Aalto University
School of Science
Master's Programme in Security and Cloud Computing (SECCLO)

Data aggregation for multi-instance security management tools in telecommunication network

Bipin Khatiwada

Master's Thesis
Espoo, July 28, 2023

Supervisors:	Professor Antti Ylä-Jääski, Aalto University
	Professor Raja Appuswamy, EURECOM
Advisors:	Juha Törrönen, Oy L M Ericsson Ab
	Ngadhnjim Plaku (M.Sc.), Oy L M Ericsson Ab
	Jari Huoppila (M.Sc.), Oy L M Ericsson Ab

Aalto University
School of Science

Aalto University
School of Science
Master's Programme in Security and Cloud Computing (SECCLO)

Thesis Abstract

Author:	Bipin Khatiwada		
Title: Data aggregation for multi-instance security management tools in telecommunication network			
Date:	July 28, 2023	**Pages:**	73
Major:	Security and Cloud Computing	**Code:**	SCI3113
Supervisors:	Professor Antti Ylä-Jääski Professor Raja Appuswamy		
Advisors:	Juha Törrönen Ngadhnjim Plaku (M.Sc.) Jari Huoppila (M.Sc.)		

Communication Service Providers employ multiple instances of network monitoring tools within extensive networks that span large geographical regions, encompassing entire countries. By collecting monitoring data from various nodes and consolidating it in a central location, a comprehensive control dashboard is established, presenting an overall network status categorized under different perspectives.

In order to achieve this centralized view, we evaluated three architectural options: polling data from individual nodes to a central node, asynchronous push of data from individual nodes to a central node, and a cloud-based Extract, Transform, Load (ETL) approach. Our analysis leads us to the conclusion that the third option is most suitable for the telecommunication system use case.

Remarkably, we observed that the quantity of monitoring results is approximately 30 times greater than the total number of devices monitored within the network. Implementing the ETL-based approach, we achieved favorable performance times of 2.23 seconds, 7.16 seconds, and 27.96 seconds for small, medium, and large networks, respectively. Notably, the extraction operation required the most significant amount of time, followed by the load and processing phases. Furthermore, in terms of average memory consumption, the small, medium, and large networks necessitated 323.59 MB, 497.34 MB, and 1668.59 MB, respectively. It is worth noting that the relationship between the total number of devices in the system and both performance and memory consumption is linear in nature.

Keywords: Data aggregation, ETL, Monitoring tool, Network monitoring

Language: English

EURECOM

Master's Programme in Security and Cloud Computing (SECCLO)

Résumé de thèse

Auteur:	Bipin Khatiwada		
Titre: Data aggregation of multi-instance security management tools in telecommunication network			
Date:	28 Juillet 2023	**Pages:**	73
Spécialisation:	Big Data Security	**Promo:**	2023
Superviseurs:	Professeur Antti Ylä-Jääski Professeur Raja Appuswamy		
Conseillers:	Juha Törrönen Ngadhnjim Plaku (M.Sc.) Jari Huoppila (M.Sc.)		

Les fournisseurs de services de communication utilisent plusieurs instances d'outils de surveillance de réseau au sein de vastes réseaux couvrant de grandes régions géographiques, englobant des pays entiers. En collectant les données de surveillance à partir de différents nœuds et en les consolidant dans un emplacement central, un tableau de bord de contrôle complet est établi, présentant l'état global du réseau catégorisé selon différentes perspectives.

Afin d'obtenir cette vue centralisée, nous avons évalué trois options architecturales : la collecte de données en interrogeant individuellement les nœuds pour les transmettre à un nœud central, la transmission asynchrone de données depuis les nœuds individuels vers un nœud central, et une approche basée sur l'Extraction, la Transformation et le Chargement (ETL) dans le cloud. Notre analyse nous conduit à la conclusion que la troisième option est la plus adaptée pour le cas d'utilisation du système de télécommunication.

Remarquablement, nous avons observé que la quantité de résultats de surveillance est d'environ 30 fois supérieure au nombre total de dispositifs surveillés dans le réseau. En mettant en œuvre l'approche basée sur l'ETL, nous avons obtenu des temps de performance favorables de 2,23 secondes, 7,16 secondes et 27,96 secondes respectivement pour les réseaux de petite, moyenne et grande taille. Il est à noter que l'opération d'extraction a demandé le temps le plus important, suivi du chargement et des phases de traitement. De plus, en termes de consommation moyenne de mémoire, les réseaux de petite, moyenne et grande taille ont nécessité respectivement 323,59 Mo, 497,34 Mo et 1668,59 Mo. Il convient de noter que la relation entre le nombre total de dispositifs dans le système et les performances ainsi que la consommation de mémoire est de nature linéaire.

Mots clés: Agrégation de données, ETL, Outil de monitoring, Monitoring réseau

Langue: Anglais

Acknowledgements

I would like to express my sincere gratitude to the European Union for granting me a place in the Erasmus Mundus program with the Erasmus+ scholarship. I am immensely thankful to Professor Antti Ylä-Jääski from Aalto University and Professor Raja Appuswamy from EURECOM for their invaluable supervision. I am also grateful to Juha Törrönen, Ngadhnjim Plaku, and Jari Huoppila from Ericsson for their guidance and support.

Studying and collaborating with the SECCLO 2021 family has been an incredible academic experience filled with joyful memories that will stay with me for a lifetime. I would like to express my heartfelt appreciation to my dear family for their unwavering support and belief in me. I also want to acknowledge all my friends from IOE Pulchowk Campus, Pinnacle Campus, and Intensive International Academy for their friendship and camaraderie.

Last but not least, I extend my sincere gratitude to Aalto University in Finland, EURECOM in France, Amazon in Luxembourg, and Ericsson in Finland for providing me with exceptional platforms for my studies, professional growth, and numerous opportunities that have shaped my career.

Espoo, July 28, 2023

Bipin Khatiwada

With the support of the
Erasmus+ Programme
of the European Union

Aalto University
School of Science

EURECOM
Sophia Antipolis

DECLARATION FOR THE MASTER'S THESIS

I warrant, that the thesis is my original work and that I have not received outside assistance. Only the sources cited have been used in this report. Parts that are direct quotes or paraphrases are identified as such.

In Espoo, Finland
Date: 28 July, 2023

DECLARATION POUR LE RAPPORT DE STAGE

Je garantis que le rapport est mon travail original et que je n'ai pas reçu d'aide extérieure. Seules les sources citées ont été utilisées dans ce projet. Les parties qui sont des citations directes ou des paraphrases sont identifiées comme telles.

À Espoo, Finlande
Date: 28 Juillet, 2023

Bipin Khatiwada

Abbreviations and Acronyms

4G	Fourth Generation
5G	Fifth Generation
6G	Sixth Generation
ACID	Atomicity, Consistency, Isolation, and Durability
API	Application Programming Interface
AWS	Amazon Web Service
BBU	Baseband unit
CAMS	Central Aggregation and Monitoring System
CC	Compliance Check
CCG	Compliance Check Group
CCR	Compliance Check Run
CCP	Compliance Check Plan
CMS	Cloud Monitoring System
CPU	Central Processing Unit
CSP	Communication Service Provider
DBMS	Database Management System
DoS	Denial of Service
DW	Data Warehouse
EL	Extract, Load
ELT	Extract, Load, Transform
ESM	Ericsson Security Manager
ETL	Extract, Transform, Load
GB	Giga Byte
GDPR	General Data Protection Regulation
GCP	Google Cloud Platform
GGSN	GPRS Support Node
GPRS	General Packet Radio Service
HTTP	Hypertext Transfer Protocol
HSS	Home Subscriber Server
ICT	Information and Communication Technology

IDPS	Intrusion Detection and Prevention System
I/O	Input Output
IoT	Internet of Things
ISP	Internet Service Provider
MB	Mega Byte
MME	Mobile Management Entity
MSC	Mobile Switching Center
NAC	Network Access Control
NMT	Network Monitoring Tool
NMS	Network Management Service
OLAP	Online Analytical Processing
OLTP	Online Transaction Processing
PGW	Packet Data Network Gateway
QoS	Quality of Service
RAN	Radio Access Network
RF	Radio Frequency
RDBMS	Relational Database Management System
S3	(Amazon) Simple Storage Service
SGW	Serving Gateway
SIEM	Security Information and Event Management
SQL	Structured Query Language

Contents

Chapter 1

Introduction

The work presented in this thesis is a part of the research conducted at Ericsson Oy, Finland. The primary objective of this research is to explore the development of a network monitoring tool at Ericsson that can establish a central monitoring system within a multi-instance deployment scenario. In order to preserve the confidentiality of proprietary information, certain component names and business-sensitive details have been anonymized or represented in alternative terms.

Large-scale enterprises, such as Communications Service Providers (CSPs), Internet Service Providers (ISPs), national health record systems, and military networks, offer online services to a vast user base distributed across diverse geographic locations. These services encompass various functionalities such as secure communication establishment, security and incident monitoring systems, and data collection. Often, these systems consist of multiple nodes or service instances operating in close proximity or across different geographic regions. Irrespective of their operational mode (distributed, single-instance, multi-node, geo-redundant), these systems handle massive volumes of data.

While these enterprises make use of public cloud service providers to some extent, the majority of their system components reside in private clouds or on in-house infrastructure. This preference is primarily attributed to the sensitive nature of the services and data they handle. For example, telecommunication providers process highly sensitive communication data and user records pertaining to a large population, which constitutes sensitive information for any nation. Similarly, military networks isolate their data and services from the public cloud due to national security concerns. Utilizing public cloud services involves placing trust in corporations such as Google, Amazon, Microsoft, or other cloud providers that may be owned, influenced, or controlled by foreign governments. Therefore, private cloud infrastructure is more commonly employed in enterprise systems of this nature.

These services must meet stringent requirements for high availability and data confidentiality. Consequently, they are time-critical in nature and necessitate extensive monitoring and data analysis to ensure uninterrupted service provision and safeguard sensitive data. Data is often transmitted from various locations to a central agent, such as a headquarters or control room. Time-critical information, such as indications of active Denial of Service (DOS) attacks, unauthorized access, or suspicious node behavior within the system, must be promptly available within a short timeframe, ranging from milliseconds to a few minutes. This enables swift responses to threats. However, non-critical information, such as data for analytical purposes, does not require real-time processing. In many cases, summarized data is sufficient for the central agent, as opposed to the entire dataset. Regardless, enterprises in this domain require the capability to aggregate information from different geographic instances or subsystems in order to obtain a comprehensive system perspective. A Central Aggregation and Monitoring System (CAMS) serves as a crucial tool for providing an aggregated overview of the system and acting as a central repository of information.

1.1 Problem Statement

Designing a CAMS for any given use case poses significant challenges, primarily stemming from the vast amount of big data collected from each instance, which presents diverse characteristics and is inherently difficult to process. The real-time aggregation and processing of this data compound the difficulty due to the sheer volume of raw data and logs that are collected. Consequently, it becomes imperative to determine which data is crucial for a given use case and how to efficiently process it. However, it remains crucial to obtain a exhaustive picture of the entire network in order to establish robust security measures and monitoring capabilities. Key decisions must be made regarding the optimization of the processing workflow, the selection of pertinent data to be collected, the execution of the processing operations, and the presentation of relevant information on the dashboard.

Another significant challenge lies in ensuring compliance with customers' data privacy requirements and various regulatory frameworks. For instance, data collected from the European Union region must adhere to the General Data Protection Regulation (GDPR) policy. In addition, health data collected by provinces or hospitals may be subject to local regulations mandating the anonymization of data, enabling the central state to access statistical information pertaining to each province, while restricting access to specific patient details to within a particular hospital or municipality. Consequently, any utilization of public cloud services must be approached with meticulous planning and stringent considerations regarding data privacy regulations.

1.2 Study Focus and Limitations

The thesis will focus on designing the CAMS in the context of security monitoring tool for CSP network. It will evaluate the different architectures and provide a detail analyses on the tools itself along with the features. Following are the main objectives of the thesis:

– Implementation and analytics decision specific to the CSP.

– Evaluate architectural choices for aggregating data from multiple nodes.

– Suggest effective data structure and data pipeline to obtain a holistic network view through the CAMS dashboard.

In order to achieve the speculated outcome, we leverage the use of public cloud, however the main objective is to validate the implementation idea and not about fixating the choice to any (public) cloud provider. The implementation can be replicated in any private cloud, if required. The behavior and implementation of each node in CAMS is out of the scope.

1.3 Research Questions

The aim of this thesis is to address the following research questions pertaining to the design and development of a CAMS tailored specifically for CSP networks:

- Q1. Which cloud architecture represents the optimal choice for constructing a CAMS for CSP networks, and how do three popular architectural options compare in this context?

- Q2. What are the essential considerations in terms of infrastructure design, data structure, data pipeline, and data visualization components within the CAMS framework?

- Q3. How does system performance and resource consumption evolve when scaling up the CAMS to handle increasing demands?

These research questions serve as the focal points of investigation in order to provide comprehensive insights into the design and operational aspects of a CAMS tailored specifically for CSP networks. By addressing these questions, this research aims to contribute to the body of knowledge concerning the efficient and effective implementation of CAMS within the context of CSP networks.

1.4 Thesis Structure

The thesis will introduce key concepts and technologies in Chapter 2 - necessary for understanding telecommunication network and CAMS. The last section of this chapter provides the systematic study of the related work along with exploring the current status and trend of data aggregation approaches. Chapter 3 explains the system in detail. It introduces the key elements in the system and how the nodes we use to collect data for designing the CAMS operates. After that, we analyze the requirements and constraints. We then present the architectural design choices considered along with the reasoning behind. Chapter 4 will discuss the implementation procedures and describe how it is set up for testing purpose. Following this, Chapter 5 will then evaluate the performance by defining metrics and different case scenarios. We then analyze the output with different graphs of data visualization. Finally, Chapter 6 will discuss the findings, evaluate strength and weakness of the system, answer the research questions, put the spotlight on the contribution of the work to the company, followed by the possible future work. In the end, we provide concluding remarks.

Chapter 2

Background

This chapter presents a comprehensive overview of mobile telecommunication systems, encompassing aspects such as security, network monitoring, security management, and the utilisation of contemporary data technologies for data aggregation. It also provides the indepth analysis of the related works.

2.1 Telecommunication network

Telecommunication networks encompass intricate systems that establish connections among diverse devices, including mobile phones, landlines, and the Internet of Things (IoT). It facilitates the transmission of a significant amount of wireless communication worldwide, at any given time. The applications of mobile communication extend beyond phone calls and includes various functionalities such as satellite communication, remote sensor data retrieval, internet communication, and other diverse applications in contemporary society. The proliferation of mobile device users has led to the development and expansion of more robust communication infrastructures.

The Radio Access Network (RAN) infrastructure facilitates connectivity for various mobile devices, including phones, sensors, and Internet of Things (IoT) devices, by establishing a link to the primary communication system known as the core network. RAN assumes the responsibility of overseeing the management of radio resources, ensuring data security, and facilitating device mobility [45]. The three primary components that are considered crucial are antennas, radios, and baseband units (BBUs).

The BBU is responsible for performing signal processing functions that enable the transmission and reception of wireless and wired communication. The device is capable of receiving radio signals from the core network and executing various digital signal processing functions, including modulation,

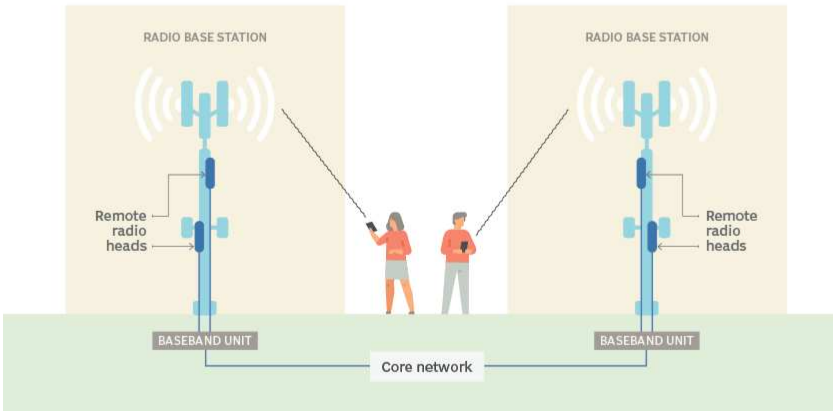


Figure 2.1: Basic RAN Architecture [45]

demodulation, encoding, decoding, encryption, and decryption. The concept of the layer exists as an intermediary component facilitating the transmission of signals between analogue and digital communication systems. Additional responsibilities of it involve functions such as data compression, resource allocation pertaining to frequency, time slots, power levels, and scheduling, interference management, mobility management, and Quality of Service (QoS) management. Therefore, it is an essential component within the RAN.

