it does not indicate that resource is compromised or malicious. In such a case using such resources require advanced as well as validated heuristics such as marking a scanned domain as bad if two more anti-virus engines have detected it as bad.

Adding non-malicious information also necessary to run the inference algorithm. Be-

nign resources have been gathered from Alexa top rank domains, High Alexa ranks are the highly reputed legit domains. The top rank is always benign always [4]s.

2.1.6 Belief propagation

In 1982, Judea Pearl [7] proposed the Belief Propagation algorithm as an inference algorithm for trees. Since then, it was proven to work on poly-trees and later on the general graph as well [8]. Belief propagation is an accurate and efficient algorithm for solving the message passing problems on graph data structures and has been found to work well with graphical models like Bayesian networks [9] and Markov random fields [10].

Belief propagation is an iterative algorithm that passes the belief iteratively. In every iteration t , belief propagation algorithm updates the belief value on the node and passes the beliefs (messages) to its neighbors based on the belief values it received from the previous iteration $t-1$. As mentioned, this interactive process happens until each message value passed by a node to a neighbor node converges to a stable number. The final beliefs can be extracted after all the iterations end and the converged belief value indicates the belief value of that node.

The main idea in the research work of Khalil et al is to define a knowledge graph and inject labels to the knowledge graph from credible sources and run the belief propagation algorithm on top of the knowledge graph by using the labeled values as initial beliefs. It will be able to uncover the probabilistic value of indicating the maliciousness of uncovered nodes in the knowledge graph. The key idea of belief propagation is that for a given node, if it has more benign nodes associated with it, the likelihood of being benign is high whereas if it has more malicious nodes associated with it, the likelihood of being malicious is high. Likewise, a probabilistic answer to the malicious score of a given node will be computed by belief propagation [figure:2.1].

The structure of the graph that is used to run the belief propagation can be of two types and the inference depends on the type of graph.

2.1.7 Formal definition about belief propagation

Consider an undirected graph $G = (V, E)$, where V represent nodes and E represent edges. Each node $v_i \in V$ contains a discrete random variable X_i where $i \in \{1, n\}$ and $n = \|V\|$ that holds the finite state s_j where $s_j \in S$ and $j \in \{1, m\}$. In our scenario, S

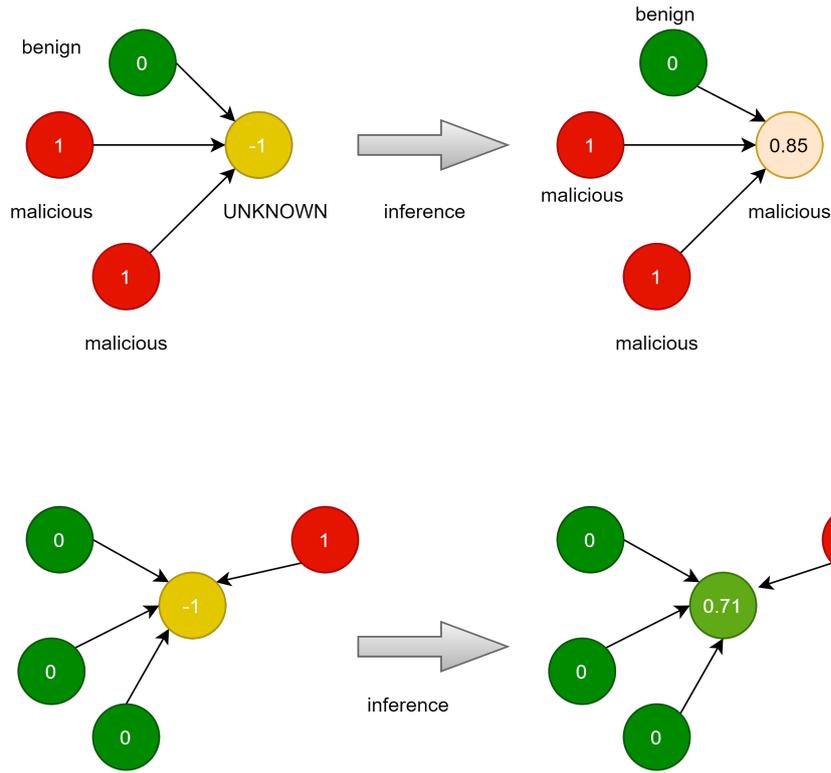


Figure 2.1: Computing the maliciousness score of a node by inference

can take three states: malicious, benign or unknown. This structural model defines the probability distribution $p(X_1, X_2, \dots, X_n)$ over the nodes in G . This is called "belief", which simply means the marginal probability being in each possible state. The exact inference algorithm computes the marginal probability distribution $p(X_i)$ of each discrete random variable X_i given a set of observed/labeled nodes.

$$p(X_j) = \sum_{X_i, i \in \{1, n\} \setminus j} p(X_1, X_2, \dots, X_n) \quad (2.1)$$

The marginal probability of the node can be computed according to the above formula by taking the sum over all possible states of all the other nodes in the graph.

The time complexity of the above marginal probability calculation is exponential to the number of nodes in the graph. The worst time complexity is $O(V^2)$. Belief propagation algorithm is an approximation algorithm to find marginal probabilities and it reduces the complexity to $O(EV)$.

The labels are the prior probability of being in each possible state and in the work of Khalil et al they have been used as the ground truth which is denoted numerically to represent the prior probability of each node. For example, a node being malicious is 1,

being benign is 0 and being unknown is -1. Formally, we use $\phi_i(s_j)$ to denote the prior probability of node i being in state s_j . By using the prior probability value $\phi_i(s_j)$, the belief propagation algorithm calculates the belief $b_i(s_j)$ for each node i by computing the marginal probability for each possible state s_j where $j \in \{1, m\}$.

The edge potential function $\psi_{i,j}(s_i, s_j)$ defines the statistical properties between any two connected nodes i and j . Here i and j are two different connected nodes, and s_i and s_j are the states of the node. In our context each node can be benign, malicious or unknown. So the relevant $\psi_{i,j}(s_i, s_j)$ edge potential function can be defined by a 3 by 3 matrix that each represents a possible state of the two edge nodes.

In each iteration, the belief propagation algorithm propagates the message vector m_{ij} of size $\|s\|$ to each neighbor j . Formally, node i passing the message to node j on state s_r is denoted as $m_{ij}(s_r)$.

$$m_{ij}^t(s_r) = \sum_{s_p \in \mathcal{S}} \left(\phi_i(s_p) \psi_{ij}(s_p, s_r) \prod_{k \in N_i/j} m_{ki}^{t-1}(s_p) \right) \quad (2.2)$$

For a given iteration at time t , the message value being updated on its previous state is $m_{ij}^t(s_r)$ (as shown in the equation).

The implementation of the algorithm can be defined to run a constant number of iterations or until the message value of each node converges (the difference between messages from two consecutive rounds for all edges is below a predefined small threshold value).

After all iterations have completed, the final belief value of node i can be defined as,

$$b_i(s_r) = C \phi_i(s_r) \prod_{k \in N_i/j} m_{ki}(s_p) \quad (2.3)$$

Where C is a constant that is used to normalize the belief values.

$$\sum_{s_p \in \mathcal{S}} b_i(s_p) = 1 \quad (2.4)$$

2.1.8 Inference-based classification

From the work of Khalil et al, our focus was on inference-based classification and the algorithmic flow is described in detail below.

Construction of Domain Graph

There are two types of nodes in the graph (domains and IPs) and information had been extracted using Passive DNS. By using the extracted A records, IP addresses have been divided into two groups; public IP addresses and dedicated IP addresses. Public IP addresses are the IPs belonging to public cloud vendors such as Google Cloud, AWS, and Azure Cloud or CDN networks such as Cloudflare, Akamai, and Fastly or web hosting services such as 00host.com. Dedicated IP addresses are the IPs belonging to particular organizations such as universities, and governments.

The construction of a heterogeneous undirected domain-IP graph called a baseline graph (G-Baseline) has been done by taking IPs and domains hosted on those IPs.

Ground Truth Collection

The nodes of the above constructed graph has been labelled using two ground truths; malicious ground truth and benign ground truth.

McAfee SiteAdvisor has been used to gather the malicious ground truth. For the malicious ground truth they have used McAfee SiteAdvisor [11]. McAfee SiteAdvisor returns four labels for a given domain; Safe, Caution, Warning and Unknown. “Safe” indicates a benign domain, “Caution” indicates the domain may bring a minor risk, “Warning” indicates that a domain has a major risk of being malicious and “Unknown” indicates that SiteAdvisor does not have sufficient information to categorize a domain.

For benign ground truth, in addition to the “Safe” domains from SiteAdvisor, the data from previous researches carried out in this area has been used [12], along with Alexa top 1 million domains [13] appearing consistently over the last 1 year. These long lived domains with high Alexa rank are very likely to be benign.

Inference

The heterogeneous undirected baseline graph has been transformed into a homogenous undirected domain-domain graph with the use of association rules. Two association rules have been created based on the results of the IP classification experiment on the paper.

First association rule is about utilizing the dedicated IPs as well as the common ASs between domains.

If two domains are associated if they have shared either at least one dedicated IP or more than one public IP from more than one AS. If they have shared one dedicated IP it's called a dedicated association rule and if they have shared more than one one public IP from more than one AS, it is called the public association rule. The strength of the type of associations depends on the number of shared, dedicated and public IPs.

The second association rule is also similar to the first one. Instead of taking a straight association, it relaxes the dedicated association rule by using the shared /24-subnets instead of the shared dedicated IPs. That means two domains are associated if they are resolved to dedicated IPs that belong to the same /24-subnet.

By using the above two association rules, research work transforms the G-baseline into the domain to domain graph and runs the belief propagation algorithm on top of it.

2.2 Current software engineering solutions

The main inspiration to understand and build a large scale graph for data manipulation is TAO: Facebook's Distributed Data Store for the Social Graph [14].

Facebook has over a billion active users. Every time any one of these users access the site they are presented with hundreds of pieces of information derived from Facebook's social graph. Users are presented with photos, check-ins, recommendations and life events from their friends; News Feed stories, comments, reactions, and shares associated with them and many more. All this information presented to the user is highly customized and has a high update rate which in turn puts an extremely demanding workload on Facebook's data back-end due to the data retrieval and rendering which should happen in a matter of a few hundred milliseconds.

Furthermore, the data set cannot be easily partitioned, and the request rates are not equally distributed as celebrity photos and current news items have more requests. This gives rise to a highly customized, constantly changing, and read-dominated workload that is incredibly challenging to serve efficiently.

Facebook architecture is designed to store and retrieve data using MySQL as the relational database technology along with Memcache as a large distributed caching layer. It was designed at a time where the popularity of graph databases were low and hence the idea was to encapsulate the graph definition (edges and vertices) in the form of a

relational structure.

TAO (“The Associations and Objects”) design is a geographically distributed collection of server clusters which are organized logically as a tree. The design, which grabs completely different sets of clusters, are used for storing objects and associations persistently. This was the secret behind the efficiency and scalability of the design of Facebook.

Although TAO is the main inspiration behind our system design, there are a few key differences in our system when compared to TAO. Due to the lack of need to cater to a global user pool, our servers will be geographically designed to be in one place. Instead of modeling the graph in a relational form, it will be modelled similar to its conceptual form with a graph definition (edges and vertices). Similar to TAO, our system will also make use of caching layers and parallel query execution (Refer Chapter 4.3). Scalability will be addressed through highly available infrastructure orchestration while achieving the maximum use of modern graph databases and parallel SQL database engines.

2.3 Competitive Analysis

2.3.1 AT&T Cybersecurity

AT&T Cybersecurity [15] provides commercial and open source services to manage cyber attacks, including the Open Threat Exchange, which is said to be the world’s largest crowd-sourced computer-security platform.

Pros

- Provides detailed threat intelligence on various malware families.
- Helps protect enterprises from malicious Internet traffic.

Cons

- Limited intelligence on Internet artifacts such as domains, IPs and certificates.
- No graph visualization.
- No inference on malicious Internet artifacts.

2.3.2 Anomali Threat Intelligence Platform

Anomali [16] is a threat intelligence and analysis platform that aggregates threat intelligence data under one platform.

Pros

- Integrates domain and IP threat intelligence data from multiple sources.
- Provides a graph representation of Internet artifacts.
- Calculates a threat score based on external threat intelligence sources.

Cons

- Intelligence is limited to external threat intelligence sources.
- Unable to dynamically expand and perform a graph based investigation.
- Unable to traverse graphs based on time.

2.3.3 Cisco Umbrella Investigate

Cisco Umbrella Investigate [17] gives a complete view of the relationships and evolution of internet domains, IPs, and files which helps to pinpoint attackers' infrastructures, and expose current and developing threats.

Pros

- Provides detailed historical view of domains based on passive DNS data.
- Provides information about blocked traffic due to malicious activities.
- Provides a maliciousness risk score for each domain.

Cons

- No graph visualization.
- Limited information on Internet artifacts such as certificates and IP addresses.

2.3.4 VirusTotal Graph

VirusTotal Graph [18] is a visualization tool built on top of VirusTotal data set. It explores the relationship between files, URLs, domains, IP addresses and other items encountered in an ongoing investigation.

Pros

- Provides threat intelligence based on 70+ antivirus scanners.
- Provides a graph view of domain under consideration.
- Incrementally loads child nodes for parent nodes with many children.
- Provides the ability to save/share graphs.

Cons

- Does not provide capability to dynamically expand or explore graphs.
- No inference based threat intelligence provided.
- No intelligence based on certificates or domain registration information.
- Unable to traverse graphs based on time.

2.4 Significance of the proposed solution

Compared to existing products in the market, our solution provides the following competitive advantages:

- Dynamic graph based visualization and search of Internet artifacts for threat intelligence.
- Ability to integrate various data sources seamlessly to expand the network coverage.
- Ability to integrate any intelligence sources seamlessly.
- Graph based inference to uncover malicious Internet artifacts based on a small seed set of intelligence data.

- Ability to traverse graphs based on time.

Chapter 3

Functional and Non-functional Requirements

This chapter explores the functional and non-functional requirements of the system through several sections. Section 3.1 gives the functional requirements of the system, while section 3.2 gives the non-functional requirements (quality attribute scenarios) of the system.

3.1 Functional Requirements

3.1.1 End user's functional requirements

- Search an exact domain name, URL, IP address, Name server, Mail server, CNAME or SOA record.
- Keyword-based search of domains names (for example, if a user types "paypal", show all apex domains containing the keyword "paypal" e.g. paypal-mysevice.com, paypal.com, paypal-login.com).
- Display results in a knowledge graph.
- Ability to expand the graph by allowing users to fetch information about any domain nodes.
- Ability to window the graph (i.e. limit the investigation to specific time window).

3.1.2 User Stories

- **As a** User (general/security researcher) **when** I type a keyword (for example, paypal) the **System** must retrieve a list of all Apex, FQDNs, NS, MX, SOA, CNAME, and associated IPs (if any) ranked by last seen and VirusTotal scan.
- **As a** User (general/security researcher) **when** I select a keyword search result **I should see** the knowledge graph, inference graph and historical information as a log.
- **As a** User (general/security researcher) **when** I click on a graph node **I should see** the inference results of the node.
- **As a** User (general/security researcher) **when** I click on a graph node **I should be** able to expand and shrink the nodes.
- **As a** User (general/security researcher) **when** I drag the timeline, **I should see** the graph (vertices and edges) and inference values dynamically changing over time.
- **As a** User (general/security researcher) **when** I change the time window of the the time line **I should see** the graph (vertices and edges) and inference values dynamically changing over time.

3.2 Quality Attribute Scenarios

The quality attribute scenarios were developed with respect to the template in "Software Architecture in Practice" by Len Bass, Paul Clements, and Rick Kazman [19].

3.2.1 Availability Scenarios

1. In the case that an unanticipated external network failure occurs during normal operation, the system should trigger an event and notify the admin within 1 second (Table 3.1).

Table 3.1: Availability Scenario 1

<i>Source of stimulus</i>	External to the system
<i>Stimulus</i>	Unanticipated network failure
<i>Environment</i>	Normal operations
<i>Artifacts</i>	System
<i>Response</i>	Event should be triggered and must notify to the admin
<i>Response measures</i>	Triggered event must be displayed within 1 second

3.2.2 Modifiability Scenarios

1. A developer should be able to add new data sources. This change will be made to the system at design time and it should take the least amount of time possible to make and test the change. Additionally, it should not violate the semantics of the inference system after the modification (Table 3.2).

Table 3.2: Modifiability Scenario 1

<i>Source of stimulus</i>	Developer
<i>Stimulus</i>	Add new data sources
<i>Environment</i>	At design time
<i>Artifacts</i>	System
<i>Response</i>	Should not violate the semantics of the inference system
<i>Response measures</i>	+/- 0.1% error in inference algorithm results

3.2.3 Performance Scenarios

1. Under normal operations, the data transformation process should complete within 2-4 hours. If not the system will lose the ability to provide up-to-date data (Table 3.3).

Table 3.3: Performance Scenario 1

<i>Source of stimulus</i>	Data extraction system
<i>Stimulus</i>	Daily gathered data
<i>Environment</i>	Normal operations
<i>Artifacts</i>	Preprocessing system
<i>Response</i>	Notify the statistics of the preprocessing jobs
<i>Response measures</i>	Average preprocessing time should bounce between 2 to 4 hours

2. Under critical operations, the data transformation process should complete within 3-5 hours. If not the system will lose the ability to provide updated data (Table 3.4).

Table 3.4: Performance Scenario 2

<i>Source of stimulus</i>	Data extraction system
<i>Stimulus</i>	Daily gathered data
<i>Environment</i>	Critical operations
<i>Artifacts</i>	Preprocessing system
<i>Response</i>	Notify the statistics of the preprocessing jobs
<i>Response measures</i>	Average preprocessing time should bounce between 3 to 5 hours

3. The graph database should be optimized in a way that users should be able to perform fast graph traversal under normal operations, and these transactions should be processed within 5 seconds (Table 3.5).

Table 3.5: Performance Scenario 3

<i>Source of stimulus</i>	Developer
<i>Stimulus</i>	Graph queries
<i>Environment</i>	Normal operations
<i>Artifacts</i>	Graph database cluster
<i>Response</i>	Retrieve nodes and edges
<i>Response measures</i>	Retrieve the data within 5 second (without any database driver crashing)

- The graph database should be optimized in a way that users should be able to perform fast graph traversal under critical operations, and these transactions should be processed within less than 15 seconds (Table 3.6).

Table 3.6: Performance Scenario 4

<i>Source of stimulus</i>	Developer
<i>Stimulus</i>	Graph queries
<i>Environment</i>	Critical operations
<i>Artifacts</i>	Graph database cluster
<i>Response</i>	Retrieve nodes and edges
<i>Response measures</i>	Retrieve the data within 15 seconds (without any database driver crashing)

- Under normal conditions, data should be orchestrated to the front end within 10 seconds in an average bandwidth internet connection (Table 3.7).

Table 3.7: Performance Scenario 5

<i>Source of stimulus</i>	User
<i>Stimulus</i>	User request
<i>Environment</i>	Normal operation
<i>Artifacts</i>	API and the middleware
<i>Response</i>	API response
<i>Response measures</i>	Return the result (JSON response) under 10 seconds (100Mbs bandwidth)

6. Under critical conditions, data should be orchestrated to the front end within 20 seconds in an average bandwidth internet connection (Table 3.8).

Table 3.8: Performance Scenario 6

<i>Source of stimulus</i>	User
<i>Stimulus</i>	User request
<i>Environment</i>	Normal operation
<i>Artifacts</i>	API and the middleware
<i>Response</i>	API response
<i>Response measures</i>	Return the result (JSON response) under 20 seconds (100Mbs bandwidth)

7. Under normal operations, inference algorithms must complete the inference process relevant to the extracted subgraph within 10 seconds (Table 3.9).

Table 3.9: Performance Scenario 7

<i>Source of stimulus</i>	Internal to the system (Inference algorithms)
<i>Stimulus</i>	Extracted sub graph from user request
<i>Environment</i>	Normal Operation
<i>Artifacts</i>	Middleware
<i>Response</i>	Inference results
<i>Response measures</i>	Run the algorithm and return results within 10 seconds

8. Under critical operations, inference algorithms must complete the inference process relevant to the extracted subgraph within 10 seconds (Table 3.10).

Table 3.10: Performance Scenario 8

<i>Source of stimulus</i>	Internal to the system (Inference algorithms)
<i>Stimulus</i>	Extracted sub graph from user request
<i>Environment</i>	Critical operation
<i>Artifacts</i>	Middleware
<i>Response</i>	Inference results
<i>Response measures</i>	Run the algorithm and return results within 20 seconds

3.2.4 Scalability scenarios

1. The database cluster should be able to scale up with the data without any availability corruption (without any downtime) (Table 3.11).

Table 3.11: Scalability Scenario 1

<i>Source of stimulus</i>	System
<i>Stimulus</i>	Data
<i>Environment</i>	Normal Operation
<i>Artifacts</i>	Database clusters
<i>Response</i>	No downtime and notify the Volume expansion
<i>Response measures</i>	Updated volume

Chapter 4

Analysis and Design

This chapter explains about the design of the system in depth through several sections, emphasizing the experiments and tests carried out to understand the data, and the design of a suitable system architecture. Section 4.1 explores the data sources used in the system which includes Farsight Security, VirusTotal, MaxMind, and Alexa Rank and the techniques used to store them, while section 4.2 explores the design of the system and how the data is preprocessed and stored.

4.1 Sources of data

The system deals with mainly four types of data sources to build the graph:

- Farsight Security [20] - Cybersecurity intelligence solutions that is the main PDNS data source.
- VirusTotal [21] - Reputation service that provides aggregated intelligence on any URL.
- MaxMind [22] - Intelligence service that provides IP geolocation data.
- Alexa Rank [13] - Website ranking service that adds the reputation value to websites based on their popularity.

PDNS data from Farsight Security was used to model the graph while data from the other sources, VirusTotal, Alexa, and MaxMind, was used as intelligence sources to add belief values to the vertices and edges in the graph in order to run inference algorithms on top of the graph.

Table 4.1: Structure and Volume of data feeds (daily)

Data feed	Structure of raw data	Volume	Structure of the transformed data
Farsight (PDNS)	Document	170 GB - 195 GB	Graph and Time series
VirusTotal	Document	2.3 GB - 3 GB	Document and Time series
Alexa Rank	Document	18MB	Document and Time series
MaxMind	Key-value	1 GB - 2 GB	Key-value and Time series

4.1.1 Passive DNS data from Farsight Security

PDNS data from Farsight Security was used as the key ingredient of the graph which consists of several types of DNS records such as A, NS, MX, CNAME, and SOA, collected daily. Initial studies were conducted in order to understand the volume and format of this data and the following fields were found to be common:

- count - Number of times that the record has been captured by Farsight Security's PDNS listeners throughout the day.
- time_first - First time the record has been captured by the PDNS listener.
- time_last - Last time the record has been captured by the PDNS listener.
- rrtype - Flag to indicate the type of record.

Table 4.2: Overview of Farsight data

Type of record	Data format	Definitions
<p>A record - Contains information about domain to IP resolutions.</p>	<pre>{ "count":6, "time_first":1596227280, "time_last":1596227280, "rrname":"1.ac.", "rrtype":"A", "bailiwick":"1.ac.", "rdata":["209.203.84.21"] }</pre>	<p>rname - Indicator of the Apex/FQDNs.</p> <p>rdata - Array of IPs that are resolved by Apex/FQDNs in the rname field.</p>
<p>NS record - Contains information about name servers.</p>	<pre>{ "count":2, "time_first":1596307649, "time_last":1596307649, "rrname":"4.ac.", "rrtype":"NS", "bailiwick":"4.ac.", "rdata":["dns1.goodspeed.net.", "dns2.goodspeed.net."] }</pre>	<p>rname - Indicates the name of the name server; this could be a root name server to FQDNs.</p> <p>rdata - Array of name servers that are associated with the name server indicated in the rname.</p>
<p>MX record - Contains information about mail servers.</p>	<pre>{ "count":7, "time_first":1601499588, "time_last":1601586025, "rrname":"encompass.co.ac.", "rrtype":"MX", "bailiwick":"encompass.co.ac.", "rdata":["10 mail.encompass.co.ac."] }</pre>	<p>rname - Indicate the name of the mail server.</p> <p>rrdata - Array of mail servers associated with the mail server indicated in the rname.</p>

<p>CNAME record - Contains information about aliases of another canonical domain.</p>	<pre>{ "count":12, "time_first":1602104578, "time_last":1602198338, "rrname":"www.acs.ac.", "rrtype":"CNAME", "bailiwick":"acs.ac.", "rdata":["tn50000215. schoolwires.net."]}</pre>	<p>rrname - Indicate the relevant domain and its original domain name.</p> <p>rrdata - Array of all aliases for the Apex/FQDNs in the rrname field.</p>
<p>SOA record - Contains administrative information about the zone.</p>	<pre>{ "count":84624187, "time_first":1596228840, "time_last":1596240622, "rrname":".", "rrtype":"SOA", "bailiwick":".", "rdata":["a.root-servers.net. nstdl.verisign-grs.com. 2020073100 1800 900 604800 86400"]}</pre>	<p>rrname - Contains information about Apex/FQDNs.</p> <p>rrdata - Relevant zone transfer information.</p>

Analysis of the volume of PDNS data

Table 4.3 gives details on the volume of data and the unique number of records received daily by Farsight Security's PDNS data feed based on their record types (A, NS, MX, SOA and CNAME).

Table 4.3: Statistics of PDNS data

Record Type	Data Volume (daily)	Number of Records (Unique)
A	68GB-75GB	350 million - 400 million
NS	51GB-60GB	200 million - 250 million
MX	2GB-3GB	10 million - 15 million
SOA	27GB-30GB	100 million - 115 million
CNAME	22GB-25GB	100 million - 115 million

Formulating graph using PDNS data

From the different types of records available in the PDNS data (A, NS, MX, CNAME, SOA) the following vertices and edges were extracted.

- Vertices: Apex, FQDNs, IPs, Name Servers, Mail Servers, CNAME, SOA
- Edges: Apex-IP, Apex-FQDNs, FQDNs-IP, Apex-NS, FQDNs-NS, Apex-MX, FQDNs-MX, Apex-CNAME, FQDNs-CNAME, Apex-SOA, FQDNs-SOA

Most of the Apex/FQDNs have used the same IPs, name servers, mail servers, and SOA. This gives rise to a large number of connected components with various underlying properties such as the same cloud vendor, CDN etc.

As shown in [Figure 4.1], the graph is stored along with time stamps which allows users to filter a subgraph for any given time interval.

