Predictive Maintenance Data System

(C00265741) - RAJAS VIJAYANAND BAKSH
(C00265740) - JACQUELINE KENNADY
(C00265987) - ELAINE KOYCE

Programme title: Master of Science in Data Science CWS07

Programme code: CWS07

Module title: Data and Data Storage Technologies

Contents

Introduction.

In an industry business, the primary driver for lost revenue is production downtime. American Productivity and Quality Center (APQC) estimates a loss between 40¢ to $1.2 for every $20 in revenue ("Avertle," n.d.). Equipment failure is a significant cause of unplanned downtime throughout the industries globally. In the earlier waves of the industrial revolution, the machines were not complicated and had fewer breakdowns. However, in the industrial revolution 4.0, the scenario has changed, and more focus has been turned towards reducing manual labor and increasing automation through sophisticated machinery. Thus, due to complex machinery, the unplanned breakdown has become a more common phenomenon, and minimizing it has become the need of the hour.

Maintenance costs between 15 to 60 percent of the goods manufactured by the industry. More than $200 billion is spent per annum maintaining the plant equipment (asset) just in the U.S. itself (Mobley, n.d.). The amount of money spent on maintenance can easily affect the profits of a company. The loss in manufacturing time and quality is an outcome of inferior maintenance management methods. Also, poor maintenance can makeover or under-utilization maintenance personals.

Thus, for this project, we have decided to take a real-world problem. The study aims to design data storage architecture for a product used to predict faults and the Remaining useful life of the machine. This product will be referred to as Prediction Unit throughout the study. This unit captures data from the machines using sensors. Sensors such as accelerometers, temperature sensors, and for some installations, even acoustic sensors capture data. The Prediction unit captures data twice every minute with high sample frequency and derives other essential parameters from the recorded data. The derived parameters are then stored in the database. As the data is captured at high frequency and apex sampling rate, storing the data and making it available to the Machine learning model maintaining low latency is quite essential. Also, the data storage system needs to be scalable as the data is being continuously captured and stored in the database. This study proposes and data storage architecture to achieve the expected results.

Problem Statement

The Prediction Unit collects the data from various sensors installed on the asset to be monitored. The parameters monitored depend on the asset and on the faults to be predicted for any particular asset. The vibration profile of an asset is captured using Accelerometers, while the temperature sensors and current transformers are used to measure temperature and current. However, for almost all the assets, the vibration and current profiles are monitored. The vibration data is used to capture faults such as unbalanced load, misalignment, defects in bearing, structural resonance, and other mechanical faults. On the other hand, current signatures are captured to detect faults such as earth faults, damage to the insulation, displacement of the conductors, thermal overloading (Siddique et al., 2005). About 37 % and 41 % of the faults occurring in an induction motor are due to the stator and the bearing, respectively. The effectiveness of detecting the fault in the asset depends on the sensitivity of capturing the data (Deeb and Kotelenets, 2020).

The Fast Fourier transform is performed on the current and vibration signature to analyze the data. The FFT helps to extract the information in the frequency domain from these signatures. These frequency components help in predicting the faults. As the faults occurring due to electrical stress can occur due to surges in operating voltage, monitoring the current signatures at high frequency is essential (Siddique et al., 2005). Thus, the Production unit captures the data twice in a minute.

The vibration signal captures the raw acceleration signal. Mathematical operations are performed to derive velocity and displacement signals from this data. Moreover, this time-domain signal is then converted into the frequency domain using FFT. An edge processing unit takes care of all these operations. The output of this edge processing unit is a JSON structure with the first 3000 frequency components of the Acceleration velocity and displacement along with the FFT component of the 3-phase current signal. The JSON structure also contains the values of temperature and time stamp at which the data was captured. Table 1 shows the high-level description of the data available in the JSON structure along with the memory required. The calculation mentioned in table 1 is by assuming that three triaxial sensors are installed on every asset.

Channels	Number of Data Points	Data Type	Data Size (Bytes)	Total Size (KB)
Acceleration FFT CH 1	3000	INT	2	5.859375

Acceleration FFT CH 2	3000	INT	2	5.859375
Acceleration FFT CH 3	3000	INT	2	5.859375
Velocity FFT CH 1	3000	INT	2	5.859375
Velocity FFT CH 2	3000	INT	2	5.859375
Velocity FFT CH 3	3000	INT	2	5.859375
Displacement FFT CH 1	3000	INT	2	5.859375
Displacement FFT CH 2	3000	INT	2	5.859375
Displacement FFT CH 3	3000	INT	2	5.859375
R-Phase FFT	3000	INT	2	5.859375
Y-Phase FFT	3000	INT	2	5.859375
B-Phase FFT	3000	INT	2	5.859375
Temperature	1	INT	2	0.001953125
Total in MB				0.0686
Total in MB for 10 Assets				0.686

Table 1

Thus, if there are at least ten assets in a plant, the memory required to store data every 30 seconds is 0.6 MB. This number does look minor, but as the data gets accumulated, the required memory to store data starts increasing. Figure 1 shows the projection of the memory requirement for one year.
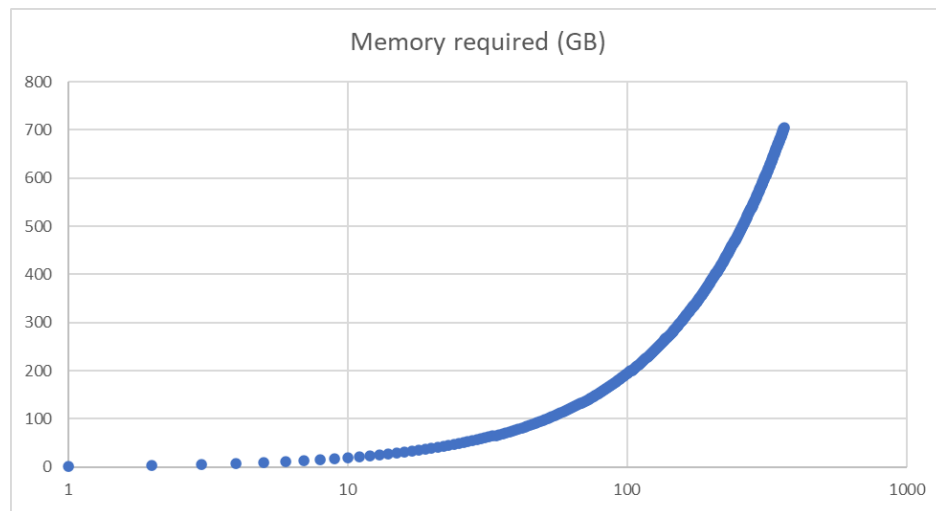

Figure 1

Figure 1 is a plot of required memory (GB) on the y-axis and the day of the year on the x-axis. The graph uses a log scale for the x-axis. The amount of required memory can be seen increasing

exponentially. Assuming at least ten assets are being monitored, by the end of the first year, ~800 GB of memory is required to store the data.

Thus, as the number of assets to be monitored increases, the slope of this graph will be steeper. Therefore, the proposed data architecture should be capable of storing data for at least five years amount of data with the feasibility of scaling. The proposed architecture should also have low latency while making available real-time data to the ML models to predict the remaining useful life of the data and archiving the older data so that the data can be further used for batch processing. Moreover, the architecture needs to provide the mapping information to the user. The data at the output of each edge processing unit is mapped to a unique id. The proposed architecture should also be able to provide the mapping information for the data acquired at each channel by the gateway.

DATABASE TECHNOLOGIES

Neo4j

Neo4j is an open-source, graph, transactional NoSQL database. It is one of the most popular graph database management systems and it is highly scalable and schema-free. It is accessed by using the Cypher Query Language and is way faster than traditional databases.