The mobile core network bundles numerous functionalities essential for establishing and maintaining communication. As shown in the figure 2.2, the core connects to multiple cell regions and manages the resources and communication policies. Some of its purposes include [47]:

- Authenticating the end devices connected to the network
- Register, Identify, Track subscriber requests, billing, charges, etc.
- Track subscriber mobility, base station hopping during connection for continuous service.
- Ensure QoS in connectivity.

Different devices are involved in the core network, including the Mobile Switching Centre (MSC), Serving GPRS Support Node (GGSN), Mobility Management Entity (MME), Serving Gateway (SGW), Packet Data Network Gateway (PGW), and Home Subscriber Server (HSS). Each device has a significant role in keeping the service alive, secure, and functioning well. The specific architecture and details depend on the network generation: 2G, 3G, 4G, and 5G. Details about them are not required for this thesis. The range of a cell phone tower is 40 km in terms of usability; however, it is usually set to cover 1.5 km to 5 km, and switching the user's connection to another cell tower happens over 0.8 to 1.6 km [43]. The cell tower range depends on the population, type of antenna used, type of signal used (radio frequency (RF) wave, millimetre wave, to name a few), choice of capacity vs. range, geography, tower position, and a lot of other technical factors. This implies that there are thousands of mobile phone base stations in a city. There are about 7 million physical cell sites world-wide and about 10 million logical sites [44]. In the typical scenario, a singular core encompasses the base bands of a city or multiple proximate cities. This core, denoted as the "network size," is contingent upon the geographical extent it covers and the quantity of base stations it interconnects. The dimensions of this core may differ depending on the operator, the population served, and the geographic region encompassed. We categorize operators as small, medium, or large based on their respective network sizes, and the distinctive characteristics of each category are outlined in section 3.2.

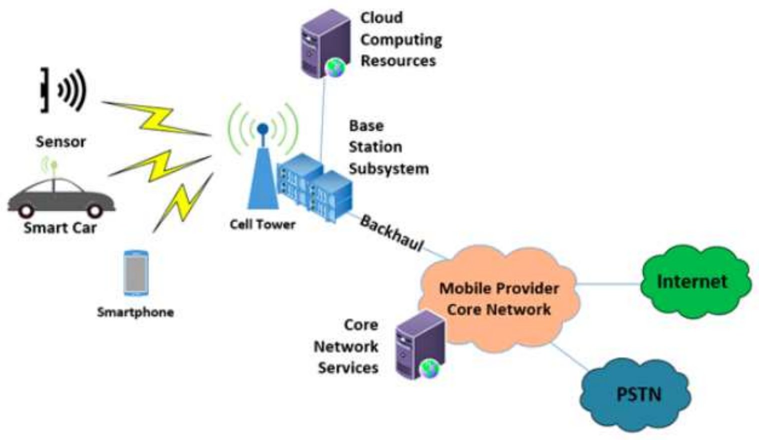


Figure 2.2: Core Network architecture [46]

.

2.2 Security of telecommunication network

Security in the telecommunication network involves a wide range of surfaces, including the core, RAN, and end devices. In this thesis, we limit the security focus to the BBUs, devices, and network related to the core. Some examples of the attacks on these surfaces include:

- False baseband
- Jamming attack
- Distributed Denial of Service (DDoS)
- Unauthorised access to network devices

Security is a critical need instead of a mere feature for such infrastructure, as the cost of its downtime is significant, affecting wide services and the operation of businesses. Further, the security and privacy of the public are associated with the telecommunication network as it handles tremendous amounts of data, including user conversations, web browsing, sensor readings, etc. The impact even extends to a national-level security threat. Therefore, threat management and the enforcement of effective security policies are must. For this, understanding the overall system state is essential for quickly identifying attacks and taking action in any of the compromised system instances, which we will also refer to as "nodes" from here on. A good approach for this is a regular check of security compliance on each device. More about security compliance is described in 3.1.2.

2.3 Network monitoring and security management

Network monitoring guides the operators through continuous surveillance and analysis of network elements and other critical components. These components include routers, switches, and base stations. It helps to understand the state of existing network behaviour for the purpose of ensuring they operate on intended behaviour and troubleshoot any deviations [32]. Monitoring tools collect different metrics and evaluate the key performance indicators to identify the operation's health and potential issues.

Security management in a telecom network involves several measures to protect against threats and ensure integrity and confidentiality. These mechanisms include the compliance of each network component in terms of security protocols, firewalls, intrusion detection and prevention, encryption and decryption, encoding and decoding, firewalls, routing, access controls, authentication measures, customer information, operation information, connection details, communication tunnels, and more. Management is done by implementing secure policies, audits, threat assessment, unusual behaviour detection, and incident response procedures.

There are several tools available for these purposes. These include services such as firewalls, Intrusion Detection and Prevention systems (IDPS), Security Information and Event management (SIEM) solutions, Anti-malware and Anti-virus solutions, and Network Access Control (NAC) systems. Different vendors incorporate one or many such services into their products. Specific to telecom network security and management, some of the market leaders include: "Ericsson Security Manager (ESM) [40] from Ericsson, "NetGuard Cybersecurity Dome" [41] from Nokia, "eSight ICT Unified Management Systems" [42] from Huawei, and "Cisco Security Manager" [48] from Cisco.

2.4 Distributed databases

A distributed database refers to a system whose data is stored in multiple storage locations. The storage location could be in a different database management system, in several geographic locations, or in a heterogeneous structure. The traditional single database is not sufficient to provide most of the services that are global or enterprise-scale. The number of users, computing power, sheer volume of data, and service requirements demand the scaling of databases. Besides, in order to separate the data based on the use case or security domain, it is often required to have a separate database instance per domain.

In the context of telecommunication, an operator can have a database close to their core network in order to facilitate the service quicker. Similarly, in the use of the health care system, separate databases per region (hospital, district, region) can be maintained separately so as to limit access. For use cases like streaming and social sites, distributed databases help serve contents to the customer fairly quickly by keeping the data source close to the user. In a nutshell, having distributed databases in many use cases provides benefits of performance improvement, massive scalability, and round-the-clock reliability [49].

Enterprise systems with multi-instances (having multiple system nodes) - as in the CSP network - keep their distributed database in the core system and could manage a central database to store the data of each core. Depending on the architecture, some data may only remain in the core, while others may be shared between the core and the central database. The advantage of this central data collection is that it provides data backup and availability to multiple nodes in different regions, thus providing a holistic picture of the system.

Having a central node that collects data from each sub-node is a good approach for central monitoring and getting an overall picture of the network.

However, it is required to consider if it affects the privacy of users at each node. And, once collected, access control is to be considered at the central node, as access to data for any user might be limited to some nodes.

2.5 Data management technologies for multi-instance architecture

As explained in the previous section, a system based on a multi-instance architecture may need to aggregate data, either fully or partially. There are different technologies and services available that cater to specific business needs. In this section, we will discuss the major ones, which include:

Data Federation

Data federation is a concept in software processes where multiple independent databases act as one under an abstract query layer. Despite data being stored in multiple locations, a query executed will combine the results of multiple data sources, combine them, and produce one single output view. The data federation helps in reducing storage space, maintaining a single source of truth, minimal coding, and easy access control [50]. Data is not physically moved; instead, the top-level component dealing with the main query calls multiple databases with subqueries. With this, there are some other challenges, such as additional overhead due to the need to integrate data from distributed sources, hassle in maintaining data consistency across multiple sources, maintaining consistent security and access control, dependency on source availability, and limited offline access.

Data Lake and Data Warehouse

Data lakes and data warehouses serve as centralized repositories for storing large volumes of data. While they both serve this purpose, they differ in their approach and functionality.

A data lake is designed to store raw data in its original format, accommodating structured, semi-structured, and unstructured data. It can store a wide variety of data types, including tables, files, social media data, sensor data, and graphs. The primary objective of a data lake is to provide a platform for exploratory and ad hoc analysis, allowing users to access and analyze data in its raw form. This flexibility enables data scientists and analysts to perform in-depth exploration and experimentation with the data, without the need for predefined structures or schemas.

On the other hand, a data warehouse is a structured environment that maintains data in a historical and consolidated form. It organizes and optimizes the data to improve query performance and is primarily geared towards Business Intelligence tasks[51]. A data warehouse structures the data according to predefined schemas, transforming and aggregating it to facilitate efficient querying and reporting. Data warehouses are well-suited for predefined queries and predefined analytics, providing a reliable and consistent foundation for generating business insights.

OLAP and OLTP

Online Transaction Processing (OLTP) is the data processing and management approach that supports the real-time needs of the system by fetching the data on demand for processing the service's immediate requests. OLTP focuses on processing high-volume transactional data to perform day-to-day business transactions. Use cases include applications such as e-commerce, banking, inventory, and order management. They have ACID (Atomicity, Consistency, Isolation, and Durability) properties, handle concurrency, have a fast response time, and typically follow data normalisation [39]. OLAP (Online Analytical Processing) is a technology that powers complex, multi-dimensional data analysis. A data warehouse, or Data lake, is the primary database technology used for OLAP. They are therefore primary for business intelligence and decision support. They allow you to analyse data from multiple dimensions, split data sources for insights, deduce relationships, aggregate, consolidate, and optimise for specific needs [39].

Data Mesh

DataMesh is an emerging approach to data engineering and management. Traditionally, distributed data has been gathered in a central location and processed for OLAP. With datamesh, the responsibility of managing data and its access control is separate by its domains. Data is organized into domains and data teams is responsible to carry out the processing in their specific product-oriented way [36]. It advocates for the domain team to take ownership and responsibility of the data instead of one central process that manages or process the data. Each team can, however, use the same common infrastructure to process the data. Therefore, the concept also goes hand in hand with Data Federation. Well-defined APIs and data contract ensure the domain team can provide required data upon requested by other teams. This gives autonomy and accountability over own data.

ETL, EL and ELT data pipelines

"Extract, Transform, and Load (ETL)" and "Extract, Load, and Transform (ELT)" are the two data processing approaches that can be implemented in a centralised repository or data mesh architecture. [37] defines each term as: Extract refers to the process of fetching a subset of information from the data source to process. Transform is the actual processing of data, which depends on the business need for data. Load refers to the storage of the information in the information distribution centre. Figures 2.3 and 2.4 show the flow diagram of ETL and ELT.

ETL, or Extract, Transform, Load, is a data integration process where data is extracted from various sources, transformed into a consistent structure, and then loaded into a target data warehouse [52]. The transformation step involves tasks like data pruning, cleaning, converting, aggregating, and applying business rules. Typically, ETL is performed on a separate server or as a separate process, and it is commonly used for handling batch-oriented data.

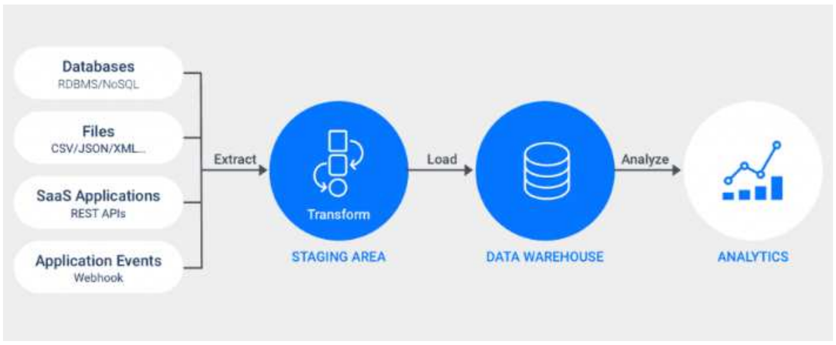


Figure 2.3: ETL process[52]

In contrast, ELT is a relatively new approach to data integration [52]. With ELT, data is extracted from the source and directly loaded into the target system. The transformation process takes place within the target system itself, leveraging its processing capabilities. This can be a data lake or a cloud data platform. Since the data transformation happens on the fly, users have the flexibility to view the raw data and decide whether it needs to be transformed. This approach enables more agile data processing, as decisions can be made based on the initial state of the data.

Lastly, in the EL (Extract and Load) process, data is consumed as it is without significant transformations [52]. This approach is best suited when the data is already clean and accurate, requiring minimal modifications.

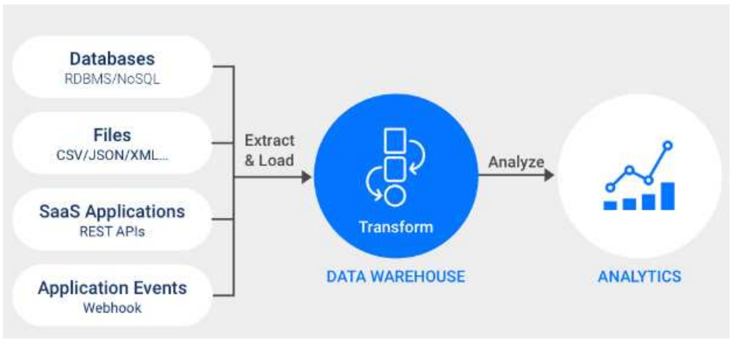


Figure 2.4: ELT process[52]

2.6 Data management services in cloud

Major cloud providers offer a comprehensive range of data management services, as previously discussed and beyond, under various names and branding. Amazon Web Service (AWS), Google Cloud Platform (GCP), and Microsoft Azure are widely recognised as the dominant players in the cloud service provider industry. The cloud offers various data management options such as Infrastructure as a Service (IaaS), Software as a Service (SaaS), and Platform as a Service (PaaS), which can be selected based on the specific needs of the organisation. Table 2.1 displays a selection of prevalent data management services offered by cloud providers.

Table 2.1: Data Services Provided by AWS, GCP, and Azure [33–35]

Data Service Type	**AWS**	**GCP**	**Azure**
Relational Databases	Amazon RDS, Amazon Aurora	Cloud Spanner, Cloud SQL	Azure SQL Database
NoSQL Databases	Amazon DynamoDB	Cloud Firestore, Cloud Bigtable	Azure Cosmos DB
Data Warehousing	Amazon Redshift	BigQuery	Azure Synapse Analytics, Azure Data Warehouse
Object Storage	Amazon S3	Cloud Storage	Azure Blob Storage
Data Streaming	Amazon Kinesis	Cloud Pub/-Sub	Azure Event Hubs, Azure Stream Analytics
Data Integration	AWS Glue	Cloud Dataflow, Cloud Composer	Azure Data Factory, Azure Data Catalog
Big Data Processing	AWS Lake Formation, AWS Glue	Cloud Dataproc, Cloud Dataflow	Azure Databricks, Azure HDInsight
Data Migration	AWS DataSync, AWS Database Migration Service	Cloud Data Transfer Service, Cloud Storage Transfer Service	Azure Data Box, Azure Data Factory
Search and Analytics	Amazon Elasticsearch Service	Cloud Datalab, Cloud Data Catalog	Azure Data Lake Analytics, Azure Analysis Services
Others	AWS Snowball, AWS Data Pipeline	Cloud Memorystore, Cloud Filestore	Azure Data Lake Storage, Azure Queue Storage, Azure Table Storage

2.7 Data aggregation using GCP

The understanding of various data management methodologies in Google Cloud Platform (GCP) can potentially be applied to analogous cloud service providers. In the realm of data aggregation from multiple nodes, Google Cloud Platform (GCP) provides a range of services that effectively support the management and aggregation of data. Several significant services that are pertinent to this thesis are outlined here:[53]

Cloud Pub/Sub: This fully managed real-time messaging service enables dependable and asynchronous communication between various distributed system components. This platform facilitates the dissemination of messages on specific subjects by data producers, which can subsequently be subscribed to by data consumers for the purposes of processing and aggregation.