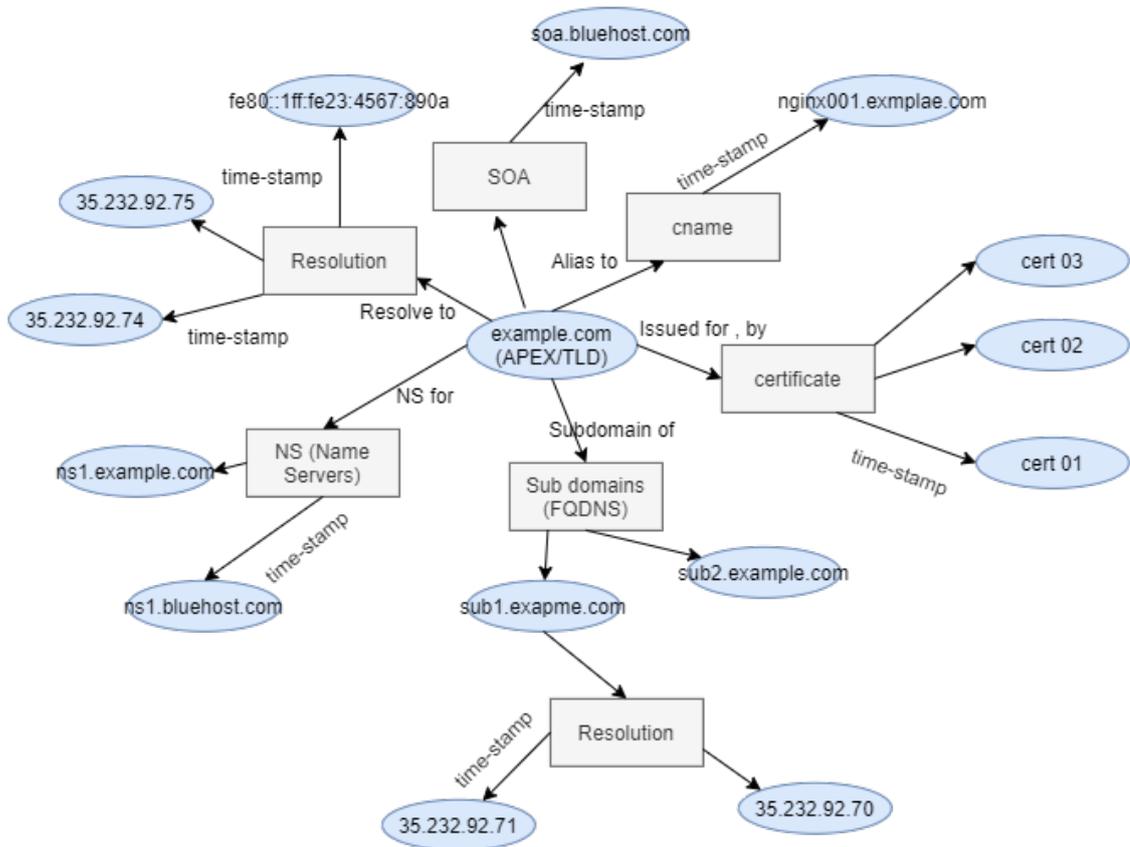


Figure 4.1: An instance of a heterogeneous knowledge graph (Apex as start node)

Storing PDNS data

Once the conceptual model of the graph is built the next challenge was to store the graph data in a manner which preserves its time properties. The inference algorithms depend on the time we run the algorithms on the graph so one of the major requirements was to build the graph to capture the time properties of PDNS data. Additionally, a large number of edges and vertices are also appended to the graph daily. This highlighted the need for a clustered environment to store and query the graph. Section 4.2.2 describes the selection process of the clustered environment.

4.1.2 Intelligence data from VirusTotal

VirusTotal is a URL reputation system which inspects items with over 70 antivirus scanners and URL/domain blacklisting services and gives a comprehensive report of that URL. cGraph uses VT data as an intelligence source for the inference algorithms in order to mark vertices as malicious or benign. While the schema of VT data gives detailed

information about the scanned URL, the system is mainly concerned with the majority vote of the indicators (malicious, spam, phishing, malware, ransomware, suspicious) and the positive count which indicates the number of indicators that have marked the URLs as infected.

Analysis of the volume of data from VirusTotal

Table 4.4 shows the daily, monthly and yearly volumes of the VT data feed and the number of records.

Table 4.4: Analysis of volume and number of records of VT data feed

	Data Volume	Number of Records (VT Scans)
Day	2.8GB-3.2GB	5 million - 8 million
Month	80GB-95GB	175 million - 250 million
Year	2GB-3GB	2 billion - 2.5 billion

Distribution VT data

Table 4.5 gives an overview of the most scanned Apexes within the study period of August 2019 to November 2019. As shown in the table, some Apexes are more prominent than others in the VT feed. These observations were considered when sharding VT data.

Table 4.5: Top Apex Domains with Most FQDNs

Apex	#FQDNs	#scans	#scanners >0%	Description
blogspot.com	2,112,783	4,144,736	25.28	Popular Blogging Site
coop.it	1,881,552	1,940,000	0.0002	URL Shortner Service
mcafee.com	1,763,914	2,342,965	0.013	McAfee Endpoint Hosts
opendns.com	1,490,547	1,987,743	0.001	Cisco Open DNS

office.com	1,068,008	1,460,318	1.43	Microsoft's Data Telemetry
footprintdns.com	764,060	883,773	0.001	Microsoft DNS Tracker
cedexis-radar.net	703,962	922,056	0.041	Cedexis Radar Tracker
amgdgt.com	233,914	330,944	0.01	Digital Advertising (Amobee Inc.)

Storing VT data

Similar to PDNS data, VT data was also stored in a manner which preserves its time properties, in order to use it as an intelligence source for graph inference and was profiled based on FQDNs as shown in Figure 4.18. A large number of records of VT data was also appended to the data store every day which resulted in a massive volume of data. This necessitates a clustered environment for storing VT data (similar to PDNS data) with a good partitioning mechanism.

First Approach

The initial idea was to keep one table for one apex and group the information of its scanned URLs in the table. However, by studying the data in depth it was clear that there is no single database that can store a large number of such individual tables and while some tables will be growing exponentially (for example, site.google.com, blogspot.com) and others will not (for example, ugvle.ucsc.cmb.ac.lk). To overcome these drawbacks, a different approach was proposed.

Second Approach

In this approach, all the URLs were stored using two level hierarchical partitioning. Specifically, the tables were partitioned based on letters, digits, and special characters (for example, !, @, # etc) and profiled them by queries. The main problem was to identify an effective way to create relations and partitions.

Each of the tables were partitioned again by letters, digits, and special characters. The first letter of the apex/FQDNs decides which table it should be stored in and the second letter decides which partition it should be in. With this conceptual model the number of relations were reduced from 2,136,347,556 to 1,369 (37*37) which is manageable by any database engine and also overcomes the problem of exponential growth of some tables. Section 4.2.2 explores how parallel SQL was used to store data according to this model and how it was queried.

Data partitioning

Storing a large number of records (Two billion per year) is quite a tedious task and querying them is also time consuming. As a solution for this relations from letter a to z, number 1 to 9 and all ASCII special characters and two indexes for table level and database level were used.

4.1.3 Geolocation Data from MaxMind

MaxMind is a geo IP locator that gives information of the geolocation of any particular IP. Compared to the other sources, MaxMind brings a considerable amount of data into the system which is easy to manipulate. MaxMind is the main source of geolocation data which gives an insight to the system's initial search results.

Storing MaxMind Data

MaxMind data is of a manageable volume and it has its own file type which is easy to manipulate. Due to these reasons, the original file format was used to store and query the data without any restructuring.

4.1.4 Data from Alexa Rank

Data obtained from the reputation system Alexa Rank includes the Alexa rank of Apex/FQDNs based on popularity and other relevant measurements. This data is used to give some insight into initial search results and rank them as well as to be fed to the inference algorithm of the knowledge graph. We use domains with high alexa ranks as benign domains for the inference algorithms.

Data format

Alexa data is represented in a key-value format, as shown below.

```
google.com, 1  
youtube.com, 2  
[apex/fqdns], [rank]
```

Storing Alexa Rank Data

Data concerning the Alexa rank is gathered every day and as a result it was a necessary requirement to maintain the time-series properties of this data. Due to this, a SQL schema was proposed which takes into consideration both the key-value properties and time-series properties of the data (Refer Appendix B).

4.2 System Design

This section explores the steps taken to store the above data in suitable databases and how the big data environment was handled under the following sub-components:

1. Data preprocessing
2. Data storing
3. Data orchestration and data visualization

4.2.1 Data preprocessing

Due to the large volume of data that is handled by the system, data preprocessing is a time consuming and computationally heavy process (Table 4.1). Distributed data processing techniques such as map-reduce and map reduce implementations such as Apache Hadoop and Apache Spark were considered for this task. According to the study of graph data processing comparison [Figure 4.2] by Lei Gu and Huan Li in the publication "Memory or Time: Performance Evaluation for Iterative Operation on Hadoop and Spark" [23], Spark was far efficient than the old school Hadoop environment for MapReduce operations. As a result, Spark was selected to distribute computing engines for pre-processing and scheduled batch processing tasks in the system.

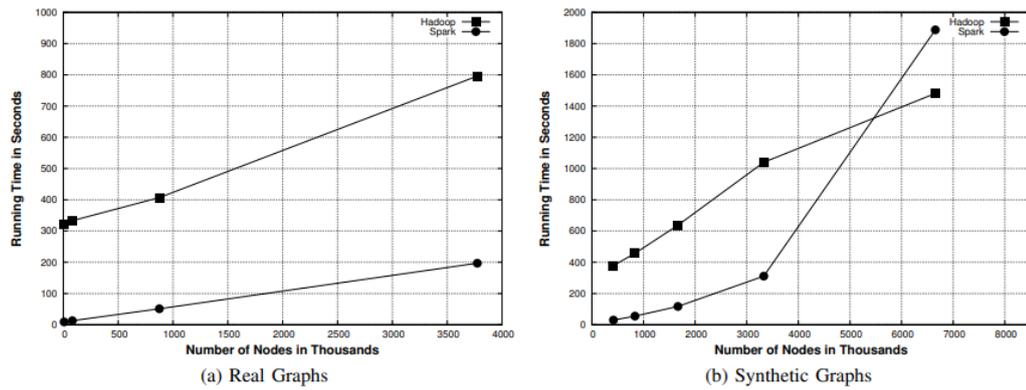


Figure 4.2: Scalability comparison

4.2.2 Data storing

Databases

Maintaining the quality and integrity of each data source was a vital factor that had to be considered when storing data. As a result, community studies and experiments were conducted in order to identify the best databases to store the data used by the system. Since each data source had its own requirement, it was decided that a single database is insufficient to control such a vast amount of data. The proposed solution for this was a cluster of databases and the following databases were selected for this purpose.

- Graph storage - ArangoDB
- VT data storage - Greenplum DB
- Alexa Rank data storage - Greenplum DB

Document/Time series Data Storage (VT and Alexa Rank data)

Greenplum DB was used for both Alexa ranked domain profiling and VT profile ranking since it is a massively parallel PostgreSQL database with the capability to run parallel SQL queries [24]. It was selected as the most suitable database to store document/time-series data based on community readings, use-case studies and various trial and error experiments that were carried out.

Graph Data Storage (PDNS data)

A benchmark test was conducted with other competitive open source graph databases available (ArangoDB vs OrientDB vs Neo4j) to analyze the performance of graph queries. It was done on a Virtual Machine in Google Cloud Platform with 16 cores and 32GB RAM and 0.01% graph data from the original PDNS data was used. Since the community studies we have previously done was complementary to the test results obtained, ArangoDB was selected as the graph data storage. ArangoDB's support for scalability and clustering for big data environments were also other major factors that was considered in this decision.

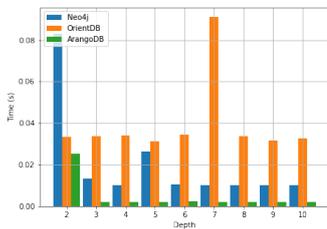


Figure 4.3: Any Direction

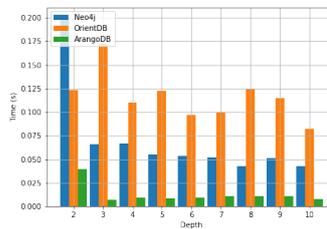


Figure 4.4: Inbound

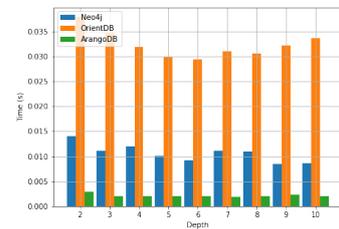


Figure 4.5: Outbound

In line with the system's goals, the system was designed to optimize Read operations compared to Write operations. Due to this, benchmark testing was performed for Read queries of inbound, outbound and any directional graphs for different Apex/FQDNs with different depth values. Figures 4.3, 4.4, and 4.5 give the mean time taken for compilation of graphs against different depth values. The time taken for graph compilation is significantly less for ArangoDB when compared to both OrientDB and Neo4j.

Furthermore, one of the main requirements in the heterogeneous graph and a necessary requirement when deploying inference algorithms is the ability to traverse in any direction. ArangoDB executes the query and returns the node and edges more effectively when compared to OrientDB and Neo4j. It was found to be 25% faster than OrientDB and 33% faster than Neo4j. Neo4j is a good competitor to ArangoDB but from an engineering point of view Neo4j had development limitations such as the lack of support for clustering and sharding in the community edition.

Database drivers

In order to select a suitable programming language to build APIs and middleware components, a study was done to obtain the performance of database drivers by using different languages. Figure 4.6 gives the performance of database drivers for ArangoDB in different programming languages against the depth of the graph.

Python and Go have an average execution time of 2-3 seconds, when compared to the other supported languages, Java and PHP. Thus, it was concluded that both Python and Go are suitable for the development of APIs and middleware components of the system. However, since time taken to execute queries is of vital importance, Go was deduced to be the better suited language for the system's needs owing to its better performance over Python.

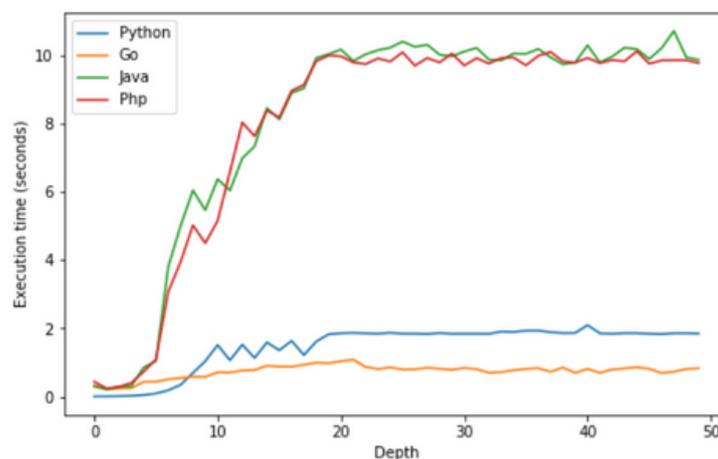


Figure 4.6: Execution time vs depth under 10 000 node limit

Infrastructure orchestration

Maintaining large database clusters is a tedious task. As a solution for this, it was decided to use a container orchestration platform such as Docker Swarm or Kubernetes [25] to deploy the middleware application. Kubernetes was selected as the more suitable containerized platform for the system due to its flexibility and scalability. Using Kubernetes to manage all computation resources together makes the manipulation of database clusters much easier [Figure:4.7]. Kubernetes brings stable and easy infrastructure orchestration as an alternative to installing binaries on VMs and creating database clusters. It

uses 'pod' or 'container' as its smallest instance of infrastructure orchestration which can consist of one or more containers inside it. This containerized architecture has made the developing and deployment process of middleware and APIs much easier than a regular development environment. Being able to deploy the algorithms and dependencies as same as its development environment (research environments) added more consistency for the middleware and its process.

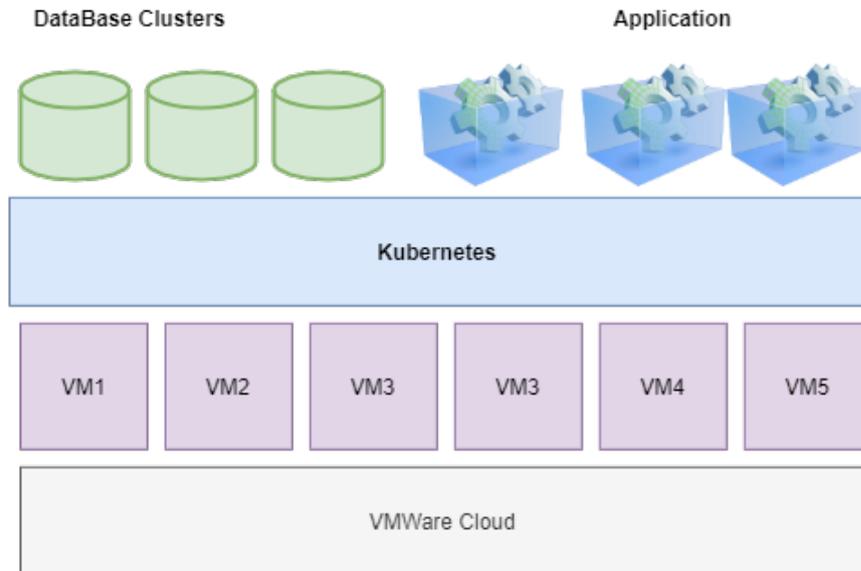


Figure 4.7: High level diagram of infrastructure orchestration with Kubernetes

As shown in Figure 4.8, different namespaces were used inside Kubernetes such that the database clusters and applications could be kept separately so that it facilitates the easy maintenance and enhancement of the system in the future.

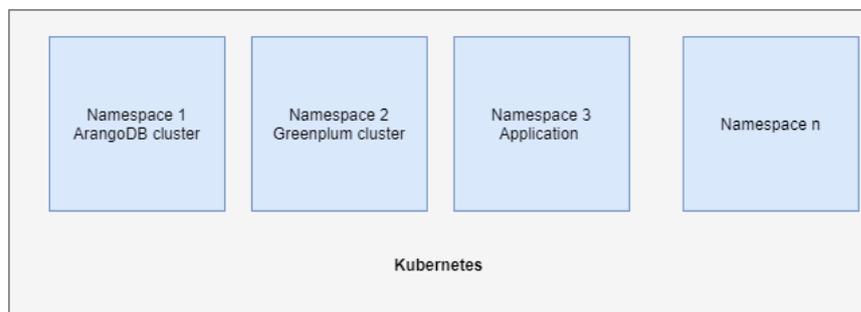


Figure 4.8: Namespaces inside Kubernetes

Data visualization

In graph data visualization, the most important requirement was effective graph rendering and good usability. Community studies and readings were done to evaluate and compare the performance and usability of open source graph rendering libraries. As a result, Graphin (AntD) and D3.js, which are two Javascript-based graph visualization libraries, were selected as most suited for the system's needs and experiments were conducted to determine the library with better rendering performance.

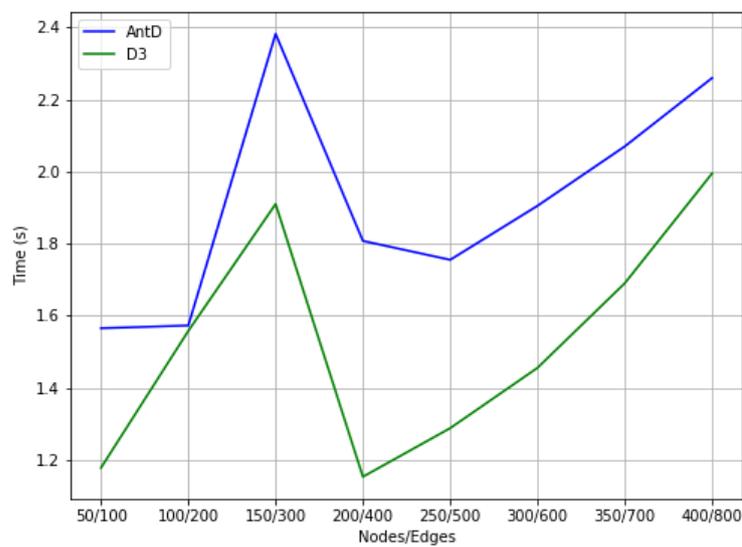


Figure 4.9: Ant Performance of Force Layouts AntD vs D3

The time taken to render the graph with different numbers of nodes and edges were measured for each library. The experiments were run on different computation environments for different browser types and versions and the average value was taken when plotting the chart for rendering time against the nodes/edges combinations. The graph was populated using real data from the databases and it was seen that the performance of the D3 library was better than Graphin (Figure 4.9). However, Graphin provided better usability and visual appearance. Therefore, it was selected as the graph visualization library.

Graphin provides several layouts to render the graph such as Random, Circle, Radial, Grid, Concentric and Force. Experiments were done on the same set of nodes/edges combinations as before on different browsers and the average time taken to render the

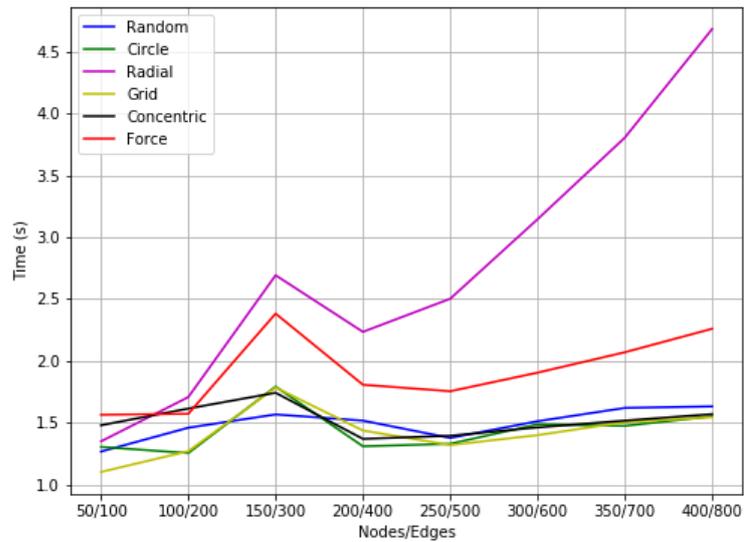


Figure 4.10: Graphin: Performance of Layouts

graph was considered for each of these layouts as seen in Figure 4.10. Although Random, Grid, Circle and Concentric layouts had better performance when compared to Radial and Force layouts, it was decided that the Force layout was better suited for data visualization of the system when considering its functionality and purpose as well as its usability. Figure 4.11 and Figure 4.12 shows the visual difference of the two layouts: Force and Random.

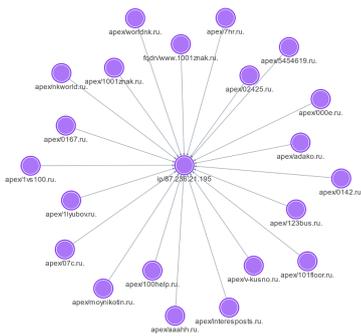


Figure 4.11: Force Layout

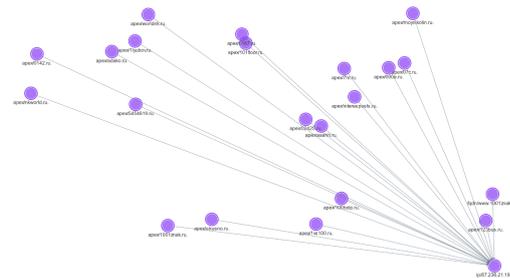


Figure 4.12: Random Layout

4.2.3 System architecture design

Designing the system architecture was done as an incremental process. Initially, a single node database instance with all the functionalities of the database was investigated. This monolithic based unscalable architecture resulted in a number of issues in the system and was a hindrance to accomplish the necessary requirements. As an alternative solution, a containerized microservices-based architecture was proposed with high availability, scalability and portability.

The said architecture is explored in 3 views below. Each view describes how the design helps to plant the algorithms that are published in a real time system and do analysis on the running system.

1. Infrastructure point of view
2. Data point of view
3. Concepts and Relationships point of view

Infrastructure point of view

Figure 4.13 depicts the system architecture from the point of view of its infrastructure. It describes how the system components, such as Spark cluster, ArangoDB cluster, Greenplum DB cluster, job schedule, middleware and the web application, are served on the private cloud environment.

All resources have been provisioned on top of VMware vSphere private cloud. Kubernetes cluster has been installed on top of the provisioned VMs and an abstracted computation resources layer has been created in order to manage all computation resources together.

The entire application has been built on top of Kubernetes, web servers, and API (middleware) ingress to the outside world via the load balancer. Behind that, we have implemented ArangoDB cluster and Greenplum cluster on top of Kubernetes cluster and the Spark cluster has been implemented outside of the Kubernetes cluster. A dedicated set of resources has been allocated to Spark itself to preserve its performance. The middleware has been deployed on top of Kubernetes in pods as replica sets to preserve the availability and get the maximum use of resources available in the Kubernetes cluster.

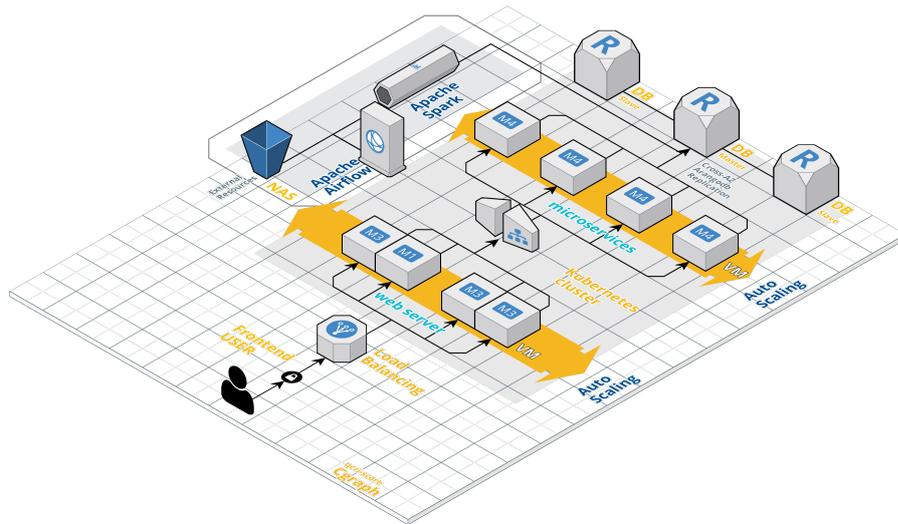


Figure 4.13: Infrastructure point of view

Apache Airflow was used to automate the daily insertion process by running daily Spark jobs and inserting new data into the database clusters.

Kubernetes was used for easy management of database clusters and to increase the availability of data nodes which are requirements of the system. It also facilitated the addition of new computation resources to the pool which makes scaling much easier.

Table 4.6: Infrastructure Components

Web Server	Serve the front-end application
Load Balancer	Balance the load arriving at the web server from outside (Both web server and load balancing happen via one NGINX Ingress Controller)
REST API	Orchestrate the data to the front-end (connect the front-end and back-end)
API Gateway	Manage all API calls happen through the microservice
Microservices	Consists of parallel data executors, data transformers, inference algorithms, MaxMind data extractors etc.
ArangoDB Cluster	Store graph data (PDNS data)
GreenplumDB Cluster	Store intelligence data sources (VT and Alexa Rank)

Spark Cluster	Run data preprocessing and insertions
Airflow	Automate the data preprocessing and data insertions

Data point of view

Data point of view describes how the data flows throughout the system design and how different data sources interact with each other to produce the expected results. As shown in figure 4.14, the graph was mainly formed with PDNS data and VT, Alexa Rank and MaxMind data sources was used to add belief values to the edges and nodes using the inference algorithms which is run on top of the formulated graph.

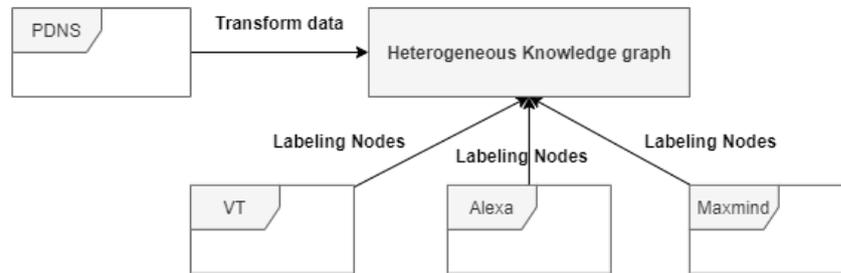


Figure 4.14: Data point of view

Concepts and Relationships point of view

The system architecture from the point of view of its concepts and relationships explores how the middleware components, parallel query execution components and inference algorithms interact with each other. The main ingress point of the system is the API. Several endpoints have been defined in the API to do queries in the backend system. Each query is treated as a separate computation work and executed parallelly. The component diagram¹ in figure 4.15 gives an overview of the interaction between system components.