Neo4j Characteristics and Relevance

- o Data is stored and displayed in a graph in Neo4j with nodes representing the data and relationships between those nodes. This differs significantly from relational databases such as MySQL, where data is presented in a tabular fashion using tables, rows, and columns to store the data.
- o Neo4j is best for storing data that has many interconnecting relationships and for that reason, graph models work best with relational data, more so than relational databases.
- o There is also no need to create a database structure before you load the data like with relational databases, as graph models like Neo4j don't require a predefined schema as the data itself is the structure.
- o Primary Keys and foreign keys are not necessary with Neo4j as you just define the relationships between the nodes you require.
- o Complex joins are not needed with Neo4j to retrieve related data as it is a graph database, and all nodes are already connected. This makes Neo4j so fast because more connected data is easier to retrieve and navigate.
- o Neo4j provides results on real-time data.

Justification of Choice for Neo4j?

Neo4j allows you to query data for real-time operational workloads and graph analytics problems. The data is stored as a graph allowing a significant performance characteristic called the index-free adjacency, meaning that Neo4j can traverse between any nodes connected by having a relationship between them. They can traverse in real-time without doing the index look-up operation, meaning even complex traversals can express safer and scale to large datasets. Neo4j emphasizes consistency, availability and scalability but does not guarantee to partition. As for scalability and real-time operational workloads are essential factors in the design of our system. This makes Neo4j an excellent choice for us.

Neo4j Data Model

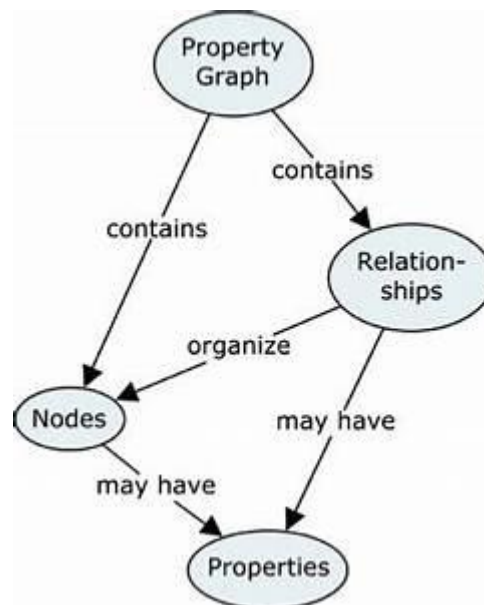Neo4j uses a property graph data model.

Figure 2

Figure 2 shows the graph model. Graph model has nodes, relationships and properties which specify data and its operation. The properties are key-value pairs. Relationship specifies the relation between two nodes and both nodes and relationships contain properties. Relationships should be directional in property graph data models. Labels are used to group nodes and each node can be assigned multiple labels. Labels are indexed to speed up finding nodes in a graph.

Neo4j Management and Usage Applications

Neo4j like are system requires constant monitoring and the Neo4j application manager's monitoring tool it can provide critical metrics identifying areas that need addressing, allowing

you to optimize the Neo4j servers. Neo4j's monitoring tool provides information about physical, virtual and swap memory usage allowing you to optimize the heap memory and non-heap memory usage so it can perform concurrent operations smoothly. Response time of database queries is quicker by identifying the threads taking the most time and prioritizing them. Store and property size details are available to forecast disk size and growth.

MySQL

MySQL is a fast, reliable, open-source, relational database management system (RDBMS) available since 1995 which has grown a dedicated community available for troubleshooting and suitable for production. The stability and reliability of MySQL are good while the database remains compact and has regular maintenance. MySQL enjoys frequent updates. It can be installed on production servers; developer computers and all operating system platforms are supported. The system is also secure.

.

MySQL Characteristics and Relevance

- o MySQL database relationship is defined in the form of tables made up of rows and columns and known as relations providing referential integrity between these rows or columns of various tables.
- o The table indexes can be automatically updated.
- o Databases created in MySQL allow you to create many tables to store and manipulate data defining the relationships between each of these tables. Clients then can make a request using SQL queries and then the server application will respond with the desired result from the query to the client.

Justification of Choice for MySQL?

MySQL is a relational database management system which is easy to use and can build and interact with MySQL using only a few simple SQL statements. MySQL is secure with a solid data security layer where sensitive data is protected from intruders and where passwords are encrypted. MySQL follows a client/server architecture where the database server and many clients communicate with the server, querying data and saving changes. It is free to download, so no cost attached. In terms of our company's system MySQL is scalable as it supports multi-threading and can handle any amount of data; however, an enormous database size isn't

supported as efficiently. The default size limit is about 4GB, but this can be incremental to a theoretical limit of 8TB data. It is also considered one of the faster database languages. MySQL also enjoys many embedded applications making it highly flexible, not to mention that it is compatible on many operating systems. MySQL allows transactions to be rolled back, commit and crash recovery. Memory is highly efficient as it has a low memory leakage problem. MySQL encourages higher productivity by developers by using Triggers, Stored Procedures and views. MySQL uses partitioning, which improves performance, adding to the fast management of the extensive database. MySQL is prone to data corruption, so it does not handle this very efficiently when it comes to transactions. It also doesn't support SQL check constraints. As we have very large datasets in our system SQL will not be the best fit for us.

MySQL Data Model

Example of tables and relations in a MySQL database. MySQL use ER Diagrams as shown in figure 3
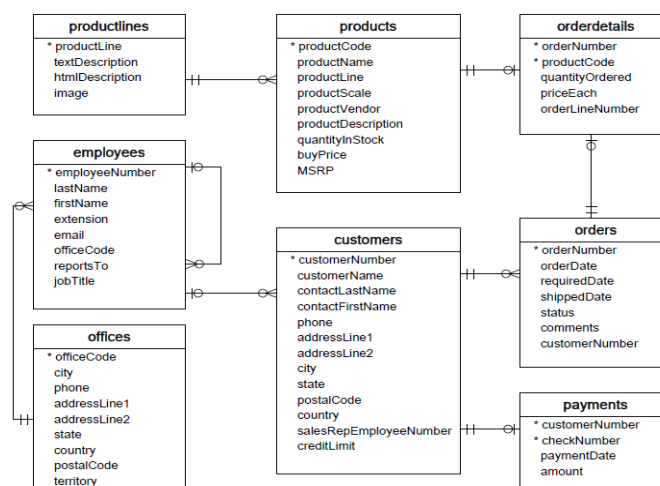

Figure 3

MySQL Management and Usage Applications

MySQL has a visual database GUI called "MySQL Workbench," used by database architects, developers, and database administrators on SQL development, data modeling, data migration, and a whole range of administration tools for server configuration, user administration and back up. There are many other applications out there designed to make working with MySQL databases easier depending on your required use of the database.

MongoDB

MongoDB is a source-available cross-platform document-oriented NoSQL database program. MongoDB uses MongoDB Query Language (MQL) and is based on JavaScript. It is easy to use and plenty of tools available to query MongoDB using SQL syntax. It relies on several drivers that allow its engine to interact with a wide variety of languages. MongoDB uses JSON like documents with optional schemas. MongoShell is based on JavaScript language and is available out of the box so to speak. MongoDB falls into the document store category. Generally, if your data is in self-contained documents and relationships rare, then document datastore is recommended.