Cloud Dataflow is a comprehensive managed service designed to facilitate the execution of parallel data processing pipelines. The utilisation of dataflow allows developers to construct data aggregation workflows that have the capability to efficiently process and aggregate data from various sources in a distributed and scalable fashion. The system offers support for both batch and stream processing modes, thereby offering flexibility in accordance with the data aggregation needs.

BigQuery, developed by Google, is a serverless data warehouse that provides robust querying functionalities for conducting extensive data analysis at scale. Data aggregation can be achieved through the execution of SQL queries across multiple nodes or partitions, enabling users to gather data from diverse sources and derive significant insights.

Cloud Dataproc is a managed service that facilitates the processing and analysis of large-scale data through the utilisation of Apache Hadoop and Apache Spark. Users are able to utilise distributed processing frameworks in order to consolidate data from multiple nodes and carry out intricate data transformations and analysis.

Data Fusion is a fully managed data integration service provided by GCP. This service streamlines the tasks of data ingestion, transformation, and aggregation from multiple sources. The platform offers a graphical user interface that facilitates the construction of data pipelines, enabling users to seamlessly consolidate data from various nodes or systems.

The services offered by Google Cloud Platform (GCP) encompass the necessary infrastructure, tools, and capabilities for efficiently consolidating data from multiple nodes. These systems provide the advantages of scalability, reliability, and user-friendliness, allowing organisations to effectively manage substantial amounts of data and derive valuable insights through the process of data aggregation.

2.8 Related Work

Reviewed literature perceives cloud database management as a computing service for handling cloud data [1]. It is guided by key design principles such as dynamic scalability, availability, and dynamic resource allocation. Various cloud database architectures exist to meet diverse requirements. Adam et al. [1] classify DBMS architectures into two types: shared nothing and shared disk. In the shared nothing approach, each node is self-sufficient and independent, while in the shared disk approach, each node can have independent memory but shared disk space. Another classification criterion is based on the measures of capacity adjustment. Cloud compute power is elastic, allowing for easier workload scaling. Scaling methods include scale up, where the database is deployed on a single virtual server instance, and scale out, where multiple instances of the same type are used, with the number of instances adjusted according to the workload [6].

Cloud Monitoring Systems (CMS) research focuses on monitoring and managing cloud activities. Birje et al. [10] provide a three-layered view of CMS: Infrastructure, Platform, and Application layers. The infrastructure layer monitors physical and virtual components such as CPU, memory, disks, and traffic. The platform layer focuses on platform and service-related metrics, such as response time, process count, VM count, threads count, and resource allocation per application. The application layer monitors system status metrics, including CPU utilization, query latency, CPU usage, and memory usage. Data generation in CMS systems can come from subnodes, cloud computing operations, log analysis, events, network management, and performance measures [10].

Silva et al. [11] propose an architecture for CAMS that comprises a message bus, producers, aggregators, and consumers. Their work emphasizes security and privacy considerations for the entire system and evaluates regional energy consumption, monthly charges, technical aspects, and homomorphic aggregators. The architecture design process begins with choosing between centralized and decentralized architectures. Monitoring, which includes collection, filtering, aggregation, analysis, and reporting, is the subsequent step.

Data collection involves gathering metrics from various subsystems into a central system. In a centralized system, all data is collected in a central node and processed, which is cost-effective but introduces a single point of failure and reduced scalability [21, 22]. In a decentralized system, CAMS acts as a lightweight client, querying information from subnodes. Literatures [23, 25–27] describes five strategies for data collection: Push, Pull, Hybrid Push, Hybrid Push Pull, and Adaptive Push. Updating data is a critical architectural decision, with two major strategies: Periodic Update, where data is updated at regular intervals [28], and Event-based Update, where data is updated when an event occurs [29]. Filtering strategies can be based on time, window, content, or threshold.

Aggregating data supports analytics, accurate results, resource optimization, and process tuning [23, 24]. The architecture of a data aggregation system depends on factors such as latency, feature requirements, security requirements, and the granularity of presented metrics [9–12]. Karthick et al. [12] conducted a survey on big data aggregation using the cloud, highlighting the importance of resource and data aggregation into data centers on the internet.

Data management has been studied in various use cases. Researchers [2, 4, 7, 8] have analyzed the use of cloud services for data management and processing. For instance, Pajic et al. [7] model data generated from laser scanning or photogrammetry with billions of points using the cloud. Chaudhary et al. [8] focus on managing IoT data using cloud services. Cooper et al. [4] analyze data engineering aspects in building Yahoo infrastructure, while Sangat et al. [2] explore cloud-based sensor data management. These applications generate vast amounts of data, requiring significant resources for processing and timely service delivery. Although the structure and processing of the data may differ across applications, the fundamental ideas of data transfer, processing, and loading are similar. Therefore, knowledge of data infrastructure in one use case can inspire the design of systems to handle Big Data.

Various tools and frameworks are used in different use cases. Previous works have employed MapReduce [3], graph theory [5], role-based approaches [1], and object-oriented approaches [18]. Tsangaris et al. [5] refer to complex tasks such as querying, search, information filtering, transformation, and analysis as rich tasks, typically represented by graphs with nodes and edges. Performance comparisons of multiple tools for cloud data management, including MongoDB, Apache Spark, Bigtable, Hadoop, and Sherpa, have been conducted in past studies [2–4, 13]. For instance, Sangat et al. [2] experiment with Apache Spark and native MongoDB for storing, ingesting, and retrieving sensor data, finding that Spark improves the ingestion process significantly and performs better than Mongo for reading and writing. Apache Spark is also shown to be effective for operations on point clouds, such as range and knn queries [7]. MapReduce and related software are designed for automating the parallelization of large-scale data analysis workloads, including Hadoop [4].

ETL and ELT are commonly used data management processes in the cloud. ETL involves applying transformations before loading data into a data warehouse for analysis, while ELT involves transforming the data at a later stage when needed. ETL is useful when end requirements and the data to be transformed are clear, but it involves significant I/O activity, string processing, and variable transformation [15]. Zdravevski et al. [13] analyze different dataflow scenarios in the ETL process that are common to many organizations, including scenarios involving no aggregation, aggregation in a fixed time period, and aggregation based on session. Cloud computing provides services that support the aforementioned stages and scenarios. Sreemathy et al. [15] present an overview of big data analytics using cloud computing, with a case study using Google's BigQuery. The study indicates that BigQuery is well-suited for complex analytical queries, often eliminating the need for simple aggregation and filtering.

Offloading data to the cloud poses its own challenges, requiring caution [3]. The security and privacy of aggregated data are crucial considerations. Privacy regulations in the data collection region, data storage region, data authenticity, and access control levels for different stakeholders are important to note [3, 9]. Access control on data can be influenced by local rules and regulations where the data is stored. For example, the US Patriot Act allows the government to demand access to data on any computer without consent or knowledge, even if it is hosted by a third party [3]. Similarly, GDPR affects the processing and management of European user data [31]. Businesses operating in multiple regions with varying data laws must comply with each regulation to operate legally. Therefore, careful consideration is required when choosing the region, cloud service provider, and database service provider [31]. Access management is a significant aspect of data security in the cloud. Abadi et al. [3] propose cloud database access and management based on three-schema and three-level object-oriented database architectures [18].

Zhang et al. [9] investigated a use case of priority-based health data aggregation with privacy preservation using the cloud. One example of an attack mentioned in the paper involves an attacker forging an emergency call to degrade network performance. The data collected itself can often include sensitive information, such as patient or user data. Approaches mentioned in the literature to mitigate these risks include encoding personally identifiable information (PII) before pushing it to the cloud, using homomorphic aggregators, employing private clouds, processing data using internal servers, and implementing role-based access control for processed results.

Chapter 3

System Study

In this section, we discuss the overview of the NMT tool of the company on which we base our study. We explain what the system is, its architecture, and the constraints and requirements of CAMS for this system.

3.1 System overview

The Network Monitoring Tool (NMT) is a software that monitors the status of the network and devices connected to it. NMT is a simplified security tool that is inspired by Ericsson Security Manager (ESM). The actual implementation of the NMT is out of scope for this thesis, as the implementation is concerned with the data output only. To uphold the secrecy of company's proprietary information, the terminologies and exact architecture of ESM are not used in the NMT node. However, the architectural considerations and data model of NMT are designed such that the knowledge gained from the result of the thesis output can be extrapolated to the multi-instance deployment of ESM.

Figure 3.1 shows the underlying physical architecture of the NMT system. It connects to a Network Management service, which connects to the different devices. Each NMT instance uses its own database to store the network monitoring results. The devices could be of different types. In the figure, device types "C", S," and "R" are shown.

Figure 3.2 shows where NMT is deployed in the physical network and how it fits in the CSP network for monitoring purposes. Each CSP node connects to thousands of cell stations. Each station has multiple devices necessary for signal coverage in the deployed region. One of the critical communication devices is the BBU. NMT connects to all the BBU devices connected to the node and many of the devices in the node itself. For the huge network

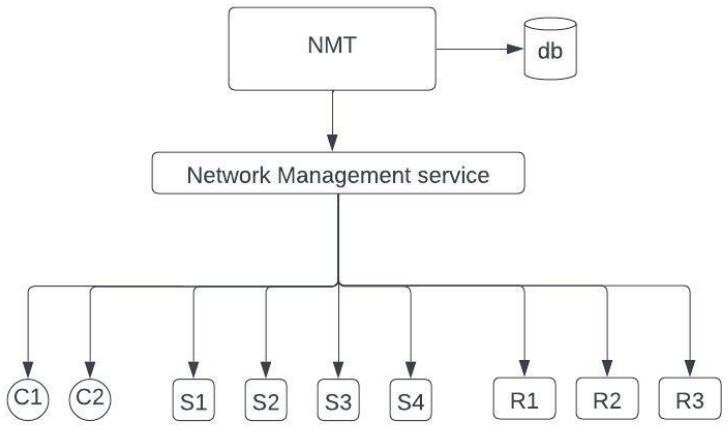


Figure 3.1: NMT physical architecture

node, multiple NMTs can be used to cover all the BBUs and node devices. The deployment can also be such that different NMT instances are used to connect different nodes. The deployment is specific to the business and operator requirements. This way, multi-instance NMT can also be deployed in geo-redundant scenarios. The fleet of such NMT instances can therefore cover the monitoring tasks of the entire network, regardless of the number of nodes and devices it has.

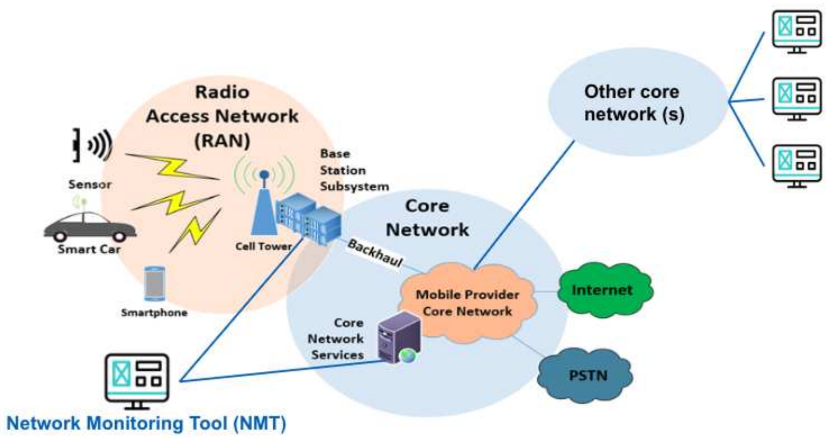


Figure 3.2: Multi instance NMT deployment

A Central Aggregation and Monitoring System (CAMS) is a platform to aggregate information from multiple instances of NMT and present a status overview of the entire network. As shown in figure 3.3, it communicates with each NMT. The protocol for communication with each NMT instance, number of invocations, type of data transferred, and information presented depend on the architectural design decisions, which will be the main focus of the thesis. CAMS, however, is not responsible for being involved in the monitoring and compliance check execution.

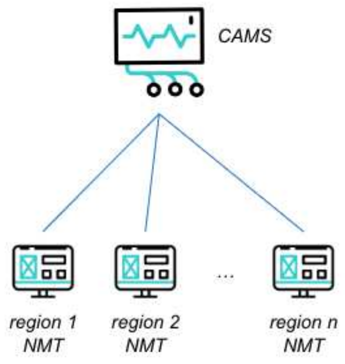


Figure 3.3: CAMS

Several terminologies and components of the NMT architecture are explained below:

3.1.1 Network Device and Device Type

Each device in the network is called a network device. In the context of NMT, a device is a physical or software element that can be tested or monitored for security policies. They are of different types, such as baseband, router, switch, firewall, network element, hub, and any such element that constitutes the telecommunication network. This does not include the end devices, for instance, mobile phones, antennas, and physical hardware devices in cell towers. In general, a network device has one or more modules, with each module integrated into it. Each module has exactly one gateway for external communication. These gateways are used directly or indirectly to perform the monitoring actions and connect devices to run the compliance checks.

Figure 3.4 shows an example of such a generic network device. The device of type D1 has 4 modules: M1, M2, M3, and M4, each of which has one gateway interface represented by ”g.”. The information associated with each network device includes the device ID, device type, number of modules,

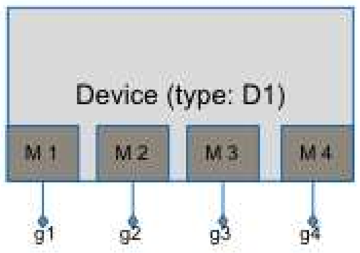


Figure 3.4: A generic Network Device

number of gateways per module, device-specific information, and supported communication protocols.

3.1.2 Compliance

Compliance is the actual security specification that the network device should adhere to. NMT will check the devices to see if they are in accordance with the guidelines enforced by “compliance”. Compliance includes information about:

- script to check compliance enforcement. This can be in Python, Ansible, or Shell.
- constraints to compare or implement
- supported device_type(s)

Examples of compliance can include “Password is of valid length”, “Login attempt count control”, and “SSH key validation”. Compliance can also be referred to as compliance Check (CC), which refers to the act of checking the compliance of a network device.

Compliance Example: Password is of valid length

```
Constraints:
    minimum_length = 8
    maximum_length = 32
    include_numbers = True
    include_special_characters = True
    special_characters = “!@$%ˆ&*()_+-[]—;:,.¡¿/?’#=”
Script:[1]
    password = env.get(“$PASSWORD”)
    has_numbers = any(char.isdigit() for char in password)
    has_special_chars = any(char in special_characters for char in password)
    pass_length_check = minimum_length ¡= len(password) ¡= maximum_length
    pass_numcheck = include_numbers or not has_numbers
    pass_specialcharcheck = include_special_characters or not has_special_chars if pass_length_check and pass_numcheck and pass_specialcharcheck then

end
    return “PASS” else

end
    raise Exception()
=0
```

Compliance Example 2: Login attempt control

```
Constraints:
    max_valid_attempt = 3
    successive_attempt_interval = 20s
    block_after_max_invalid_attempts = True
Script:
    ... // omited for brevity // ... =0
```

[1] *In a well-designed system, passwords are usually not saved in raw form and is not (or, should not be) accessible as shown in the example above. This script is for the demonstration purpose and is not related to any real world implementation.*

3.1.3 Compliance Check Group (CCG)

A compliance check group is a logical group of two or more compliance checks. All the related compliances can be grouped under a meaningful name. For example, one small CCG can be named "Incoming API Request CCG" and include compliances related to all the API calls being made to the service, such as "number of API calls per client for a given interval", "wait time between successive API calls", "maximum payload size", "accepted HTTP verbs," etc. CCG is often a broad umbrella term that can include tens to hundreds, if not thousands, of compliances. CCG also has information on what device types or lists of assets it can run compliance checks on.