The system functions as a set of microservices. When a search query is executed for an IP, Apex, FQDNs, NS, MX, CNAME or SOA from the frontend of the system, the relevant microservice executes parallel queries on an ArangoDB cluster and fetches the relevant data. If the scanned resource is an IP, the search component interacts with MaxMind IP database and gather necessary data. All other resources will additionally

¹UML 2.0 Specification

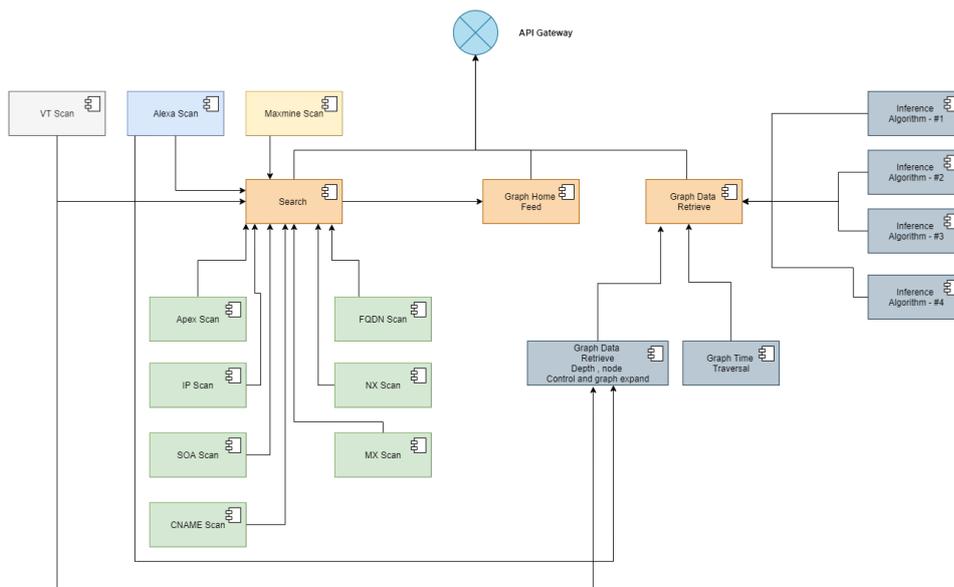


Figure 4.15: Concepts and Relationships point of view

fetch data from the VT profile database and Alexa Rank database. The retrieved results will be sorted based on Alexa Rank and VT feedback. The results will be displayed on the front-end as a ranked list of search results.

When a search result is selected, the graph data retrieval endpoint will fetch the data necessary to render the graph in the front-end. Graph data retrieval endpoint consists of two children microservices; a microservice to handle the depth and the number of nodes returned in the graph traversal, and another microservice to handle the traversal of the graph using time property. VT scan and Alexa scan microservices will fetch data relevant to the vertex and assign a maliciousness score to each domain of the graph using the graph inference algorithm.

The diagram in figure 4.15 shows a single instance of the microservices. However, each microservice has been deployed as a replica set in Kubernetes which helps to avoid single point of failures.

4.2.4 Design of the inference system

Khalil et al in their work has designed the inference system for a homogeneous domain graph which consisted of only of two types of nodes; domains and IPs only. The baseline graph has been converted into a domain-domain graph by using association rules and the inference algorithm has been run on top of it. The knowledge graph constructed in our

project is of a heterogeneous nature and it is exponentially large when compared to the graphs used in the publication.

There has been recent research showing that the belief propagation works on heterogeneous graphs of different types of network nodes [26]. Following this approach, we run belief propagation on the heterogeneous knowledge graph we build. An advantage of this approach is that even if the association rules change over time, this approach can withstand the changes and produce acceptable results.

Ground truth extraction for inference

Similar to the work in the publication by Khalil et al, our system also has two ground truths; malicious ground truth and benign ground truth. Malicious ground truth was defined by using the intelligence provided by the daily feed of VirusTotal. For a given resource the VT positive count was taken for seven days prior to the current date and if the count was ≥ 2 , it was considered to be malicious [4]. Benign ground truth was defined by using the intelligence provided by Alexa Rank data. If the Alexa Rank for a given resource consistently below 100K for a week, considered to be benign [4].

Node labeling

For the node labeling process, the node types of the nodes extracted from the graph were synced with the relevant ground truth information and they were labeled by following the heuristics given below:

- If the given Apex/FQDNs/IP/SOA/CNAME/NX/MX is in the ground truth list and its VT count is ≥ 2 , the node is marked as Malicious (indicated by 1)
- If the given Apex/FQDNs/IP/SOA/CNAME/NX/MX is in the ground truth list and its Alexa Rank is $\leq 100,000$, the node is marked as Benign (indicated by 0)
- If the given Apex/FQDNs/IP/SOA/CNAME/NX/MX is not in the ground truth list, the node is marked as Unknown (indicated by -1)

Inference process

The implementation of our version of belief propagation has a worst case time complexity of $O(EV)$. Even for the implementation of the algorithm using a low-level language

based (C-language based) parallel code it was seen to take days to execute such a large volume of data. Initially, our attempt was to map the belief propagation process to the MapReduce operation and run on it. However, it was extremely tedious to interpret the inference algorithm to MapReduce operations. As a solution, it was decided to run the inference algorithm on user requested depth 2 sub graph that extracted from the main knowledge graph, and then run inference on top of the immediately connected components as requested by the user. This scenario is depicted in figure 4.16.

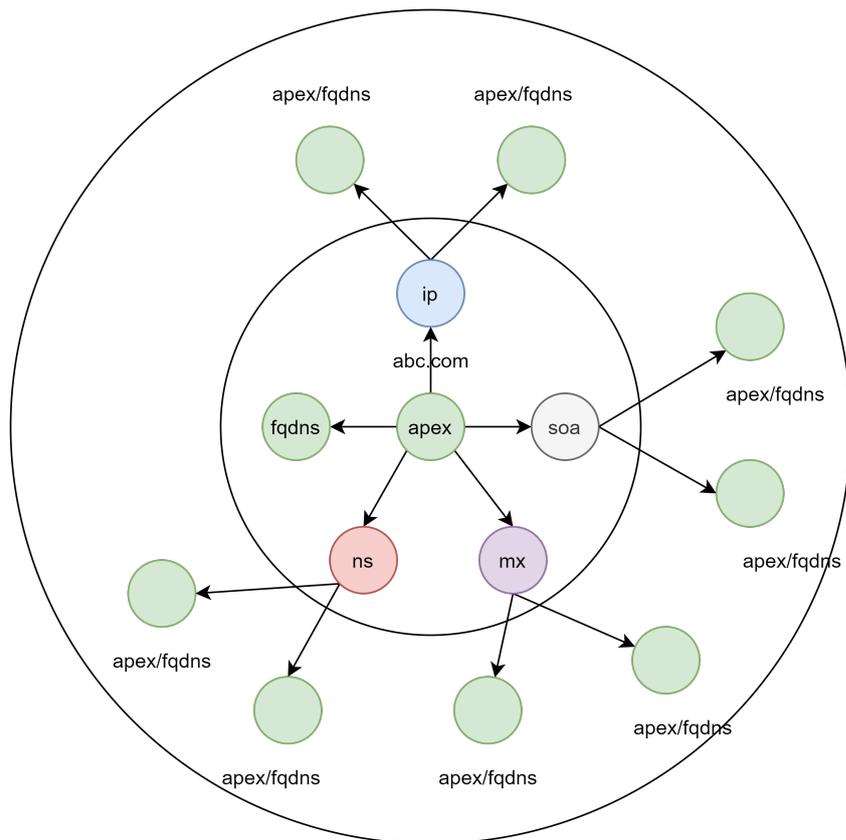


Figure 4.16: Subgraph with a depth of 2

For the inference process a subgraph with depth 2 was extracted from the knowledge graph with the resource scanned by the user as the root node. For an example, if the user scans the Apex abc.com, the latest DNS formation is taken from the knowledge graph with a depth of 2 as shown in figure 4.16. Then it is synced with the data of malicious and benign nodes and the inference algorithm is run on top of it and the graph is rebuilt in a heterogenous manner as shown in figure 4.17.

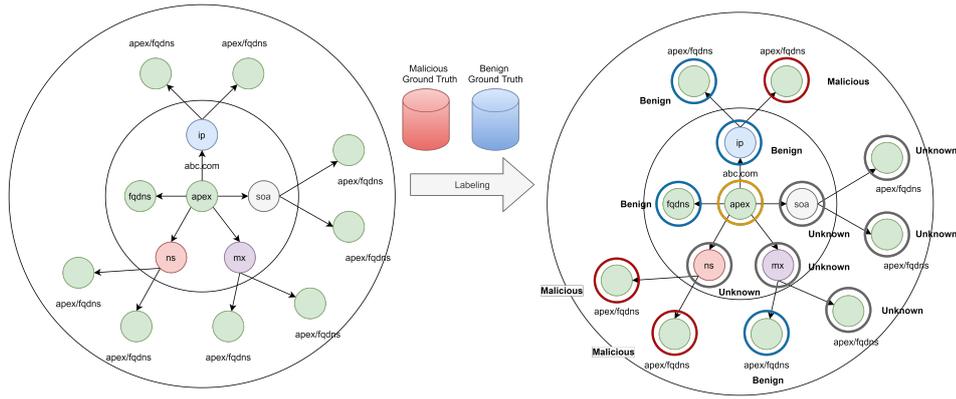


Figure 4.17: Labeling the depth two subgraph before Inference

4.2.5 Technology stack

Table 4.7: Overview of the Technology Stack

Scenario	Technology
Alexa Rank and VT data storage	Greenplum DB
PDNS data storage	ArangoDB
Data Preprocessing	Apache Spark
Managing high available system	Kubernetes
API/Middleware (parallel query executors and inference algorithms)	Go, C, Python
Front-end	ReactJS with Redux, Graphin
Documentation	Swagger, MkDocs
Spark Jobs Automation	Apache AirFlow
Caching	Redis
Traffic Manipulation	KrakenD
CI/CD	GitHub Actions

4.3 Engineering Challenges and Solutions

Various software engineering challenges were encountered during the development process and this sections explores them and how they were overcome.

4.3.1 Profiling VT records

VirusTotal is the main source of intelligence data in the system. The total number of VT scan records that had to be profiled based on their FQDNs is 2,136,347,556. In order to fulfill this task, two different approaches were used. One was just using flat-files and the Apache Spark distribution system, and the other approach was to preprocess the data using Apache Spark and save in a database. Before going into further details about each process, it is important to take a closer look at the profiling process. As shown in Figure 4.18, it was required to profile each scanned URL under its FQDNs and separately retrieve the profile based (time series) data to feed the inference algorithm and label the vertices of the graph. One of the major requirements of this profiling is to monitor the behaviors of FQDNs relative to the time.

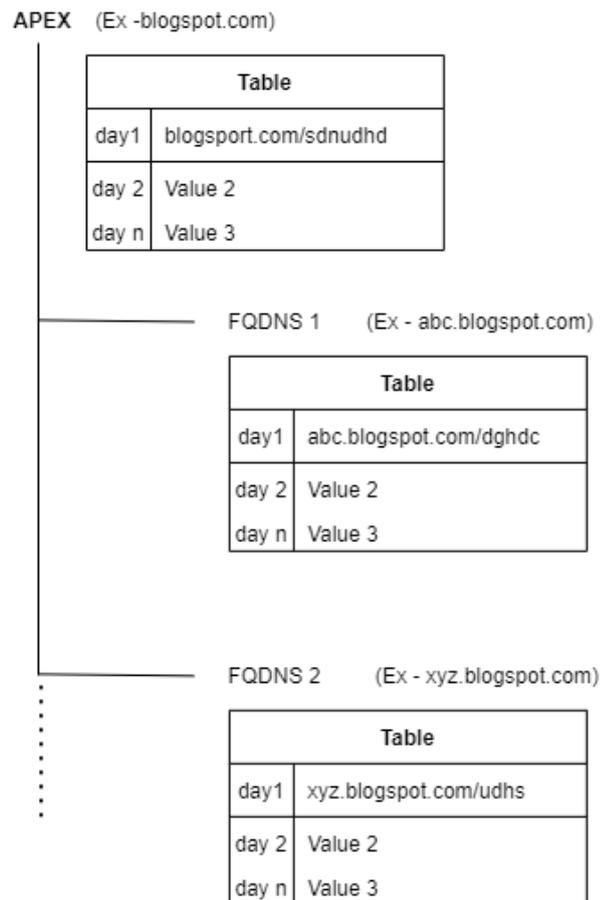


Figure 4.18: VT profiling

As shown in figure 4.18, the requirement was to profile the scanned URL feed and scan results in a hierarchical structure.

Using Spark

Since the requirement of the system was to build a hierarchical profiling model, the initial idea was to use a model similar to the file hierarchy model found in operating systems. Apache Spark was used to preprocess the data from VT and this data preprocessing stage consisted of running a set of MapReduce operations followed by a set of groupBy operations on the data and saving the grouped results on the file system. A single file was used to store one scanned apex. However, this approach had some drawbacks owing to the difficulties encountered while inserting new data.

Spark has been designed to load all the necessary data into memory at once using a fundamental data structure called Resilient Distributed Datasets (RDD). RDD is a read-only, partitioned collection of records, and each dataset in RDD is divided into logical partitions which may be computed on different nodes of the cluster in parallel. Due to this, it is impossible to track the necessary data in a straightforward manner without running any map filter operations which makes insertion of new data an expensive and time-consuming task.

Furthermore, running search queries on the data is challenging due to the difficulty in maintaining an indexing mechanism. As a solution, it was proposed to use a binary tree. However, integrating a custom-built binary tree with Spark MapReduce operations was proved to be ineffective due to the amount of time taken to run operations such as keyword searches. Additionally, since VT data amounts to over 2 billion records in total, maintaining such a large number of files was a tedious task and the data partitions had to be maintained as ext4 file type.

Taking these maintenance and scalability issues into consideration it was decided to use a dedicated database or database cluster to store the data

Using Database

One of the major conflicts encountered when using databases was to integrate the profiling model as same as the conceptual model. Due to this reason, our first attempt was to keep one table per one FQDNs. However, it was an impossible task to find any single database that could handle 2 billion tables, so it was realized that it was a naive approach. The next step was to identify a model that was compatible with both conceptual models and the database. Since this data is structured and well-formed, it was decided that a

relational database was most suited to store for this data. This resulted in two solutions: one was to design a normalized design, and the other was to keep some redundancies and save the data without a normalized design.

Normalized design

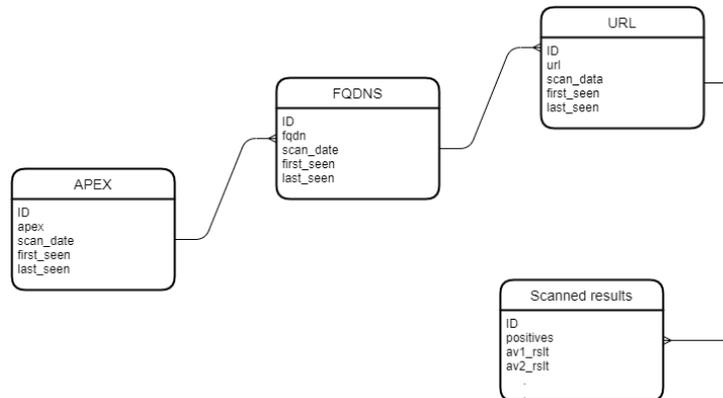


Figure 4.19: ER diagram for the Normalized design

All the apexes were encapsulated into one relation while FQDNs, scanned URLs, and antivirus engine results were encapsulated into another relation². When a single apex is considered there were multiple FQDNs associated with it. Therefore, a one-to-many relationship was created between Apex and FQDNs relations. Likewise, a single FQDNs has multiple scanned URLs associated with it. Therefore, a one-to-many relationship was created between FQDNs and URL relations. Finally, a similar one-to-many relationship was created between URL and scanned results relations (Figure 4.19).

After the data was normalized into the above structure, each relation contained nearly 200 million records, and both URLs and scanned results relations contained nearly 800 million and 2 billion records respectively. Even with indexing and partitioning of the relations, running aggregate queries with join operations took a considerable time to complete (20 seconds to 1-2 minutes). This was a major barrier in achieving the system's quality requirements. To overcome this, vertical scaling was done by giving more resources to the database. However, the goals were still not reachable and the performance was poor.

The solution for this was identified as having clusters of the database in which vertical scaling could be done exceptionally well, distributing the data among the database in-

²UML 2.0 Specification

stances (segments) and running parallel queries on them. In order to do that, the database had to be sharded. By keeping the same normalized schema, Random and Hash-based data sharding methods were used in the cluster. Random sharding distributes data in a round-robin manner while the hash-based sharding uses a hash of the data in columns. Still, it was observed that this normalization makes some unnecessary traffic between database instances resulting in the reduction in performance of the database cluster.

Only hash-based sharding could be used in the normalized design. The reason was that the database does not allow sharding in a random manner whenever there exists a constraint in the relation. This was a critical way of distributing data due to the fact that poor distribution of data led to communication between segments when executing queries which led to reducing the performance. Although indexing and partitioning mechanisms were used to overcome this, the results were not satisfactory. Thus, it was concluded that the best practice of avoiding redundant data had to be traded to reach the expected query performances.

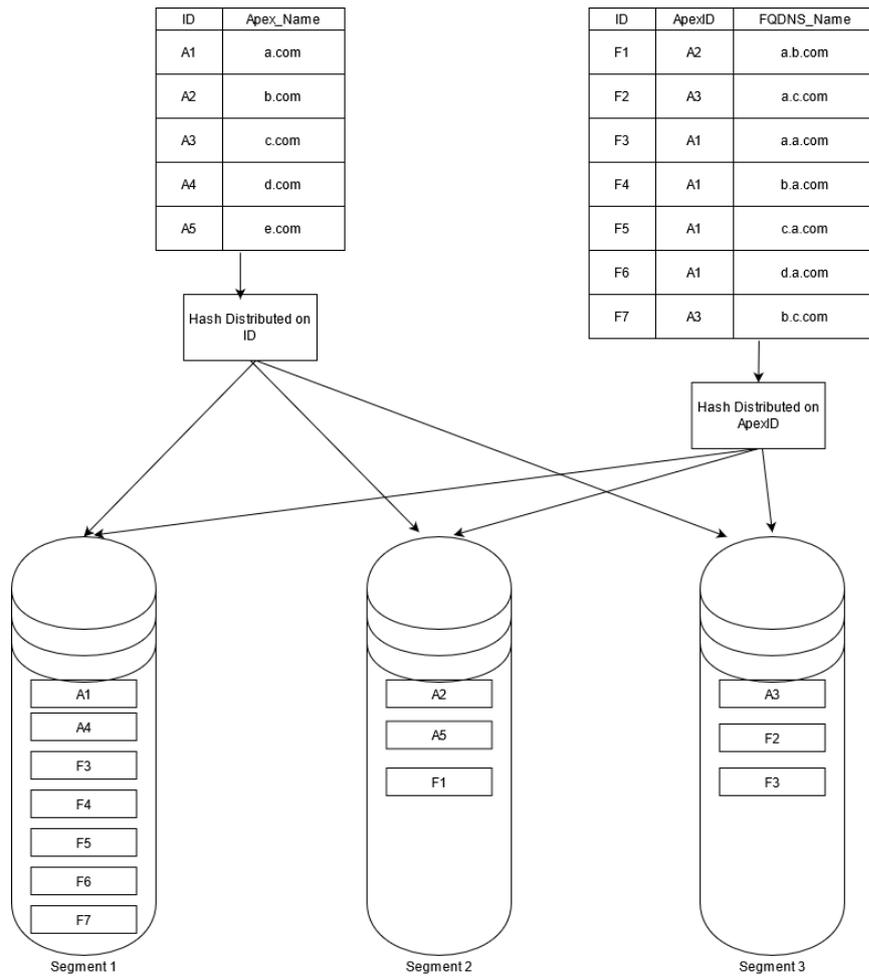


Figure 4.20: Sharding using APEX-ID (Hash distribution)

Figure 4.20 shows how data was sharded using APEX-ID (Hash distribution). When considering very large scanned domains such as Blogspot, one segment can grow exponentially. This method was quite effective with join operations, but the benefits for parallel queries were lost because of the non-uniform record distribution within the segments.

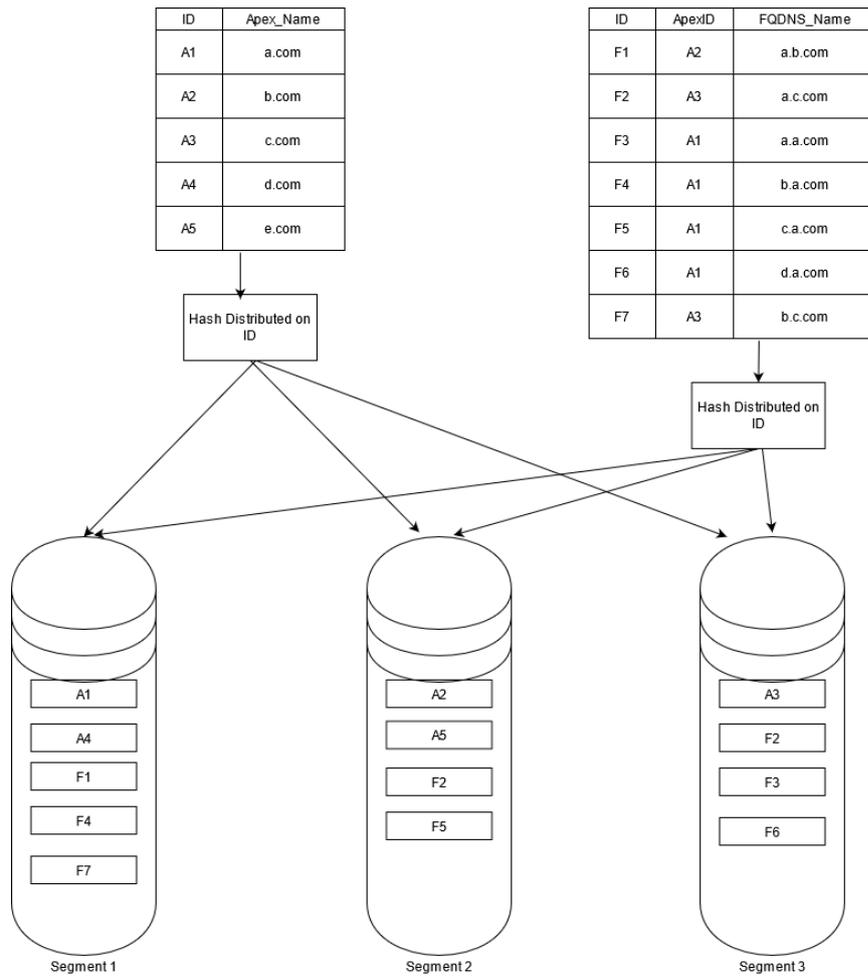


Figure 4.21: Sharding using FQDN-ID (Hash distribution)

Figure 4.21 demonstrates an alternative investigation done to overcome the above problem. Here the relations were shared using FQDN-ID and by using the same hash-based sharding strategy. Considering the fact that FQDN-ID is not the foreign key of the FQDNs relation, the situation where most records are stored in the same segment can be avoided. As a result, data are more likely to be uniformly distributed within the segments. However, this strategy also fails when running parallel queries since join operations are quite costly, which resulted in a large amount of unnecessary traffic between segments.

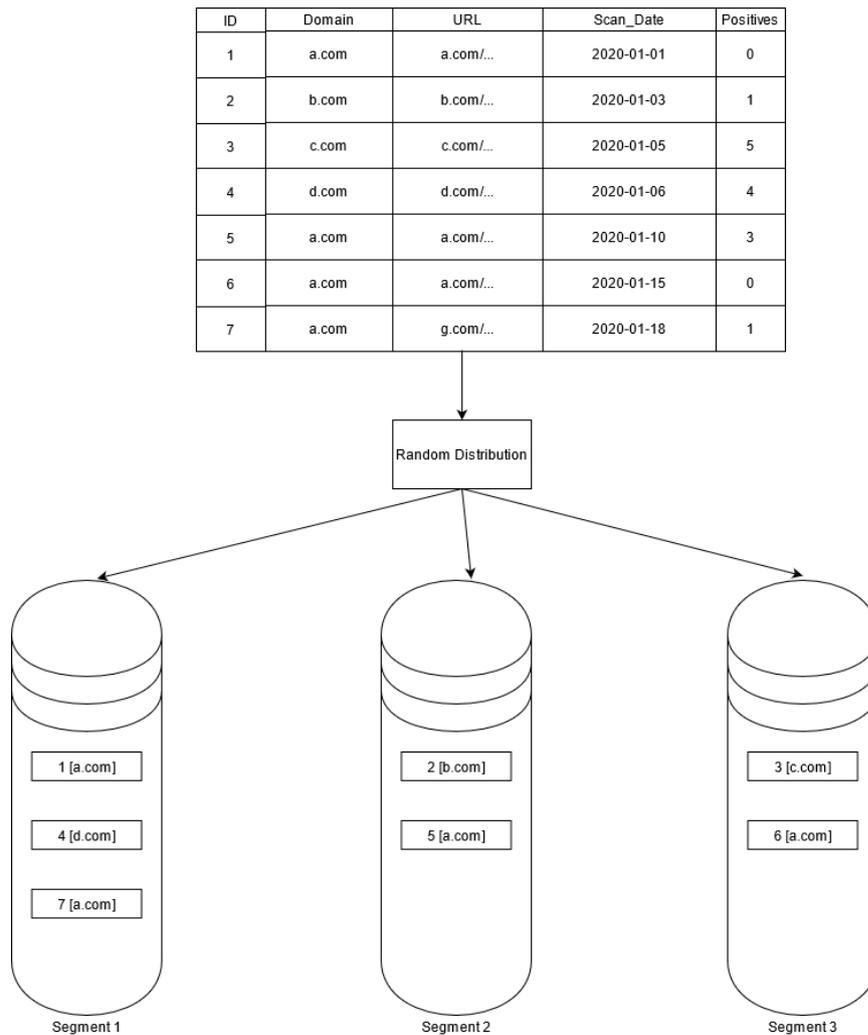


Figure 4.22: Sharding using FQDN-ID (Random distribution)

As shown in Figure 4.22, in the final design decision of VT profiling the normalized design has been omitted. Storage was traded for query performance. In this scenario, the relations were sharded by using a random sharding mechanism (round-robin manner) which resulted in data being distributed more uniformly. Since each data silo is monolithically captured within the segment, queries without any join operations can be executed in parallel on each segment.

Not Normalized design

In this design, the relations were designed based on the letters of the scanned URLs. The relations were created based on the first letter of the scanned URL and each directory was partitioned by the second letter of the scanned URL. This method was not visually the same as the conceptual model of profiling but consists of all the necessary semantics

of the conceptual model. Then experiments were done by using both random and hash-based sharding methods to shard the relations within the database instances in the cluster.

While experimenting with both sharding methods, it was observed that in hash-based sharding some shards were growing exponentially. The reason for this is when the URL was hashed and the relevant segment was selected, some special cases of the dataset were affected. As explained in the experiments (Table 4.5), some of the Apex/FQDNs had been scanned considerably more times than other Apex/FQDNs. In such cases hash-based sharding would lead to the creation of unbalanced segments with either a million records or very few hundred records.

Due to these unsatisfactory observations it was decided to use a random distribution method which runs on each segment in a round-robin manner. With this mechanism it was possible to reach the expected query execution results (Read and Write). The ER diagram in figure 4.19 shows the relations and its partitions. Aggregate queries could be avoided by trading extra space on the disk due to the monolithic nature.