MongoDB Characteristics and Relevance

o   MongoDB is ideal for small and continuous read and writes where the data may be volatile but needs fast-streaming processing and in-memory access.

o   Wide variety of access patterns across many different data types where the requirements are breadth search and depth search, pattern findings, long running queries, deep analytic type of usage.

o   CRUD applications where the requirements are to access complex data without joins.

o   When programmer friendliness is a priority, and the requirements are rapid application development and deployment.

o   Large media - BLOB types that support large media data types, caching of web pages or saving complex objects that were expensive to join in a relational database.

o   Easier upgrade options where the requirements are fluid schema system that supports optional fields, adding fields, deleting fields without building an entire schema migration framework.

o   Transactions are supported, so it can be considered if transactional data you want to store; SQL or relational databases are not the only options in this case.

MongoDB stores data in collections, documents, and fields. This is in contrast with traditional RDBMS that uses tables, rows and columns as shown in figure 4.
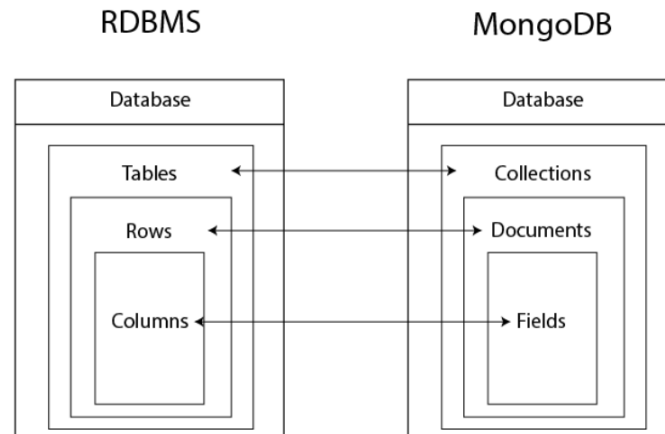
Figure 4

Justification of Choice for MongoDB?

MongoDB supports replications and sharding like other NoSQL datastores. If you have extensive scale data, you will use the distributed features of MongoDB. Multi-document transactions support transactions, unlike most other NoSQL databases. Transactions in MongoDB guarantee atomicity and consistency, meaning immediate consistency as cannot read until all is updated. Data changes are going to be invisible until that transaction is completed. Atomicity is all or nothing.

A transaction is done across multiple documents or collections, or datasets and transactions are supported even if you must update multi-documents. They might be on different servers or hardware devices, but you need to update all these devices. Therefore, multi-document problems are associated with performance. So, to avoid this issue, we use denormalization. Therefore, you need to design your database correctly, bearing in mind performance implications.

MongoDB as multi-cloud data distribution allows you to distribute or partition data across over 75 cloud regions on AWS, Azure and Google Cloud. MongoDB enjoys sophisticated security controls and advanced functionality to satisfy both new and existing data privacy and compliance measures. You can manage security over three different areas. Access to MongoDB itself, database level and network level, setting up users, permissions, and passwords. You can invite users and set up teams of users using the access manager tab.

Document database like MongoDB eliminates lots of joins and no immediate need to normalize data like you would in SQL. MongoDB makes a trade-off between efficiency and consistency. The rule is, changes to a single document are always atomic, while updates to multiple documents should never be assumed to be atomic.

MongoDB is an excellent choice for our system considering the types of data we are working with.

Range partitioning or Hash partitioning in terms of MongoDB

Partition documents in MongoDB into multiple shards where each shard has replica's and are the secondary nodes. There need to go through the server for clients or the application to access the data or documents on the various shards. It cannot directly communicate with them but has to go through the primary router. Slave nodes are used for backups.

The partitioning for shards in MongoDB can be used for range or hash partitioning. Range partitioning is preferable for queries that look for a range of items from for e.g. (n to 20). Hash partitioning is used for uniform distribution of the data across the servers and allows for better load balancing. Range can cause hotspots or blockages, but hash partitioning is more evenly balanced.

The hash function that is used to create the hash keys for the documents does not have the ability to have strong cryptography as it is not about security and more about having those random hash keys to distribute data across the different partitions. (Availability over security)

Keys divided into chunks and each chunk has what will be stored in the shards. You don't have to partition all the collections that you have in your system either. You can partition the big collections between multiple servers and if you have a collection that can be hosted in one of the servers you host in the first shard of the cluster. These are the things we identify in our cluster when designing.

MongoDB Data Model

Normalized or Embedded data models are the choices for MongoDB. Model one-to-one relationships with embedded documents presents a data model where embedded documents to describe one-to-one relationships between connected data. Model one-to-many relationships with embedded documents presents a data model where embedded documents describe one-to-many relationships between connected data. Model one-to-many relationships with document references presents a data model that uses references to describe one-to-many relationships between documents.

It is designed to aid developer performance, allowing them to build applications faster with drivers, integrations, and tools to manipulate, analyze and visualize data. Availability is good with distributed fault tolerance and backup options to meet data recovery needs. Backups offer snapshots in the last 24 hours so that if you need to restore your database, you can do so at any given point of time from when those snapshots were taken.

MongoDB has on-demand scaling and real-time insights on your database performance. With real-time monitoring, you can see what is happening right now. You can get information about operations for e.g., Query execution times, query targeting and read and writes.

You can use the profiler to identify in a graph and title view any slug queries that are appearing and then in the performance advisor it will suggest indexes that could be implemented to improve performance. You can set up alerts on the project alert page, which is useful for e.g., setting up storage use warnings.

STORAGE ARCHITECTURE TECHNOLOGIES

The storage architecture of your system is one of the most critical components of data transfer and accessing information. It provides the foundation for data access across an enterprise.

The reason for not storing databases in the main memory is the size of the data. The main memory is usually limited in terms of size but growing fast in recent years but still hard to accommodate large amounts of data in the main memory with the cost. Another reason is that it is also volatile. If your system shuts down, you lose the data. There are many in-memory databases but not used for persistent data but transient data. Buffering data enables control of the size of the buffers within memory and adjusts buffer size to improve the performance of the database.

Organizing data on disc on the continuous block with magnetic disc read/write if closed to each other then it makes things faster than if they are spread everywhere across the magnetic disc…data being continuous.

Reading data ahead of when required, usually move data ahead of time and copy into main memory before it is needed. Use algorithms to predict what data is going to be needed next by queries if the database wails until the query needs the data it is going to be slow.

Proper scheduling of in/out requests allows magnetic discs to access the discs more easily and usually moves one way.

Ways to improve performance

Some of the data can be more critical than the rest of the data in the database and can be arranged so more critical ones are stored in faster storage. Some technologies allow you to control that automatically and after some time, if data is not accessed it is moved to another storage layer where it is less expensive to maintain, and you still have the data if you need it.

RAID

Raid is used for improving performance and was developed in the 80's as that time, CPU speed was increasing over the years, was very fast and so was a memory in size, but magnetic discs were not catching up with the processing speed. Physic laws restricted them as read/writes head cannot be made much faster than it is. This problem was solved by combining several discs together and writing data to these multiple discs. The result was the better speed in writing and reading of the data. The discs can combine into the array in different RAID levels, with each level having its own characteristics. The levels are from 0 to 6 however with level 1 and 5 being the most popular.

With RAID if one of the discs fail it is going to fail the whole RAID of the data is going to be lost as you lost one disc. There are ways around that, so the solution is to have redundancy. Parity blocks these would allow you to store another copy of the data in the case on copy fails you still have data to recover. Increase the write input/output as have to write to multiple discs at a time and do a lot of computation. Increase disc users and replicated data on multiple discs.

Mirroring means that you have multiple discs that store the same data like replication as disc in database before. Parity discs stores error correction data itself and in case of failure use the error correction data. The different levels of RAID as are follows.