3.1.4 Compliance Check Run (CCR)

Also known as Compliance Run. While performing a security check of a compliance against a device, the script will run in an isolated environment to check if the device conforms to all the constraints defined in the given compliance. This way, desired checks can be defined using compliance constraints and a script. Associating each compliance with the supported device type, a structure is defined to perform different security checks against different device types.

3.1.5 NMT node

The NMT node refers to a deployment instance of NMT. There can be other software alongside NMT, a wrapper, and different processing components used beside NMT, as required. This all-in-one combination is called an NMT node. This is responsible for the monitoring of the regional network. It defines the CCG, compliances, and association of device type with the CCG or compliance, creates plans to run the CCG, executes the CCG plans, and stores the result in the local database. SQL is a commonly used option to store all the information regarding the network and the run results.

3.1.6 CCG Plan and Execution

A CCG plan, or Compliance Check Plan (CCP), includes the execution plan of a given CCG in any NMT node. The plan defines what CCG to run, for which devices to check compliance, when to run, scheduling information, and so on. Once the plan is ready, it is run by NMT. The process is called Compliance Check Run (CCR). Since each compliance of CCG is run against the defined number of assets, we can calculate the total number of records in the output using the following equations:

$$\text{Number of gateway execution point} = \sum_{\text{device type}} \text{Number of device} \times \text{Number of modules} \tag{3.1}$$

$$\text{Number of CC results} = \text{Number of gateway execution point} \times \text{ Number of compliances in CCP} \tag{3.2}$$

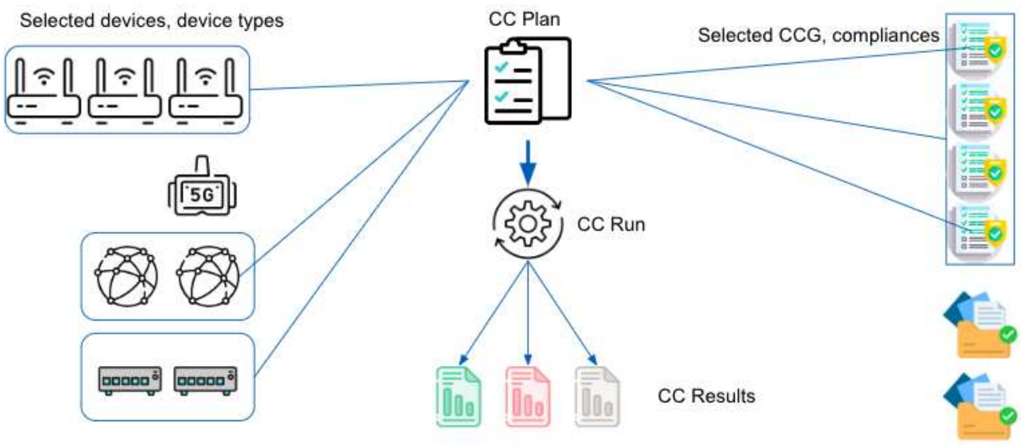


Figure 3.5: Compliance Check workflow

Figure 3.5 shows how the compliance check is planned and executed. A single CCP is formed by selecting some devices or all devices of a particular device type. It is then matched with a subset of compliances under one CCG. Once the plan is ready, it is executed. During the run, a script inside each compliance is run against the selected device to check its compliance. The CCR results are then generated, which contain the result status and output logs.

3.1.7 Network Management Service (NMS)

This layer has the responsibility of identifying all network devices and establishing connections with them to enable the execution of compliance checks when required by the Network Management Tool (NMT). The maintenance of the device connection and the forwarding of commands from NMT to the devices take place within this layer. Subsequently, NMT retrieves information about the network devices and compliance check specifics from a database or other related services, and carries out the necessary actions. A relational database management system (RDBMS) is utilized to store details about the network as well as the outcomes of the Compliance Check Group (CCG) runs. The NMS can establish direct connections with the devices themselves or establish connections through an interface for a group of devices.

3.2 System Profiling: Requirements and Constraints

Each NMT node contains a certain number of basebands and multiple devices of each type. Through the study of real-world systems and interviews with the System Architect, we determine the quantity of devices of each type for different operator sizes. Table 3.1 presents the number of devices of each type present in different nodes based on the operator size. For security purposes, we encode the device types as `t1`, `t2`, `t3`, `t4`, and so on. `t1` is the baseband type, which is also encoded as `bb`. The number of gateways for each device type is provided thereafter. On average, 30 controls are applied per gateway during the CCR, enabling us to calculate the total number of executions. This table will serve as a reference during the implementation process.

The specific information necessary in the central node or aggregation node may vary over time, contingent upon the use case and business requirements. Consequently, the system implementation is devised with the awareness that any subsequent new feature or information can be seamlessly incorporated. In essence, it should possess feature-scalability. In this implementation, the following output requirements are taken into account for the central node.

1. Trend of each CCG over whole network
    - Number of compliance checks output distribution by CCG
    - Percentage of output status distribution by CCG
    - Performance of each CCG by node
    - Sort the top few CCG based on latest run or result statuses
2. Trend of each node
    - Number of compliance checks output distribution by node
    - Percentage of compliance checks output distribution by node
    - Performance of each node by CCG

- Sort the top few devices based on latest run or result statuses

3. Trend of each device type
    - Distribution of different result statuses grouped by CCG
    - Distribution of different result statuses grouped by nodes
    - The top problematic device type to skew the result
    - Sort the top few devices based on latest run or result statuses
4. Number of CCRs executed in certain nodes using certain policies for certain time range

Table 3.1: Device type and Operator Sizes

Type	**Operator Size**			**gateways**	**controls**	**Executions**		
	S	**M**	**L**			**S**	**M**	**L**
bb	5000	20000	80000	1	30	150000	600000	2400000
t2	2	6	23	259	30	15540	46620	178710
t3	1	1	1	1	30	30	30	30
t4	16	64	255	5	30	2400	9600	38250
t5	48	192	765	5	30	7200	28800	114750
t6	2	6	24	1	30	60	180	720
t7	1	1	2	1	30	30	30	60
t8	5	18	72	1	30	150	540	2160
t9	2	6	24	1	30	60	180	720
t10	3	12	48	1	30	90	360	1440
t11	1	4	16	1	30	30	120	480
t12	1	2	8	1	30	30	60	240
t13	2	7	25	10	30	600	2100	7500
t14	1	2	8	1	30	30	60	240
Total	**5085**	**20321**	**81271**	**Total executions**		**176250**	**688680**	**2745300**

Note:
"Type" refers to device type
"controls" refers to the controls applied per gateway
"S", "M", "L" refers to network sizes: Small, Medium, and Large.

3.3 Architectural Design Choices

The requirements can be fulfilled through different architectures, offering a wide range of possibilities. NMT nodes are deployed in multiple geographic locations, necessitating the aggregation of data from each node via data transfer over private or public networks. In order to attain the desired output, we examine several options that were initially considered. These choices were influenced by recommendations from the system architect and decisions based on findings from the literature reviews conducted.

3.3.1 Architecture Option I: Poll-based / Synchronous Approach

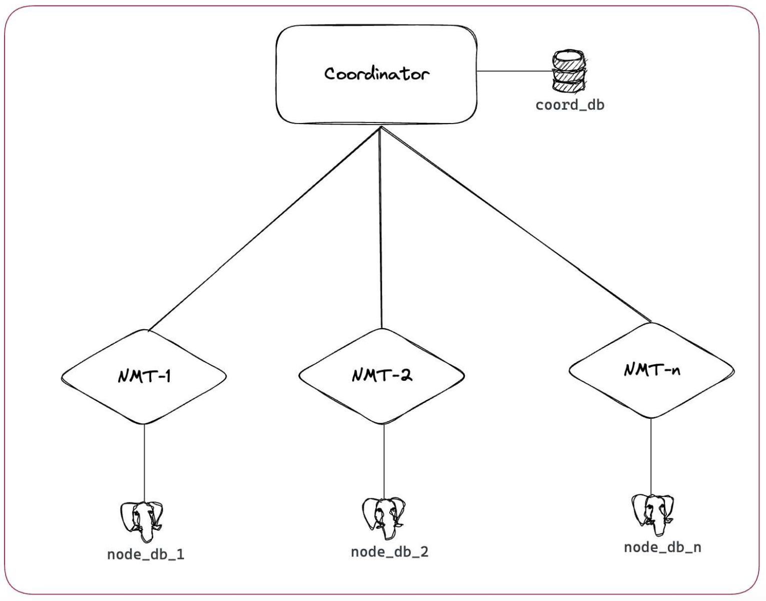


Figure 3.6: Polling-based Architecture

This option, depicted in Figure 3.6, entails the presence of a ”coordinator” application that regularly sends requests to all NMT nodes and gathers the data. Each NMT node maintains its own local database to store the results of CCG runs. The coordinator has the option to cache the results or not.

Advantages: This approach is straightforward to implement since it requires minimal modifications to the API of existing nodes for integrating polling from the coordinator. By using query parameters, the coordinator can specifically request the required data. Additionally, the polling frequency can be adjusted based on the needs. Implementing caching reduces the number of API calls to each node, thereby improving data access speed. Furthermore, this approach can be extended to allow the coordinator to act as a proxy for any node, enabling users to access more information or exercise control over specific nodes.

Disadvantages: The coordinator needs to establish and maintain active connections with each node, and it is responsible for monitoring the status of active and inactive nodes. Depending on whether caching is enabled on the coordinator and the number of requests received by the coordinator regarding the status of any CCG run, the number of API calls to each node can increase significantly.

3.3.2 Architecture Option II: Push-based/Asynchronous Approach

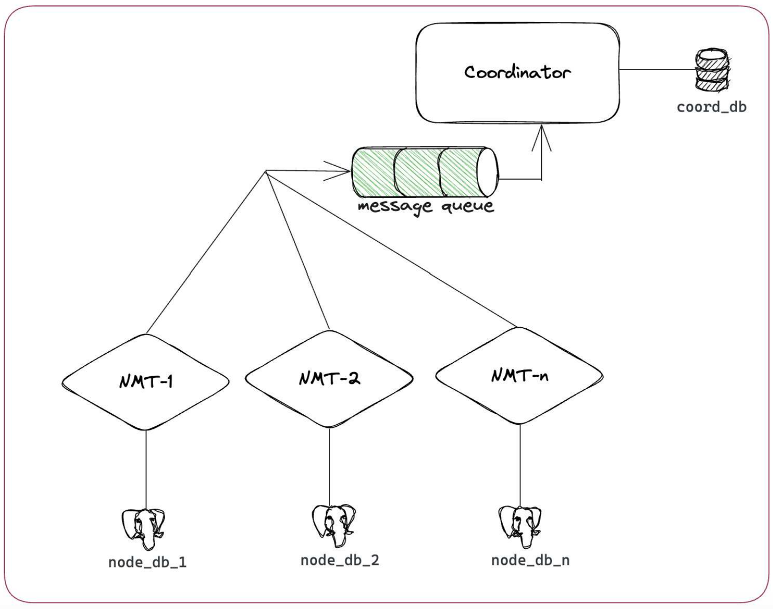


Figure 3.7: Push-based Asynchronous Architecture

In this approach, illustrated in Figure 3.7, each node pushes the new CCG run result to a message queue topic. The coordinator reads the results from the message queue, processes them, and stores them locally. The coordinator can then utilize the local database to provide aggregated statistics for the entire dataset.

Advantage: This architecture eliminates the need for the coordinator to establish and maintain active connections with each node. Once a run result is available on a node, it is simply pushed to the queue, ensuring that the latest result is immediately accessible in the message queue.

Disadvantage: To handle asynchronous events, an additional layer is required in the coordinator to set up the message queue and categorize the messages from each node based on the topic or source. Moreover, since there is no direct communication established between the coordinator and the nodes, the coordinator cannot directly pass user requests to the nodes.

3.3.3 Architecture Option III: ETL Approach

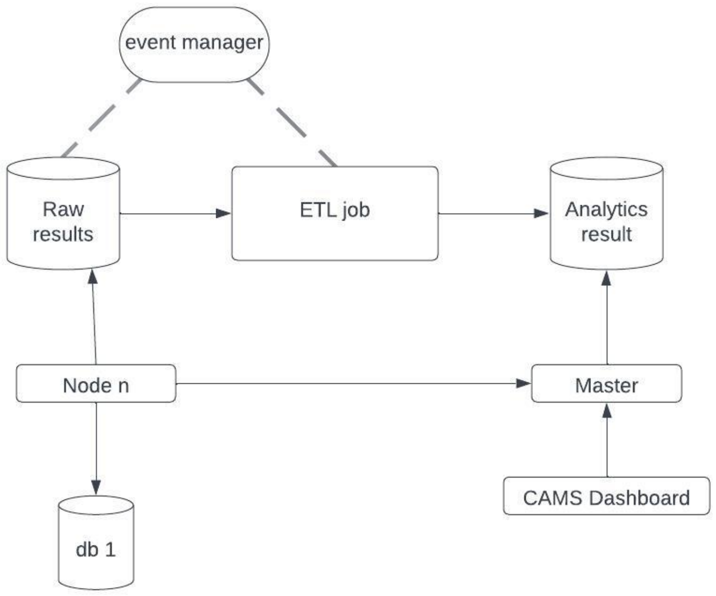


Figure 3.8: Cloud ETL Approach

In the ETL approach, depicted in Figure 3.8, each node pushes the result of every CCG run or the combined results of multiple CCG runs within a certain interval to a central cloud storage service. Once a new result is available in the storage, an ETL job is triggered. The job reads the latest file, processes the data according to the requirements, and stores the analytics result in a database. The ETL job is implemented as a serverless function, although it can also be accomplished using other services provided by the cloud provider. The processed result is then stored in a separate output storage component, such as a database system, file storage, or querying platform. This layer, either directly or with the assistance of a new API layer, delivers the analytics result to the user.

Feature Scalability: Since the ETL function is triggered by the presence of a new file in the cloud storage, multiple ETL functions can be triggered for a single event. This allows the implementation of new features or output statistics requirements through additional ETL functions.

Advantage: By separating each component, changes in the transformation function can be easily updated or horizontally scaled. This flexibility allows the infrastructure size to be adjusted based on the information requirements of the central node.

Disadvantage: The use of multiple cloud services introduces cost considerations, which depend on factors such as data volume, bandwidth, traffic, invocation, and memory usage. While the pay-per-use feature is beneficial for networks with simple central nodes and fewer run results, the cost can increase for larger, high-traffic nodes.

3.3.4 Choosing the best architecture plan

Upon implementing the Minimum Viable Product (MVP) for each of the aforementioned options, it became evident that the ETL-based approach outperformed the others in terms of scalability and customization. After thorough discussions with project stakeholders, several compelling reasons emerged in support of adopting the ETL-based approach.

Firstly, the ETL approach provides a highly scalable solution capable of efficiently handling a large number of nodes and accommodating future growth. Given that the NMT node aims to support an increasing number of devices and expand compliance check coverage, the anticipated growth in run results necessitates a scalable architecture. The ETL approach effectively separates the computational load on the node side, alleviating strain on the node's API. In contrast, achieving scalability with the other options requires significant effort and meticulous planning during system development.

Secondly, the ETL-based approach offers real-time processing capabilities by allowing easy configuration of memory, processing power, and invocation frequency. The inherent flexibility of the ETL approach enables seamless system reconfiguration as needs evolve. It supports multiple data sources and facilitates tailored transformations to adapt to specific formats. This adaptability is crucial for accommodating changing requirements. The transformation function can be easily updated or parallelized, ensuring resilience and agility in the face of evolving needs.

Furthermore, leveraging cloud-based ETL provides access to powerful tools and services for data transformation and enrichment. The availability of cloud services reduces development time and allows a focus on the core business logic rather than infrastructure implementation and maintenance. This consideration was prioritized by the company's stakeholders.