4.3.2 Modeling PDNS data to graph

Graph modeling was an extremely tedious task, when considering the 310,092,177 DNS records that were received in the feed, since it had to undergo multiple preprocessing operations. Getting the DNS records of A, NS, MX, CNAME, and SOA, and finding the associated edge and vertex relationships and interest in the graph database cluster was a computationally heavy task. Apache Spark was used as the distributed computing framework to carry out this task. However, there were some challenges to overcome with respect to the big data environment. Due to the massive data volume, once the data was loaded to Spark's RDD, Java heap memory overflow issues were encountered. Furthermore, network issues were encountered where Spark was not able to transfer data within nodes because of the volume of RDDs. These problems were overcome by re-partitioning and some context declarations.

The next challenge in graph modeling was to save the time-series properties of the graph. The nodes and edges had to be time-stamped based on the first seen and last seen dates of the PDNS feed. None of the open-source graph databases supported the time-stamping properties itself. Community studies were done on Neo4j, OrientDB, and ArangoDB and it was realized that there is no explicit way to save the time-series

property of the graph. After completing the community studies and our own benchmark tests on the sample data and picking ArangoDB as the best candidate to store PDNS data, the next step was to figure out an explicit way to store the time-series properties of the nodes and edges.

Figure 4.25 shows how the time-stamping problem was overcome. New edges were added daily with a timestamp even if the edge already exists in the graph. This process costed some storage but the time-series property could be preserved. Later, ArangoDB Query Language (AQL) which is ArangoDB's own query language, was used to filter out the edges and nodes for a given time frame. Considering the time-stamping nodes, ArangoDB's inbuilt UPSERT operation was used to overcome this problem.



Figure 4.23: Timestamping graph edges

Graph sharding (Figure 4.24) was a remaining unsolved problem in the design. Arango DB suggests sharding the graph in a way that minimizes the edges between database instances. But due to the nature of the graph it was almost impossible to figure out the way to balance the graph. One experiment was to understand the nature of connected components and reinsert the data based on those findings. However, with random sharding, good performance could be achieved to retrieve queries under 10 seconds.

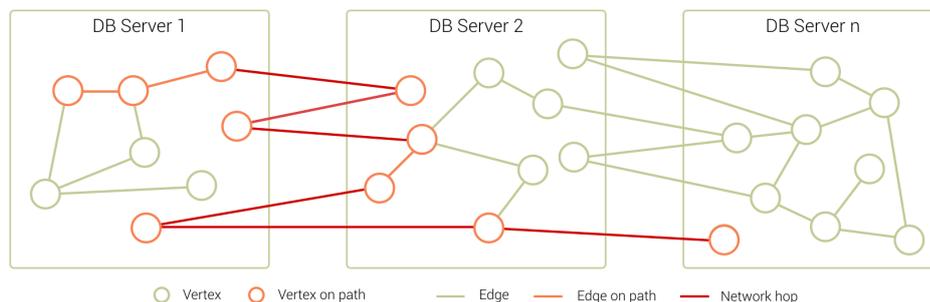


Figure 4.24: Graph sharding

4.3.3 Bootstrapping and daily insertions (Populating databases)

Populating databases was carried out in two steps: bootstrapping the databases with pre-collected data (each source has data collected for the duration of one year), and automating daily insertion process of data. Apache Spark was used for preprocessing and bootstrapping data from each source.

4.3.4 MaxMind Data Storage

MaxMind data has two formats: MaxMind's own file format (.mmdb), and Comma-separated values format (.csv). The initial attempt was to store the CSVs in the database as it preserves its time series properties. However, the IPs were compressed into groups by their subnets in the CSV format and it was difficult to determine the subnet that a given IP belongs to.

Experiments were carried out to obtain an idea about the subnet distribution (Figure 4.25), to investigate a method to map a given IP to the subnet. As seen in the figure it was hard to distinguish the subnet by just giving the IP. Therefore, two approaches were proposed: to expand all subnets (2 billion IPs per day) and keep repeating the data or to move forward with MaxMind's own file format. However, it became clear that the first approach of expanding subnets leads to a waste of storage space and reduction in performance.

As a result, MaxMind's own file format was chosen to store data in the database since it supports straightforward IP to subnet mapping and it was proposed to implement the middleware to query these files in a way that its time series properties are preserved.

4.3.5 Alexa Data Storage

Initial experiments on storing Alexa data were done using InfluxDB [27], which is an open source time series database. The performance of InfluxDB was tested with 100 days of Alexa Rank data (approximately 100 million records) and it was determined to be satisfactory. However, when the database was deployed in the cluster and data insertion was done, with increasing amounts of data various issues arose. Some of these issues were timeout errors during database operations, and the database getting frozen and automatically restarting during query execution.

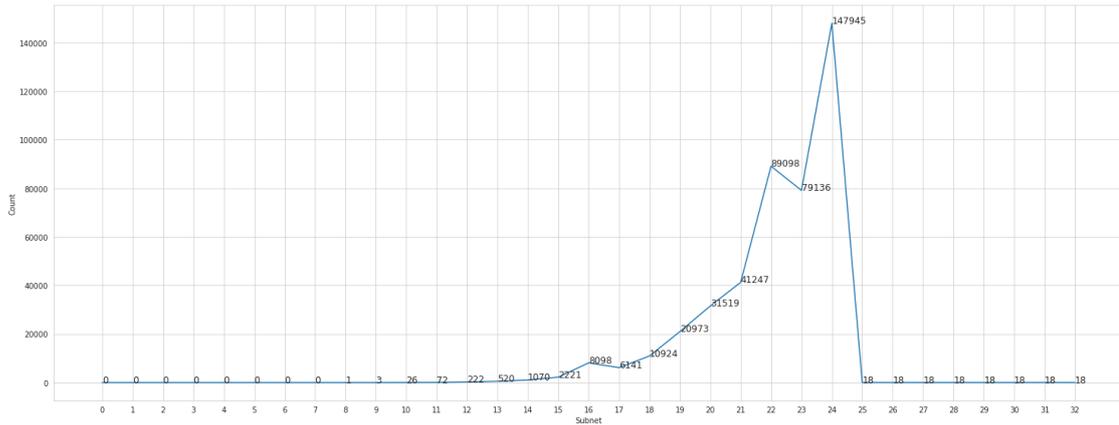


Figure 4.25: MaxMind's subnet distribution

This was an unexpected challenge since although community readings and benchmarks suggested that it was the better time series database, it was not suited for the purpose of our system due to its inability to handle extremely large amounts of data. Therefore, Greenplum DB was chosen to store Alexa Rank data as the alternative.

4.3.6 Designing a scalable architecture

With the volume of data that had to be dealt with daily, the time that could be spent to insert new data each day was quite limited. The data insertion bandwidth that needed to be achieved did not match with the current capabilities of our existing graph database cluster which led to the failure of the database cluster during data insertion. To avoid this failure more database coordinators had to be added to the Arango DB graph database cluster, which was a very resource intensive decision. However, even after the insertion of one week of data it was observed that the database cluster was not able to handle the volume of graph data that was been generated. As a technique to overcome this challenge, new VMs were added to the Kubernetes cluster and an entirely new ArangoDB graph database cluster was created on top of the newly added resources and our day granule graph was built on top of it, as shown in figure 4.26. This gave the need to add 16 new VMs each month and at the end of the period of a year a total of 216 VMs will be present in the Kubernetes cluster. Since the scale of the VMs that is used for this purpose is 8-16 core, 16-32GB RAM, it was quite an expensive design decision to take. Due to this, it was decided to gradually archive the previous year's data starting from the current date.

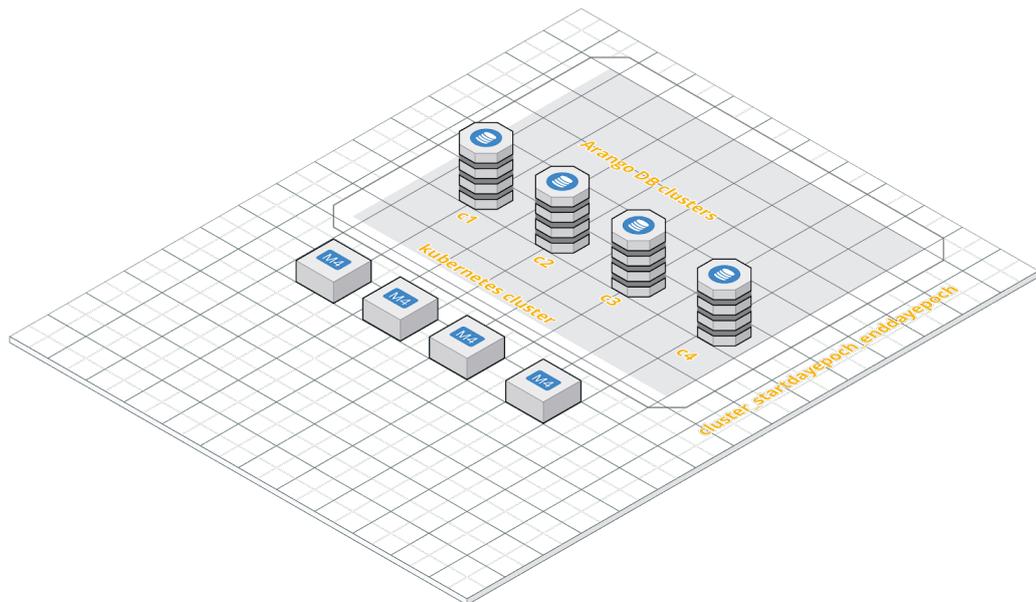


Figure 4.26: Resource intensive architecture design

Since this was an expensive and highly resource intensive design decision, the need for a less expensive solution to this problem was needed. After revising the system requirements, the time taken for the daily bootstrapping process of new data insertion was increased from 1 hour to 4 hours by slowing down the data insertion pipeline. This reduced the number of database coordinators required to handle the daily insertion process. Additionally, ArangoDB's default configuration was overwritten with some native configurations in order to enhance its buffer space. With this solution it was possible to have a single cluster that could hold one month of data and 4 new VMs does not need to be added each week. Instead, 4 new VMs could be added per month which was a 50% cost-cutting solution (Refer section 7.5)

Chapter 5

Implementation

This chapter discusses the implementation details of the project under three sections: implementing the data processing pipelines using Spark, implementing the middleware and implementing end consumer products (web application, Chrome extension and developer API).

5.1 Implementation of the infrastructure and database clusters

5.1.1 Implementing the Kubernetes cluster

A Kubernetes cluster was implemented using 16 VMs with 16 cores 32GB each in order to create the initial Kubernetes cluster for this project. Docker was used as the container management engine for Kubernetes. Considering the cluster provisioning, our own bash scripts were used to send SSH keys to nodes and the necessary commands were executed to install binaries in each node to create the cluster. Once the configuration was set in place for each node, one node was assigned as the Master node and 15 other nodes were assigned as Worker nodes in the cluster. Next, a pod network was installed on top of the cluster using Flannel [28]. In the final stage, the Kubernetes dashboard was set up and configured for real-time monitoring of the cluster and a couple of 25 terabyte network-attached storages were included to maintain the persistent volumes.

5.1.2 Implementing the ArangoDB cluster

An ArangoDB cluster with 5 agents, 5 coordinators and 15 database engines was implemented on top of the Kubernetes cluster.

5.1.3 Implementing the GreenplumDB cluster

Our own Kubernetes operator was created on top of the container image of Greenplum DB extracted from Pivotal Cloud and it was pulled into our container registry. 15 segments and 2 Master Greenplum DB clusters were implemented on top of the Kubernetes cluster using this Kubernetes operator.

5.2 Data processing pipelines using Spark

Two main types of pipelines can be found in the Spark data processing pipeline. One was implemented to bootstrap the historical data and the other was implemented for daily insertion of new data. The initial design and implementation of both types of pipelines are the same but the strategies followed in order to bootstrap the initial data were different.

5.2.1 Spark jobs for processing PDNS data

The Spark pipeline is designed in a way to work as five parallel Spark jobs, each for processing A, MX, NX, CNAME, and SOA records as indicated in figure 5.1]. Each job is automatically triggered by Airflow with new data for each day. Each pipe has three phases. Phase one cleanses and transforms the raw data into a structured form. Underneath these jobs there are a series of MapReduce operations which is executed to clean the data and do the transformation. In the phase two, the cleaned and transformed data are taken and synced with data from the daily feed of VirusTotal and it undergoes the labelling process. In the phase three, the data is taken and it undergoes the labelling process again with Alexa Rank data which adds more value and information to the raw data. Later this data is sent to be stored in the relevant databases.

Apart from the data in the process the initial structure of each of the six pipelines was the same. In abstract, the job was to take documents from PDNS records and transform

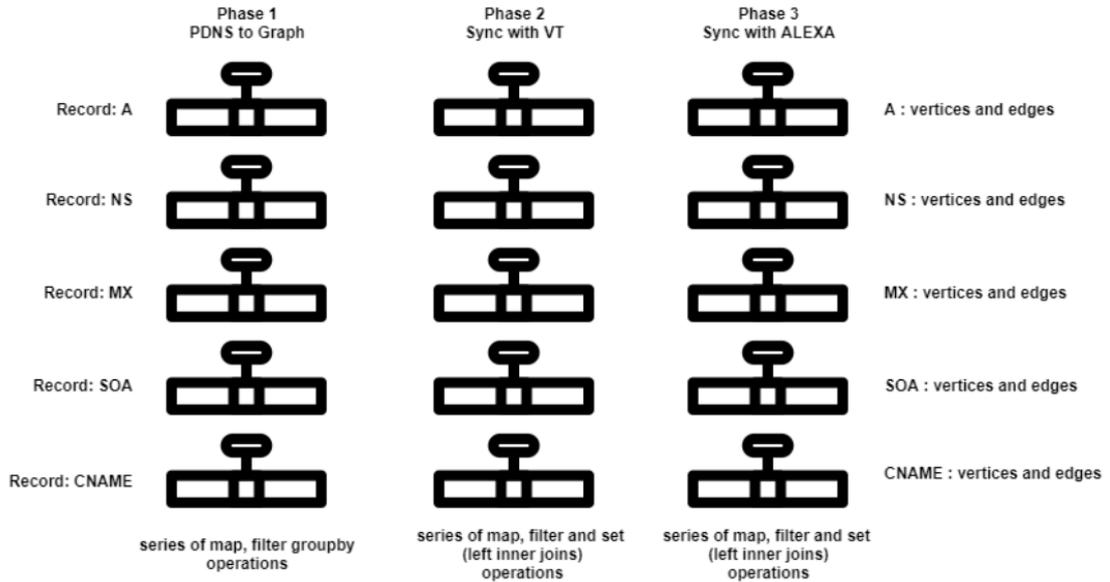


Figure 5.1: Spark data pipeline with 3 phases

the data to graph schema that ArangoDB graph database cluster can store. Considering the PDNS records, Type A records needed some slightly different design and implementation when compared to other record type such as NS, MX, SOA, and CNAME.

Processing of Type A records

The job for processing Type A records was the most critical processing job. This was due to the fact that Type A records contain much more transformative information which needs to be synced with other sources and inserted into the ArangoDB cluster. Figure 5.2 shows the row schema of a type A record and the nodes and edges that have been derived to store inside the graph database cluster using the record.

The first stage of the Spark job for processing Type A records was to remove the duplicate entries. Farsight has several sensors around the world to capture DNS traffic and as a result the feed may have multiple PDNS records for a single Apex/FQDN. Considering these duplicate records, we can not simply take one and ignore the others since those records may have newly exposed IP addresses as most of the domains use CDN nowadays. So the first stage was to take the PDNS records and load it to a RDD which is Spark's native data structure and do the group by operation by domain(Apex/FQDNs) and aggregate the rrddata array. This aggregation process of rrddata array is a union operation since we get all the exposed IPs relevant to that particular domain without having

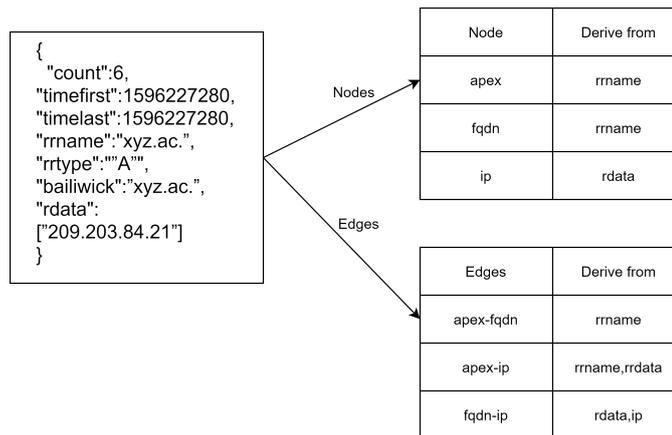


Figure 5.2: Converting Type A record to graph nodes and edges

any duplicate IPs. The following code snippet shows how the group by operation was done in the Spark job.

```

apex = DF.filter(lambda x:x.TYPE=='apex')
apex = spark.createDataFrame(apex)
apex = apex.groupBy('apex')
    .agg(F.collect_list('ip'),F.collect_list('count'),
    F.collect_list('time_first'),F.collect_list('time_last'),
    F.collect_list('rrtype'),F.collect_list('bailiwick'))
apex = apex.rdd.map(apex_vertex_edges)

```

The next stage was the Spark job, which divides the aggregated data into nodes and edges as it is needed to be stored. This is also a MapReduce operation as given in the following code snippet.

```

#node trasformation
return {
    "apex_vertex":{
        "_key": x.apex,
        "_id": "apex/{}".format(x.apex),
        "rrtype": "A1",
        "type":1
    },
    #edge trasformation
    "edges":{

```

```

    "apex_ip": combine_apex_ip(
x['collect_list(ip)']
,x['collect_list(time_first)']
,x['collect_list(time_last)'],x.apex)
}
}

```

Next the pipeline runs a series of left join operations with VirusTotal data and Alexa Rank data to insert knowledge information to the relevant domain (Apex/FQDNs).

```

#Alexa
JOIN_APEX = JOIN_APEX.join(
alex_a_day,
JOIN_APEX._key==alex_a_day.domain_alex_a,
how='left')
#VT
JOIN_APEX = apex.join(VT_APEX, apex._key==VT_APEX.apex, how='left')

```

After syncing with the knowledge data, the Spark pipeline flushes the data to relevant database clusters with relevant timestamps. The above code snippet simply shows the core parts of a complex Spark job design.

Processing of records of Type NS, MX, SOA, and CNAME

Design and implementation of the processing jobs for NS, MX, SOA, and CNAME records are also similar to the design and implementation of the job for A records. The only difference is the number of derivations is lesser than that of the A records as indicated in figure 5.3. Similar to the processing of A records, the jobs for all these record types also run as a series of groupBy operations to remove the redundancy and extract full information. It is followed by a MapReduce operation to transform the document form to nodes and edges. Next, set operations are carried out to sync the data with other data from knowledge sources and then the data is flushed to the relevant database clusters with timestamps of the data.

The major difference and critical part of the Spark job for processing NS, MX, CNAME, and SOA records is that the design has one extra stage of MapReduce and

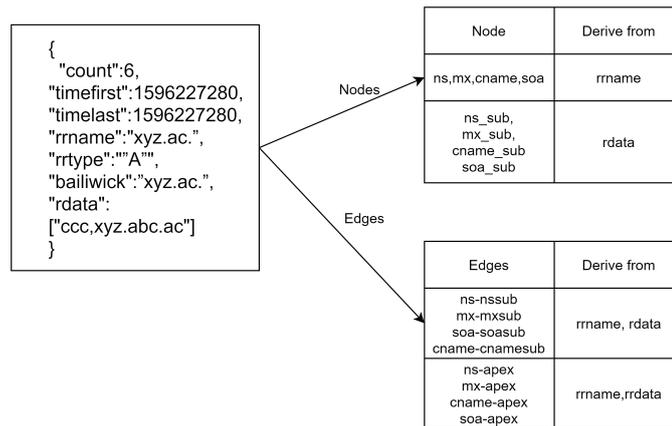


Figure 5.3: Converting Type NS, MX, SOA, and CNAME records to graph nodes and edges

set operation (left join operation within RDD) to define the relationship of NS, MX, CNAME, and SOA and previously processed A records.

5.2.2 Spark jobs for processing VT data

Similar to the Spark job for processing PDNS data, Spark jobs for processing VT data are set to trigger automatically with new data by Airflow. This job prunes the raw JSON output and transforms it into the database schema of the Greenplum DB as shown in figure 5.4.

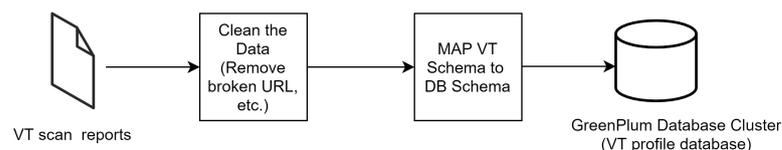


Figure 5.4: Spark jobs for processing VT data

5.2.3 Spark jobs for processing Alexa Rank data

The data cleansing and transformation process for Alexa Rank data is exactly similar to the data processing of VT data as shown in figure 5.5. The only difference is the schema of raw Alexa Rank data and database schema that is used to store the data.

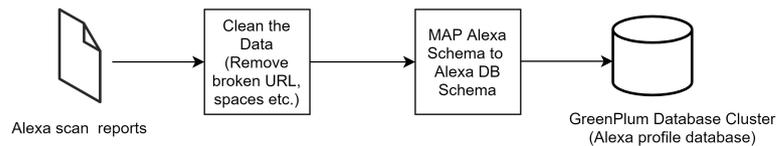


Figure 5.5: Alexa cleansing and transformation process

5.2.4 Bootstrapping pipelines and daily pipelines

Designed Spark jobs has been used for two different tasks. One is for bootstrapping the historical data initially and the other is to insert the daily feed to the database clusters. The way each job has been structured in order to build the overall pipeline has some differences. Initial bootstrapping jobs run parallel replicates of each job at the same time for different data portions and the daily insertion job was automated using Airflow with one replica of each job.

5.3 Middleware

This section explores the detailed architecture we have used in building the middleware using Microservices. The purpose of microservices is to solve the issues encountered when working with monolithic architectures.

Monolithic architectures have been the main approach to software design for a long time and still today, a large number of applications run through a monolithic approach. In software engineering, a monolithic application describes a single-tiered software application in which the user interface and data access code are combined into a single program from a single platform. Monolithic architecture has some considerable drawbacks which is why we have moved to the microservices architecture as shown in figure 5.6.

The microservices architecture has appeared lately as a new paradigm for programming applications by means of the composition of small services, each running its own processes and communicating via light-weight mechanisms. This approach has been built on the concepts of Service-Oriented Architecture (SOA) brought from crossing-boundaries workflows to the application level and into the applications architectures.

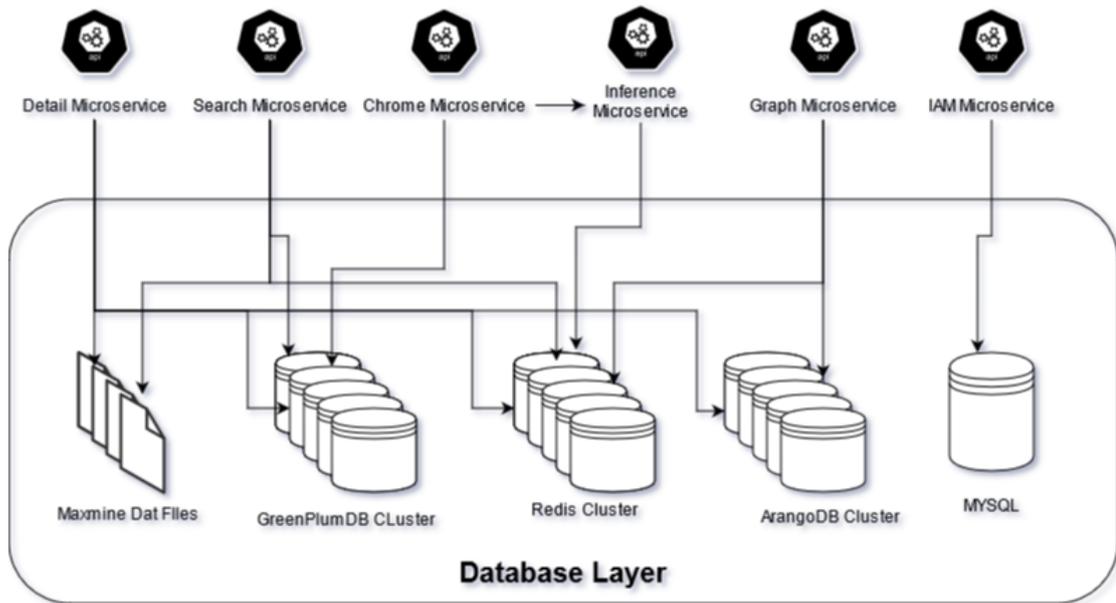


Figure 5.6: Microservices Architecture

5.3.1 IAM Microservice

IAM Microservice is used to manage user access and identity in the system. It is implemented using Echo, a high performance, extensible, and minimalist Go (Golang) web framework, and the data storage used to implement the service is MySQL. Figure 5.7 gives an overview of the endpoints implemented for identity and access management.

POST	<code>/signin</code> signin to the system
POST	<code>/signup</code> register to the system
POST	<code>/sendVerifyemail</code> send email verification
GET	<code>/verifyemail</code> Make the user verified.
POST	<code>/forgotpassword</code> send password reset link.
POST	<code>/resetpassword</code> Update the password.

Figure 5.7: Identity and access management endpoints

Users can register with the system by using the /signup endpoint. The endpoint validates the data (unique constraints are applied for the username) and insert them to the users table. Then the user will receive an email to confirm their email address provided when registering. The state of the validation is kept in the database and users will not be able to get access to the system until their email address is confirmed. Once it is confirmed, the users get the access token to access the system using the /signin endpoint by providing valid credentials that were given by the user during the registration. In case they forgot their password, users get the chance to get the reset password link to their email using the /forgotpassword endpoint by providing a valid email address. The users will get an endpoint to reset the password which includes a unique token for each user which validates that before resetting the password of the user. Figure 5.8 gives an overview of how the authentication works in the system.

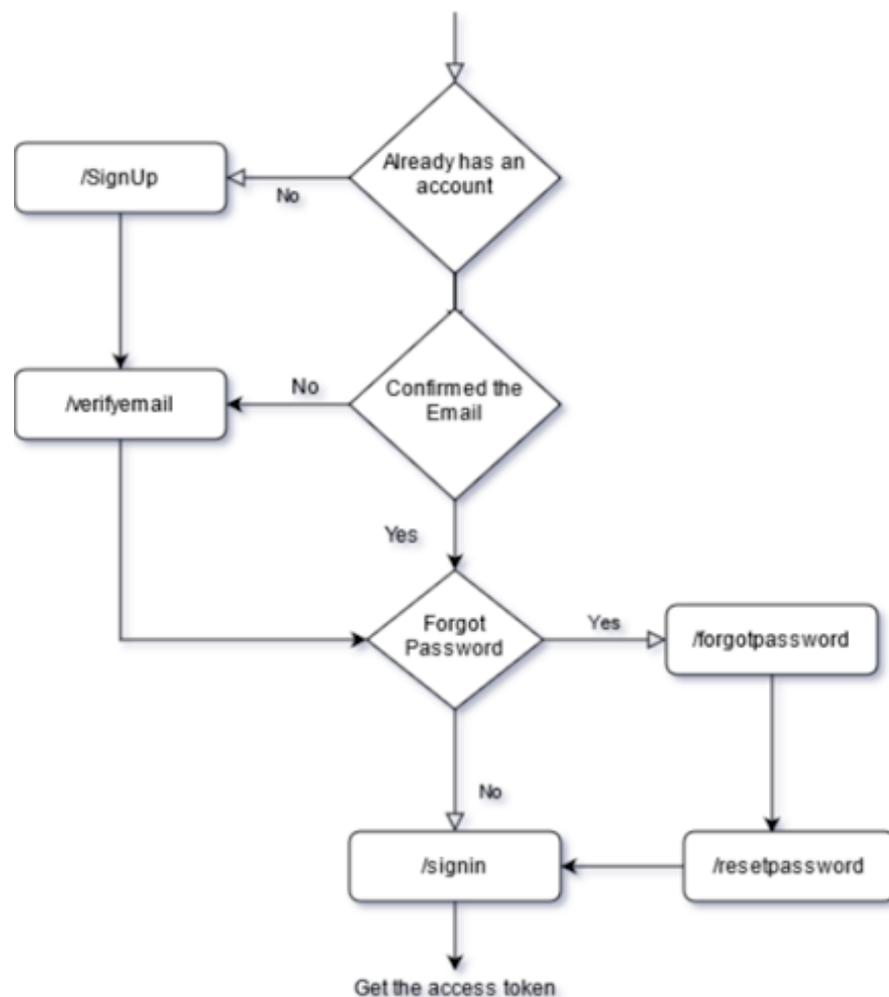


Figure 5.8: Flowchart for Authentication

5.3.2 IAM Authentication

When users sign up to the system a unique token is created for each user. That token is 37 characters long and that unique string is used to confirm the email address of the user.

