Contents

1. Introduction 1
2. Technological Challenges 2
3. Technology Analysis 3
    - 3.1 Ingestion of meta-data 3
        - 3.1.1 Intro The Issue 3
        - 3.1.2 Alternatives 4
        - 3.1.3 Chosen Approach 6
        - 3.1.4 Proving Feasibility 6
    - 3.2 Model Training Framework 7
        - 3.2.1 Intro The Issue 7
        - 3.2.2 Alternatives 7
        - 3.2.3 Chosen Approach 9
        - 3.2.4 Proving Feasibility 10
    - 3.3 Type of learning 10
        - 3.3.1 Intro The Issue 10
        - 3.3.2 Alternatives 11
        - 3.3.3 Chosen Approach 14
        - 3.3.4 Proving Feasibility 14
4. Technology Integration 15
5. Conclusion 15

1. Introduction:

We are Smart Cloud Shield, our team members are Cole Neubauer, Zhaolu Yang, and Austin Torrence, and our mentor is NAU graduate student Jun Rao. Our project is sponsored by Daniel Boros, a staff software engineer in the Spectrum Protect Server Development department of IBM.

When data is uploaded to one of Spectrum Protect's storage options, it needs to be categorized into a storage tier. These storage tiers are varying degrees of hot or cold. Hotter storage allows for quick access to data and is generally for data that is frequently used. Colder storage takes longer to access data and is usually only used for rarely accessed data. The problem for our client is the same for all storage services in the market, the miscategorization of data into the wrong tier.

IBM Spectrum Protect focuses on data protection for large amounts of data. They offer data hosting in physical, virtual, software-defined, or cloud environments. Our project will be focusing on all of the storage options.

Currently, backup administrators at IBM manually configure policies to allow for data to demote from hot to cool storage. This system has the following issues:

- Increases overhead costs by having data unnecessarily sit in hot storage before being demoted
- Wastes work hours by forcing backup administrators to configure policies for correction
- Hot storage costs more, so data going into hot storage unnecessarily before being demoted raises operating costs

We have been tasked with designing a product that preemptively categorizes data uploaded to Spectrum Protect into its correct storage tier to avoid unnecessary overhead costs. We envision an end-to-end application which will ingest file metadata and use a machine learning framework to create a model which then can be used to classify data into storage tiers. Our solution will be able to:

- Read extracted features from metadata, potentially petabytes at a time
- Use a machine learning algorithm to classify incoming data
- Gather data as it runs to continually test the accuracy of the model
- Automatically use run-time prediction accuracy results to continually improve learning module

The purpose of this document is to present our expected challenges and our anticipated solutions for these challenges. The first section is the Technological Challenges section, where we will analyze each major technological challenge we expect to encounter and the solutions we plan to implement. This is followed by the Technology Analysis section. In this section, we will look at the previously described problems, examine any solutions to handle the problem, and explain why we chose to go with our chosen solution. The following section, Technology Integration, will bring together all the solutions described in the Technology Analysis section into a coherent overall solution. Finally, we will conclude this document summarizing why we have chosen our proposed solution.

2. Technological Challenges

We have come up with a solution to our problem outlined in the last section and have foreseen some technological challenges regarding the types of technologies we will use and methodologies we will incorporate. We have created a rubric to be able to judge each technology based on the requirements we have for the project to be successful. The challenges we expect to be the most challenging follow:

1) Ingestion of meta-data

Our first problem involves the ingestion of given meta-data. The meta-data has been extracted for us, but we will need to be able to load, pre-process, and process the data. Our solution needs to be scalable so it is a necessity the solution can distribute across nodes.

2) Model Training Framework

The second problem we will face will be in the training of our model. We will need to find a tool to properly apply machine learning methods to the data to be able to produce an accurate model. We can not be sure which methods will produce the best result as we have not seen the data. Because of this, we will need a tool that has the option to explore different machine learning methods to avoid changing frameworks if we find the data's characteristics do not match our expectations. We will also need a tool that is flexible in how it interprets features as a requirement is we need to allow for easy manipulation of which features are used.