Cost-effectiveness also played a pivotal role in selecting the ETL-based approach. The inherent flexibility of the cloud infrastructure, with its pay-as-you-go pricing model and on-demand scalability, provides superior cost efficiency compared to the other options. Additionally, the ETL-based approach requires fewer engineers for development and maintenance compared to the other models, resulting in reduced project costs and shorter timelines.

Based on these comprehensive comparative analyses and taking into account the input from stakeholders, the ETL-based architectural option has been chosen for implementation.

Chapter 4

Implementation

In this section, we will elucidate the implementation details of the selected architecture. To execute the ETL approach, we employ the utilization of Google Cloud Platform (GCP). The choice of GCP for this thesis is predicated on the availability of cloud resources provided by the company during the course of this work. Nevertheless, users have the liberty to opt for any other public cloud platform or their own internal cloud system for production, while ensuring the consistent functionality of the components.

We will now delve into the specifics of how each component is constructed, fitted, and executed on GCP, which will be elaborated upon in the subsequent subsections. The NMT system is designed to facilitate experimentation with all three types of networks: low-traffic, medium-traffic, and heavy-traffic.

The source code is available on a Github repository [1].

4.1 Simulating NMT Node

The implementation of the NMT node does not focus on actual device scanning and tasks related to network monitoring. The company already has existing solutions for these purposes, which are beyond the scope of this thesis work. For implementation purposes, the NMT node will simply simulate the output result of the node.

To automate the creation of multiple different configurations more efficiently, the `config.py` script generates a config file. This script enables faster creation of various configurations, allowing each node to have a different set of `config.yaml` files for each run, thereby generating a new set of test data. The `config.yaml` file includes details on how the test data should look, what data it should include, and how many instances of data should be generated.

[1]https://github.com/bipinkh/data-aggregation-telco-network

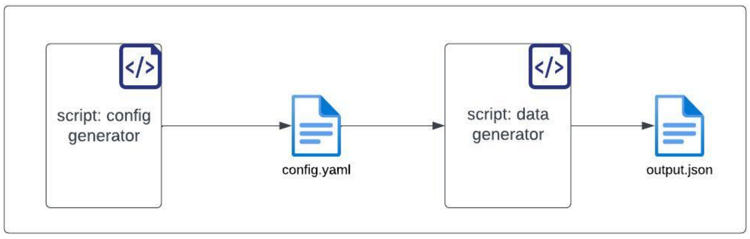


Figure 4.1: NMT node simulation

A sample of the config file can be found in Figure 4.2. In this example, we generate a config file for node *node1l* with 78164 basebands, 5842 devices of type *t2*, 1 of type *t3* and so on. The output will have about 10% error outputs, 17% fail outputs, 4.7% noreach, and 67% pass outputs. It also provides value for CCG id, CCG name, run id, total controls and run time.

The information available in config file include:
- node name
- id, total controls, begin timestamp, and end timestamp used for the CCR
- number of devices of each device type to use for CCR
- id and name of CCG
- the distribution probability of different result statuses

To generate the output data, the `script.py` is executed by passing the generated config file as a parameter. The command syntax is presented in Figure 4.3. A sample of the generated output file is provided in Figure 4.4. The output data file contains various details about the CCR, including the ID, CCG details, and the runtime range. For each run, it includes the device ID, compliance ID, and the run result. Additionally, information about each device type corresponding to each device ID is available, enabling analysis of the result distribution based on device types.

```
node: node11
runs:
- devices:
    bb: 78164
    t10: 42
    t11: 11
    t12: 9
    t13: 150
    t14: 10
    t2: 5842
    t3: 1
    t4: 510
    t5: 2295
    t6: 25
    t7: 2
    t8: 70
    t9: 25
  ccg_id: ccg1
  ccg_name: sample ccg 1
  result_chances:
    error: 0.10374874098684847
    fail: 0.1787522560020067
    noreach: 0.0474815149848024
    pass: 0.6700174880263424
  run_id: '231302624799'
  total_controls: 30
  ts_from: '2023-06-27T23:13:02+03:00'
  ts_until: '2023-06-28T00:58:21+03:00'
total_runs: 1
```

Figure 4.2: Sample of config file

```
SYNTAX:
    python script.py [config_file] [data_json_file]

config_file = location of the config file to be used
data_json_file = location of the output (json) data file

Example:
    python script.py configs/config_region1.yaml
    data/data_region1.json
```

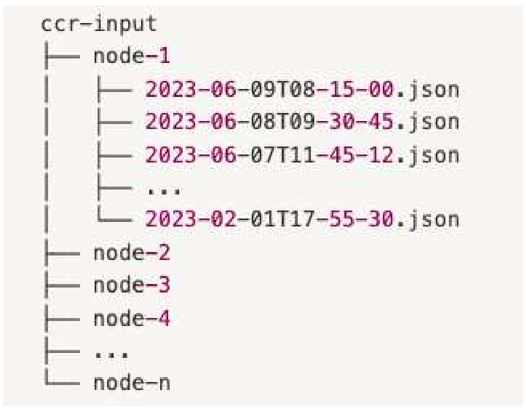
Figure 4.3: Command syntaxes for generating CCR results

```
{
    "node_id": "node1xs",
    "total_runs": 1,
    "runs": [
        {
            "run_id": "231831669110",
            "ccg_id": "ccg1",
            "ccg_name": "sample ccg 1",
            "total_assets_checked": 0,
            "total_controls": 30,
            "ts_from": "2023-07-02T23:18:31+03:00",
            "ts_until": "2023-07-03T02:01:27+03:00",
            "results": [
                {
                    "id": 1,
                    "device_id": "a0bb",
                    "compliance_id": "c1",
                    "result": "pass"
                },
                {
                    "id": 2,
                    "device_id": "a0bb",
                    "compliance_id": "c2",
                    "result": "fail"
                }
                //omitted for brevity //
            ],
            "device_types": {
                "bb": ["a0bb", "a1bb", "a2bb"],
                "t1": ["a0t1", "a1t1"],
                "t2": ["a0t2"],
                "t3": ["a0t3"]
            }
        }
    ]
}
```

Figure 4.4: Sample of CCR result

4.2 ETL infrastructure

The data-ingestion process utilizes Google Cloud Storage bucket as the storage layer. The bucket is organized with nested buckets to store the results from each node separately. The data file is named based on the date and time when the node exports the file, facilitating easy tracking of the last export time for any node. The structure of the bucket is depicted in Figure 4.5. Authorized API requests can insert new files into the bucket. Each node utilizes its service account token to connect to the bucket.



```
ccr-input
├── node-1
│   ├── 2023-06-09T08-15-00.json
│   ├── 2023-06-08T09-30-45.json
│   ├── 2023-06-07T11-45-12.json
│   ├── ...
│   └── 2023-02-01T17-55-30.json
├── node-2
├── node-3
├── node-4
├── ...
└── node-n
```

Figure 4.5: GCP data-ingestion bucket structure

When a new file is present in the cloud storage, the ETL step is triggered by Google EventArc's trigger service. Event type *google.cloud.storage.v1.finalized* is used for this purpose. This event type is activated when a new object is created and finalized in the GCP bucket. Any write or modification process on the object is considered "finalized".

For the ETL operation, Google Cloud Function is employed. A Python function is triggered with the event object, which contains information about the newly added file and the target bucket. The pseudocode outlining the ETL process is presented in Algorithm 1.

4.3 BigQuery for output data layer

The BigQuery layer serves as the data warehouse and analytics layer in this implementation. While it is possible to query the raw json files directly from the cloud bucket, preprocessing the raw files and storing the computed statistics can help reduce duplicate computation, improve processing efficiency, and achieve faster query times. This, in turn, helps optimize costs

Algorithm 1 Pseudocode for ETL process

```
Constants: PROJECT_ID, BQ_DATASET_ID, BQ_TABLE_ID
Function etl_cloud_event(cloud_event):
    Extract bucket_name and file_name from cloud_event data
    Call etl function with bucket_name and file_name as parameters
    return "OK"
Function etl(bucket_name, file_name):
    Retrieve data from Cloud Storage using bucket_name and file_name
    Extract node_id from data
    Perform data transformation to compute result totals
    Store results in BigQuery using node_id
Function compute_result_totals(data):
    Initialize empty lists all_results and device_specific
    foreach run in data['runs'] do
        Initialize stats dictionary
        Initialize device_stats dictionary
        foreach result in run['results'] do
            Increment the corresponding result type in the stats dictionary
            Get or create the device stats information for this device type from
              device_stats;
            Increment the corresponding result type information for device
              stats;
        end
        Add the stats dictionary to run
        Add run to all_results
        Add device_stats to device_specific
    end
    return all_results, device_specific
Function store_result_in_bigquery(node_id, results, device_specific):
    Create BigQuery client
    Create BigQuery dataset if it does not exist
    Define the BigQuery table schema
    Create BigQuery table reference
    if table does not exist then
        Create the BigQuery table with the schema
    end
    Prepare rows to insert
    foreach data in results do
        Create a row object with the data
        Remove previous stats for the given node and run
        Add the row to the rows_to_insert list
    end
    Insert the rows into BigQuery
    foreach next 10000 data in device_specific do
        Create a row object with array of the data
        Insert the row object into BigQuery
    end
```

for cloud resources.

There are two tables in BigQuery that store the results of the ETL process. Table 4.1 is used to store statistical overview data, while Table 4.2 is used to store statistical data based on asset types. To optimize query performance, the `ccr_stats_overview` table is first clustered by the `ccg_id` column and then by the `node_id` column. Clustering by `ccg_id` improves querying time as the central dashboard primarily accesses information based on the CCG. Similarly, as queries based on `node_id` are also popular in the dashboard, the second clustering by `node_id` further aids in grouping the information according to the node ID.

Table 4.1: *ccr_stats_overview* table schema

column_name	**datatype**	**default**	**remarks**
run_id	STRING	NULL	
ccg_id	STRING	NULL	CCG id
ccg_name	STRING	NULL	CCG name
node_id	STRING	NULL	
stats_pass	INTEGER	NULL	total CC with pass result
stats_fail	INTEGER	NULL	total CC with failed result
stats_error	INTEGER	NULL	total CC with script error
stats_noreach	INTEGER	NULL	total CC for which devices were unreachable
total_devices_checked	INTEGER	NULL	
total_compliances	INTEGER	NULL	total unique CC
ts_entry	TIMESTAMP	NULL	timestamp when data was stored in BigQuery
ts_from	TIMESTAMP	NULL	CCG run start timestamp
ts_until	TIMESTAMP	NULL	CCG run stop timestamp

BigQuery offers several ways to access the data from the tables. We use following two in the implementation.

1. Using Query language
   The SQL query language in the GCP console is the most straightforward method for data access. It allows for obtaining the data in the precise format required. This approach is commonly used during development and can be utilized by individuals with knowledge of SQL queries and access to the query console.

2. Using APIs
   BigQuery provides APIs that enable programmatic access to the data.

Table 4.2: *ccr_stats_devicetype* table schema

column_name	**datatype**	**default**	**remarks**
run_id	STRING	NULLABLE	
device_type	STRING	NULLABLE	type (enum) of device
node_id	STRING	NULLABLE	
ccg_id	STRING	NULLABLE	CCG id
stats_pass	INTEGER	NULLABLE	total CC with pass result
stats_fail	INTEGER	NULLABLE	total CC with failed result
stats_error	INTEGER	NULLABLE	total CC with script error
stats_noreach	INTEGER	NULLABLE	total CC for which devices were unreachable

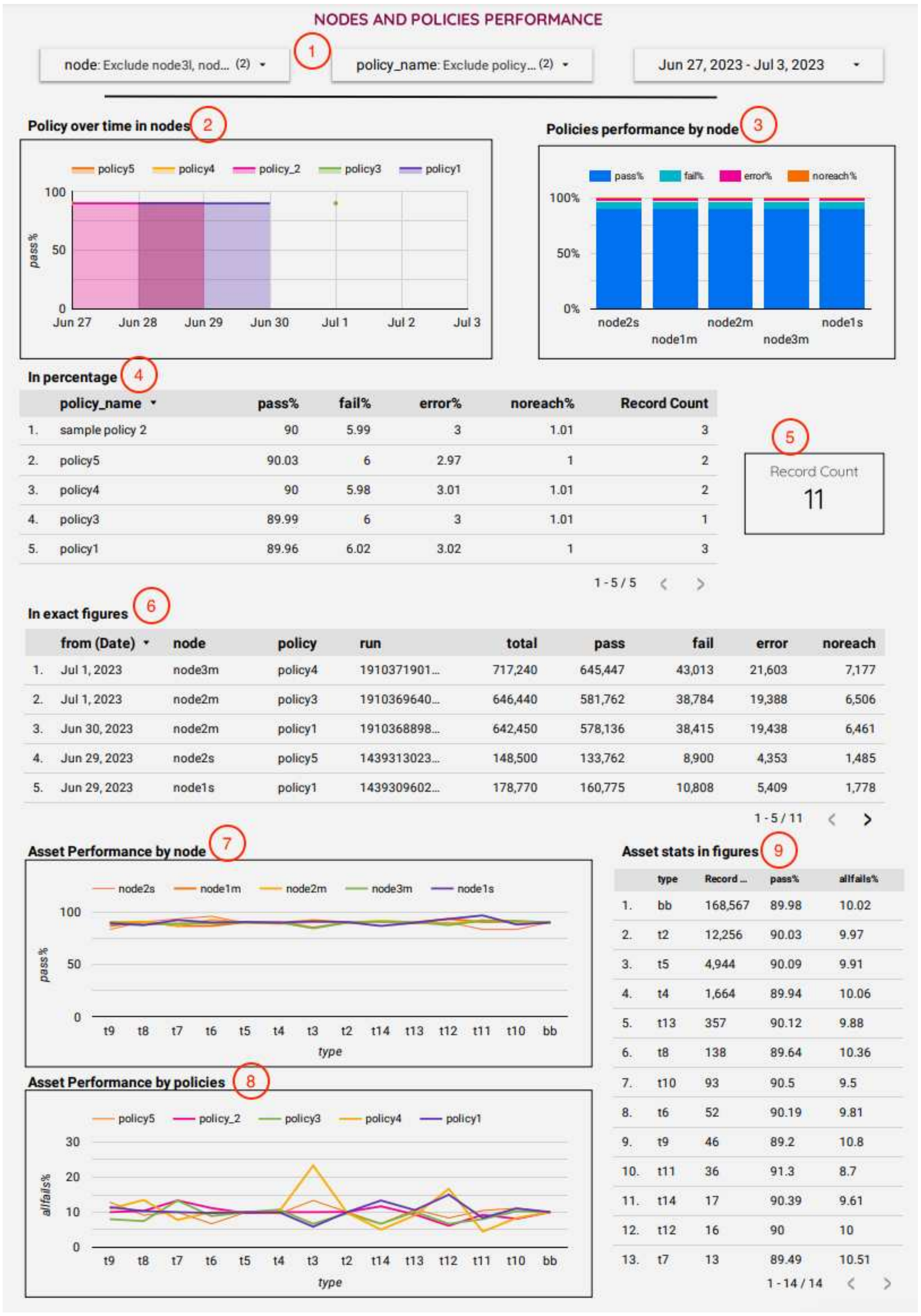
These APIs can be invoked directly or integrated with other cloud services, such as Lookerstudio, to achieve the desired outcomes. Utilizing APIs offers flexibility in accessing and retrieving data from BigQuery tables.

4.4 Lookerstudio for Dashboard and Data visualization

Lookerstudio, another Google Cloud service, facilitates the creation of dashboards using various data sources, including BigQuery tables. It internally invokes the BigQuery APIs as required to retrieve results. Lookerstudio offers a range of tools to select different graphs or tables and define axes and metrics. It can effortlessly connect to multiple table sources and blend the data to provide unified results.