LEVEL 0 (Stripped discs) - if one disc fails you lose data. This is used where you are more concerned about speed than fault tolerance. Level 0 architecture is shown in figure 5.
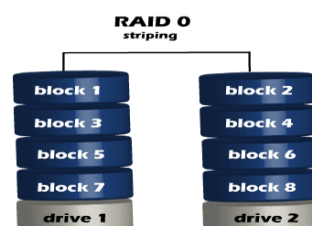

Figure 5

LEVEL 1 (Mirrored discs) - if you need fault tolerance use level 1. Block 1 exists on both discs. Level 1 architecture is shown in figure 6
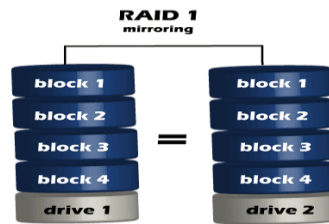

Figure 6

Level 1 is for more critical data and ok with redundancy in case of failure with one disc you have it on another disc.

LEVEL l 2,3, 4 - not used as much.

LEVEL 5 (Stripped discs with parity across drives) - used more frequently and splits at block level. It uses what is called parity blocks so this allows us in case one disc fails you can recover data using those parity blocks. This improves performance as you can read from multiple discs at the same time and improves redundancy as failure of one disc can still recover. Level 5 architecture is shown in figure 7.
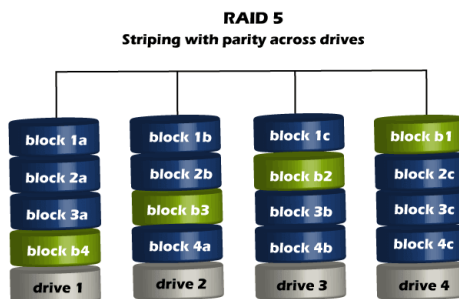

Figure 7

LEVEL 6 - similar to level 5 but main difference is that you have two parity blocks. If two discs fail, you can still recover your data.

Most common levels used are 0,1 and 5. 0 has no redundancy, 1 has parity discs and 5 has parity blocks mirrored.

RAID has scalability but not to the level of big data as you have more discs it is going to cost more. You also and have to be able to deal with redundancy and tolerate the costs of that redundancy. It is ok for terabytes of data but nothing bigger. For these reasons RAID is not one I would be recommending we use in our system.

Network attached storage (NAS)

This is storage attached to your network and is accessible from all the clients and is used by small and medium sized business. It has protocols that allow you to see the drive on your system as if they are directly attached to the system while they are actually in the network. The idea is to improve scalability for storage while also reliability, flexibility and performance. You can scale for all the users in your network.

Storage Area Network (SAN)

This allows high speed storage and each company that is offering storage in networks will be using fibre channel techniques specifically made for these types of users. It does not use IP but its own protocols to access the data. Storage area networks can also use fibre channels over the ethernet and IP network which is already in place rather than have to invest in fibre channel. Users can see the storage area network as if it is connected to their system.

The difference between NAS and SAN is that NAS is simply storage connected to the existing network and SAN could be specifically made to provide storage to the companies' network. SAN architecture is shown in figure 8
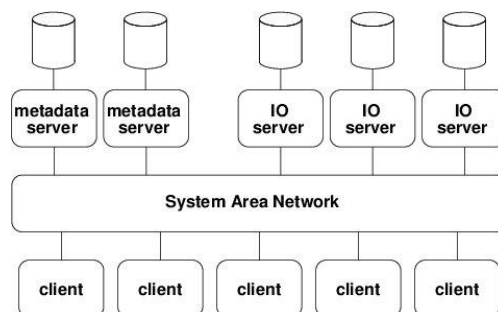

Figure 8

Object based storage

Object based storage is newer storage used to store data as objects rather than files. It is very scalable and can store vast amounts of data, usually unstructured data. It can retrieve the data using the global identifier which makes accessing the data much faster. It can also store transactional data. Facebook use it to store media file, images etc. The way it is done is to store references to images and other files instead of storing the object itself as that will be stored in other forms of storage and use databases to access them from storage. Another advantage of object storage is there is less need for system administrators, and it is fast, and you simply use an

API to access the objects.Figure 9 shows the object based storage. The main advantages of Object based storage is its resource optimisation, reduced cost, faster data improvement, infinite scalability can scale boundlessly to petabytes and the availability for data analytics.
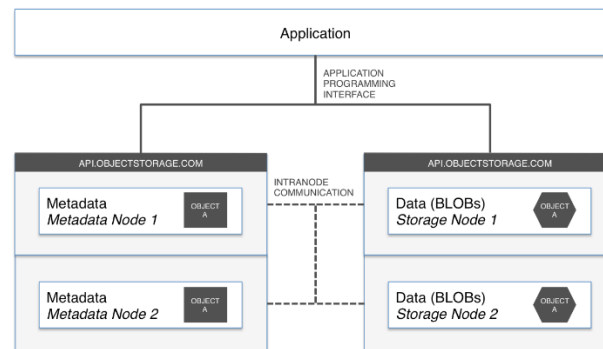

Figure 9

Hadoop Distributed File System (HDFS)

Hadoop Distributed File System is a core component of Hadoop making it possible to store large data sets of different types of data such as structured, unstructured, and semi-structured data. Data is distributed over a number of machines and duplicated to protect their durability to failure and high availability to parallel application. It is cost effective as it works on low-cost hardware. You can see the whole HDFS as a single unit even though your data is stored across various nodes in the cluster. The HDFS will maintain a log file about all the stored data and has two core components which are DataNode and NameNode. The NameNode contains metadata i.e., the log file and needs less storage as doesn't store actual data. The DataNodes need much more storage as all your data is stored on these but these nodes are typically very inexpensive and widely available.

HDFS can scale hundreds of nodes in any one cluster making it highly scalable. Should a node containing the data be lost the HDFS maintains a copy of the data on a different machine. The HDFS is highly fault tolerant, should any machine fail, another machine containing a copy of that data automatically takes over. The distributed data storage is the main feature of the HDFS, here data is spread into multiple blocks and stored into nodes and is what makes Hadoop so powerful. Finally, such is the design of the HDFS that it is easily portable from one platform to another.

MQTT Protocol

Message Queuing Telemetry Transport (MQTT) is an internet of things connectivity protocol. It is machine to machine, lightweight and we can publish and receive the messages as a client making it simple to communicate between multiple devices. As we are considering the internet of

things applications in our design, MQTT is an ideal choice for use in our system also. MQTT provides real time messaging protocol which is so important in our system design.

Some of the components of MQTT;

- **Message** this is the data which is carried out by the protocol across the network for the application.
- **Client** is a device which opens the network connection to the server.
- **Server** is a device that allows a client to publish and subscribe to the messages.
- **Topic** is when the label attached to the messages is checked against the subscription which is known by the server.

AWS S3 Bucket

S3 which stands for Simple Storage Service provides secure, durable, highly scalable object storage. A user-friendly interface makes it easy to store and retrieve huge amounts of data from the internet. It is object-based storage meaning that you can store the word documents and image files and these files can range in size from 0 bytes to 5 terabytes. It enjoys unlimited storage so you can store as much data as you require.

S3 has bucket which is like a folder that stores the files. The bucket requires a unique name so it can create a unique DNS address. These buckets have no max limit for data you can store in them. Data can also be downloaded from this bucket allowing you to grant permission to others to also download this data if required. A developer assigns a unique key by which each object can be stored and retrieved. The data is secure due to this authentication method. These numerous features of S3 makes it a good object-based storage solution for our system.

S3 is a key-value store and is object-based meaning that it contains the following.