3) Type of Learning

We need to decide on a type of learning for this project. As we have not seen the data, we must currently work off of what we know from our client and make some informed assumptions about the data. The learning type will greatly change the outcome of our model and deciding on the best one will be dependent on how the model fits into the learning type and the characteristics of our data. Because we are making assumptions about our data, we do need to plan for the case the data does not match our assumptions. Planning for this potential challenge now will allow us to seamlessly switch learning types without changing our system architecture and not waste valuable development time.

3. Technology Analysis:

3.1 Problem 1: Ingestion of meta-data

3.1.1 Intro to the Issue

The first large problem our team needs to address is the ingestion, pre-processing, and processing of meta-data. We will need to be able to process amounts of data that exceeds the amount of memory at our disposal. To do so, we need a tool that can handle data on disk and in memory. We have defined a set of requirements for a tool to be viable for our project, some being more necessary than others. The tool should preferably be easy to use and accessible in case we need to alter the source code of the product. The tool's environment needs to be easy to set up as our project will be deployed on machines by our sponsor. Its scalability is another important aspect as it needs to be able to handle petabytes of data once we hand over our product to our sponsor. One of the IBM Spectrum Protect's key selling points is the safety of their clients' data, so security should be of the utmost importance. Memory efficiency and long-term support are also necessary. Lastly, all of the tools we may use needs to be either open source or free to use.

3.1.2 Alternatives

There were many possibilities we considered before making a final decision. Apache Spark, Amazon Redshift, Apache Hive, Presto, MemSQL, and Upsolver were all viable options at first glance but, after careful consideration, there was an obvious choice for our project. Some tools may have outscored our choice in specific categories but overall did not satisfy all of our requirements.

Upsolver is a data lake platform that manages, integrates, and prepares streaming data for analysis. This tool has a reputation for being very secure which is a necessity for our sponsor. Its UI is very user-friendly and has extensive directions on setup which are also very easy to follow. It is also fairly easy to integrate with other tools such as Apache Spark or Presto. Its ability to run locally or on the cloud is also a very desirable feature. Speed is not an issue with this tool, but it does not have a reputation for working on big projects. The biggest issue with we found with this tool is that it is not free. Its demo version does not provide all the functionality we need for this project, and its cost breaks our requirements.

MemSQL is a database management system for processing large amounts of data quickly and efficiently. It is one of the most secure software as it even offers protection from “insider threat.” This tool is very easy to use as it has a very user-friendly UI along with command line capabilities. A detailed instruction manual on setup is also provided on their website. MemSQL can handle processing data in memory or on disk, although its speeds for processing data on disk are not as fast other tools we are surveying. One downside to this software is that the server can only run on Linux which lessens the versatility of the end product. It is a paid tool which also breaks one of our requirements.

Apache Hive is a data warehousing infrastructure tool based on Apache Hadoop. This was probably the least promising software as it is only able to be run locally and not on the cloud, it does not provide innate processing so it would bottleneck data queries, and it doesn’t do much other than act as an SQL database. There are just too many issues with it to even consider it at this time.

Presto is an open source distributed SQL query engine. This tool has a reputation for being used with large companies such as Facebook, Airbnb, and DropBox and is open source. This tool is extremely secure with multiple forms of authentication at varying levels and has secure internal communication. One

downside to Presto is that it can only be interacted with on a CLI or through some convoluted Java that does the CLI calls directly. It has an easy setup with no requirement on specific data storage and can pull from different types of data storage on one query. Its speeds consistently exceed those of its competitors such as Apache Hive in benchmarks. The biggest issue is that it is not supported for machine learning as we would like it to. Using this tool for machine learning isn't necessarily impossible but would be difficult as this would be something this tool has never been used for before or at least documented on. Although it has fast speeds and is open source, the machine learning aspect is the main reason as to why this tool was not our chosen solution.

Amazon Redshift has met almost all of our standards, and at first, glance appeared to be a prime choice for our project. It is easy to use with a user-friendly UI, is known to have been used in projects related to large healthcare, retail, and government workloads and is more secure than any of the tools on this list. It is easily scalable for large amounts of data without slowing speeds with increases in nodes. This is capable of processing petabytes of data at extreme speeds but the one major downfall, like most of the tools in this list, is the fact the software is paid and only offers a free two-month trial. This is a big flaw as we are looking primarily for open source or free to use the software.