Figure 4.6 presents a snapshot of the complete dashboard. We will now explain each section marked by numbers inside red circles. It is possible to add multiple sections and pages within the dashboard. The selection of sections displayed in the dashboard is determined based on the company's requirements and the desired information from the system.

Section 1 is the control area where control variables are located. These variables can be used to filter the source dataset. All selections, sorting, and filtering in other sections are applied to the data filtered by these control variables. The first control variable allows the selection of nodes to visualize. Only the results belonging to the selected nodes are displayed. Selecting a single node allows for in-depth analysis of that node's statistics, while selecting all nodes provides a holistic view of the network status. The second



In percentage

	policy_name ▾	pass%	fail%	error%	noreach%	Record Count
1.	sample policy 2	90	5.99	3	1.01	3
2.	policy5	90.03	6	2.97	1	2
3.	policy4	90	5.98	3.01	1.01	2
4.	policy3	89.99	6	3	1.01	1
5.	policy1	89.96	6.02	3.02	1	3

1 - 5 / 5

Record Count
11

In exact figures

	from (Date) ▾	node	policy	run	total	pass	fail	error	noreach
1.	Jul 1, 2023	node3m	policy4	1910371901...	717,240	645,447	43,013	21,603	7,177
2.	Jul 1, 2023	node2m	policy3	1910369640...	646,440	581,762	38,784	19,388	6,506
3.	Jun 30, 2023	node2m	policy1	1910368898...	642,450	578,136	38,415	19,438	6,461
4.	Jun 29, 2023	node2s	policy5	1439313023...	148,500	133,762	8,900	4,353	1,485
5.	Jun 29, 2023	node1s	policy1	1439309602...	178,770	160,775	10,808	5,409	1,778

1 - 5 / 11

Asset stats in figures

	type	Record ...	pass%	allfails%
1.	bb	168,567	89.98	10.02
2.	t2	12,256	90.03	9.97
3.	t5	4,944	90.09	9.91
4.	t4	1,664	89.94	10.06
5.	t13	357	90.12	9.88
6.	t8	138	89.64	10.36
7.	t10	93	90.5	9.5
8.	t6	52	90.19	9.81
9.	t9	46	89.2	10.8
10.	t11	36	91.3	8.7
11.	t14	17	90.39	9.61
12.	t12	16	90	10
13.	t7	13	89.49	10.51

1 - 14 / 14

Figure 4.6: Dashboard

control variable enables the selection of policies, filtering the CCRs generated by applying the chosen policies. The third control variable allows for choosing a date range to retrieve data from. By combining these three control variables, users have the option to select CCR data from specific nodes, policies, and within a certain date range.

Section 2 presents the result status of different CCGs over the chosen date range across all selected nodes. This section allows users to choose the metric to display: pass%, fail%, noreach%, and error%. It provides insights into the average behavior of each compliance on a particular day.

Section 3 displays the average CCG performance grouped by each node. Each bar represents how the selected CCGs performed on average for all runs made within the selected date range.

Section 4 illustrates the overall compliance performance, averaged across all execution results from all selected nodes during the chosen time period. The columns depict the percentages of different result statuses and the total number of times the CCG has been applied.

Section 5 shows the number of CCRs that match the criteria set by the control variables.

Section 6 presents the latest CCRs executed on all chosen nodes for all selected CCGs. The table can be sorted by each column in ascending or descending order. It provides information such as the CCR ID, the node it was run on, the policy applied to it, the day of the run, the total number of generated records, and the distribution of record statuses.

Section 7 and **Section 8** provide a detailed breakdown by different device types. Each section displays all device types on the X-axis, along with the status metrics of pass% or allfails% (which includes the combined sum of fail%, error%, and noreach%). Section 7 breaks down the result statuses grouped by nodes, while Section 8 groups them by CCGs. These sections help in understanding how different device types behave across nodes and CCGs, allowing for the identification of problematic device types in the network.

Section 9 provides a breakdown of device types with the number of records available for each type. This represents the sum of all compliances across all selected nodes and policies. It shows the actual number of records used for each device type. This information can help identify if the graphs in Section 7 or 8 are skewed by a few device types only.

Chapter 5

Evaluation and Analysis

5.1 Setup and Performance Metrics

To test the behavior of the system with different input sizes and network sizes, separate tests are conducted for each network size category. For each network type, five sets of CCGs are run on each of the three nodes. This generates a total of fifteen sets of CCR results for the entire network. During each compliance run, the number of each device type is randomly generated, with a variation of approximately 20% more or less compared to the values provided in Table 3.1. Similarly, the probabilities of different result types are randomly chosen within certain ranges. The probability ranges for each result status are set as follows: `pass` - (0.3, 0.99), `fail` - (0.09, 0.4), `error` - (0.05, 0.3), and `noreach` - (0.03, 0.3). These values are then normalized so that the sum of all probabilities is 1.0. This process generates different sets of compliance results for all fifteen CCGs, with slight variations in the number of assets checked, the number of records generated, and the distribution of result statuses. Each CCR result is stored in a json file with the provided structure shown in Figure 4.4. All the json files are subsequently uploaded to the GCP bucket for data ingestion. The ETL process then processes each result file in a separate instance of Cloud Function. During execution, the following metrics are collected for each ETL process:

- **etl_total**: Total time taken for the ETL process to complete.
- **etl_extract**: Total time taken for the cloud function to fetch the file from the Cloud Storage bucket.
- **etl_transform**: Total time taken for the cloud function to process the CCR result file and produce the output.

- **etl_load**: Total time taken for the cloud function to load the processed result into the output table in BigQuery.
- **bq_prepare**: Time taken for the cloud function to prepare the records for insertion into the BigQuery table.
- **bq_insert_np**: Time taken for the cloud function to load the computed result regarding node and policy-level details.
- **bq_insert_a**: Time taken for the cloud function to load the computed result regarding asset type details. Memory consumed by each instance.
- **records**: Number of compliance check result items generated by each ETL process.
- **devices**: Combined number of gateways from each device used in the CCR.

Therefore, the following relation hold:

$$\text{etl\_total} = \text{etl\_extract} + \text{etl\_transform} + \text{etl\_load}$$

$$\text{etl\_load} = \text{bq\_prepare} + \text{bq\_insert\_np} + \text{bq\_insert\_a}$$

Once the metrics are available, a detailed analysis is conducted to examine the behavior of each metric for different network sizes. The focus is on understanding how memory, performance, and resource consumption vary across different network sizes.

The objective is to gain insights into the factors that affect the system and assess its scalability as the number of devices in the network increases. Additionally, the goal is to analyze resource consumption patterns and identify any potential bottlenecks or areas for optimization. By understanding these aspects, we can make informed decisions about system enhancements, resource allocation, and scalability measures to ensure efficient operation and performance.

5.2 Devices and Records

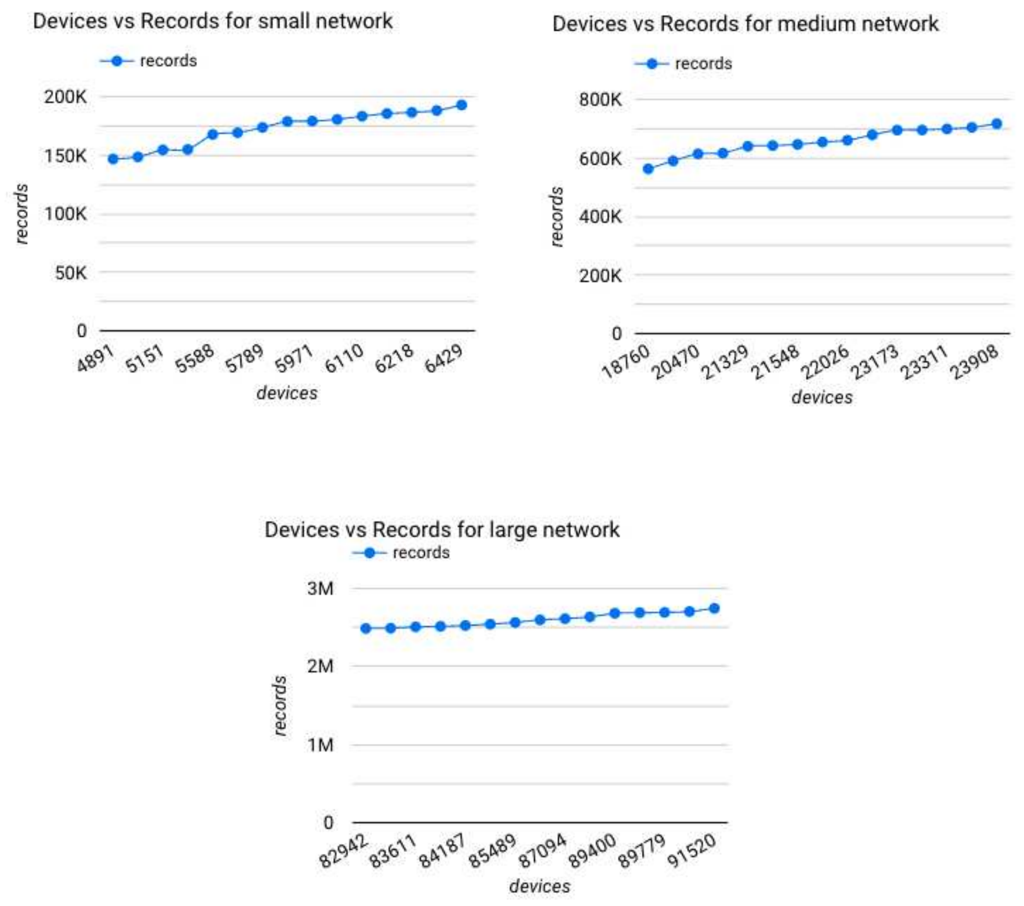


Figure 5.1: Number of devices vs number of records

Figure 5.1 depicts the different number of devices used for compliance checks across all fifteen CCRs, along with the corresponding number of records generated for each CCR. On average, the number of devices for small, medium, and large networks is approximately 5,000, 20,000, and 85,000, respectively. Similarly, the average number of records generated per CCR in small, medium, and large networks is close to 175,000, 650,000, and 2.6 million, respectively.

In Figure 5.2, we observe the overall relationship between the number of devices and the number of records generated during CCR for all network sizes. Plotting the values for all network sizes reveals a linear relationship between the number of devices and the count of records. A trendline is drawn along with its equation to represent the linear relationship. This linear relationship

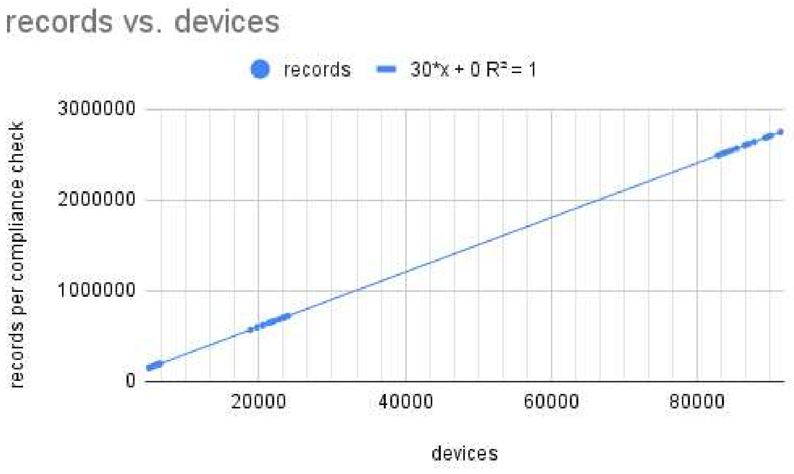


Figure 5.2: Relation of number of devices and records

supports the formula for calculating the number of records,

$$\text{records} = \sum_{\text{device type}} (\text{devices} \times \text{gateways} \times \text{compliance per gateway})$$

$$\text{records} = \sum_{\text{device type}} (\text{devices} \times \text{modules} \times 30) \tag{5.1}$$

This equation corresponds to the equation of the linear regression generated for the given dataset in the figure 5.2:

$$30x + 0 = 1$$

5.3 ETL performance

Figure 5.3 illustrates how the performance of the ETL process varies for each given network size. The specific values for the time taken in each step are presented in Table 5.1. For each network size, the extraction of the CCR output file from the cloud storage bucket is the most time-consuming process. Loading the computed result into BigQuery follows as the second-most time-consuming task. The transformation task, on the other hand, takes less time compared to the extraction and loading operations. On average, the complete ETL process takes approximately 2.2 seconds to process 172 thousand records, 7 seconds to process 654 thousand records, and 28 seconds to process 2.6 million records.

Table 5.1: ETL performance values on different network size

nwsize	etl_total	extract	transform	load	#devices	#records
Small	2.234	1.235	0.216	0.779	5755.733	172672
Medium	7.156	4.581	0.822	1.743	21819.666	654590
Large	27.955	18.797	3.431	5.695	86666	2599980

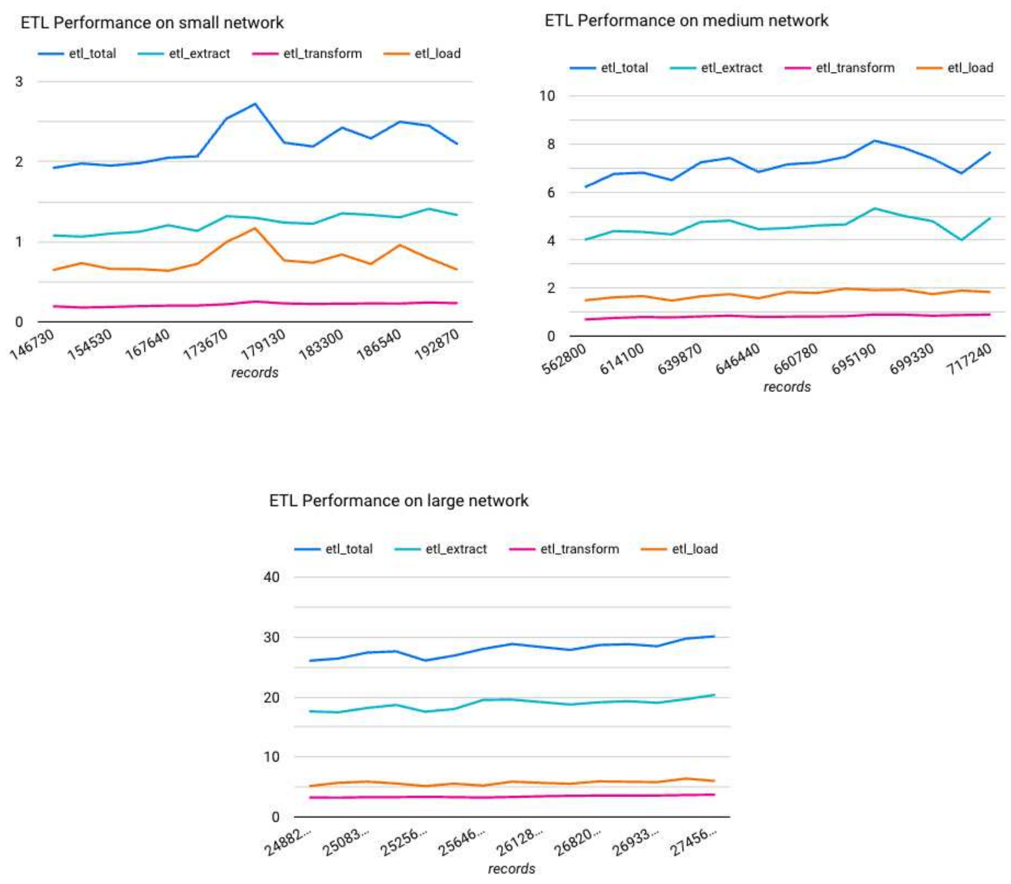


Figure 5.3: ETL performance graph on different network size

Figure 5.4 presents the system's performance scalability by comparing the time taken by each ETL process with the number of devices used to generate CCR results. It is observed that each ETL step exhibits a linear relationship with the number of devices in the network. The graph includes values for small, medium, and large networks, and a trendline is drawn to measure the linear regression. This allows us to determine the equations for each linear relationship.