- The name of an object is called the **Key** and is used to retrieve the object, e.g. project.txt.
- The data inside of the file itself is called the **Value** consisting of a sequence of bytes.
- Version ID is a string generated by S3 which uniquely identifies the object when added to a S3 bucket.
- The data about the data you are storing is called the **Metadata**.
- You can add permissions to individual files using **Access Control Information.**

IoT Gateway

An internet of things (IoT) gateway is a device that connects IoT devices, equipment systems, sensors and the cloud. The devices in the field are connected to a centralised cloud and this gateway allows processing locally, storage solutions and also the power to independently control field devices based on data input by sensors. They can be a virtual or hardware tool and receive their data from IoT sensors which can be sent or received from the cloud which means the data can go to the device itself also and therefore goes through the connected gateway in either direction. IoT gateways security is excellent and prevent outside parties from unauthorised access.

IoT Rules

You interact with the AWS services using rules. The rules are inspected and executed based on the MQTT topic stream. Rules are used to perform a number of tasks like filtering data received from a device, writing data received from a device to a database, saving a file to your S3 bucket, extracting data or sending message data to a web service as only some of the examples. AWS IoT rule actions specify what to do when a rule is triggered and supports sending a message to an Apache Kafka cluster. I will advise on Apache Kafka next.

Apache Kafka

Apache Kafka is used to handle real-time data storage. It is open-source and a distributed stream processing software platform. It works as an intermediary between two parties and can handle huge amounts of data events each day. It is a publish-subscribe message system which allows exchange of data between processors, servers, and applications. The source system or producer data is sent to the Apache Kafka, here it decouples the data, and the target system consumes the data from Kafka. Apache Kafka has latency value of under 10ms which is low and earns it praise as a well-versed software. Apache Kafka maintains fault-tolerance and has resolved the problem of reprocessing data. Apaches Kafka's core capabilities are its high throughput, scalability, permanent storage, and high availability and is a good fit for our system.

Proposed System Architecture

The data related to predictive maintenance system are time series data from various sensors and the

metadata – plant details, sensor information, etc. The main database used for storing time series data is MongoDB due to its configurations which supports for storing high volume of data. The data captured from the sensors are used for both real-time analysis and also as an architecture resource for later usage. Hadoop hdfs is used for storing these architecture data. Figure 2 is the pictorial representation of the back-end architecture of the predictive maintenance system.

Solution Architecture Diagram

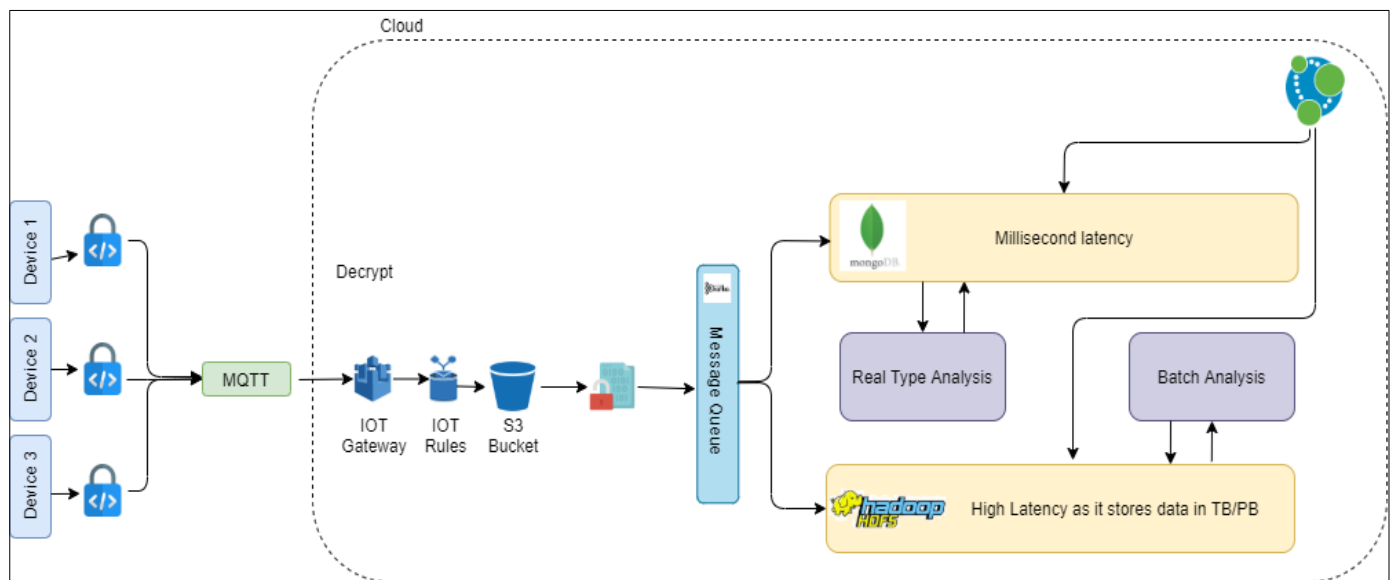The proposed architecture is shown in figure 10.

Figure 10

Data Encryption and Decryption

Data from a variety of devices is encrypted and transfer to the cloud. DES encryption is used to encrypt/decrypt these data. DES offer wide range of specifications for electronic data encryption. DES is a 64-bit symmetric block encryption algorithm which works on 64-bit blocks of plain text. As it is symmetric in nature same key can be used for encryption and decryption. Initially it divides 64-bit block of plain text into two 32-bit blocks by applying initial permutation. The two each blocks then performs 16 identical operations, called rounds. Then first inversion of the permutation is performed after connecting the two halves. The purpose of the first implementation is clear which does not affect the security of the algorithm. Therefore, small blocks of plain text and cipher text can be loaded into an 8-bit chip. Only one divided portion of the original 64-bit block is used in

one run. So, we need to   alternate the rounds between the two halves.

A messaging protocol MQTT is used for sending high volume of sensor messages to analytical and cloud storage. MQTT is a lightweight publish/subscribe messaging protocol were designed for low-bandwidth, high latency, unreliable networks and it is residing on the top of TCP/IP network. These characteristics make MQTT as a good option for sending high volumes of sensor messages to cloud solutions. This protocol consists of three components such as subscriber, publisher and broker. The broker manages the message distribution, and these messages are stored as topics.

Why MQTT used in this architecture

The following characteristics make MQTT as a good choice for our architecture:

- **Lightweight and Efficient**: MQTT client are smaller in size as well as require minimal resources, thus   it can be used in small microcontrollers.

- **Reliable message delivery:** MQTT defined 3 levels of quality of service, which makes the message delivery more reliable.

- **Scale to millions of things**: MQTT can scale to connect with larger number of IoT devices.

Internet of Things

IoT Gateways: The main function of IoT gateway is receiving the data from sensor devices and transmit it to the data centers for further processing. The gateways can also be used as a filter which may drops the noise, select data of interest and also aggregate data from different IoT devices. The gateways transform data into a usable format, and this can be used for data analysis in later. It can be also used to optimize routing in the network layer. In most implementation the MQTT broker run on the server machine networks. Here Gateways listen to downlinks by subscribing to the topic and sends uplinks on the topic. When an uplink is sent, the Gateway writes the physical payload received from the device with additional data about it.

Significance of IoT Gateway in this system

- IoT Gateway provide a single point of contact for sensors with external networks by using any type of network connectivity.

- It gathered information from all sensor devices and perform pre-processing on the collected data before it sends to the cloud storage.

- It should act as a single point of access for monitoring the selected area of the operational field.

IoT rules in the system send data from one device to other by listening specific MQTT messages and the data format in the message payloads. Based on the MQTT topic stream rules are analyzed

and then the actions are performed. The rules to support task may like this:

- Filter the data received from the devices.
- Write data received from a device to the data store.
- Save a file to Amazon S3.
- Send the data from an MQTT message for the predictions based on ML model.