When sending the password reset links it first creates a token and updates the password reset table in the database with the token and sends the link to the users email address. When resetting the password the system always checks the token in the database to validate the request.

5.3.3 Search microservice

Search Microservice is used to search any keyword and get a list of suggestions that matches the keyword, which includes some filters to facilitate the filtering of data. This was implemented using Python and Flask since MaxMind .dat files have drivers only for Python language. Given below is the endpoint implemented for the search. (Figure 5.9)

- Term - The keyword used to fetch the results
- Page - Only 10 results per page will be retrieved therefore need to include the page number
- Filter - The factor that is used to filter the data (Alexa Rank, VT, firstSeen and lastSeen)
- Order - the sorting order (either ASC or DESC)
- Type - Type of the nodes (Apex, FQDN, Name Server, Mail Server, SOA, CNAME or all)

GET	<code>/search/<term>/<page>/<filter>/<order>/<type></code>	Retrieve the data of the selected node for past week
------------	--	--

Figure 5.9: Search endpoints

As explained in the data modelling section, ArangoDB consists of nodes in different collections which is used to create the graph. When the user searches the data for the first

time the queries are executed in each collection parallelly and the data is retrieved using a fulltext search and it is combined together as depicted in figure 5.10. Then retrieved data is stored in the Redis cluster and the filters are applied to get 10 records that need to be sent as the response and the MaxMind ASN and location information is bound to these 10 records and the response is sent back. This data is stored in the Redis cluster for 1 day. That is, whenever someone searches the same keyword or the url the system first checks whether the result is already available in the redis cluster or not. If it is, then the data will be retrieved from the Redis cluster and the filters will be applied. If not, the queries will execute parallelly to retrieve the data more efficiently.

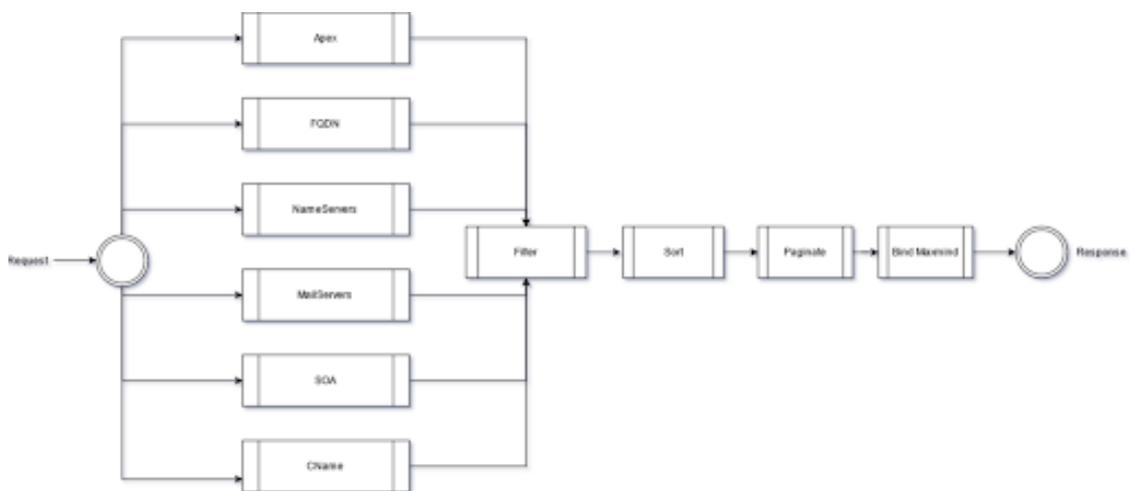


Figure 5.10: Component diagram for search

5.3.4 Graph microservice

Graph Microservice is used to retrieve the nodes and edges of graph traversal for a given depth and node as well as to retrieve the last 7 days graph traversal of the given node. This microservice includes 2 endpoints as /graph and the /timeline as shown in figure 5.11.

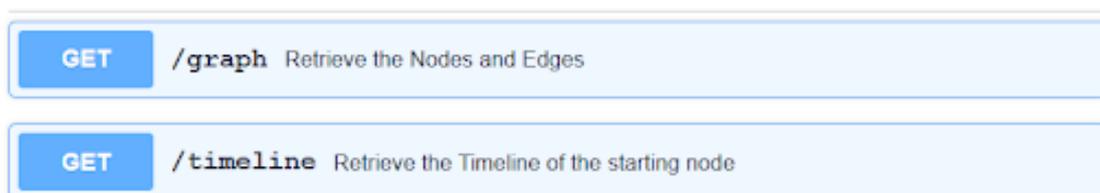
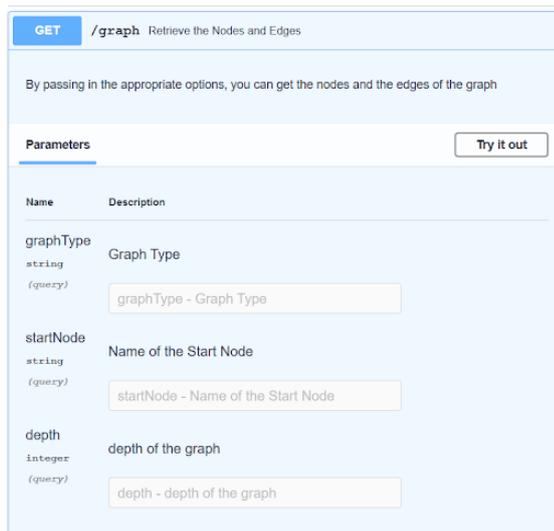


Figure 5.11: Graph endpoints

/graph

This endpoint (figure:5.12) is used to retrieve the graph for the current date. It requires three query parameters in order to carry out the graph traversal.

- GraphType - Whether the starting node is an Apex, FQDN,CNAME, SOA, Name Server or Mail Server
- Startnode - key of the starting node
- Depth - the depth for the graph traversal



GET /graph Retrieve the Nodes and Edges

By passing in the appropriate options, you can get the nodes and the edges of the graph

Parameters Try it out

Name	Description
graphType string (query)	Graph Type
startNode string (query)	Name of the Start Node
depth integer (query)	depth of the graph

Figure 5.12: /graph endpoints

When the request comes in the graph traversal will execute in two parallel processes to fetch the Edges and the Nodes (figure:5.13). Then the data combines together and it is sent as the response. The Color codes are binded to the node data during the data processing. The data is stored in the Redis cluster once the graph is traversed. Since the depth is passed as a query parameter, we have the ability to use this same endpoint as the expand endpoint by passing the depth of 1 to the endpoint. So the endpoint acts as the graph traversal endpoint and also used to expand the graph in the frontend.

For the parallel execution we have used 'goroutine' which is a lightweight thread of execution. When using goroutine the function which is invoked using goroutine will execute concurrently. And then it uses wait groups to wait till all the processes to end and then sends back the response.

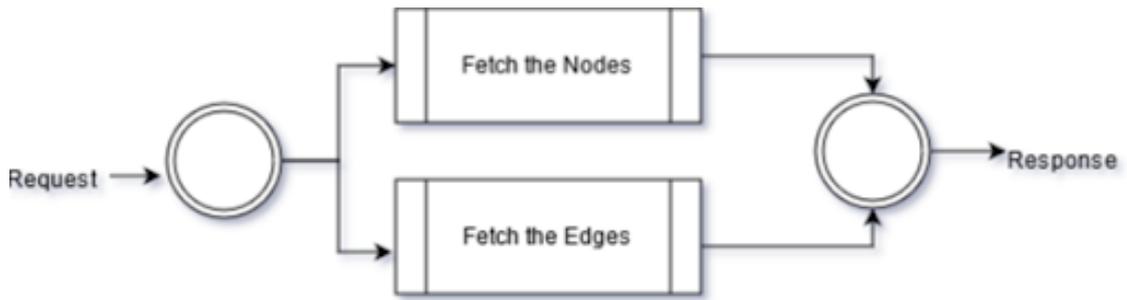


Figure 5.13: Component diagram of graph

/timeline

This endpoint (figure 5.14) is used to retrieve the graph for the last seven days. It requires three query parameters similar /graph in order to carry out the graph traversal.

- GraphType - Whether the starting node is an Apex, FQDN,CNAME, SOA, Name Server or Mail Server
- Startnode - key of the starting node
- Depth - the depth for the graph traversal

Name	Description
graphType string (query)	Graph Type
startNode string (query)	Name of the Start Node
depth integer (query)	depth of the graph

Figure 5.14: /timeline endpoints

This works the same as in the /graph endpoint but the only difference is that this will create 2 processes parallel for each day. Therefore, altogether 14 processes are created to fetch the data for the past 7 days. Figure:5.15 illustrates how the 14 processes are made. Similar to above, goroutines are used here for the processes along with wait groups.

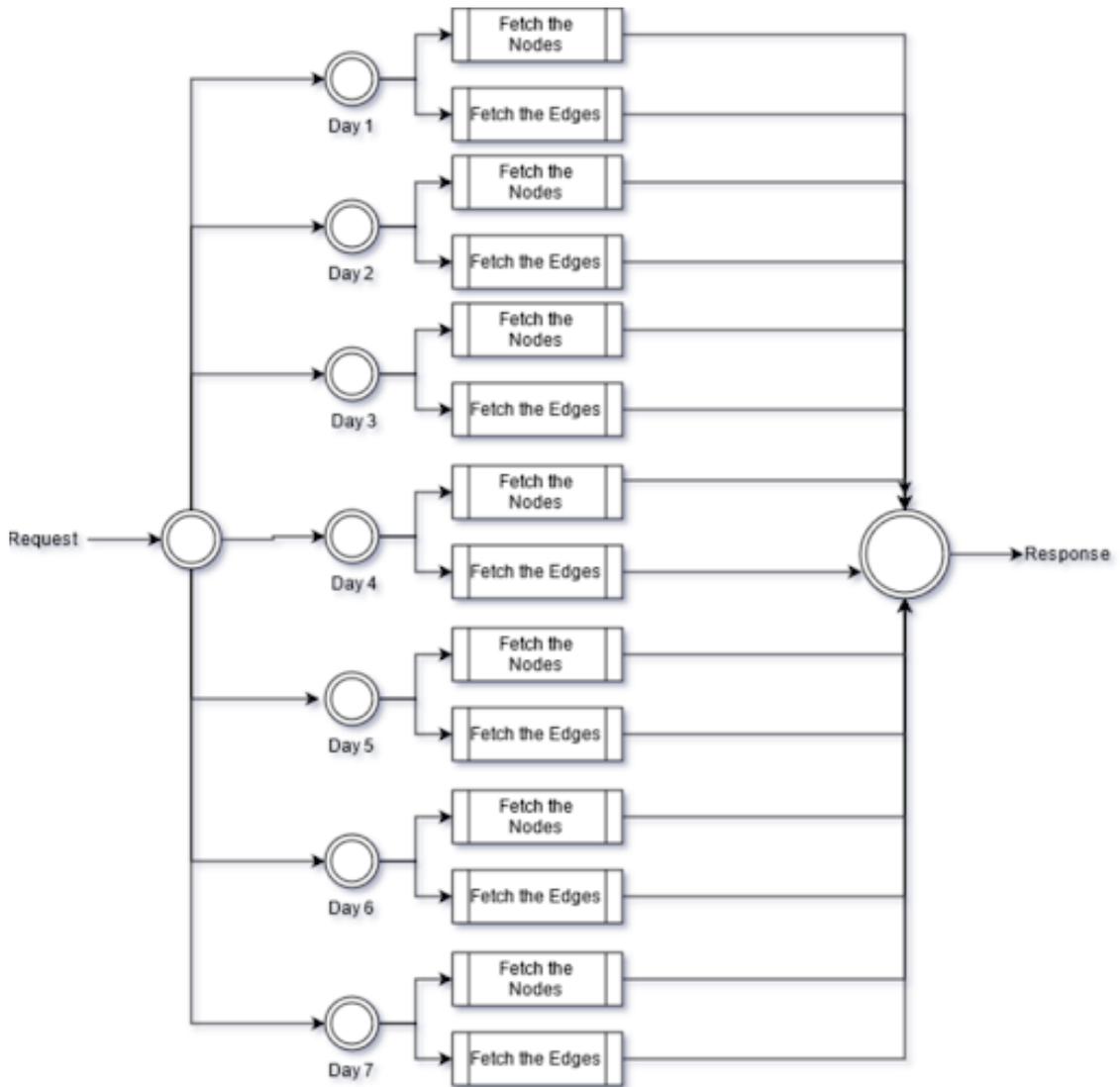


Figure 5.15: Component diagram of timeline

5.3.5 Inference microservice

Inference microservice is the component which is used to get the inference output of the graph. Relevant implementation of belief propagation (Refer section 2.1.6) is exact to the variation of the paper. Belief propagation process can be encapsulated to the following flow as shown in figure 5.16.



Figure 5.16: Belief propagation process

Purging process removes the IP address that belongs to public clouds, web hosting

and CDN networks since the system designed in the works of Khalil et al does not work with public IP addresses. Then filtered graph undergoes the labelling process, and the inference algorithm is run and the inference value is taken and reconstruct the heterogeneous graph and return it.[figure:5.16]

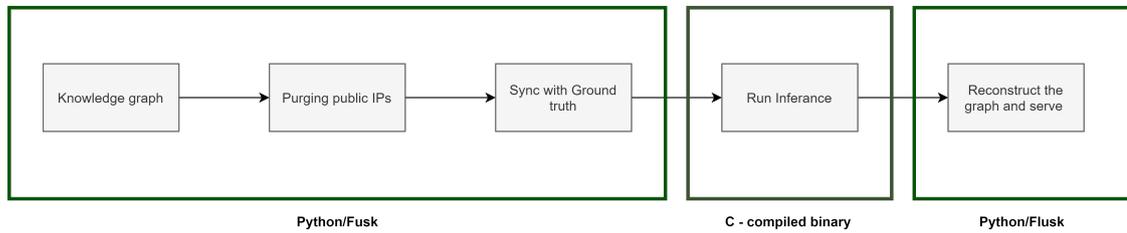


Figure 5.17: Programming languages in implementation

Being more specific to the implementation (figure 5.17), two main programming languages were used to implement this service. The core of belief propagation has been implemented using C language since the execution time is a critical factor for the algorithm. Apart from that, extracting the graph, syncing with ground truth data, and reconstructing and serving the results has been implemented using Python and Flask framework.

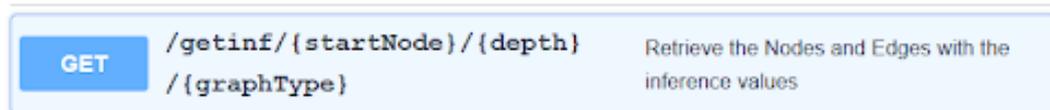


Figure 5.18: Inference endpoints

Python OS module was used to load the pre-compiled C binaries and pass the graph to the inference algorithm and take the outputs and return it to the API. Figure 5.18 shows the API endpoint for retrieving the inference output.

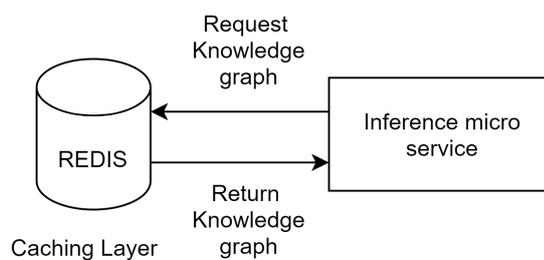


Figure 5.19: Inference microservice and caching layer

In this implementation we are not directly extracting graphs to run the inference algorithm from the ArangoDB cluster. Since time and performance are critical factors it has been taken from the caching layer as seen in figure 5.19.

As shown figure 5.20 once a user invokes the knowledge graph, it first checks if it's on our cache. If not it goes and searches for the relevant data on ArangoDB and serves the graph. While serving the knowledge graph, it is cached in the Redis caching layer. Later when user invoke the inference output, inference system directly invoke the knowledge graph from Redis cluster and execute its process and return the output via HTTP endpoint. Our Entire system design is designed on pluggable architecture as here we plug the inference intelligence. Any graph based intelligence can be planted as a scalable microservice implementation to the system.

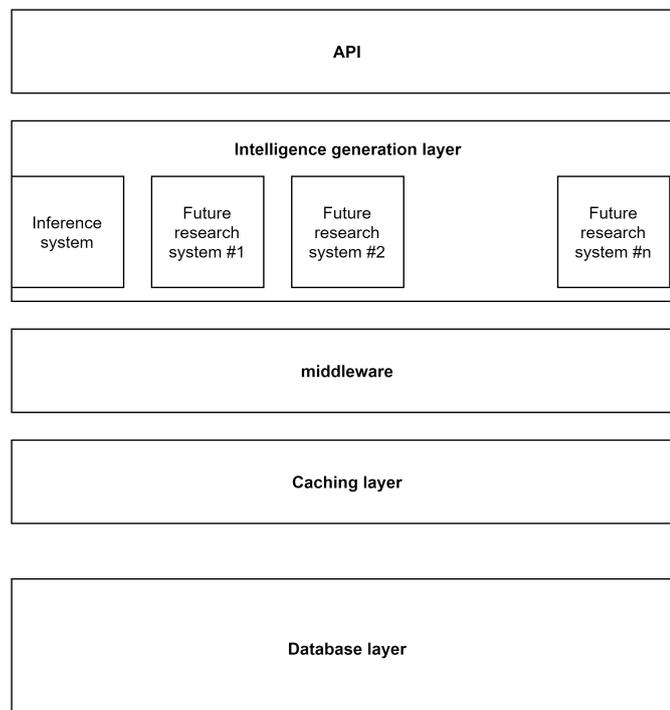
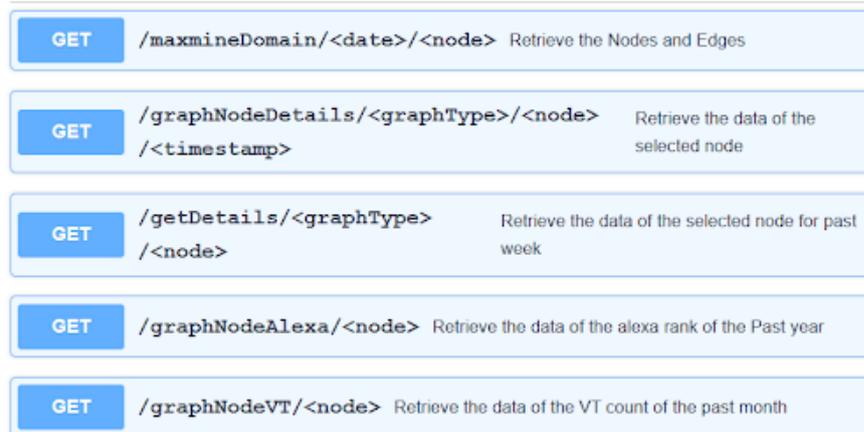


Figure 5.20: pluggable architecture

5.3.6 Detail microservice

Detail Microservice [figure:5.21] is used to retrieve the Alexa Rank history, VirusTotal history and MaxMind data for the given domains. This was implemented using the Python and the Flask since the MaxMind .dat files have drivers only for the Python language. Given in figure 5.21 is the endpoint that can be used to retrieve the data.



GET	<code>/maxmineDomain/<date>/<node></code>	Retrieve the Nodes and Edges
GET	<code>/graphNodeDetails/<graphType>/<node>/<timestamp></code>	Retrieve the data of the selected node
GET	<code>/getDetails/<graphType>/<node></code>	Retrieve the data of the selected node for past week
GET	<code>/graphNodeAlexa/<node></code>	Retrieve the data of the alexa rank of the Past year
GET	<code>/graphNodeVT/<node></code>	Retrieve the data of the VT count of the past month

Figure 5.21: Details endpoints

5.3.7 Monitoring and matrices

In big data systems one of the most critical requirements is maintaining the availability. Since a number of computational tasks are running in both background and foreground there's a possibility of causing a failure. Also, responding to the consumer end requests can cause huge traffic within the middleware and database segments.

Therefore, monitoring the health of the entire system is essential in designing a big data system. For that, we have defined three levels of monitoring matrices.

The first level of monitoring is designed to monitor the big data pipeline. As shown in figure 5.22 we have full control of how the Apache Spark cluster works and the jobs that are running.

The second level is designed to monitor the entire cGraph infrastructure where we have run all database clusters, microservices, web servers, IAM servers, etc. The entire infrastructure was designed on the Kubernetes cluster. Figure 5.23 shows how the Kubernetes monitoring system has been configured to monitor the infrastructure and its failures.

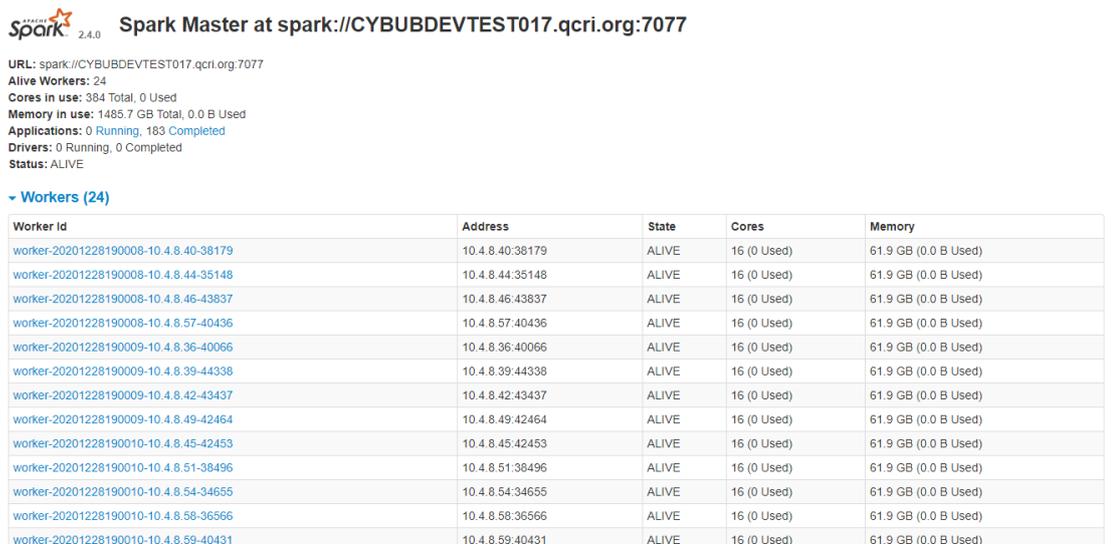


Figure 5.22: Spark Monitoring

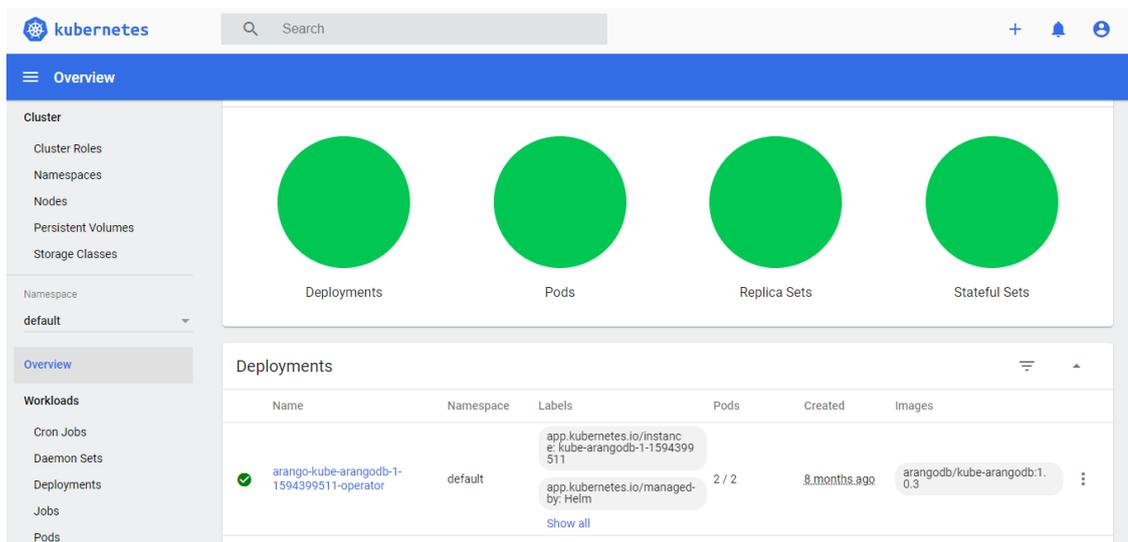


Figure 5.23: Kubernetes Monitoring

The third level is the API monitoring which is depicted in figure 5.24. This level of monitoring was not a straightforward development similar early stages since we have used a single instance Influx time-series database to store the metadata generated from KrakenD API gateway and Grafana to visualize the matrices.

5.3.8 Deployment strategies

The continuous delivery strategy was used to add new features to the system. For that, GitHub actions (figure 5.25) was used to take the new commit on the deployment branch

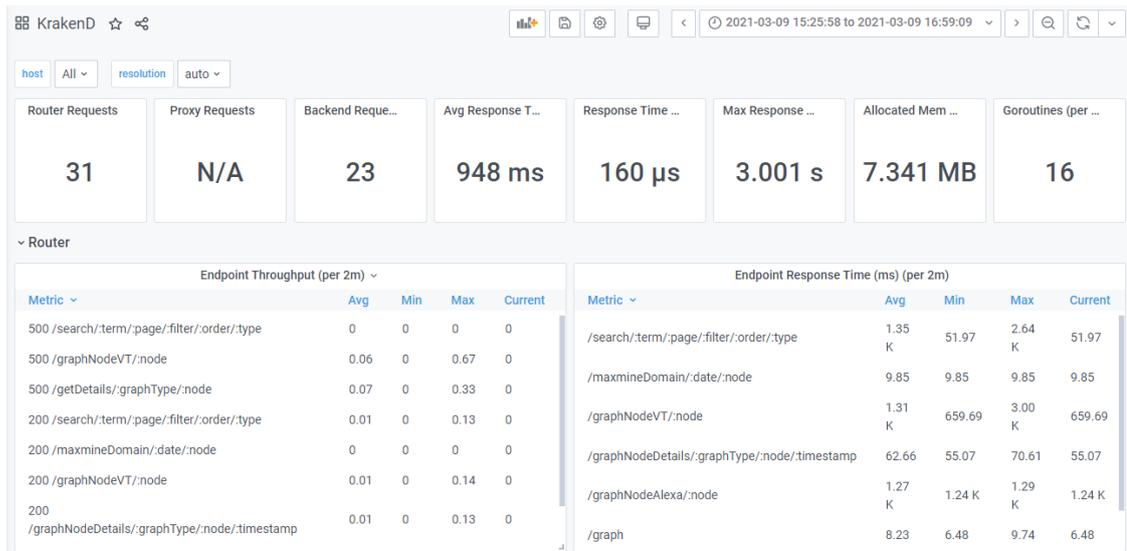


Figure 5.24: API monitoring

and build the docker image and push it to the container registry. Then Kubernetes pulls the image for its new deployment and initiate it on the Kubernetes cluster. This semi-automated process makes it much easier to add new features to the system.