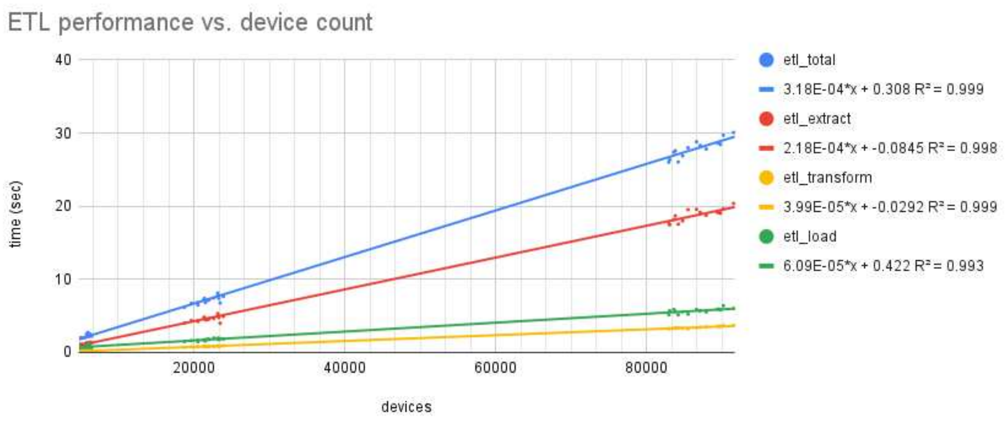


Figure 5.4: Scalability of ETL Performance

The slope of the trendline for the extraction step is the highest compared to the slopes for the load and transform steps. As the number of records increases, the ETL process takes more time to fetch the records from the Cloud storage to its memory. The trend is similar for the load and transform steps, but the increase in time is significantly less compared to the extraction step. One reason for this is the benefit of maximum memory size provided by the cloud function. With the provision of allowing maximum memory, the transform and load steps can be processed faster with only a slight increase in time. However, loading the file from Cloud storage is not as impacted by the available memory size for the process.

The "Load" step encompasses various actions, each of which can be further evaluated in terms of timing.

Table 5.2: "Load" performance in ETL on different network size

nwsize	records	etl_load	bq_prepare	bq_insert_a	bq_insert_np
Small	172672	0.7797	0.3705	0.3631	0.0453
Medium	654590	1.7432	0.4211	1.2801	0.0414
Large	2599980	5.6956	0.4879	5.1489	0.0582

Table 5.2 provides the average time taken for the "load" step in the ETL process for different network sizes. For loading 172 thousand records, the operation takes 0.8 seconds, for 654 thousand records it takes 1.74 seconds, and for loading 2.6 million records, it takes 5.7 seconds.

The "load" step comprises two components: preparing the records for insertion and inserting the records. The time taken to insert the "np" record, which contains statistics about the count of individual result types, is constant, averaging around 0.4 to 0.5 seconds. This is because this record consists of only one row, regardless of the number of records processed.

The preparation phase involves calculating the time required to establish a connection to BigQuery and obtain the table reference. This time remains approximately the same for all network sizes. However, the number of rows containing device type information increases with the number of records. This is because the number of device-level rows is directly proportional to the number of records. As a result, the larger the number of records, the longer it takes to load the device-level information. This relationship can also be observed in Figure 5.5, where the size of the purple bar indicates the increase in time taken to insert asset-level information as the number of records increases.

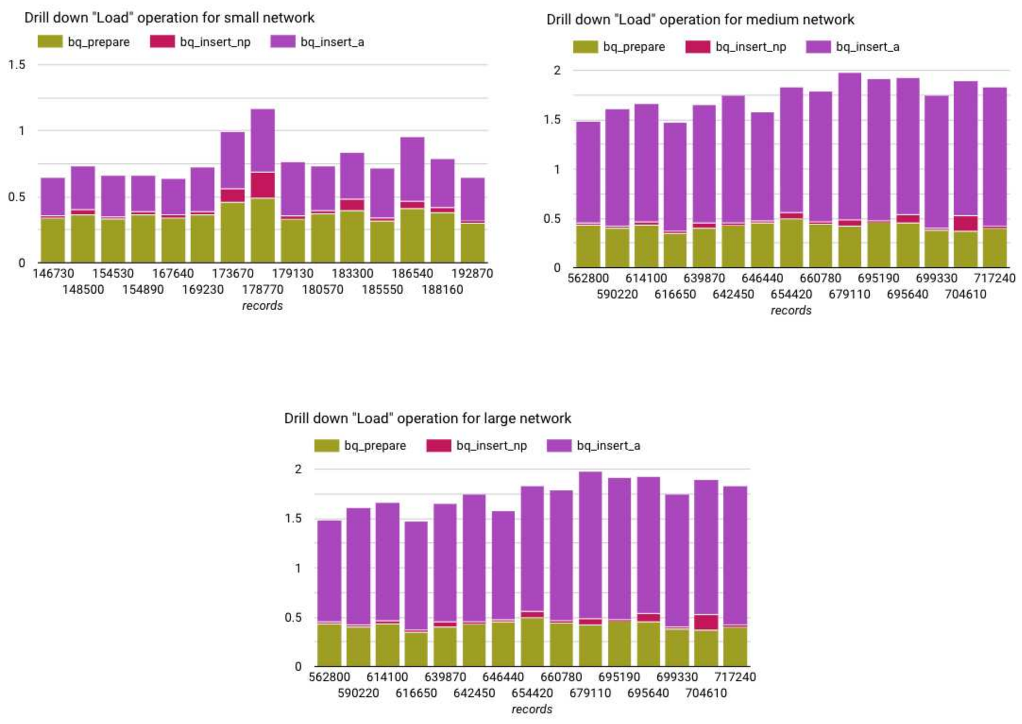


Figure 5.5: “Load” time in ETL process for different network sizes

5.4 Latency

This timing also provides the latency value. We define latency as the time period between the generation of CCR and observing the result in the central dashboard.

$$\begin{aligned}\text{latency} = {} & \text{Time to push CCR to cloud bucket} \\ & + \text{EventArc trigger time} \\ & + \text{ETL time} \\ & + \text{BigQuery execution time} \\ & + \text{Delay in refreshing dashboard}\end{aligned} \tag{5.2}$$

The time taken to push CCR to the cloud bucket depends on the implementation of the node and the size of the records. The EventArc trigger itself is negligible in terms of time. However, if the maximum number of instances for the Cloud Function is reached, events may be queued until a free instance becomes available. In such cases, the trigger time can increase. The execution time of BigQuery depends on the number of records present in the table. In simple terms, as the number of CCRs processed by the ETL process increases, the size of the BigTable also increases, resulting in longer BigQuery execution times. The analysis of the trend for this value is not covered in this thesis. Additionally, the equation takes into account the delay time from clicking the refresh button on the dashboard once the data is available in BigTable.

5.5 Resource Consumption

Resources required to run the ETL process include processing power, database size, concurrency, logging, and memory. However, in our application, external API calls are not served, so the invocation count is not considered.

In this analysis, we will focus on memory consumption as it is a major factor for billing and determining ETL time.

In Figure 5.6, we depict the memory consumption required by each Cloud Function instance to process one CCR file. For the smaller network, with a record size ranging from 147,000 to 193,000, the average memory used by the Cloud instance is 323.59 MB, with a maximum of 352.06 MB and a minimum of 286.61 MB. For the medium-sized network, with records ranging from 563,000 to 717,000, the average memory consumption is 497.34 MB, with a maximum of 573.46 MB and a minimum of 379.32 MB. In the case of the larger network, memory consumption ranges from 1,941.95 MB to 1,205.84 MB, with an average of 1,668.59 MB. This indicates that running a Cloud Function with a maximum memory allocation of 2 GB is sufficient to handle the system. Considering the potential increase in record count, a safe limit for the maximum memory size would be between 2.5 GB to 3 GB.

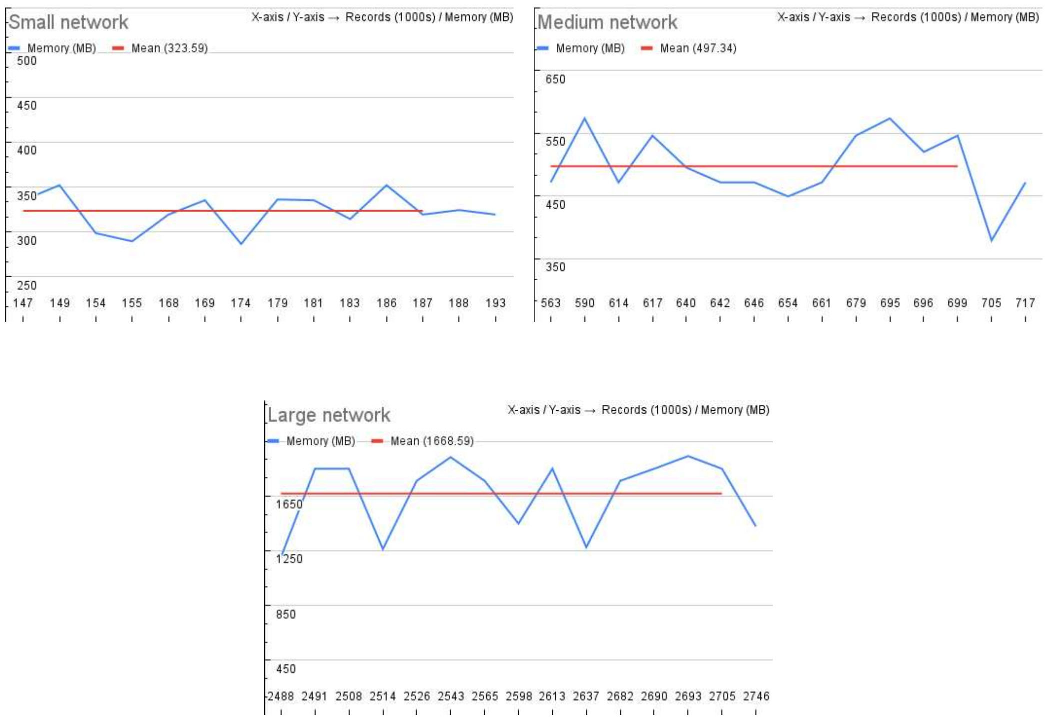


Figure 5.6: Memory consumption for different network size

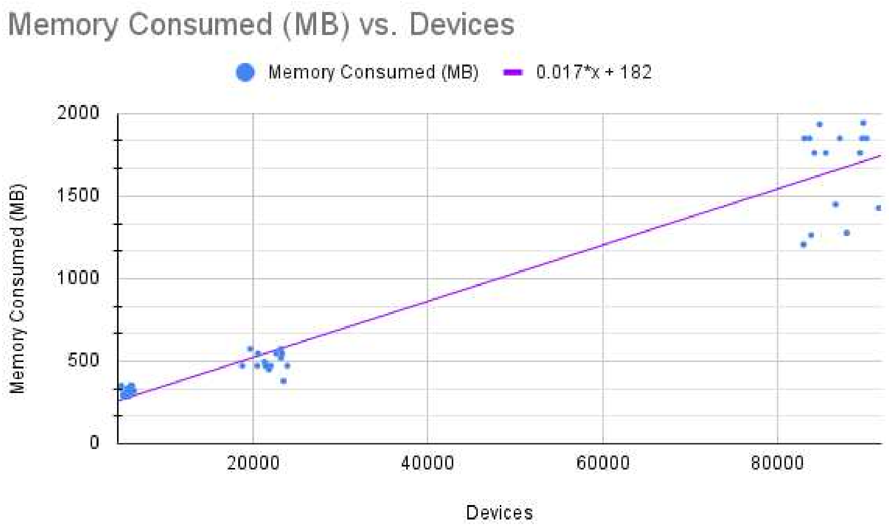


Figure 5.7: Memory consumption over different device size

Each CCR file is processed by an individual Cloud Function instance, each of which has its own dedicated memory with a maximum limit. This advantage of Cloud Function allows us to be less concerned about the changes in memory limit with the increase in records coming from multiple nodes simultaneously.

In Figure 5.7, we show how memory consumption varies when processing CCRs of different device sizes. We plot the values for different network sizes and observe the trendline. The relationship between memory consumed and the number of devices is linear, with a slope of 0.017. We observe that the variance in memory consumption is lowest for the small network and highest for the larger network. This is due to the fact that as the device count increases, the number of records generated during each CCR varies. The number of records depends on the combined number of gateways, which further depends on the number of different device types present in the network.

Chapter 6

Discussion

This section discusses how the result and evaluation be useful for the company and CSP system in general.

6.1 Evaluation of system's effectiveness

The terminologies used in the implementation employ generic terms for the system components and are based on a similar system architecture used for NMT. Therefore, the results and analysis can be extrapolated to real-world systems. The analysis demonstrates that the system's performance grows linearly with an increase in system size and data volume. The resources utilized for each invocation remain below 1700 MB for the large network, which is easily manageable for all the company's customers. The average processing time of less than 30 seconds for the large network and below 10 seconds for the medium-sized network is considered acceptable and does not impose significant delays in system performance. Furthermore, the resources can be tuned and resized according to the customers' system size, allowing for scalability and optimization based on individual requirements.

6.2 Considerations for system configuration

The current implementation provides the proof of concept, however, use of it in real CSP network require some decisions and considerations.

Data security
The current implementation is deployed on the Google Cloud public service. However, the business requirements and data protection policies in a

CSP environment may prohibit the use of public cloud services. There are two possible approaches to address this issue.

The first approach involves processing the data by removing sensitive information and details about the resources such as nodes, CCGs, and devices. These details can be replaced or encoded with alternative identifiers, allowing for the use of cloud services solely for data processing. The results can then be decoded to provide meaningful insights into the original data.

The second approach is to build the entire infrastructure pipeline within the customer's premises, using in-house machines and servers. This approach eliminates the need to rely on cloud service providers and allows for greater trust and control over data security. Many cloud service providers offer the option of bringing their cloud services to the customer's premises, providing private cloud security. One example of such an offering is AWS Outpost, where AWS brings racks and servers to the customer's building, allowing the cloud services to run locally while still adhering to data policies and regulations.

Performance configuration options

There are various configuration options that can be adjusted to meet the performance requirements. These configuration options are available for cloud functions and other cloud services. For cloud functions, some of the configuration options include:

- Maximum number of cloud function instances to run in parallel: This determines the level of concurrency and can be adjusted based on the workload and performance needs.
- Memory usage limit per instance: This defines the maximum amount of memory allocated to each cloud function instance. Choosing an appropriate memory limit can optimize performance.
- CPU choice: The option to select a specific CPU type for the cloud function instances, which can impact performance depending on the workload.
- Timeout: The maximum allowed execution time for a cloud function. Setting an appropriate timeout value ensures efficient resource utilization.
- Retry policies: Configuring retry policies for handling transient errors or failures during execution.

Other configurations available include resource labeling, access control settings for storage, BigQuery, and cloud functions, specifying the function entry point, runtime language, trigger type, service account for cloud function updates, partitioning and clustering of tables, and choice of cloud storage class.

Additionally, the choice of cloud storage class is an important consideration. Different cloud providers offer different classes of storage options for storing result files, with pricing varying based on factors such as storage region, usage frequency, and backup requirements. These options can be configured based on specific needs, regulatory requirements, and cost considerations.

Furthermore, there are additional options for query optimization in BigQuery, including query caching and query plan analysis. Based on user requirements, frequently used or resource-intensive query operations can be optimized. The use of EXPLAIN statements can help in understanding and improving query performance. Techniques such as views, appropriate join strategies, and efficient grouping and aggregation within the query itself can also have a significant impact on performance.

6.3 Future Work

There are opportunities for performance improvement, dashboard updates, and process logic enhancements in the implementation. The current implementation has demonstrated that the ETL total time scales linearly with the number of devices and records, showcasing its scalability compared to other approaches. However, further performance improvements can be achieved by optimizing the algorithm to enable parallel thread counting. Additionally, splitting CCR files into multiple chunks and processing them in parallel can help reduce the ETL total time to a logarithmic scale. It is important to note that achieving better performance may require additional processing power and larger resources.