Finally, our most promising tool we surveyed is Apache Spark. Apache Spark is an analytics engine for large-scale processing. This software met each of our requirements and almost all of our preferred features. This product is a very active open source project meaning this tool will most likely be supported for years to come. It has a few security vulnerabilities, but each is extensively documented on their website and won't be an issue as these attacks require the attacker to either have access to the users account information, be another local user on the machine, or use social engineering to acquire information or trust from the user. This tool is very user-friendly but has an extensive setup process. The instructions are clear, but when following them on a machine we tested it on, there were many errors we had to fix which added extensive time to the setup. This tool has been used on hundreds of large projects and has been or is currently being used by over 1000 companies. Its ability to process petabytes of data is its most desirable feature, and it has been known to use clusters with 8000 nodes. This tool can also be run either locally or on the cloud. It also balances memory and disk usage better than most of the tools we surveyed. Apache Spark also has a built-in machine learning library that will be very useful when used in combination with the chosen learning solutions.

3.1.3 Chosen Approach

Our chosen approach is Apache Spark. Apache Spark met each of our requirements and almost all of our preferred features. As detailed above Apache Spark was the most impressive product we researched. Its superiority to our other options is clearly outlined in the Table 1 below.

	Apache Spark	Amazon Redshift	Apache Hive	Presto	MemSQL	Upsolver
Security	4	5	4	5	5	5
Ease of Use	5	5	3	3	5	5
Ease of Setup	3	5	4	5	5	5
Accessibility to Product	5	1	5	5	1	1
Scalability	5	5	5	5	5	3
Capability	5	5	3	3	4	3
Future Support	5	5	5	5	5	5
Memory Efficiency	5	5	5	4	5	4
Total Ranking	4.75	4.5	4.125	4.375	4.375	3.875

Table 1. Research Rubric for Data Tool

3.1.4 Proving Feasibility

1. **Creating a Demo**
   We will create a demo that can ingest and process data too large to be done in memory. This demo will be around 50 GB and will be data given to us by our sponsor.
2. **Testing Pipeline**
   We will check that the data tool can properly communicate with some machine learning framework.
3. **Check for properly formed output**
   We will then check that the model produced in the end is properly formed.

3.2 Problem 2: Model Training Framework

3.2.1 Intro the issue

Our second problem is the model training. We need to design and train a machine learning model to correctly assign new data into one of the five data storage tiers offered by Spectrum Protect service. We need to determine the most suitable machine learning product for our project. We require the learning product to support many machine learning methods, be scalable, and have long-term support. We are also grading based on the ease of use and ease of setup, although these are not requirements. The tool should also support deep learning as well as traditional machine learning as we may find our data may not match our expectations.

3.2.2 Alternatives

There are many possible options to our problem, and after researching the problem better, we were able to narrow down the scope of those options down to six. Our first option is Tensorflow, a popular machine learning framework developed by Google. Our second option is Microsoft Cognitive Toolkit (CNTK), which is an excellent open-source toolkit for distributed deep learning. Keras is the third option, and it is a high-level API to build and train deep learning models. MXNet will be the fourth option, and it is a good deep learning framework designed for maximizing the efficiency and productivity of the product. The fifth option is Caffe, a deep learning framework made with expression, speed, and modularity. PyTorch will be the last option since it is a good deep learning framework for designing and evaluating deep learning models.

Tensorflow is a popular machine learning framework developed by Google. It is an open source software library that uses dataflow graphs for numerical calculations. The advantages of the framework are its scalability with multi-GPU distributed training, easy to use interface with model visualization in TensorBoard, and versatility with various kinds of third-party libraries supported. Cons of the tool are the cumbersome underlying interface and inflexible high-level APIs. If we want to train the model using our custom functions, it is very inconvenient based on predefined high-level APIs.