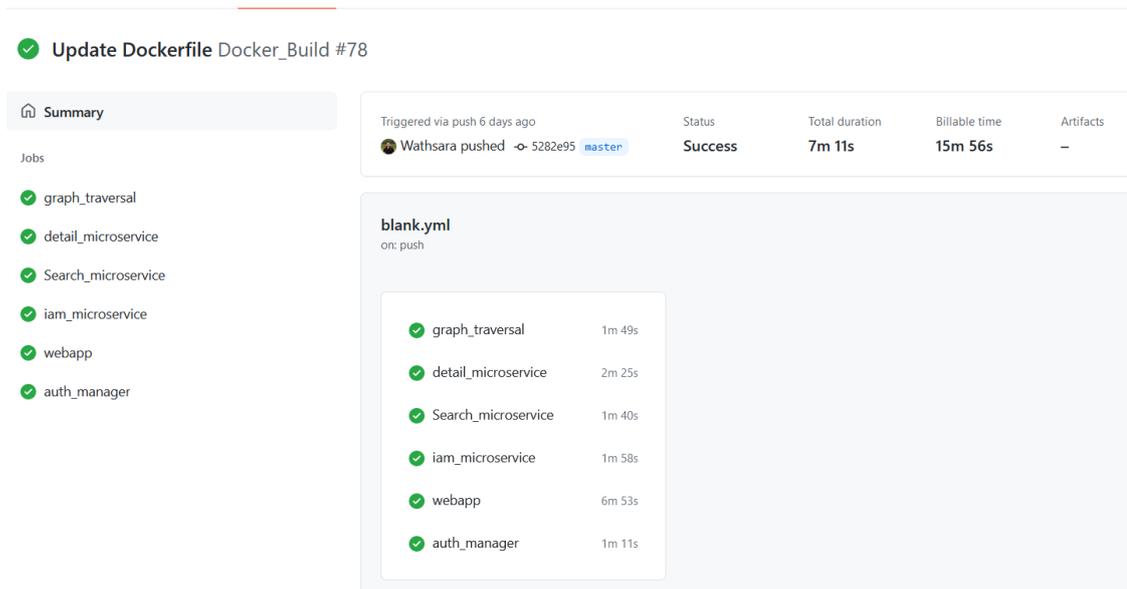


Figure 5.25: Github Actions Page

5.4 End consumer products

5.4.1 Web Application

A single-page application (SPA) is a more modern approach to app development. SPA is a web application in which content is loaded dynamically without needing to reload the page. A multiple page application (MPA), on the other hand, is considered a more classical approach to app development. The multi-page design pattern requires a page reload every time the content changes. By considering the advantages of SPA, the front end was designed as a single page application. The communication between frontend and back end happens via REST API calls.

React JS is a front end development framework developed and maintained by Facebook which runs on NodeJs server. Node Package Manager (NPM) is used to install and maintain the packages. Webpack, Babel, React hot loader, and Nodemon are some necessary tools used for React development.

Ant Design specification was used to develop a React UI library that contains a set of high quality components and demos for building rich, interactive user interfaces. It is compatible with browsers such as Edge, Chrome, Safari, Opera, Firefox and etc. For graph rendering in the web application we are using Graphin which is a big data visualization package under Ant Vision which was built specially for React using the G6 library in Ant Vision.

Maintaining the application state is difficult in a single page application. Redux JS is used to maintain state in SPA. Redux is a predictable state container for JavaScript apps. The state is stored in the Redux Store. It helps to write applications that behave consistently, run in different environments (client, server, and native), and is easy to test. We have used redux-logger along with Redux which made it easy for debugging and identifying issues especially when working with Axios to manage REST API calls.

We have not used a single template from outside and all the CSS and components are implemented and designed by ourselves.

Using Redux

The whole global state of the application is stored in an object tree inside a single store. The only way to change the state tree is to create an action, an object describing what happened, and dispatch it to the store. To specify how a state gets updated in response to an action, we have written reducer functions that calculate a new state based on the old state and the action.

Decribed below is how Redux works with an example from our System. Assume that the user enters a keyword in the search box and press submit.

- Once the user presses the submit button, the Search function dispatches three Redux Actions.
- First SEARCH_START action is dispatched. The action modifies the Redux state of the search state as loading true, data to null and the error null

```
searchData:{
  loading: true,
  error: null,
  data: null
}
```

- This loading true state is used to show the loading state of the search screen.
- The frontend sends the GET request to the search endpoint with the relevant path parameters. Then the backend retrieves the relevant data and sends it to the frontend
- When the front end receives the successful response the SEARCH_SUCESS action is dispatched. This action modifies the loading to false, data to payload and error to null.

```
searchData:{
  loading: false,
  error: null,
  data: payload
}
```

- Then the search page used the data in the state to show them in the front page.
- If somehow the response failed for a reason the `SEARCH_FAIL` action is dispatched. This action updates the loading to false, data to null and error to payload.

```
searchData:{  
  loading: false,  
  error: payload,  
  data: null  
}
```

- This error object is used and shows an error notification in the front end.

Structure of the front end

The application is structured and refactored to make the maintenance easy. The accepted React-Redux application directory structure is used in the front end.

- `components/` - All the UI components are implemented in this directory.
- `store/actions/` - All the Redux actions are defined in this directory.
- `store/reducers/` - The reducers which binds Redux actions with Redux state are defined in this directory.
- `services/` - Used to manage the localStorage and the authentication.
- `assets/` - All icons, images and other assets are included in here
- `components/[component]/style.less` - SCSS style files are structured in this form.
- `index.js` - This file serves as the entry point to the application.

Authentication

The users need to have an account to access some of the routes in the application. We have implemented an Authentication service to protect these routes. Whenever user tries to go to a protected route the service looking for the token in the localStorage and validate the token and if only a valid token is present in the store the application lets the

edges at a single time since it performs well without freezing the browser. It even increases the user experience and the usability of the system. This implementation details of this process is explained in detail below.

- First when the user goes to the graph views an action named `GET_NETWORK_GRAPH_START` is dispatched. The actions request the nodes and the edges from the backend via a REST API call using Axios and once the action receives a successful response the system dispatches an action named `GET_NETWORK_GRAPH_SUCESS`.
- `GET_NETWORK_GRAPH_SUCESS` action gets the payload and updates the `networkGraphData` objects `finalNodes` and the `finalEdges` with the payload and sets the nodes and edges to empty arrays.

```
case actions.GET_NETWORK_GRAPH_SUCESS:
  return{
    ...state
    networkGraphData: {
      nodes: [],
      edges: [],
      finalNodes: payload["nodes"],
      finalEdges: payload["links"],
      loading: false,
      error: false,
      expanding: false,
      expandedCout: null,
      messageShow: false
    }
  };
```

- Then `GET_NETWORK_GRAPH_SUCESS` action dispatches another action called `GET_EXTRACT_NETWORK_GRAPH_START`.
- This actions get the states of the `networkGraphData` and perform a small function to extract 10 edges (if there are more than 10) and get the nodes and edges relevant to them. Then the same function creates a payload which includes nodes

and edges that were taken out and nodes and edges that are remaining taken out. And then the action updates the networkgraphData states by dispatching GET_EXTRACT_NETWORK_GRAPH_SUCESS action.

```
case actions.GET_EXTRACT_NETWORK_GRAPH_START:
  return{
    ...state
    networkGraphData: {
      loading: false,
      nodes: payload["nodes"],
      edges: payload["edges"],
      finalNode: payload["finalNodes"],
      finalEdges: payload["finalEdges"],
      error: null
    }
  };
```

- We are using nodes and edges to render the graph and finalNodes and finalEdges includes the remaining nodes and edges to render.
- Same action dispatches when the View More button is clicked in the frontend. It dispatches the GET_EXTRACT_NETWORK_GRAPH_START action passing the key of node ID. Then 10 edges and nodes relevant to it are fetched and concatenated to the networkGraphData nodes and edges and removed from the finalNodes and finalEdges by dispatching the GET_EXTRACT_NETWORK_GRAPH_SUCESS action.

5.4.2 Chrome Extension

This is a plugin that can be installed into Chrome. Using the extension the users could investigate whether the domain they are currently accessing is benign or malicious based on the inference values. The extension is fed with an API called the Chrome API which was developed using Flask. This API calls the previously implemented Graph API and the Inference API to retrieve the inference value of the domain that the user is currently accessing. The Chrome extension passes the current domain to the endpoint and retrieves

the inference values to the Chrome extension. Based on the inference value the status of the domain is shown in the browser extension. Figure 7.2 gives a situation where the domain has a probability for being benign while figure 7.3 gives a situation where the domain has a probability for being malicious.

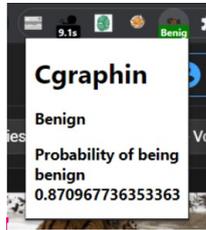


Figure 5.27: Benign indication



Figure 5.28: Malicious indication

5.4.3 Developer API

For researchers or people who need to would like to do research using our data a developer API was created publicly exposing the API to access the data. However, users need to have a unique access token and their JWT token. This could be generated using the Token Manager page in the web application as shown in figure 5.29. These public APIs are secured with the tokens and JWT tokens. Whenever a request is received the API validates both the JWT token and the token before retrieving the data to the public users.

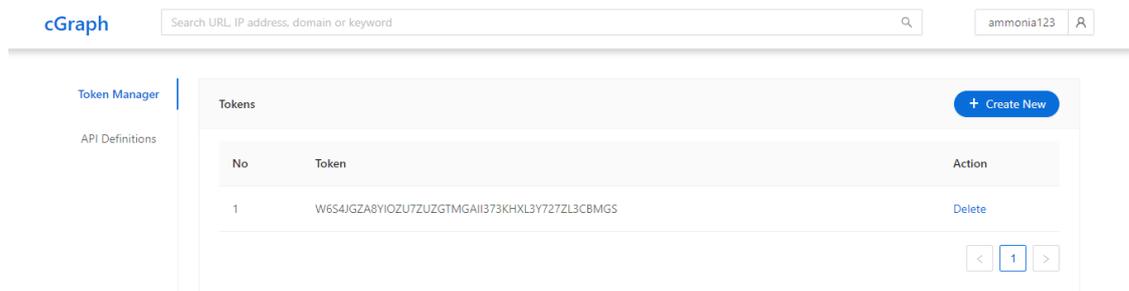


Figure 5.29: Token Manager (Public API key generator)

As given in figure 5.30 the publicly exposed API has two end points to the users. First end point is the the /search endpoint where users could have the available domain in our database and another endpoint is the /graphsearch to retrieve the nodes and edges for any analysis they are willing to carry out.

cGraph Search URL, IP address, domain or keyword ammonia123

token Manager

API Definitions

CGraph Public API 1.0.0 OAS3

<https://raw.githubusercontent.com/Wahsara/waggeer-og/main/cg.yml>

This includes Endpoints for getting details to public using Tokens

Apache 2.0

Servers

graph get the Details of the graph

GET `/graphsearch/<term>/<token>` Retrieve the data of the selected node for past week

search Search domains and keywords

GET `/search/<term>/<page>/<token>` Retrieve the data of the selected node for past week

Figure 5.30: Public API endpoints

Chapter 6

Results and Analysis

This chapter explores the results of the inference process and gives an analysis on it. Section 6.1 discusses the evaluation of the inference process and section 6.2 gives a demonstration of the inference process.

6.1 Evaluation of the inference process

Apex domains with different VT levels and benign domain with a high Alexa Rank was extracted from the PDNS A record data in order to run the inference algorithm on it.

6.1.1 Ground truth creation for the evaluation process

The ground truth extraction process was conducted in two stages; benign ground truth extraction and malicious ground truth extraction.

For the benign ground truth extraction Alexa Rank data was used. Alexa Rank data feed for the latest seven consecutive days was taken and the domains with Alexa Rank ≤ 10000 was taken for each day. From this, the unique set of domains was extracted and the topmost 700 Alexa Rank domains were taken as the benign ground truth. The reason for using domains with Alexa Rank ≤ 10000 which appear continuously throughout a week is under the assumption that such domains are less likely to be attacker manipulated [4].

For the malicious ground truth, the data from the daily feed of VirusTotal was used. VirusTotal reports for seven consecutive days were taken and the records with a VT count of more than 2 is filtered each day. From this, the unique set of records is taken

and a random sample of 600 records was taken as the malicious ground truth.

After the extraction of malicious and benign seeds, a few random samples were selected in order to carry out manual verification to verify that the extracted samples are correct. It was observed that the sample set had no ambiguous seeds, so the collection of 1300 ground truth data was used to run the inference algorithm and extract the belief value after the inference process was done.

6.1.2 ROC and AUC

To understand the best threshold value from the given inference values as given in table 6.1 the ROC curve was plotted for different threshold values and the AUC value was taken from it.

Table 6.1: TP rate and FP rate against different threshold values on inference results

False Positive (FP) Rate	True Positive (TP) Rate	Threshold
0.006667	0.175714	0.692308
0.008333	0.175714	0.600000
0.078333	0.994286	0.500000
0.140000	0.994386	0.400000
0.161667	0.984286	0.307692
0.170000	0.994267	0.250000
0.573333	0.995714	0.181818
1.000000	1.000000	0.129032

With the different threshold values, the best performance was obtained when the threshold value is 0.1290 as shown in figure 6.1.

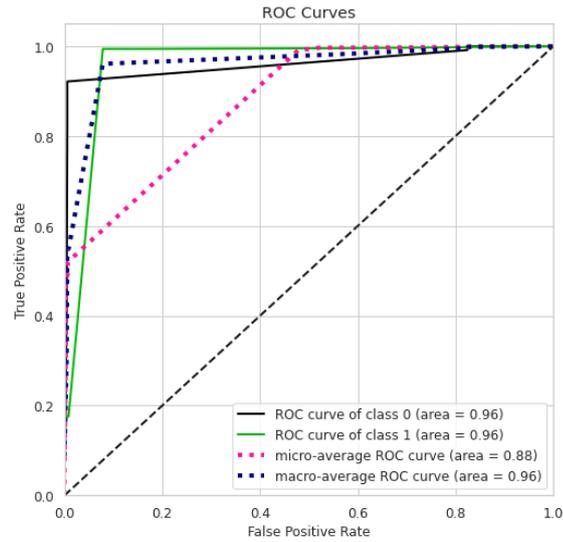


Figure 6.1: ROC curve

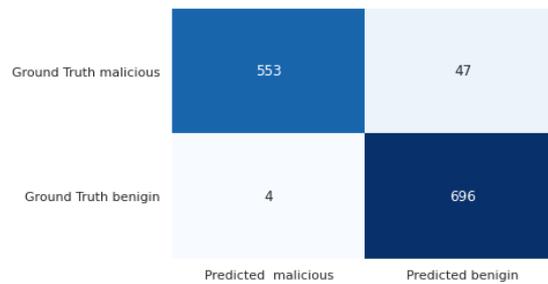


Figure 6.2: Confusion Matrix

With the selected best threshold value, the confusion matrix given in figure 6.2 was obtained. The relevant evaluation matrix calculated using these values are given in table 6.2.

Table 6.2: Evaluation matrix for the inference process

precision	0.92166
recall	0.99428
PPV	0.99281
PNV	0.93674

6.1.3 Evaluation of the inference process against different maliciousness levels indicated by VirusTotal

The following experiment was done in order to verify how well the inference algorithm has performed. FQDNs with different levels of maliciousness as provided by the VirusTotal data was extracted and the inference algorithm was run on top of it. As seen in figure 6.3, it was observed that there was a reduction in False Positive rate as the positive indicators of VT data increases. Thus, we can conclude that the inference algorithm has performed very well.

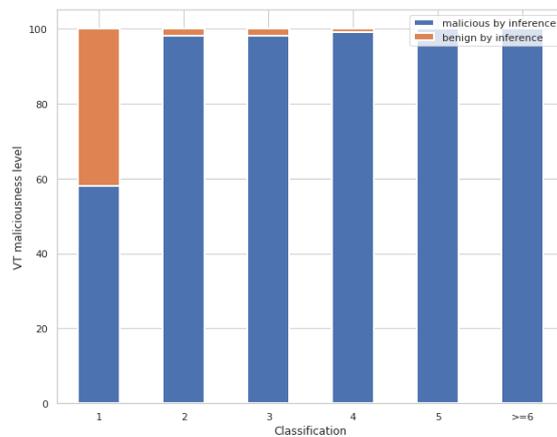


Figure 6.3: Evaluation of the inference process against different maliciousness levels indicated by VirusTotal

6.2 Demonstration

Most of the phishing attacks are targeted at the average user and most attacks run via the social engineering corner. A main method that attackers use to mislead a user is squatting the domain. For example, a user may think that since paypalcom.co is similar to paypal.com that it is also a legitimate site. That is, an average user may not know whether a certain domain is legitimate or not.

In the following tscenario a specific type of squatting attack called typo attacks is considered for demonstrtion purposes. Consider an average user who receives an email from <http://www.goole.com/> (which is an actual site). But users may be confused whether its the original google.com or not.

In general major companies such as Facebook or Google does not allow their domains be vulnerable to phishing attacks. In order to do this they buy all the domains that look similar to their original one and create CNAME records in the DNS and redirect the traffic to the original site. Figure 6.4 shows the search results for the keyword "Instagram" in our system where such a situation has occurred.

Domain	ASN	Location	First Seen	Last Seen
instergram.com. (Apex)	None	Australia	2020/9/7	2020/9/8
	AS133618 Trellian Pty, Limited			
instergram.online. (Apex)	None	Ashburn,United States	2020/9/7	2020/9/7
	AS32181 ASN-GIGENET			

Figure 6.4: Search results for the keyword "Instagram"

All the above domains belong to the legitimate Instagram corporation and the grey dash mark indicates that existing reputation systems have never explored them. Additional information also gives some insight as where they are coming from etc.

Domain	ASN	Location	First Seen	Last Seen
pahe.in. (Apex)	None	United States	2020/9/7	2020/9/8
	AS13335 CLOUDFLARENET			
pahe.me. (Apex)	None	Nuremberg,Germany	2020/9/7	2020/9/7
	AS51167 Contabo GmbH			
pahe.ph. (Apex)	0	Menifee,United States	2020/9/7	2020/9/8
	AS30148 SUCURI-SEC			

Figure 6.5: Advanced filtering of search results

Figure 6.5 gives another example page where a domain originates from various locations and is marked with a green tick indicating that it has been explored by an existing reputation system (VirusTotal). This means that it is recognized as a non-malicious site.

Furthermore, our system can be used for exploring the resources attached with a given domain or IP in order to determine whether it is legitimate or not. Such a situation is shown in figure 6.6.

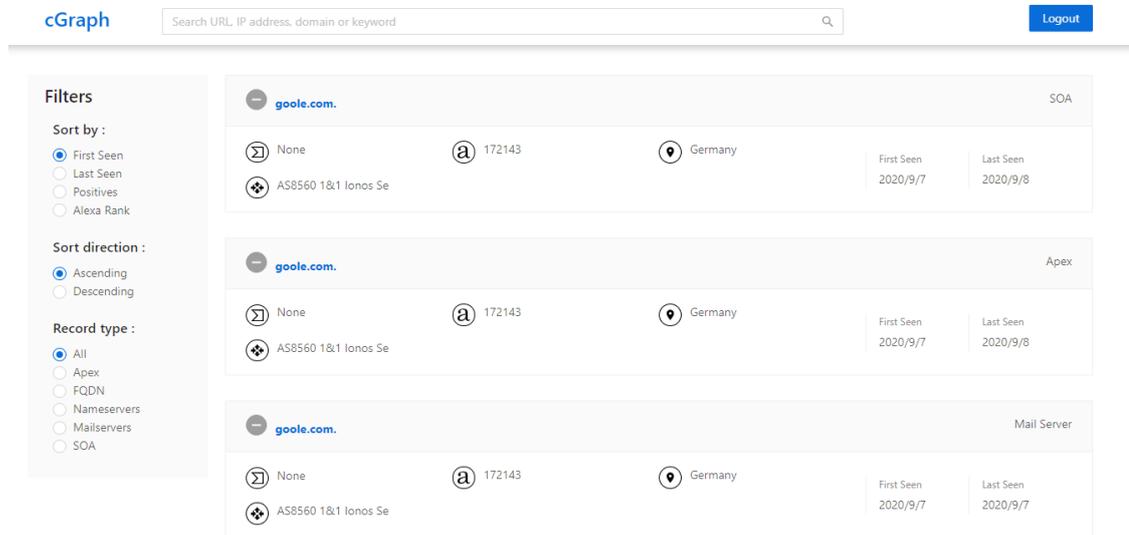


Figure 6.6: Catching malicious acts from search results

We can sort all our internet resources (Apex, FQDNs, Name Servers, Mail Servers etc.) using the Alexa Rank in the ascending order (Figure 6.7). Alexa Rank indicates the traffic statistics and gives an idea whether the given domain is popular or not. But still, the attacker has a chance to compromise the site and use it for malicious acts.

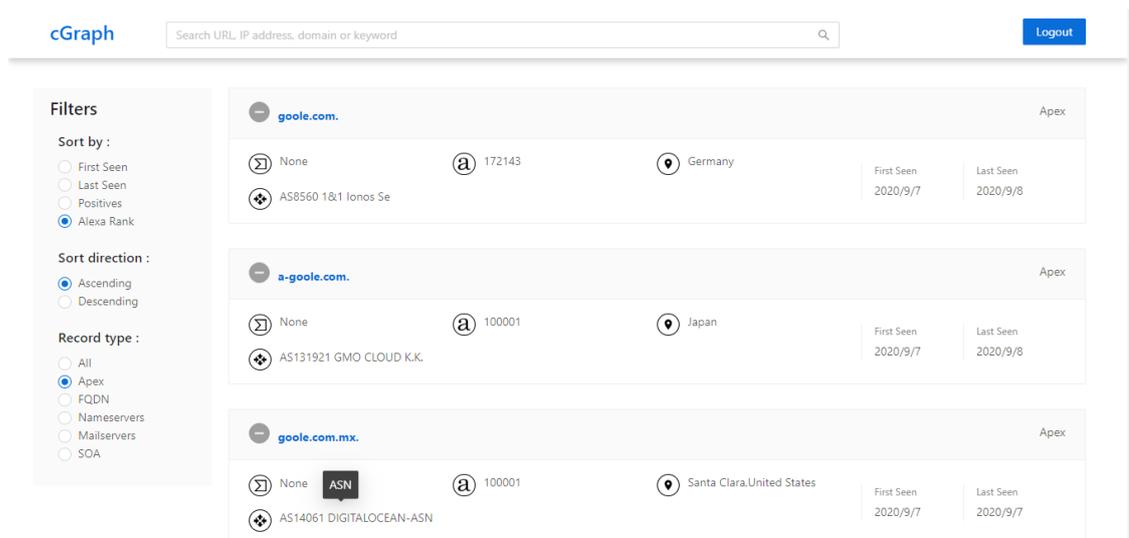


Figure 6.7: Sorted search results

When filtered goole.com is at the top and it indicates that it has never been explored

by VirusTotal and its Alexa Rank is 172143 which indicates that it is not much popular. Therefore, at this stage, user can arrive at the conclusion that it is not attacker created, but there is a chance to be compromised. That is the next step of our system as the search result could be visualized using a graph as shown in figure 6.8.



Figure 6.8: Visiting the graph view

As seen in figures 6.9 and 6.10, the basic information for the search result does not indicate that it could be a malicious domain.

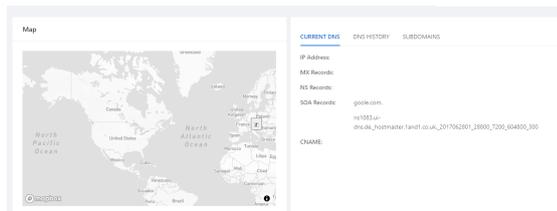


Figure 6.9: View historical DNS information

When exploring the DNS profiles of the domain we can see what existing reputation systems indicate about the domain. Since its Alexa Rank is within the top 1M, it's a good indicator that this site does not have any malicious intentions.

We can expand the graph and explore the relations of the captured resources as shown in figures 6.11 and 6.12.

We can expand the graph traverse it through time to see how things have changed over the time.

In conclusion, in this scenario all the information points to the resource being labeled

CURRENT DNS		DNS HISTORY	SUBDOMAINS	
A	MX	NS	SOA	CNAME
Positives		SOA	First See	
None		google.com.	1601497	
None		ns1083.ui-dns.de_hostmaster.1and1.co.uk_2017062801_28800_7200_604800_300	1601497	

< 1 2 3 4 5 6 7 >

Figure 6.10: View historical DNS information (Summarized View)

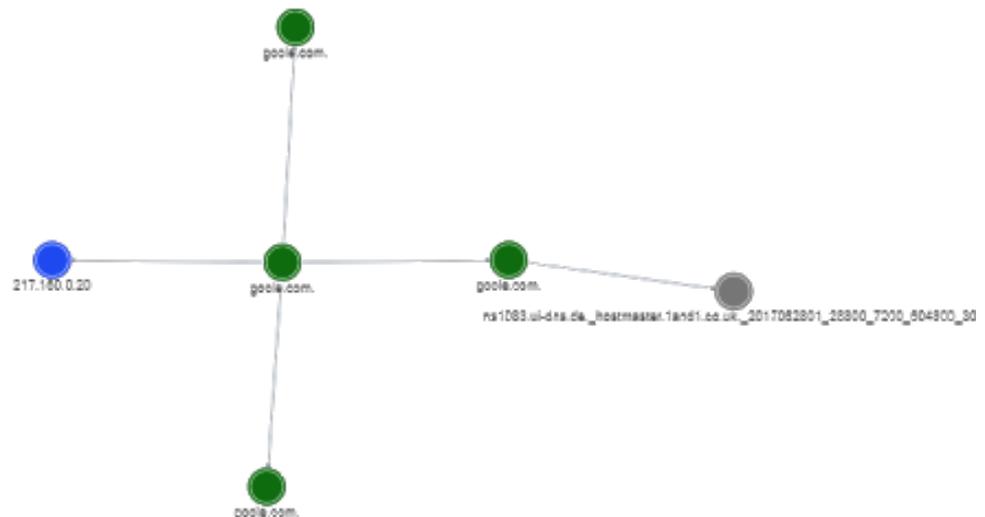


Figure 6.11: Initial graph view

as a benign domain, even if it's more likely to be a typosquatting domain of original google.com. Also we can conclude that it's neither an attacker created or compromised domain.