S3 buckets are the storage containers for the objects which provide industry-leading scalability, data availability, security, and performance. Here the objects in the S3 bucket are data files which contain time series data from the sensors as well as the metadata. We created an object in s3 bucket with specific key name which uniquely identifies the object in the bucket. The data inside the bucket is the captured temperature from the various sensor devices. Then a version Id is generated and attached to each object while it adding to the s3 bucket.

Kafka as a Message Queue

Apache Kafka, an open-source program developed to manage large volumes of data ingestion, which is one of the core components of a stable IoT data network. It serves as an entry point for the data processing pipeline. Kafka is a quick messaging system that makes good use of IO by batching and compressing records. Data sources are decoupled using Kafka.

It acts as a high-performance ingestion layer for sensor data. The data points generated by various sensors are gathered at high frequency. These massive set of data are ingested into the data processing pipeline for storage, transformation, processing, querying, and analysis. The gateway pushes the data to Kafka cluster and then the data driven to multiple paths. Data that need for real time analysis go to the MongoDB data store and architecture data writes to Hadoop HDFS perform batch analysis in later.

Why Apache Kafka is used?

- It acts as a unified platform for handling all the real-time data and supports low latency message delivery which gives guaranteed fault tolerance.
- The ability of handling large number of diverse consumers makes Kafka more suitable for IoT architecture.
- Increased ingestion rate which makes faster transferring of data from a page cache to a network socket.
- Supports scheduled insertion in data stores.

MongoDB as Primary Database

While working on time-series data we need to understand how data is going to be created, queried, and expired. But the main challenges about designing a data store for this kind is it not only meet the current needs and expectations but meets all the requirement of future projects in whatever sizes and shapes they come in. Basically, the solution needs to be scalable. So, the document database is the best choice. The better write performance and throughput make MongoDB as good choice for ingesting time series data.

In mongo DB, a popular approach for organizing time-series data is to divide it into buckets, each of which represents a uniform unit of time, such as a day or a year. Bucketing organizes small categories of data to make it easier to find what you're looking for:

- Discover historical trends
- Forecast future trends
- Optimize storage usage.

Using Bucket Pattern in MongoDB

Time series data come like a stream of data over a period of time, so we need to store each measurement in its own document. However, this method is a very relational approach to handling the data. Suppose we have a sensor taking the temp and saving it to the database every minute, our data stream might look something like:

```
{

sensorId: 12345,

timestamp: ISODate("2019-01-31T10:00:00.000Z"),

temp: 40

}

{

 sensorID: 12345,

timestamp: ISODate("2019-01-31T10:01:00.000Z"),

temp: 40

}

{
```

```
    sensorID: 12345,

    timestamp: ISODate("2019-01-31T10:00:00.000Z"),

    temp: 40

}

{

    sensorID: 12345,

    timestamp: ISODate("2019-01-31T10:01:00.000Z"),

    temp: 4}
```

Here the data is stored as a single document like one document per seconds. But this is a very relational approach for storing data. So, we apply bucket pattern to our data model. There are two type of bucketing.

- Time based bucketing
- Size based bucketing

The time-based bucketing collects the data as a single document in every minute. The data generated in time-based application are not in a fixed interval of time, so in this case time-based bucketing is not an optimal option

```
{
_id: ObjectId(),
deviceid: 1234,
sensorid: 3,
nsamples: 5,
day: ISODate("2018-08-29"),
first:1535530412,
last: 1535530432,
samples : [
 { temp: 50, time: 1535530423},
{ temp: 55, time : 1535530427},
 { temp: 56, time: 1535530432},
{ temp: 55, time : 1535530440},
```

```
{ temp: 56, time: 1535530451}
] }
```

In size-based bucketing insert one documents per a certain number of emitted sensor events, or for the entire day, whichever threshold is reached first.

Secondary Indexing

Secondary indexing in mongo dB provide better way to accessing the documents. The indexes can be defined in ascending or descending order. Mongo DB provide a default indexing in each document. But querying a document with this primary index may increase the read latency and decrease the query performance. So, we set additional index to each document. In this system sensor Id and Timestamp set as a secondary index.

Data Retention in MongoDB

Data Retention is one of the most important things to address in time series application. In our system we need to store the real time data for 1 year and after that specific period it can be discarded. For this purpose, use methods such as TTL indexing, remove documents using Remove Statement or by using drop collections.

A TTL index is very similar to the regular index which specify a time interval to automatically remove the document from database.in our case we need to retain the collections for one year only, so create a TTL index to remove records older than one year.

The remove statement can be used to remove data from a MongoDB list, but we'll need to write an automated script to clean out the data. By using the remove command or TTL index cause high disk I/O where the system is already in high load these techniques are not desirable.

Using drop collection command is the most efficient and faster way to perform deleting data and indexes.

Reason for selecting MongoDB for real time processing

- **Flexibility:** Dynamic nature of data store helps to add new elements to schema easily. So, the storing and **processing of any kind of data can be done without any difficulties.**

- **Secondary Indexing:** It provide a flexible access to the data and improved the query performance.

- **Size-based Bucketing**: It reduce the number of documents in collections and improve indexing performance.

- **Data Retention**: Retain the data in real-time database for a specific time period and after that it removed by dropping the data collection as well as the index.

Neo4j for Storing Metadata

Neo4j is a graph database software designed to store, query, interpret, and process closely linked data faster than most databases. Both vertically and horizontally, the Neo4j database is extremely elastic without compromising data confidentiality or accuracy.

Present architecture reference models were considered and components of each synthesized into a software stack for smart application creation to meet the need for controlling the convergence of connected devices and enabling new business models from heavily interconnected networks. This implements a graph database for managing and maintaining connected components. In this system each component's virtual and physical connectivity, technical functionalities, and state are considered nodes.

The nodes in the data model are the physical components such as sensors or other data gathering devices. Neo4j emphasizes on consistency, availability and scalability but does not guarantee partitioning. As scalability and real time operational workloads are essential factors in the design of our system. This makes Neo4j an excellent choice for us.

Hadoop HDFS

Hadoop distributed file system is the primary storage in the Hadoop architecture which provide high throughput access to the application data.

The main reasons Hadoop is best suited to work with IoT are given below:

- Increased ability to store vast amounts of heterogeneous data: In IoT architecture, need to handle a large volume of structured and unstructured data. So, the hdfs have the ability to accommodate the data of such kinds.
- Fault Tolerance: Hardware failure does not affect the data and data processing applications. Jobs are automatically routed to other nodes if a node fails, ensuring that distributed computation does not crash. All data is simultaneously duplicated and stored in several locations.
- Flexibility: Data pre-processing is not necessary to store data to the data store.
- Low cost: For storing large quantity of data, it uses commodity hardware and also it is a free open source framework.

Less administration is required and can be easily scalable.

Batch Process Analysis

Batch processing is a periodical job that performs the computation in stored data. It operates in a batch of data and then stores it in some where else. It is a cost-efficient method, and it can process

a large volume of data efficiently.

Migration of existing system to proposed system.

After understanding the benefits of the proposed architecture, the next question left to answer is how we can migrate the system from older architecture to new architecture without downtime? The potential problem can be seen while committing the data into MongoDB while handling the real-time data.

In MongoDB, about 30% of existing databases are migrated from a relational database. Many industries in the market face the problem that we are addressing here. The unrelating growth in the data sources is pushing many organizations towards nonrelational databases. MongoDB has published a white paper addressing ways to migrate from relational database to nonrelational database ("SQL Database to MongoDB Migration Guide," n.d.).