Computational Network Toolkit (CNTK) is an open source deep learning framework developed by Microsoft Research. It depicts the neural network as a series of computational steps through a directed graph. CNTK has many neural network components that allow users to easily design new complex layers by

combining these components without writing the underlying code of C++ or CUDA. It is easy to set up with lines of code using pip in Linux system. In addition to these advantages above, it has great scalability because it allows for distributed learning and makes it very easy to set up for parallel training. However, it has massive documentation with different series of tutorials to ease us into the system.

Keras is a high-level API running on the top of Tensorflow to build and train deep learning models for fast prototyping, advanced research, and production. It has several key advantages. It is super easy to set up based on Tensorflow environment since it is running on the top of it. Also, It is easy to make models by connecting configurable building blocks since it provides a simple and user-friendly interface optimized for common use cases, with few restrictions. At last, it is scalable and easy to extend by writing custom building blocks. However, this tool is required to take up too many GPU memory resources and more compile time to train a model.

MXNet is a fast, full-featured, and scalable deep learning framework that supports the most advanced deep learning models. It is originated from the labs of Carnegie Mellon University and the University of Washington, Its most prominent advantage is the combination of symbolic programming and imperative programming. It uses symbolic programming to build various networks with high speed and uses imperative programming to better handle complex environments. However, documents for some APIs are limited and not clear. There appear to be limited plans for future support. Also, it is not easy to set up since it needs to install packages like CUDA and OpenMP, and configure environmental variables.

Caffe is an open-source deep learning framework developed by Yangqing Jia, a doctor at the University of California, Berkeley and it supports many different types of deep learning architectures geared towards image classification and image segmentation. Caffe is of great scalability and is being used in academic research projects, startup prototypes, and even large-scale industrial applications. But it mainly focuses on applications in processing vision, speech, and multimedia. So it might be not a good choice for our project. Additionally, it is not easy to set up, especially in CUDA, CUDNN path setting and opencv version.

PyTorch is an open source machine learning library for Python, based on Torch, used for applications such as natural language processing. It is primarily developed by Facebook 's artificial-intelligence research group. Even though its advantages of quick setup and training with a few lines of code, it is a brand new tool. It does not have a long history of support although there are plans for future support. As one of popular machine learning framework, PyTorch seems like something that would be interesting, but we can't trust and choose it at this stage.

3.2.3 Chosen Approach

We have chosen Tensorflow as our solution because there are many useful and efficient APIs and predefined functions provided for us to easily create and train a machine learning model. As the most popular machine learning framework developed by Google, Tensorflow has the most active development community where programmers globally work together to develop and improve Tensorflow.There are now various kinds of useful and powerful third-party python libraries which help us ingest metadata from our data tool. Additionally, with well-designed and informative documents, users can easily set up a runtime environment and design the neural network model with several lines of code. Another important factor is its great scalability and portability. It is easy to deploy models to PCs, servers, and even mobile devices with any number of CPUs or GPUs, which then we can apply built-in parallel programming methods to speed up the training process. There exists a user-friendly GUI in the form of TensorBoard, a powerful visualization component of Tensorflow, which is useful for visualizing our neural network structure and training process. Tensorboard can provide guidelines for us to help improve and train the model. We came up with a rubric to grade each tool which led Tensorflow to be chosen as our solution; the rubric is represented in table 1, the rating of each value is on a 1-5 scale; 5 being best.

	TensorFlow	CNTK	Keras	MXNet	Caffe	PyTorch
Ease of Use	4	4	5	4	4	5
Ease of Setup	5	5	5	4	3	3
Scalability	5	5	4	5	5	3
Speed	5	5	4	5	4	5
Capability	5	4	4	4	3	3
Future Support	5	5	4	4	4	4
Deep Learning	5	5	5	5	5	5
Total Ranking (Average)	4.8	4.7	4.4	4.4	4	4

Table 2. Research Rubric For machine learning Tool

3.2.4 Proving Feasibility

1. **Testing Pipeline**
   We will check that the machine learning framework can ingest the extracted features from our data tool.
2. **Train the model**
   We will attempt to train a model using said features and check that the output is well-formed.