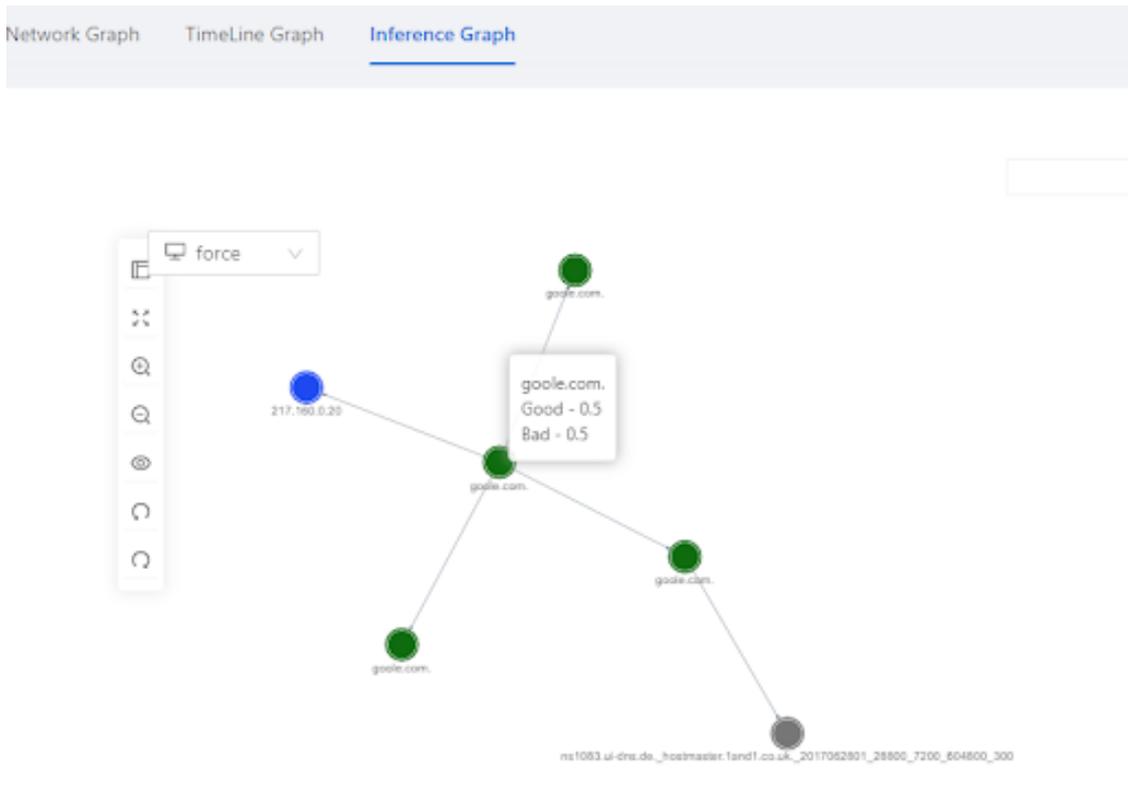


Figure 6.12: Inference graph

Chapter 7

Evaluation and Testing

This chapter explores the performance of the designed system against different measurement matrices and how the entire system is evaluated in terms of Performance, Availability, Security, and Usability, etc.

7.1 Spark data processing pipelines

Table 7.1 gives the aggregated summary of each Spark job process. It shows the required resources and processing time, as well as the volume of data that has been processed.

The total number of processors indicates the accrued number of processor cores while the total number of executors indicates the number of spark executors that was designed to distribute the data. Total memory gives the total memory capacity required and swapping indicates the disk swapping while the in-memory computation is running on the job.

Each spark job consists of series of MapReduce operations. The failure rate indicates the number of times a particular stage had to be resubmitted in order to complete the job successfully. The causes for this type of failures was Java VM heap overflows and bad partitioning. The skipping rate indicates the intermediate stage crashes that occurred while processing VirusTotal data. It was identified that this was due to oversized RDD partitions. With high optimizations the skipping rate was reduced to 0.01% level, which led to high efficiency in processing of VirusTotal data.

When considering the time taken to complete each job, which is the most significant evaluation criteria, new data could be processed each day within 1.5 hours to 2 hours

Table 7.1: Measuring the performance of the Spark big data pipeline

Job name	Number of processor cores (Total)	Number of executors	Number of processor cores for executor	Memory	Swapping	Input	Output	Failure rate (stages)	Skipped (stages)	Time to complete
A record processing stage one	360	24	15	1.4TB	0B	68GB	76.1GB	0%	0%	15:28min
A record processing stage two	360	24	15	1.4TB	0B	76.1GB	84.2GB	0%	0%	16.30min
NS record processing stage one	360	24	15	1.4TB	0B	51GB	62.3GB	0%	0%	10.12min
NS record processing stage two	360	24	15	1.4TB	0B	62.3GB	68.1GB	0%	0%	11.00min
MX record processing stage one	360	24	15	1.4TB	0B	2.1GB	3.0GB	0%	0%	4.20min
MX record processing stage two	360	24	15	1.4TB	0B	3.0GB	3.4GB	0%	0%	5.23min
SOA record processing stage one	360	24	15	1.4TB	0B	27GB	32.1GB	0%	0%	8.12min
SOA record processing stage two	360	24	15	1.4TB	0B	32.1GB	36GB	0%	0%	9.30min
CNAME record processing stage one	360	24	15	1.4TB	0B	22GB	27.4GB	0%	0%	8.40min
CNAME record processing stage two	360	24	15	1.4TB	0B	27.4GB	27.1GB	0%	0%	10.10min
VT data processing	360	360	1	1.448TB	100MB-200MB	2.3 GB - 3 GB	2.8 GB - 3.6 GB	0%	~0.001%	23.20min
Alexa Rank data processing	4	4	1	1GB	0B	11MB	~28MB	0%	0%	1min

time. Processing such a large volume of data within that amount of time leads to keep the system up to date while serving global consumers.

7.2 Database Clusters

7.2.1 Evaluation of data insertion in ArangoDB

When a database is at its maximum capacity, it may lead to some unexpected failures. Therefore, the system had to be designed in such a way so as to ensure that none of the database clusters arrive at its maximum capacity. Considering the database clusters that have been used in the project, the most critical database cluster to the system is ArangoDB since it contains the data needed to formulate the entire graph and it also has to handle the most number of Read and Write requests. Figure 7.1 shows the outcome of aggressive data insertion for the ArangoDB cluster. As shown, the data insertion speed is ~23MBps. Due to the data capacity it was possible to insert the entire day graph data into the database cluster within 3 hours.

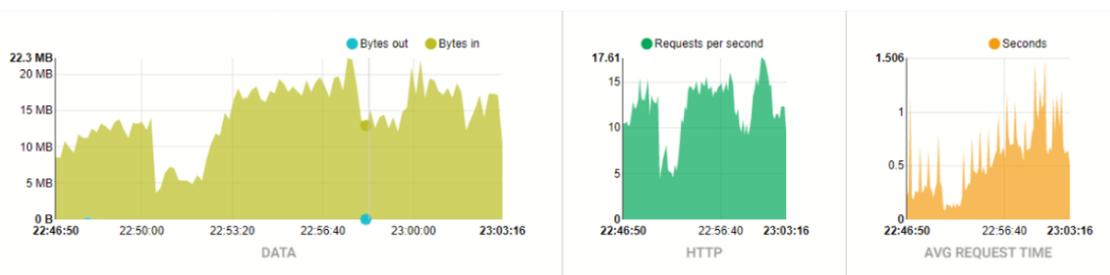


Figure 7.1: Evaluation of the Write speed of the ArangoDB graph database cluster

7.2.2 Evaluation of the indexing performance of the Greenplum DB cluster

Graph databases needed less search space compared to the greenplum databases , Given a node extracting a graph from the only search space used in the Graph database cluster. Considering the two databases (Virus total profile database and Alexa profile database) in green plum required more search space. Since it's used to retrieve apex/fqdns belongs to given keywords etc. With the new data insertion it is required to update the index of each partition of the database. Figure 7.2 shows the amount of data records in the alexa database partitions and figure 7.3 shows the time of completion of the index update. The time required to update the index is orthogonal to the amount of the data in the partition. With design we could active average 50 second index updation time for each new data insertion.

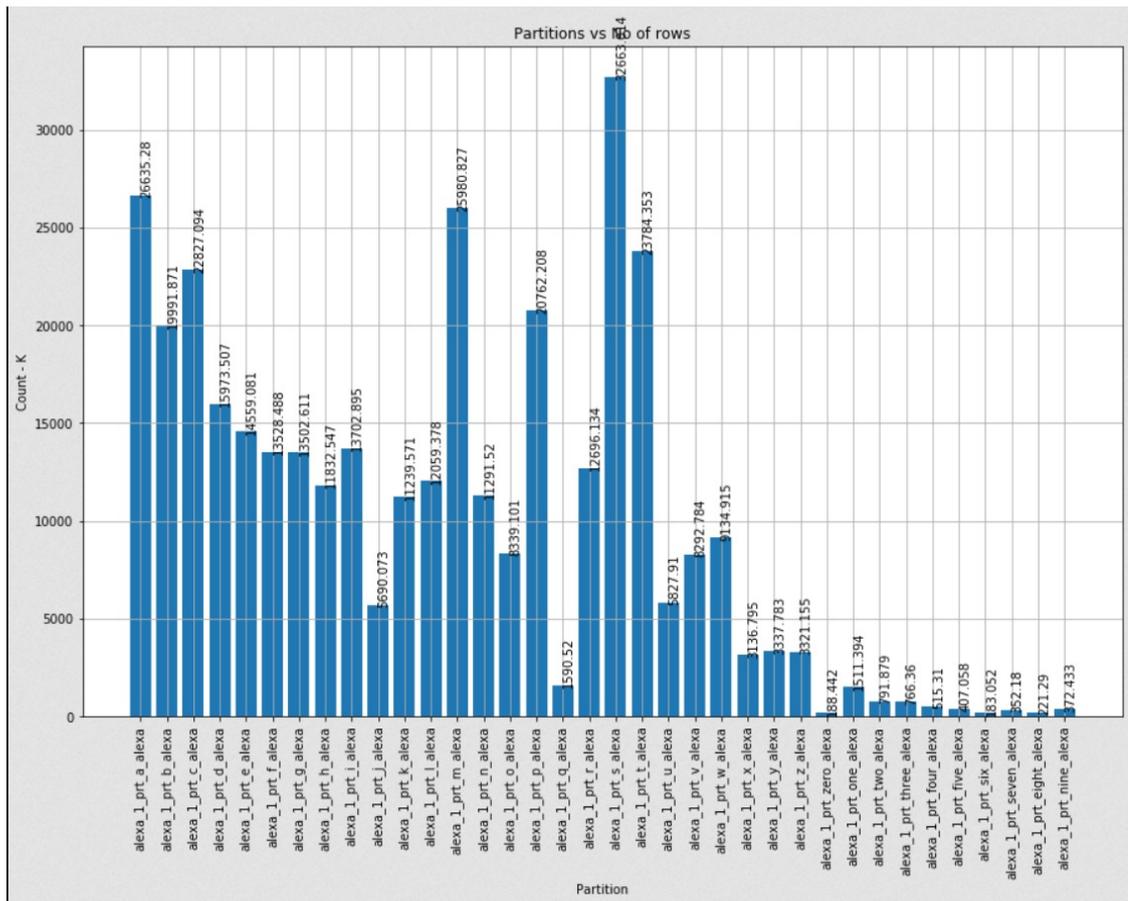


Figure 7.2: Partitions Vs Number of Records

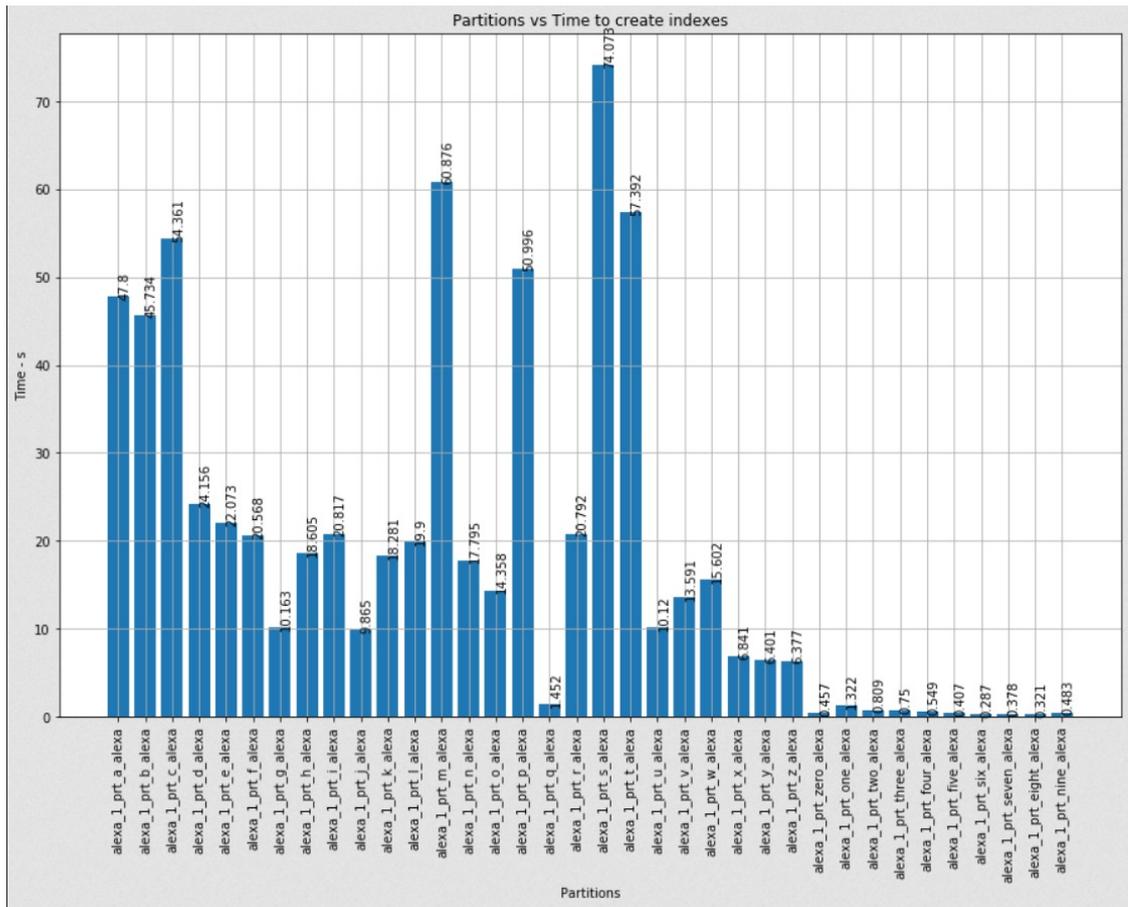


Figure 7.3: Partitions Vs Index Creation Time

Encapsulating the index creation time of VirusTotal database is much more complex than the Alexa DB. VirusTotal database has been partitioned into 1332 partitions and each partition. Those partitions have high diversity within the volume of data. Figure 7.4 shows the cumulative measure of index creation time in each partition. On average 50% partitions do the index creation process within 500s and the maximum time required to update the index was 1300s.

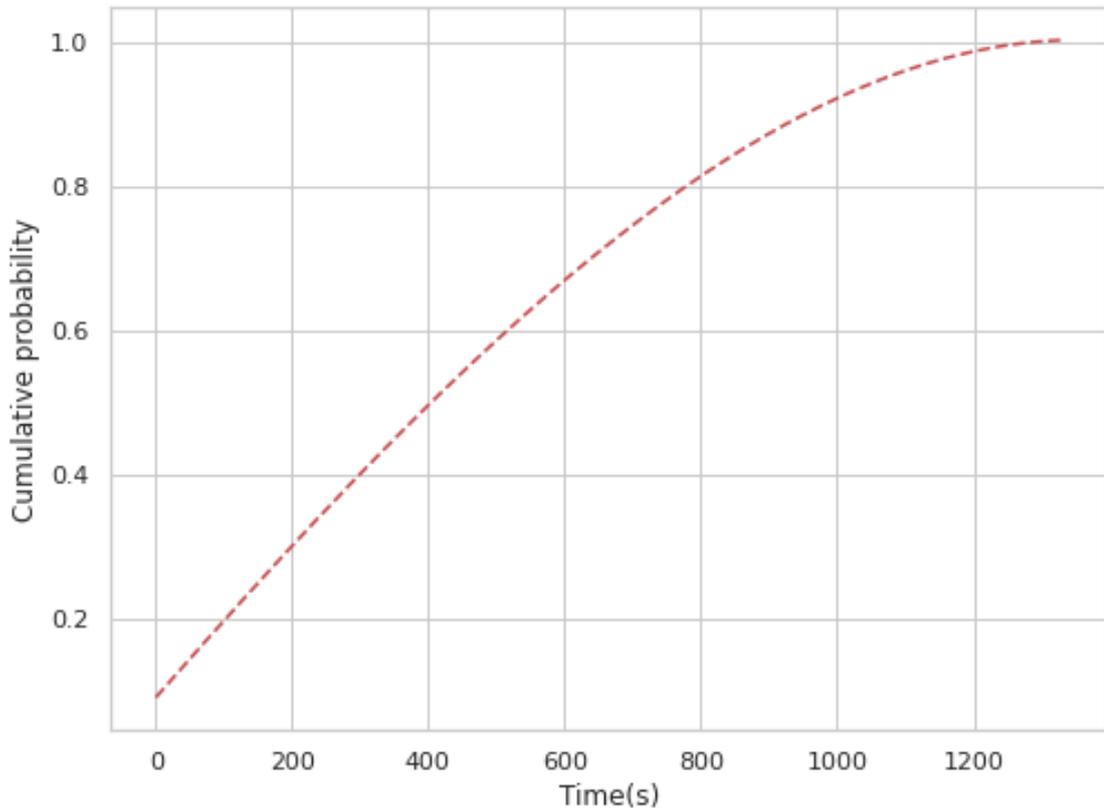


Figure 7.4: Cumulative measure of index creation time in VirusTotal profile database

7.3 Middleware

7.3.1 Performance

For evaluating the performance of each microservice Apache JMeter [29] was used, which is an industry-standard open-source software used for load testing and performance evaluation. The performance evaluation process was done using hundred concurrent JMeter artificial clients, sending the request to each service individually to its endpoints for ten rounds and calculating the average, min, max, and median access time/response time.

The results of the performance testing carried out using JMeter is summarized in table 7.2. The response times and error rates achieved by the system indicates that the optimization strategies implemented in the system is satisfactory.

Table 7.2: Performance evaluation using JMeter

Service	Average(ms)	Median(ms)	Min(ms)	Max(ms)	Error	TP(ms)
GraphNodeAlexa API	2138	2058	874	3424	0.000%	0.060
Maxmind API	306	288	248	1028	0.000%	0.059
Graph API	469	258	252	1756	0.000%	0.058
GraphInference API	1213	1068	773	2881	0.000%	0.460
Timeline API	3058	2880	2055	5509	0.000%	0.059
Search API	360	332	250	859	0.000%	0.059
GetNodeDetails API	354	328	264	671	0.000%	0.575
GetNodeDetails API	2041	1992	1302	3245	0.000%	1.061
Forgotpassword API	1758	1732	1671	2132	0.000%	0.939
signup API	470	485	388	527	0.000%	0.657
signIn API	475	489	374	683	0.000%	0.681

7.3.2 Availability

For evaluating the availability of the application an industry-standard open-source performance evaluation tool named Locust [30] was used. The number of concurrent users was gradually increased using this tool against our platform and the mean response time was recorded (Figure 7.5). With the increased number of users, the number of requests per second was also gradually increased as shown in figure 7.6] and the average response time was measured. As seen in figure 7.7 it remained constant, and it was concluded that the load balancing and auto-scaling of API is functioning as designed.

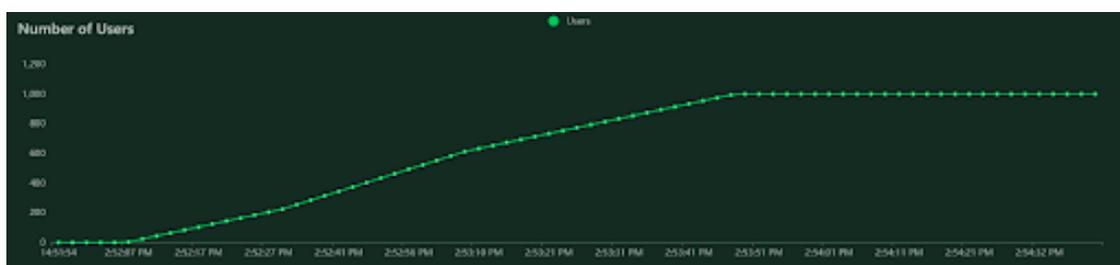


Figure 7.5: Increasing the Number of users with time

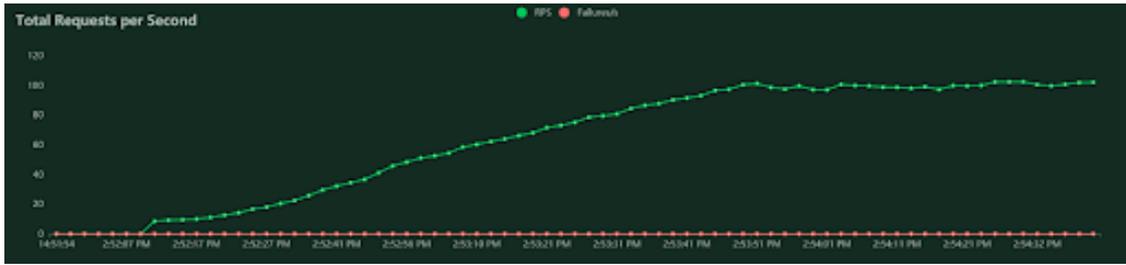


Figure 7.6: Increasing the Number of requests with time

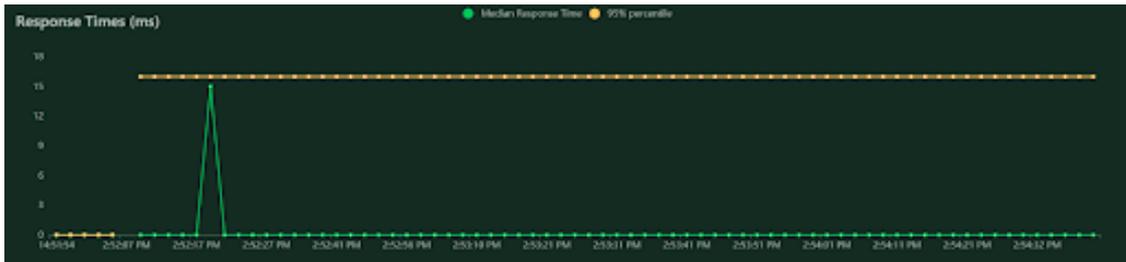


Figure 7.7: Average response time

7.4 Web application

7.4.1 Front-end testing and end-to-end evaluation

Cypress [31], which is a JavaScript end-to-end testing framework built for the modern web was chosen as the front-end testing and API testing tool. Figure 7.8 shows the Cypress dashboard integrated with our system.

The test cases are run for each Pull Request (PR). As shown in figure 7.9, test cases for all the functionalities defined were executed and tested.

Run status

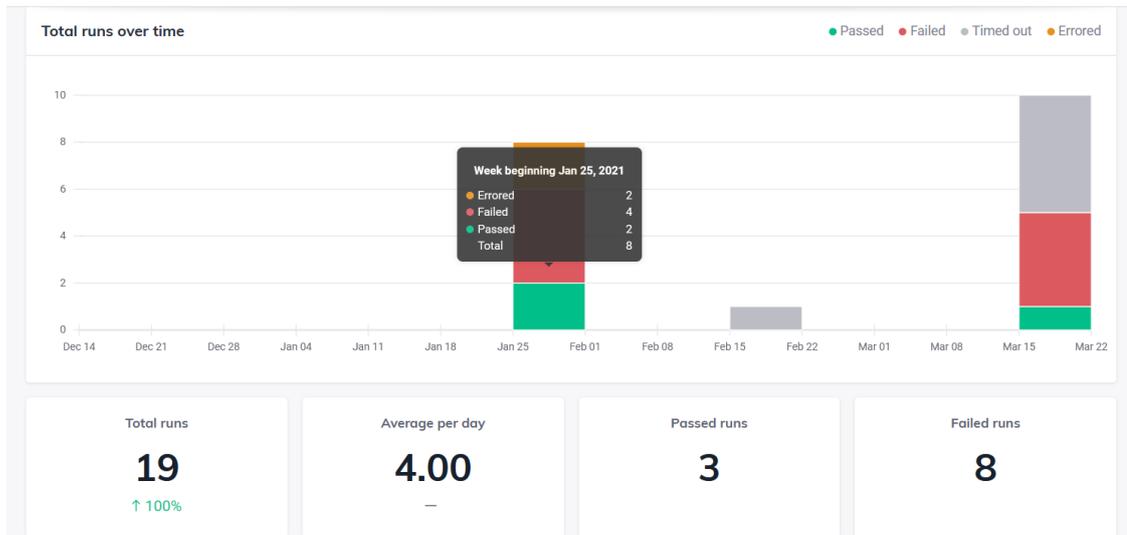


Figure 7.8: Cypress dashboard (Summary)

Latest runs

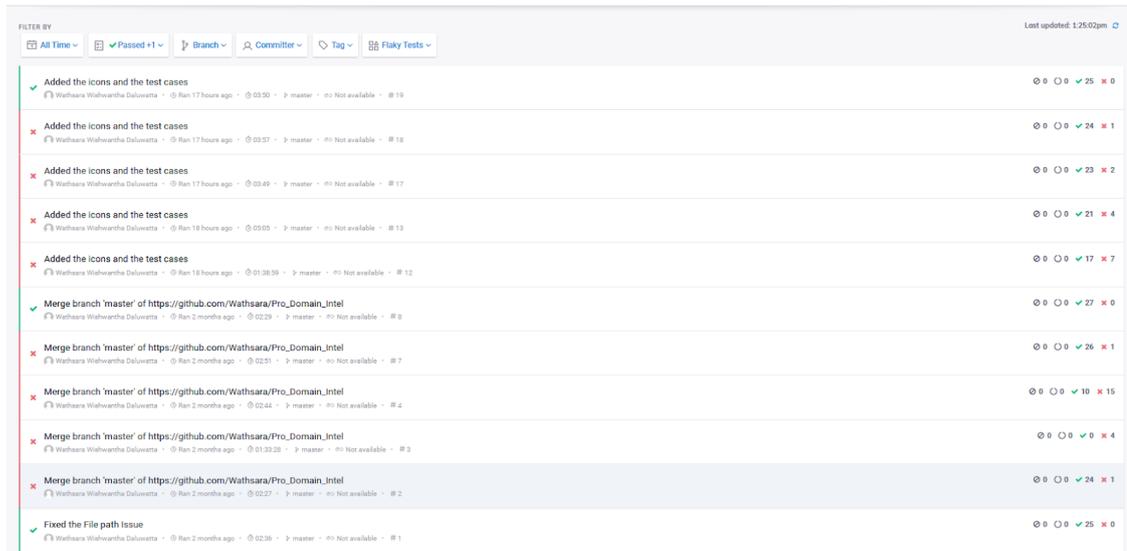


Figure 7.9: Cypress dashboard (Test case)

7.5 Cost evaluation

According to the Google Cloud Resource Calculator the initial system cost was calculated as given in figure 7.10, without calculating the egress charges for each VM [32].

With this initial cost of USD 8,850.93, our initial design had to add 16 new VMs to the cluster each month. The cost of adding 16 VMs to the cluster was USD 6,261.55 which was an increment cost for the month as shown in figure 7.11.

Your Estimated Bill *

Estimated Monthly Cost: USD 8,850.93

 20 x	e2-standard-16	14600 total hours per month	USD 7,826.93
 Cloud Storage	Storage	51200 GiB	USD 1,024.00

Total Estimated Monthly Cost

USD 8,850.93

Your Estimated Bill *

Estimated Monthly Cost: USD 6,261.55

 16 x	e2-standard-16	11680 total hours per month	USD 6,261.55
--	----------------	-----------------------------	--------------

Total Estimated Monthly Cost

USD 6,261.55

Figure 7.10: Initial cost

Figure 7.11: Incremental cost

Thus, the estimation cost for one year was USD 519,473.45. With deep configuration of database clusters it was possible to reduce 4 new VMs per week when scaling the Kubernetes cluster into 4 new VMs per month. This reduced the overall cost to USD 209,526.89 with a reduction in cost of USD 309,946.56, which is a 40% cost cut.

Chapter 8

Conclusion

The main purpose of the project was to design and implement a scalable and efficient big data graph system that can process and store a large amount of data. The challenge was to understand raw data sources and transform them into intelligence and store them in a time series heterogeneous graph that could be retrieved within milliseconds. Additionally, a web platform for cyber threat intelligence was developed using the research findings of QCRI for graph-based inference, along with a Chrome extension and Developer API.