Importing data from current relational databases to MongoDB can be done in a variety of ways. Own scripts to convert source data into a hierarchical JSON format imported into MongoDB with the mongoimport tool. In our case, we can create feeds from the source systems, dump regular updates from an existing RDBMS to MongoDB to execute concurrent operations, or perform application creation and load testing It's essential to think about how to manage data deletions in the source system while using this method. One approach is to use MongoDB to build "A" and "B" target databases, then swap regular feeds between them. Database A provides one regular feed in this case, after which the program transfers the next day's feeds to Database B. Meanwhile, the original Database A is decommissioned, and as new feeds to Database A are made, a whole new instance of the source database is produced, meaning that deletions are synchronized with the source data.

Conclusion of our findings

What our problem was?
Our study was to design data storage architecture for a product used to predict faults and the remaining useful life of the machine. The unit captured data from the machine using sensors and the derived parameters are then stored in a database.

What solutions did we use and why?

- Encryption – DES as the encryption as encryption standard.
- MQTT - The protocol's light weight makes it suitable for use on both compressed computer hardware and low-bandwidth networks.
- IOT Gateway - Receive the data sent from device
- IOT Rules - To perform the basic operations on the received data.
- S3 Bucket - Received the data and stored it into the S3 bucket.
- Kafka – Used to queue the encrypted message and schedule the insertion of data into the MongoDB an HDFS

The technologies which we used can help us to scale easily. The solution is horizontally scalable. Multi indexing allows for quick access to the documents in MongoDB providing us with real time access to these documents. The archived data after one year can be found in the Hadoop distributed file system and can be used occasionally for data analysis. As every data is uniquely identified by its tag ID and the tag ID is highly relational data, Neo4j makes it easier to analysis and map this type of data

What result did we achieve?

We achieved architecture that can store data for real time data processing as well as for batch processing or archiving the data. Even though the data appears to be structured, the benefit of storing it into nonrelational databases is that it helps us to perform quicker operations on the databases and scalability. The proposed system is designed to cater for the scalability needs for the future.

Team Collaboration

- We agreed communication platforms in first lab also setting up a project what's app group.
- Discussed ideas for projects and if anyone had idea they wished to run with.
- Rajas had idea from company he previously worked for and we agreed to use that.
- At first, we thought project was to be practically implemented so at this point. Rajas set up cloud SDK and databases, SQL and MongoDB, gave us access and sent documentation to inform us about using cloud SDK.
- We all accessed it and ensured we were able to access databases via Jupiter notebook.
- We all looked for different types of datasets for data we could use vibration, thermal images and acoustic datasets such would be the data we would be working with in the project.
- All information was shared and discussed once collated.

- Once clarification that project was more theoretical rather than practical, we divided up the tasks of completing the report into the following.

  Introduction and problem statement – Rajas

  Individual Technologies – Elaine

  System Design - Jacqueline

  Solution Architecture Diagram – Rajas

  Solution Decision – Discussed by all written by Jacqueline

  Conclusion - All

  Team Collaboration -Discussed by team and written by Elaine

- Discussion and meetings via Microsoft teams in discussion for possible solutions and technologies to use.

- Everyone prepared the slides from the research they prepared for report and Rajas collated at the end to put theme to these slides.

- Meetings for presentation preparation where we practised to ensure we were within the required time limit.

- All sections of report when finalised shared via email and collated into report with table of contents by Elaine

References

https://www.ltts.com/solutions/avertle (accessed 4.21.21).

Deeb, M., Kotelenets, N.F., 2020. 2020 Int. Youth Conf. Radio Electron. Electr. Power Eng. REEPE Radio Electron. Electr. Power Eng. REEPE 2020 Int. Youth Conf. On 1–6.

Mobley, K., n.d. An Introduction to Predictive Maintenance - 2nd Edition [WWW Document]. URL https://www.elsevier.com/books/an-introduction-to-predictive-maintenance/mobley/978-0-7506-7531-4 (accessed 4.21.21).

Siddique, A., Yadava, G.S., Singh, B., 2005. IEEE Trans. Energy Convers. 20, 106–114.

Some Database Technologies information is from previously submitted report by Elaine Koyce c00265987 on Lab Report 3 for this same module as I also completed some similar topics in this paper.

Agarwal, S., n.d. Importance of RAID in Databases. URL https://www.sqlservercurry.com/2013/09/importance-of-raid-in-databases.html (accessed 4.26.21).

AWS IoT rule actions - AWS IoT Core [WWW Document], n.d. URL https://docs.aws.amazon.com/iot/latest/developerguide/iot-rule-actions.html (accessed 4.26.21).

Building with Patterns: The Bucket Pattern | MongoDB Blog [WWW Document], n.d. . MongoDB. URL https://www.mongodb.com/blog/post/building-with-patterns-the-bucket-pattern (accessed 4.26.21).

Devulapalli, A., Dalessandro, D., Wyckoff, P., Ali, N., Sadayappan, P., 2007. Integrating parallel file systems with object-based storage devices. p. 27. https://doi.org/10.1145/1362622.1362659

MongoDB Design: Tips AND Tricks - DZone Database [WWW Document], n.d. . dzone.com. URL https://dzone.com/articles/mongodb-design-tips-amp-tricks (accessed 4.26.21).

MQTT - The Standard for IoT Messaging [WWW Document], n.d. URL https://mqtt.org/ (accessed 4.26.21).

Object storage, 2021. . Wikipedia.

Perform CRUD Operations in Atlas — MongoDB Atlas [WWW Document], n.d. . Httpsgithubcommongodbdocs-Bi-Connect.-3279sourceindextxt. URL https://docs.atlas.mongodb.com/data-explorer/ (accessed 4.26.21).

Rules for AWS IoT - AWS IoT Core [WWW Document], n.d. URL https://docs.aws.amazon.com/iot/latest/developerguide/iot-rules.html (accessed 4.26.21).

Team, E., n.d. Amazon S3 and its Benefits [WWW Document]. URL https://www.linkeit.com/blog/what-is-amazon-s3-and-its-benefits (accessed 4.26.21).

The Architecture of IoT Gateways - DZone IoT [WWW Document], n.d. . dzone.com. URL https://dzone.com/articles/iot-gateways-and-architecture (accessed 4.26.21).

Time Series Data and MongoDB: Part 2 – Schema Design Best Practices | MongoDB Blog [WWW Document], n.d. . MongoDB. URL https://www.mongodb.com/blog/post/time-series-data-and-mongodb-part-2-schema-design-best-practices (accessed 4.26.21).
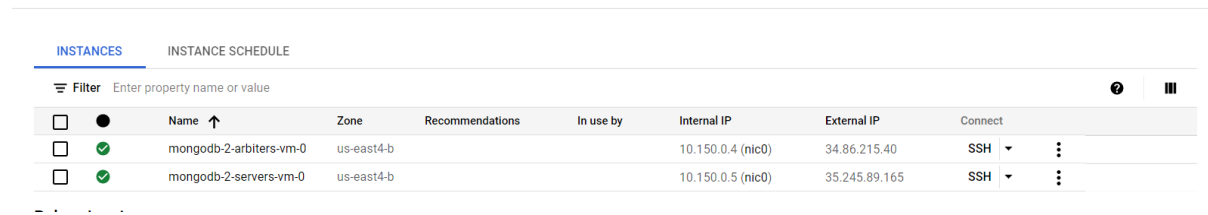
What Is Hadoop & How Does It Work? [WWW Document], n.d. URL
https://www.sas.com/en_ie/insights/big-data/hadoop.html (accessed 4.26.21).