3.3 Problem 3: Type of Learning

3.3.1 Intro the Issue

A requirement of our project is to use machine learning techniques to produce a machine learning model. To be able to create a proper model we first need to find which type of learning is most appropriate. The type of learning defines the overall process our system will use to create the model. The type of learning will be determined by looking at three main factors, the data we will analyze, the desired behavior of the model, and the complexity of the learning type. However, since we have not had access to the data set we must work off of informed assumptions; because of this, we may find our initial decision to not be the best choice when we begin prototyping.

The goal of our model is accurate classification. We expect our model to be given metadata and then be able to use that to correctly give one of five classifications of data storage. Classification is a classic machine learning problem, and most of the learning lends itself to classification fairly well. Because

of this, our decision will need to be based more on the data and the complexity of the learning type.

We know the data we will work with is metadata extracted from IBM's cloud service users' data. We are making the following assumptions: the metadata is well formed making feature extraction easy and efficient, the data is correctly classified, the data is not balanced, and the data will require preprocessing before learning can begin. We make these assumptions based on what we have been informed from our sponsor but cannot guarantee because we have not been seen the data.

We are considering the complexity of the learning type to be able to avoid unnecessary complexity. For example, if we have two options that fit our model's behavior and the data equally well, we would want to be able to formally discern between them based on the complexity of their processes.

Using what we know about the desired behavior of the model, the expected characteristics of the data, and the complexity of the learning type we can begin comparing our options for learning types.

3.3.2 Alternatives

The four main types of learning are supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning. These types are known informally, and perhaps more recognizable, as machine learning, deep learning, mixed learning, and reinforced learning; respectively. Each type defines its unique process of creating a model and are most useful when applied to certain characteristics of data and model behavior.

Supervised Learning

The most well-known choice is supervised learning. Also known as traditional machine learning, this learning type has the easiest to follow process for creating a model. This process predicates the data must be well-labeled observations to make for proper input and output. The data is then split into two categories: test data and training data. The test data is fed into a machine learner and after sufficient data has been passed through a model is created. This model is then tested for accuracy using the test set. The stats of the model's performance are sent to the learner to improve the teaching of the model. This process is repeated until the model is acceptably accurate.

This process lends itself to classification problems where features used for classification are well defined. Specifically, when there are a finite number of input parameters that determine a single objects classification into a finite number of outputs. As noted above, this process requires the data to be well labeled and well-formed. Without the data structured this way, the process is unable to define traits that can be used to classify the object or have reliable training and test data.

If our assumptions are correct, then there will be extra overhead involved in choosing supervised learning. The creation and designation of training and test data will require preprocessing before we can begin creating a model. However, supervised learning's most common use case is classification. Because of this, we would have more documentation and resources available to us compared to the other options. The process is also much more deterministic than other choices making it easier to recreate and confirm our work as we progress on the system.

Unsupervised Learning

Our second option is unsupervised learning or deep learning. This learning type is most similar in concept to a black box. We know the input going in and know what output to expect, but determining what is going on in any specific case is difficult; if not impossible. The process has no specific requirements of data. This lack of requirements allows deep learning to be applied to unlabeled data where features are not well defined. Examples of this are things that are difficult for a person to define such as a human face.

The general process works by first providing a dataset that has no specific outcome or correct answer accompanying the data. The data is then passed through a neural network, the learning algorithm that will be used to model the input-output relationship. The neural network will then attempt to automatically find structure in the data and extract useful features from the structure it finds. This would allow us to get an accurate model if it is difficult to extract features from our data or too difficult to define useful features.

Following our assumptions, this would not be the best option for us. Because we are making the assumption our data will have easily defined features; we are making an unnecessary tradeoff. We lose the deterministic

behavior in the process, but because the pros for deep learning relate to defining features, we do not appear to gain anything of immediate value.

Semi-supervised Learning

The next option is semi-supervised learning. Also referred to as mixed learning, this process uses some data that is labeled and some that are not for training. This is usually used when data can be well defined but is very time intensive or expensive to define making it so getting enough data is near impossible. This would occur situations where an expert is necessary to label the data such as medical images or thermal anomalies in volcanic flows. The most common implementation of semi-supervised learning involves using general adversarial networks.