In terms of the dashboard, there is room for improvement by including more analyses and results from complex queries. The specific analyses and visualizations can be tailored to the use cases and requirements of the Security Officer. While the current implementation provides various charts to analyze the network from multiple angles, it is not an exhaustive list of potential outputs. Depending on the requirements, additional features and visualizations can be incorporated to enhance the dashboard.

Furthermore, the process logic employed in the implementation can be adapted based on specific requirements. Currently, the results are grouped by result status and further categorized by device types. However, if new information needs arise in the CAMS requirement, the process logic can be updated to accommodate the new data requirements. This would involve modifying the cloud ETL function and potentially the BigTables to ensure the updated logic is properly implemented.

Chapter 7

Conclusion

In this thesis, we investigated the imperative need for CSP networks to comprehensively monitor their entire infrastructure by gathering information from individual network nodes regarding the status of their network devices. The approach for data aggregation from these nodes to the central control node is contingent upon several factors, such as the use case of the system, data type, data security considerations, and acceptable latency levels.

The primary objective of this study was to explore and evaluate various methods of data aggregation from network monitoring tools deployed across multiple nodes within the CSP network, encompassing the entire geographical region. We conducted a comparative analysis of each option and subsequently focused on the most promising alternative, which we then implemented and measured. The impetus behind these endeavors was the company's aim to enhance the capacity of their network monitoring systems, enabling a centralized control dashboard for comprehensive node activity observation. This capability would facilitate prompt and informed decision-making processes.

To begin, we conducted a meticulous evaluation of the existing system, profiling it, and identifying pertinent issues and constraints that needed to be addressed. Furthermore, we established specific requirements tailored to networks of varying sizes — small, medium, and large. Subsequently, we examined three different architectural approaches: synchronous pull-based, asynchronous push-based, and cloud ETL, while also reviewing related works in similar projects. In addition, we specified the data type to be employed, customizing it to suit the existing system.

Our thorough evaluation of these three architectural options for establishing a central view of a telecommunications network revealed that the cloud ETL-based approach stood out as the most suitable choice. As part of our research, we successfully implemented the ETL pipeline utilizing the Google

Cloud Platform. Furthermore, we devised a test scenario comprising three nodes, over five CCGs, and multiple compliance checks within each node.

The chosen implementation approach demonstrated good performance, with processing times ranging from 2.23 seconds to 27.96 seconds across various network sizes, considering a ratio of 30 monitoring results per device. The memory consumption exhibited a linear correlation with the number of devices. Overall, the cloud ETL-based approach provides an efficient and scalable solution for collecting and analyzing network monitoring data in large-scale telecommunications networks. It effectively addresses the challenges of data collection, processing, and visualization, making it a favorable choice for achieving a centralized view of the network.

Bibliography

[1] Mansaf Alam and Kashish Ara Shakil, “Cloud database management system architecture,” *UACEE International Journal of Computer Science and its Applications*, vol. 3, no. 1, pp. 27–31, 2013.

[2] Prajwol Sangat, Maria Indrawan-Santiago, and David Taniar, “Sensor data management in the cloud: Data storage, data ingestion, and data retrieval,” *Concurrency and Computation: Practice and Experience*, vol. 30, no. 1, p. e4354, 2018.

[3] Daniel J Abadi, “Data management in the cloud: Limitations and opportunities,” *IEEE Data Eng. Bull.*, vol. 32, no. 1, pp. 3–12, 2009.

[4] Brian F Cooper, Eric Baldeschwieler, Rodrigo Fonseca, James J Kistler, PPS Narayan, Chuck Neerdaels, Toby Negrin, Raghu Ramakrishnan, Adam Silberstein, Utkarsh Srivastava, and others, “Building a cloud for yahoo!,” *IEEE Data Eng. Bull.*, vol. 32, no. 1, pp. 36–43, 2009.

[5] Manolis M Tsangaris, George Kakaletris, Herald Kllapi, Giorgos Papanikos, Fragkiskos Pentaris, Paul Polydoras, Eva Sitaridi, Vassilis Stoumpos, and Yannis E Ioannidis, “Dataflow Processing and Optimization on Grid and Cloud Infrastructures,” *IEEE Data Eng. Bull.*, vol. 32, no. 1, pp. 67–74, 2009.

[6] Gang Chen, HV Jagadish, Dawei Jiang, David Maier, Beng Chin Ooi, Kian-Lee Tan, and Wang-Chiew Tan, “Federation in cloud data management: Challenges and opportunities,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 7, pp. 1670–1678, 2014.

[7] Vladimir Pajić, Miro Govedarica, and Mladen Amović, “Model of point cloud data management system in big data paradigm,” *ISPRS International Journal of Geo-Information*, vol. 7, no. 7, p. 265, 2018.

[8] Rajat Chaudhary, Gagangeet Singh Aujla, Neeraj Kumar, and Joel JPC Rodrigues, "Optimized big data management across multi-cloud data centers: Software-defined-network-based analysis," *IEEE Communications Magazine*, vol. 56, no. 2, pp. 118–126, 2018.

[9] Kuan Zhang, Xiaohui Liang, Mrinmoy Baura, Rongxing Lu, and Xuemin Sherman Shen, "PHDA: A priority based health data aggregation with privacy preservation for cloud assisted WBANs," *Information Sciences*, vol. 284, pp. 130–141, 2014.

[10] Mahantesh N Birje and Chetan Bulla, "Cloud monitoring system: basics, phases and challenges," *International Journal of Recent Technology Engineering (IJRTE)*, vol. 8, no. 3, pp. 4732–4746, 2019.

[11] Leandro V Silva, Pedro Barbosa, Rodolfo Marinho, and Andrey Brito, "Security and privacy aware data aggregation on cloud computing," *Journal of Internet Services and Applications*, vol. 9, no. 1, pp. 1–13, 2018.

[12] N Karthick and X Agnes Kalrani, "A survey on data aggregation in big data and cloud computing," *International Journal of Computer Trends and Technology (IJCTT)*, vol. 17, no. 1, pp. 28–32, 2014.

[13] Eftim Zdravevski, Cas Apanowicz, Krzysztof Stencel, and Dominik Slezak, "Scalable cloud-based ETL for self-serving analytics," in *2019 IEEE International Conference on Cloud Engineering (IC2E)*, pp. 124–131, 2019.

[14] M Vijayalakshmi and RI Minu, "Incremental Load Processing on ETL System through Cloud," in *2022 International Conference for Advancement in Technology (ICONAT)*, pp. 1–4, 2022.

[15] J Sreemathy, R Brindha, M Selva Nagalakshmi, N Suvekha, N Karthick Ragul, and M Praveennandha, "Overview of etl tools and talend-data integration," in *2021 7th International Conference on Advanced Computing and Communication Systems (ICACCS)*, vol. 1, pp. 1650–1654, 2021.

[16] Blend Berisha, Endrit Mëziu, and Isak Shabani, "Big data analytics in Cloud computing: an overview," *Journal of Cloud Computing*, vol. 11, no. 1, p. 24, 2022.

[17] Peter Garraghan, Paul Townend, and Jie Xu, "An analysis of the server characteristics and resource utilization in google cloud," in *2013 IEEE*

International Conference on Cloud Engineering (IC2E), pp. 124–131, 2013.

[18] Piotr Habela, Krzysztof Stencel, and Kazimierz Subieta. ”Three-level object-oriented database architecture based on virtual updateable views.” *International Conference on Advances in Information Systems*, pp. 80-89, 2006.

[19] Benjamin Reed Christopher, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins. ”Pig latin: a not–so–foreign language for data processing.” *In Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pp. 1099, vol. 1110, 2008.

[20] Ronnie Chaiken, Bob Jenkins, Per-Åke Larson, Bill Ramsey, Darren Shakib, Simon Weaver, and Jingren Zhou. ”Scope: easy and efficient parallel processing of massive data sets.” *Proceedings of the VLDB Endowment*, vol. 1, no. 2, pp. 1265-1276, 2008.

[21] Hadassa Daltrophe, Shlomi Dolev, and Zvi Lotker. ”Data Interpolation: An Efficient Sampling Alternative for Big Data Aggregation.” *arXiv preprint arXiv:1210.3171*, 2012.

[22] Linquan Zhang, Chuan Wu, Zongpeng Li, Chuanxiong Guo, Minghua Chen, and Francis CM Lau. ”Moving Big Data to the Cloud: Online Cost-Minimizing Algorithms.” *Networking for Big Data*, 2016. CRC Press, Taylor & Francis Group.

[23] Mingwei Lin, Zhiqiang Yao, and Tianqiang Huang. ”A hybrid push protocol for resource monitoring in cloud computing platforms.” *Optik*, vol. 127, no. 4, pp. 2007-2011, 2016. Elsevier.

[24] Xuan Liu and Feng Xu. ”Cloud service monitoring system based on SLA.” *2013 12th International Symposium on Distributed Computing and Applications to Business, Engineering & Science*, pp. 137-141, 2013. IEEE.

[25] JA Perez-Espinoza, Victor J Sosa-Sosa, and José Luis Gonzalez. ”Distribution and load balancing strategies in private cloud monitoring.” *2015 12th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE)*, pp. 1-6, 2015. IEEE.

[26] Mauro Andreolini, Michele Colajanni, Marcello Pietri, and Stefania Tosi. ”Adaptive, scalable and reliable monitoring of big data on

clouds." *Journal of Parallel and Distributed Computing*, vol. 79, pp. 67-79, 2015. Elsevier.

[27] Gokce Gorbil, David Garcia Perez, and Eduardo Huedo Cuesta. "Principles of pervasive cloud monitoring." *Information Sciences and Systems 2014: Proceedings of the 29th International Symposium on Computer and Information Sciences*, pp. 117-124, 2014. Springer.

[28] Mauro Andreolini, Marcello Pietri, Stefania Tosi, Andrea Balboni, and others. "Monitoring large cloud-based systems." *CLOSER 2014-Proceedings of the 4th International Conference on Cloud Computing and Services Science*, pp. 341-351, 2014. SciTePress.

[29] Javier Povedano-Molina, Jose M Lopez-Vega, Juan M Lopez-Soler, Antonio Corradi, and Luca Foschini. "DARGOS: A highly adaptable and scalable monitoring architecture for multi-tenant Clouds." *Future Generation Computer Systems*, vol. 29, no. 8, pp. 2041-2056, 2013. Elsevier.

[30] L. Degambur, A. Mungur, S. Armoogum, and S. Pudaruth, "Resource Allocation in 4G and 5G Networks: A Review," *International Journal of Communication Networks and Information Security (IJCNIS)*, vol. 13, no. 3, Dec. 2021.

[31] Li, He and Yu, Lu and He, Wu. "The impact of GDPR on global technology development" *Journal of Global Information Technology Management*, vol. 22, no. 1, pp. 1-6, 2019. Taylor & Francis.

[32] Lee, S., Levanti, K., Kim, H. S. "Network monitoring: Present and future." *Comput. Netw.*, vol. 65, no. 2, pp. 84-98, 2014. DOI: 10.1016/j.comnet.2014.03.007.

[33] Sherif Sakr, Anna Liu, Daniel M. Batista, and Mohammad Alomari. "A Survey of Large Scale Data Management Approaches in Cloud Environments." *IEEE Communications Surveys & Tutorials*, vol. 13, no. 3, pp. 311-336, 2011. doi: `10.1109/SURV.2011.032211.00087`.

[34] Muhammad Fajrul Falah, Yohanes Yohanie Fridelin Panduman, Sritrusta Sukaridhoto, Arther Wilem Cornelius Tirie, M. Cahyo Kriswantoro, Bayu Dwiyan Satria, Saifudin Usman. "Comparison of cloud computing providers for development of big data and internet of things application." *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 22, no. 3, pp. 1723-1730, June 2021. ISSN: 2502-4752. doi: `10.11591/ijeecs.v22.i3.pp1723-1730`.

[35] Carey, Michael J., Onose, Nicola, and Petropoulos, Michalis. "Data Services." *Communications of the ACM*, vol. 55, no. 6, pp. 86-97, June 2012. ISSN: 0001-0782. doi: `10.1145/2184319.2184340`.

[36] Machado, I., Carretero, J., & Filipe, J. (2022). "Data Mesh: Concepts and Principles of a Paradigm Shift in Data Architectures." *Procedia Computer Science*, 196, 263-271. International Conference on ENTERprise Information Systems / ProjMAN - International Conference on Project MANagement / HCist - International Conference on Health and Social Care Information Systems and Technologies 2021. ISSN 1877-0509. doi: `https://doi.org/10.1016/j.procs.2021.12.013`.

[37] Tayade, D. M. "Comparative study of ETL and E-LT in data warehousing." *International Research Journal of Engineering and Technology*, vol. 6, pp. 2803–2807, 2019.

[38] Bogdanov, D., Kamm, L., Laur, S., Pruulmann-Vengerfeldt, P., Talviste, R., Willemson, J. "Privacy-Preserving Statistical Data Analysis on Federated Databases." In *Privacy Technologies and Policy*, edited by B. Preneel and D. Ikonomou, pp. 30–55, Springer International Publishing, 2014.

[39] G. Satyanarayana Reddy, Rallabandi Srinivasu, M. Poorna Chander Rao, and S. Kanth Reddy Rikkula. "Data Warehousing, Data Mining, OLAP and OLTP Technologies are Essential Elements to Support Decision-Making Process in Industries." *International Journal on Computer Science and Engineering (IJCSE)*, vol. 02, no. 09, 2010, pp. 2865-2873.

[40] Ericsson. "Security Manager." Available online: `https://www.ericsson.com/en/portfolio/new-businesses/security-and-risk-management/security-manager` (Accessed on June 11, 2023).

[41] Nokia. "Cybersecurity Dome." Available online: `https://www.nokia.com/networks/security-portfolio/netguard/cybersecurity-dome/` (Accessed on June 11, 2023).

[42] Huawei. "eSight." Available online: `https://e.huawei.com/en/products/esight` (Accessed on June 11, 2023).

[43] Dgtl Infra. "Cell Tower Range: How Far Can They Reach?" Available online: `https://dgtlinfra.com/cell-tower-range-how-far-reach/` (Accessed on June 11, 2023).

[44] Operator Watch. "How Many Cell Towers/Base Stations Are There?" Available online: `https://www.operatorwatch.com/2020/08/how-many-cell-towers-base-stations.html` (Accessed on June 11, 2023).

[45] TechTarget. "Radio Access Network (RAN) - Definition." Available online: `https://www.techtarget.com/searchnetworking/definition/radio-access-network-RAN` (Accessed on June 11, 2023).

[46] RouterFreak. "Demystifying 5G RAN." Available online: `https://www.routerfreak.com/demystified-5g-ran/` (Accessed on June 11, 2023).

[47] 5G.SystemsApproach. "5G Architecture." Available online: `https://5g.systemsapproach.org/arch.html` (Accessed on June 11, 2023).

[48] Cisco. "Security Manager." Available online: `https://www.cisco.com/c/en/us/products/security/security-manager/index.html` (Accessed on June 11, 2023).

[49] Fauna. "The Why and How of Distributed Databases." Available online: `https://fauna.com/blog/the-why-and-how-of-distributed-databases` (Accessed on June 11, 2023).

[50] TIBCO. "What is a Data Federation?" Available online: `https://www.tibco.com/reference-center/what-is-a-data-federation` (Accessed on June 11, 2023).

[51] Amazon Web Services (AWS). "Data Warehousing." Available online: `https://aws.amazon.com/data-warehouse/` (Accessed on June 11, 2023).

[52] Rivery. "ETL vs. ELT: What's the Difference?" Available online: `https://rivery.io/blog/etl-vs-elt/` (Accessed on June 11, 2023).

[53] Google Cloud. "Google Cloud Storage." Available online: `https://cloud.google.com/storage`.