What is Storage Architecture | Storage Architecture Tips [WWW Document], 2020. . Worldw.
Serv. URL https://worldwideservices.net/what-is-storage-architecture/ (accessed 4.26.21).

SQL Database to MongoDB Migration Guide [WWW Document], n.d. . MongoDB. URL
https://www.mongodb.com/lp/white-paper/trail/migration-rdbms-nosql-mongodb (accessed
4.26.21).

Appendices

Figure 11 shows the installed instance of MongoDB instance on GCP

Figure 11

Figure 12 is a snapshot of the sample document inserted into MongoDB. The use of multi-indexing can be seen in the below screenshot.

Figure 12

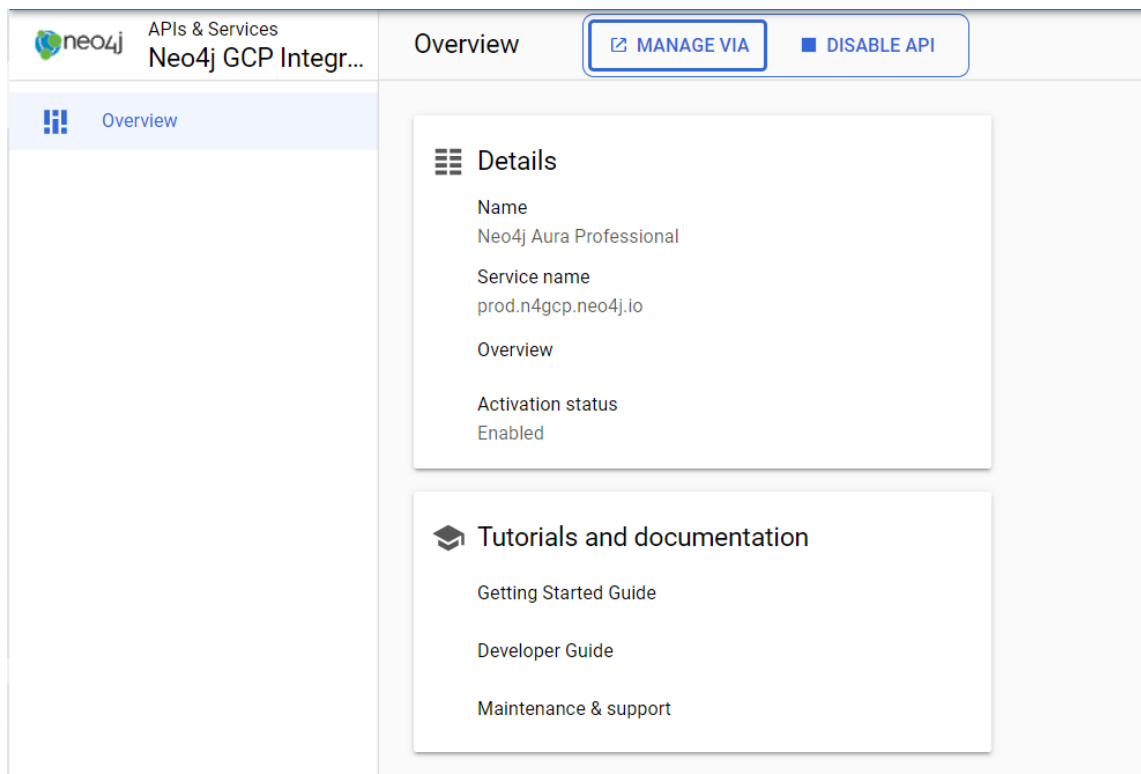Figure 13 and figure 14 shows the installation of Neo4j professional using GCP
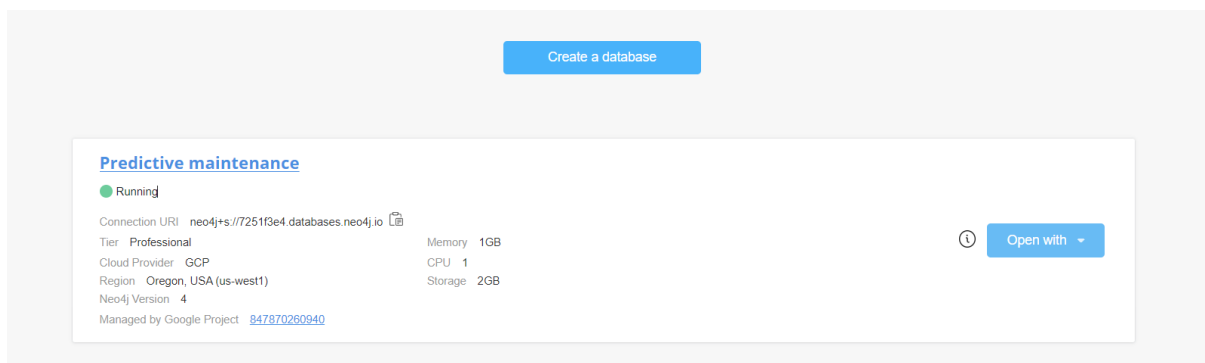
Figure 13

Figure 14
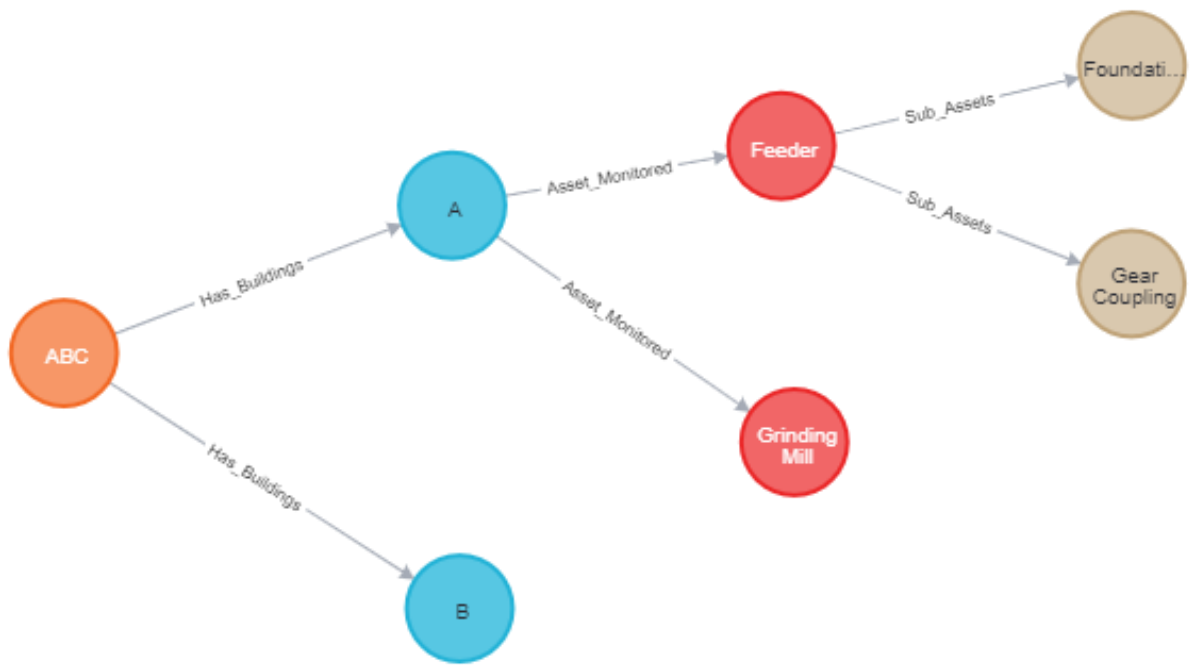
The sample structure of data in Neo4j is shown in figure 15

Figure 15