This study includes a well-focused background study covering the inference algorithm research done by QCRI, and a number of existing cyber threat intelligence solutions. Furthermore, it provides an analysis of the competitive advantage of our system over the existing systems.

System implementation was done in three main streams as data engineering, back end development and front end development. Data engineering includes the necessary data preprocessing, transformation and storage components. Back end development holds implementation of the middleware with the use of microservices and REST APIs to provide services of the system. Additionally, the back end development also includes integration, monitoring and deployment of the system. Front end development includes implementation of the web platform which provides the users an interactive web application in order to derive cyber threat intelligence using graphs.

The entire big data system has been evaluated in order to ensure that it meets its functional and non-functional requirements. The inference system has been evaluated to ensure that the underlying methodologies and algorithms of the original research is

preserved when being adapted into a system with real-world data.

8.1 Limitations

The system only allows traversal through a prior week's worth of historical data. Beyond one week of data existing open-source tools were unable to handle the rendering process. Due to this the system only allows the user to travel one week, which avoids flooding of the user interfaces from thousands of edges and nodes and crashing of the browser. Since transferring the data is not cumbersome, retrieving more historical data is only allowed via the developer API.

Additionally, the inference process is being run on depth two subgraph only. It is run on the entire knowledge graph at once, since there is no existing tool to load such a massive amount of data and run the graph inference algorithm on it.

8.2 Future Work

- Introducing a Pricing Plan and limiting the retrieved results based on the packages for the public API.
- Introducing other graph based research outcomes of QCRI to the system and make it available to the end consumer.
- Enhancement of the chrome Extension.
- Creation of a Data archive system.
- Creation of TAXII/Indicators of Compromise Service(IOC).

References

- [1] I. Khalil, B. Guan, M. Nabeel, and T. Yu, “Killing two birds with one stone: Malicious domain detection with high accuracy and coverage,” 2017.
- [2] A. Metzger, K. Pohl, P. Bellavista, J. Butler, M. Franz, and J. Garbajosa, “Software Engineering: Key enabler for innovation,” in *NESSI White Paper*. Networked European Software and Services Initiative, Jul. 2014. [Online]. Available: http://www.nessi-europe.eu/Files/Private/NESSI_SE_WhitePaper-FINAL.pdf
- [3] N. P. Hoang, A. A. Niaki, M. Polychronakis, and P. Gill, “The web is still small after more than a decade,” 2020.
- [4] R. De Silva, M. Nabeel, C. Elvitigala, I. Khalil, T. Yu, and C. Keppitiyagama, “Compromised or attacker-owned: A large scale classification and study of hosting domains of malicious urls,” january 2021, usenix security. [Online]. Available: <https://sec21fall.usenix.hotcrp.com/paper/309?cap=0309aokMMWq1Zwm8>
- [5] J. S. Yedidia, W. T. Freeman, and Y. Weiss, “Generalized belief propagation,” in *Proceedings of the 13th International Conference on Neural Information Processing Systems*, ser. NIPS’00. Cambridge, MA, USA: MIT Press, 2000, p. 668–674.
- [6] S. Ramanathan, J. Mirkovic, and M. Yu, “Blag: Improving the accuracy of blacklists,” 01 2020.
- [7] J. Pearl, “Reverend bayes on inference engines: a distributed hierarchical approach,” in *in Proceedings of the National Conference on Artificial Intelligence*, 1982, pp. 133–136.
- [8] J. Yedidia, W. Freeman, and Y. Weiss, “Understanding belief propagation and its generalizations,” in *Exploring Artificial Intelligence in the New Millennium*,

- G. Lakemeyer and B. Nebel, Eds. Morgan Kaufmann Publishers, Jan. 2003, ch. 8, pp. 239–236. [Online]. Available: <https://www.merl.com/publications/TR2001-22>
- [9] N. Friedman, D. Geiger, and M. Goldszmidt, “Bayesian network classifiers,” in *Machine Learning*, 1997, pp. 131–163.
- [10] H. Rue and L. Held, *Gaussian Markov Random Fields: Theory And Applications (Monographs on Statistics and Applied Probability)*. Chapman Hall/CRC, 2005.
- [11] “McAfee SiteAdvisor,” <https://www.siteadvisor.com/>, [Online].
- [12] P. Manadhata, S. Yadav, P. Rao, and W. Horne, “Detecting malicious domains via graph inference,” in *Proceedings of the 2014 Workshop on Artificial Intelligent and Security Workshop*, ser. AISEC ’14. New York, NY, USA: Association for Computing Machinery, 2014, p. 59–60. [Online]. Available: <https://doi.org/10.1145/2666652.2666659>
- [13] “Alexa Top Sites,” <https://aws.amazon.com/alexa-top-sites/>, [Online].
- [14] N. Bronson, Z. Amsden, G. Cabrera, P. Chakka, P. Dimov, H. Ding, J. Ferris, A. Giardullo, S. Kulkarni, H. Li, M. Marchukov, D. Petrov, L. Puzar, Y. J. Song, and V. Venkataramani, “TAO: Facebook’s distributed data store for the social graph,” in *2013 USENIX Annual Technical Conference (USENIX ATC 13)*. San Jose, CA: USENIX Association, Jun. 2013, pp. 49–60. [Online]. Available: <https://www.usenix.org/conference/atc13/technical-sessions/presentation/bronson>
- [15] “AT&T Cybersecurity,” <https://cybersecurity.att.com/>, [Online].
- [16] “Anomali Threat Intelligence Platform,” <https://www.anomali.com/>, [Online].
- [17] “Cisco Umbrella Investigate,” <https://umbrella.cisco.com/products/umbrella-investigate>, [Online].
- [18] “VirusTotal Graph,” <https://www.virustotal.com/gui/graph-overview>, [Online].
- [19] L. Bass, P. Clements, and R. Kazman, *Software Architecture In Practice*, 01 2003.
- [20] “Farsight Security,” <https://www.farsightsecurity.com/>, [Online].
- [21] “VirusTotal,” <https://www.virustotal.com/>, [Online].

- [22] “MaxMind,” <https://www.maxmind.com/en/home>, [Online].
- [23] L. Gu and H. Li, “Memory or time: Performance evaluation for iterative operation on hadoop and spark,” in *2013 IEEE 10th International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing*, 2013, pp. 721–727.
- [24] “Greenplum Database,” <https://greenplum.org/>, [Online].
- [25] “Kubernetes,” <https://kubernetes.io/>, [Online].
- [26] E. Choo, M. Nabeel, M. Alsabah, I. Khalil, T. Yu, and W. Wang, “Device-watch: Identifying compromised mobile devices through network traffic analysis and graph inference,” 2019.
- [27] “InfluxDB,” <https://www.influxdata.com/>, [Online].
- [28] “Flannel: Pod network for Kubernetes,” <https://github.com/flannel-io/flannel>, [Online].
- [29] “Apache JMeter,” <https://jmeter.apache.org/>, [Online].
- [30] “Locust,” <https://locust.io/>, [Online].
- [31] “Cypress,” <https://www.cypress.io/>, [Online].
- [32] “Google Cloud Resource Calculator ,” (<https://cloud.google.com/products/calculator/#id=e07bb625-e7c1-4325-844f-fe6c907299af>).

Appendices

University of Colombo School of Computing
Sri Lanka

RE: Letter of Appreciation

To whom it may concern,

I am writing to express my satisfaction and appreciation of the successful completion of cGraph: Graph Based Extensible Cyber Threat Intelligence Platform by W. W. Daluwatta (16000226), L. R. S. De Silva (16000252) and S. N. Kariyawasam (16000684) from University of Colombo School of Computing, Sri Lanka.

cGraph is a project to develop a pluggable and scalable big data graph processing and storing system as a backbone of graph-based research outcomes of QCRI. The goal was to design and implement a big data architecture and implement one of our research outcomes on predicting malicious Internet domains and expose it to the end user as a proof of concept.

The research outcomes show how to run inference-based models on a controlled environment with static datasets. Taking such research outcomes to production involves several engineering challenges on their own and requires innovative engineering solutions especially due to the sheer volume of data that needs to be handled and the dynamic nature of the data involved. The three students were able to work as a team and overcome the above challenges to build a highly scalable system.

QCRI was responsible for providing the raw data, hardware infrastructure, and research outcomes of the inference algorithms. The aforementioned group of students has done an immense job, among other aspects, in gaining the required domain knowledge, understanding the non-uniform and dynamic raw data, finding better ways to optimally process and store data, running microbenchmark on database tools and frameworks to understand the most suitable technology stack for the project, designing a highly available and scalable architecture, building a Kubernetes cluster and



database clusters, designing and implementing the microservices based middleware and consumer end applications, solving the problem of large graph visualization in consumer products, adopting our research outcomes to the system, and evaluating both the algorithm and the system.

Thank you.

Sincerely,



Dr. Mohamed Nabeel
Senior Scientist

Qatar Computing Research Institute
Hamad Bin Khalifa University
HBKU Research Complex
P.O. Box 5825, Doha, Qatar
Tel: +974 44541426
www.qcri.qa

Appendix A

Contribution

This project was successfully completed with equal contribution from all members of the group at all phases of the project. The workload was divided horizontally and vertically within each member. Detailed descriptions of the contribution by each member of the group is given below.

W.W. Daluwatta (16000226)

Mainly worked on designing and building the microservices/API and designing the database schemas along with their optimizations, explored how to use parallel routines for fast execution, explored better ways for faster data orchestration to the end consumer, explored how to get the max use of Kubernetes to build a high available big data processing/storing system and worked on application level security. As a result was able to develop a highly available and effective graph retrieval and graph time traversal middle-ware.

Additionally worked on designing and implementing databases, worked on solving data engineering problems, worked on designing big data processing pipelines using Spark, worked on designing continuous delivery pipelines (CI/CD) and solved the challenges of adopting the QCRI research outcomes to the system. Also worked on developing the public API, Chrome extension and the web application.

L.R.S. De Silva (16000252)

Mainly worked on designing and implementing the high available infrastructure and data processing pipelines. Was responsible for understanding the volume and shape of raw data, designing the database schemas, designing and implementing the data processing pipelines using Spark and building the Kubernetes cluster and other relevant database clusters on top of the Kubernetes clusters.

How to keep the system without failures, how to achieve the best combinations of infrasture orchestration, how to trade data redundancy against efficiency, how to optimize the pipelines using Spark, how to see every problem as a MapReduce operation when writing Spark jobs, and how message passing algorithms such as belief propagation works in our context were the major concerns.

Apart from that understanding the research outcome of QCRI and converting it to a microservice that could be plugged into the designed system was one of the other highlighted works. Collaborating to the design phase of middleware and continuous delivery pipelines and frontend were other contributions

S.N. Kariyawasam (16000684)

Mainly Worked on designing and implementing the data visualization component and the relevant APIs. Smooth rendering of a large amount of nodes and edges, finding the most suitable data visualization libraries, state management of the rendering process in React using Redux, developing the timeline feature from the sketch, and graph node expansion were the major concerns.

Apart from that the API is the most important link between the backend and frontend, designing how the messages are passed within the backend and implementing the orchestration were highlighted works.

Also contributed to designing the database schemas, solving the data engineering problems and adopting the inference algorithm to the system. And worked on developing the web application and Chrome extension.

Group contribution

Conducting the literature study and competitive analysis of similar systems, exploring various technologies through trial and error in order to select the best suited technologies for the project, running microbenchmarks on several databases and libraries, designing the overall system, overcoming data engineering challenges and evaluating the system were a collaborative effort with more or less individual responsibilities.

Appendix B

SQL Schema for Alexa data

```
CREATE TABLE alexa (  
    domain text,  
    partition_key char(2),  
    date DATE Null,  
    rank int Null  
)  
  
DISTRIBUTED RANDOMLY  
PARTITION BY LIST (partition_key) (  
PARTITION a_alexas VALUES ('a'),  
PARTITION b_alexas VALUES ('b'),  
PARTITION c_alexas VALUES ('c'),  
PARTITION d_alexas VALUES ('d'),  
PARTITION e_alexas VALUES ('e'),  
PARTITION f_alexas VALUES ('f'),  
PARTITION g_alexas VALUES ('g'),  
PARTITION h_alexas VALUES ('h'),  
PARTITION i_alexas VALUES ('i'),  
PARTITION j_alexas VALUES ('j'),  
PARTITION k_alexas VALUES ('k'),  
PARTITION l_alexas VALUES ('l'),  
PARTITION m_alexas VALUES ('m'),
```

```
PARTITION n_alexas VALUES ('n'),
PARTITION o_alexas VALUES ('o'),
PARTITION p_alexas VALUES ('p'),
PARTITION q_alexas VALUES ('q'),
PARTITION r_alexas VALUES ('r'),
PARTITION s_alexas VALUES ('s'),
PARTITION t_alexas VALUES ('t'),
PARTITION u_alexas VALUES ('u'),
PARTITION v_alexas VALUES ('v'),
PARTITION w_alexas VALUES ('w'),
PARTITION x_alexas VALUES ('x'),
PARTITION y_alexas VALUES ('y'),
PARTITION z_alexas VALUES ('z'),
PARTITION zero_alexas VALUES ('0'),
PARTITION one_alexas VALUES ('1'),
PARTITION two_alexas VALUES ('2'),
PARTITION three_alexas VALUES ('3'),
PARTITION four_alexas VALUES ('4'),
PARTITION five_alexas VALUES ('5'),
PARTITION six_alexas VALUES ('6'),
PARTITION seven_alexas VALUES ('7'),
PARTITION eight_alexas VALUES ('8'),
PARTITION nine_alexas VALUES ('9'),
DEFAULT PARTITION other
);
```

Appendix C

Product Documentation

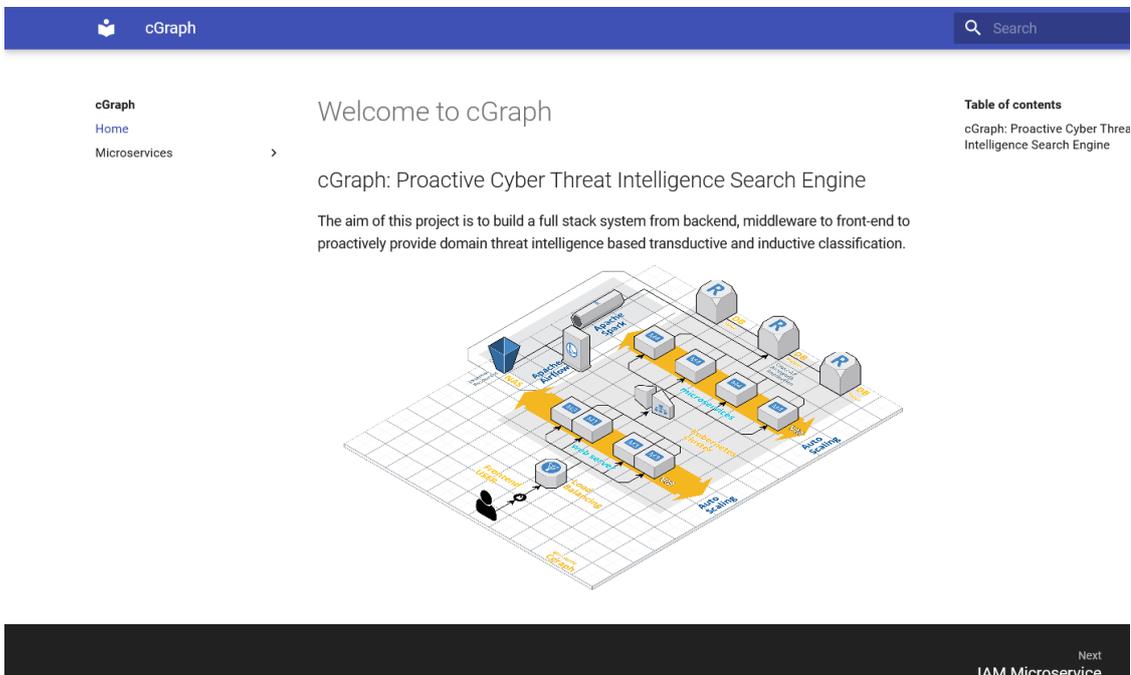


Figure C.1: Documentation - MkDocs

Appendix D

System View

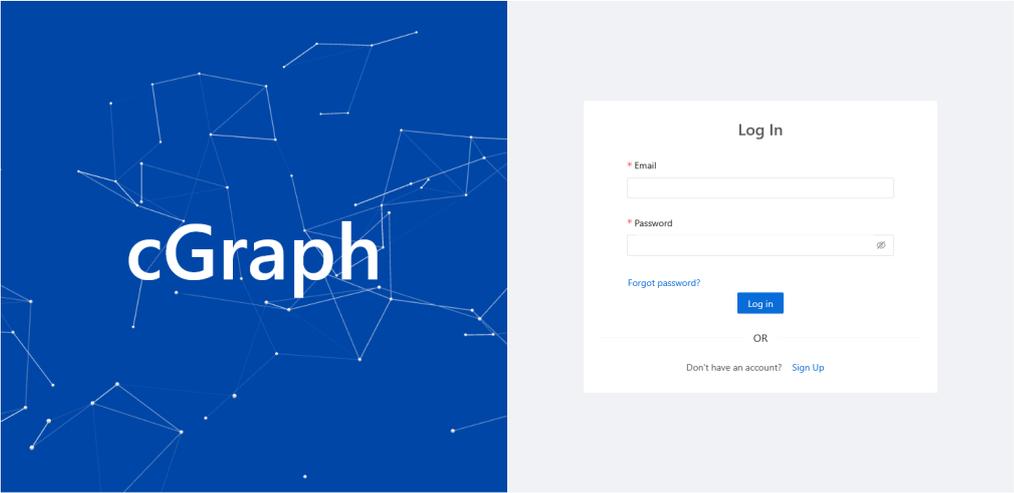


Figure D.1: SignIn

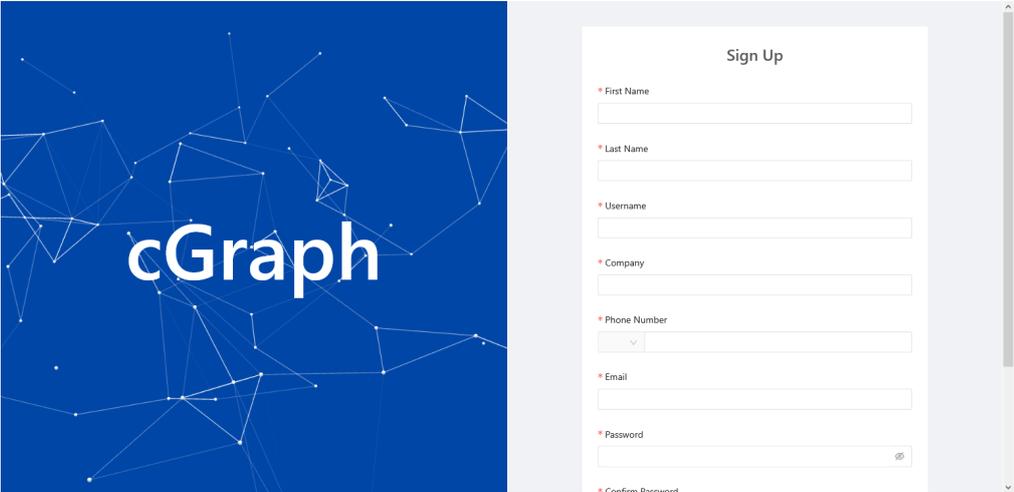


Figure D.2: SignUp

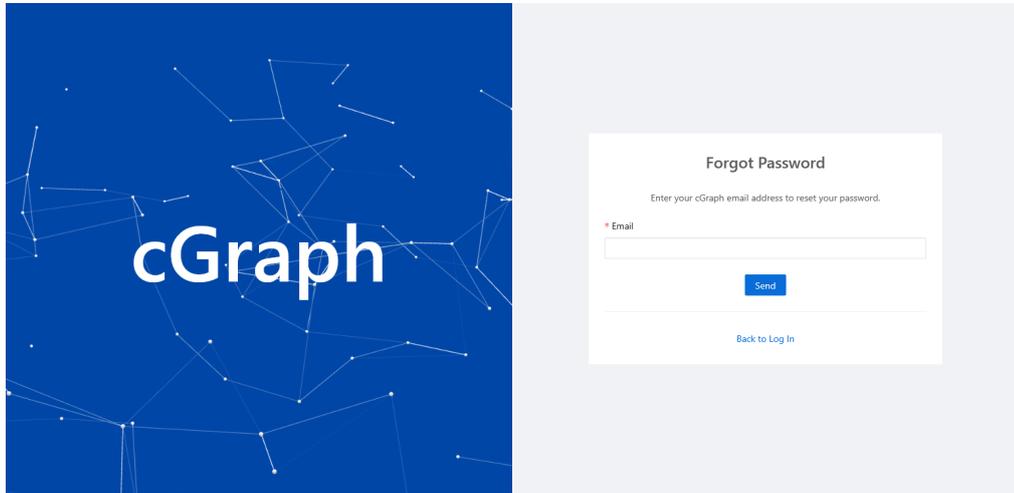


Figure D.3: Forgot Password



Figure D.4: Search View

Filters

Sort by :

- First Seen
- Last Seen
- Positives
- Alexa Rank

Sort direction :

- Ascending
- Descending

Record type :

- All
- Apex
- FQDN
- Nameservers
- Mailservers
- SOA

paypal-accounts-verify.com.		Apex		
16	Unknown	Malaysia	First Seen: 2020/7/31	Last Seen: 2021/2/15
AS55639 Asia Web Service Ltd				
paypalalert.org.		Apex		
13	Unknown	Kansas City United States	First Seen: 2020/8/13	Last Seen: 2021/0/26
AS15169 GOOGLE				
paypal-assist.com.		Apex		
11	Unknown	Singapore	First Seen: 2019/10/25	Last Seen: 2020/10/25
AS45102 Alibaba (US) Technology Co., Ltd.				
paypal-adder-generator.com.		Apex		
10	Unknown	Ashburn United States	First Seen: 2019/8/2	Last Seen: 2021/0/19
AS26496 AS-26496-GO-DADDY-COM-LLC				
paypal3-ankjdns.com.		Apex		
9	Unknown	United States	First Seen: 2020/7/22	Last Seen: 2021/1/6
AS46606 UNIFIEDLAYER-AS-1				
paypal2go.com.		Apex		
8	Unknown	United States	First Seen: 2020/6/1	Last Seen: 2021/2/19
AS26496 AS-26496-GO-DADDY-COM-LLC				
paypal-assistance-secure.com.		Apex		
8	Unknown	Russian Federation	First Seen: 2020/5/20	Last Seen: 2020/9/2
AS49335 LLC Server v arendy				
paypal-account-support.com.		Apex		
7	Unknown	Germany	First Seen: 2015/6/3	Last Seen: 2021/2/17
AS47846 SEDO GmbH				
paypal1-dankjnwds.com.		Apex		
6	Unknown	United States	First Seen: 2020/7/22	Last Seen: 2021/1/19
AS46606 UNIFIEDLAYER-AS-1				
paypalcase24.com.		Apex		
6	Unknown	United States	First Seen: 2020/6/27	Last Seen: 2021/1/20
AS46606 UNIFIEDLAYER-AS-1				

Figure D.5: Search Results

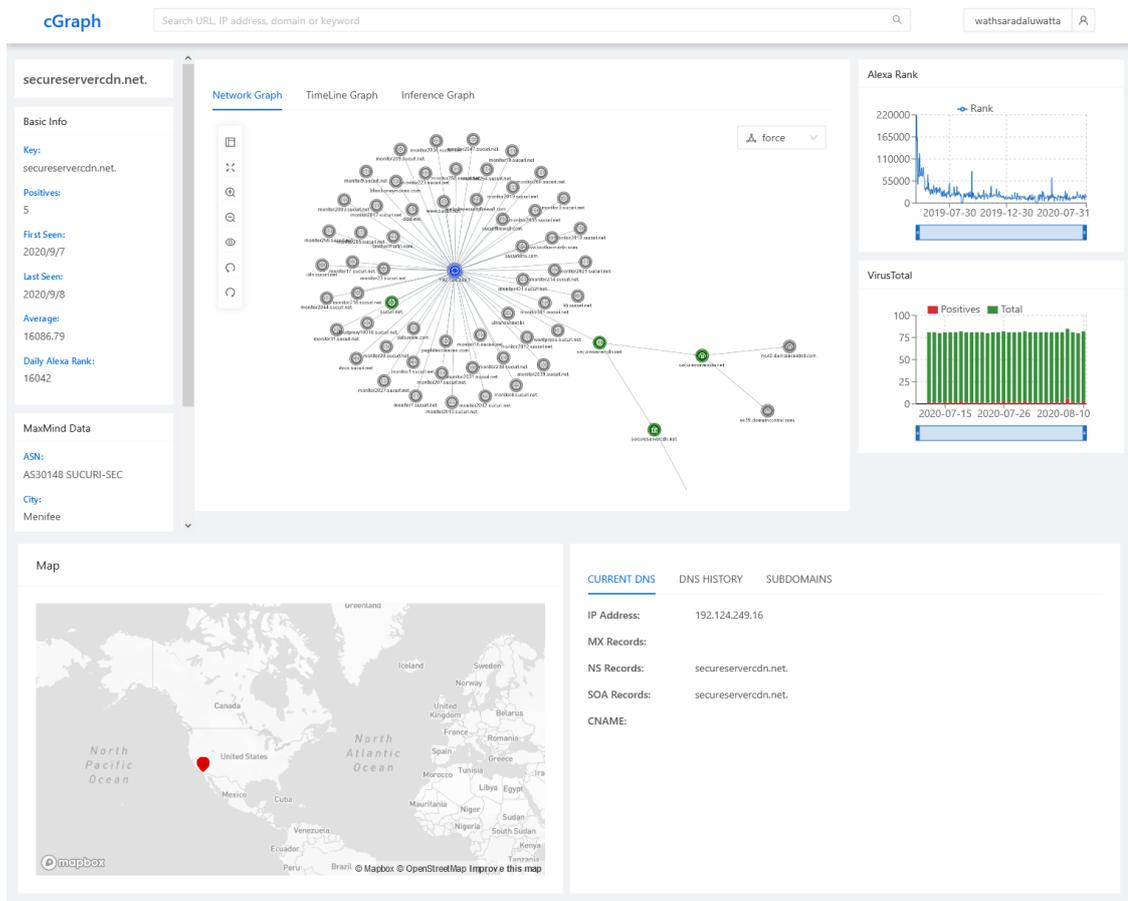


Figure D.6: Search Results

Appendix E

Container images

cgraphdcr / detail-microservice Updated 19 hours ago	Not Scanned	☆ 0	↓ 52	Public
cgraphdcr / krakend Updated 19 hours ago	Not Scanned	☆ 0	↓ 275	Public
cgraphdcr / webapp Updated 19 hours ago	Not Scanned	☆ 0	↓ 215	Public
cgraphdcr / auth-manager Updated 2 days ago	Not Scanned	☆ 0	↓ 105	Public
cgraphdcr / graph-traversal-microservice Updated 2 months ago	Not Scanned	☆ 0	↓ 62	Public

Figure E.1: Container images - DockerHub#1

cgraphdcr / detail-microservice Updated 19 hours ago	Not Scanned	☆ 0	↓ 52	Public
cgraphdcr / krakend Updated 19 hours ago	Not Scanned	☆ 0	↓ 275	Public
cgraphdcr / webapp Updated 19 hours ago	Not Scanned	☆ 0	↓ 215	Public
cgraphdcr / auth-manager Updated 2 days ago	Not Scanned	☆ 0	↓ 105	Public
cgraphdcr / graph-traversal-microservice Updated 2 months ago	Not Scanned	☆ 0	↓ 62	Public

Figure E.2: Container images - DockerHub#2

Appendix F

External References

Microbenchmark codes and results used in cGraph project can be accessed by using the following link:

<https://github.com/Wathsara/graphdb-benchmark/tree/main>