General adversarial networks or GANs put two deep learning networks in a system of competition. One network works to create an image that mirrors the training set, and one network attempts to designate given input as either from the original dataset or produced from the generating network. The system repeats this process, improving both networks.

This option will most likely not apply to us. From what we know of our data it has its features already defined in the form of metadata, the questionable part is whether we can define the useful features. If we can, we would more than likely use supervised learning. If we cannot then deep learning would be a more appropriate option than semi-supervised learning. It would add, as can be seen above, unneeded complexity to the process without benefit.

Reinforcement Learning

Reinforcement learning is used to improve a process or technique with an end goal. It is not given data to train or test with but instead uses an AI and is given an environment to experiment in. The AI has a base set of operations which it may use randomly in the environment to start. As it gets closer to an end goal, the AI is incrementally rewarded; leading the AI to an end goal and a larger reward state. Each trial is saved, and the AI tries to recreate trials that were rewarded more optimally. This process repeats until the AI is optimal at working through similar environments or tasks.

This type of learning can be definitively ruled out. Its use cases do not apply to our end model goal and therefore can not be used to satisfy one of our known requirements.

3.3.3 Chosen Approach

The following table (Table 3) outlines the comparison (0 - 5) of the considered approaches and their scoring relating to how well they applied to our end model behavior, how well the type fit our data's expected characteristics, and the complexity of the type's processes.

	Supervised	Unsupervised	Semi-supervised	Reinforcement
Fit desired model behavior	5	5	5	0
Fit expected data characteristics	4	3	2	0
Relative Complexity	5	3	4	4
Total Ranking (Average)	4.6	3.6	3.6	1.3

Table 3: Research Rubric For Learning Type

From Table 3, it appears clear the best option is supervised learning. Supervised, unsupervised and semi-supervised all fit our desired model behavior very well as they all could produce a classification model. However, the supervised expected input most closely resembled the expected characteristics of our data only losing one point because it does create overhead from requiring preprocessing and designation into different data pools for testing and training. It also seemed to be the least complex of our options because the process steps are well defined, deterministic and mostly well understood. We do acknowledge that unsupervised could end up fitting our data more because the actual data characteristics could potentially not match our assumptions. In this case, we do not see it as a potential issue switching because of our options for learning products support supervised and unsupervised learning.

3.3.4 Proving Feasibility

1. **Compare Expected Characteristics of Data**

   We will first compare the actual characteristics of the data with our previous assumptions. If there are major

discrepancies, we will re-evaluate our best option for learning type.

2. **Testing Process on Data**
   We will test that our process works as expected. We will do so by using only the chosen learning framework or library to avoid potential missteps in our pipeline. We will use a smaller group of data and confirm it is going through our process as expected.

3. **Check for Properly formed Output**
   We will check that the output is properly formed. This is because in this early state we do not expect an accurate model, so we will only check that a model was produced.

4 Technology Integration:

The primary challenges with our project are finding the right technology to use. The technology we have proposed in our document will need to come together to create one cohesive system that will be able to create a model to classify data.

First, our application will read in meta-data collected on users data. Using Apache Spark, our application will analyze the data and determine if any preprocessing or balancing needs to occur. We will then process the data as needed to extract the required features.

After extracting our features, we will pass the features from Apache Spark to our machine learning framework, TensorFlow. With the release of Apache Spark 2.3, TensorFlow is inherently supported by Apache Spark and allows for seamless integration through their Python APIs. Here we will use supervised learning to iteratively create an increasingly accurate model. We will continue this process; testing the accuracy of our model. We will continue this process testing and documenting the results with a goal of creating a model with at least 80% accuracy.

5 Conclusion:

In conclusion, data uploaded to IBM's Spectrum Protect product is often miscategorized into the wrong storage tier. This requires manual intervention by system administrators to correct miscategorizations after the files have been in the wrong tier for too long. The general solution to our problem is as follows:

1) Ingest metadata and extract features from the metadata. Then use the features in a machine learning framework to create a model which can be used to classify data correctly.
2) The model will then be exported to be used to correctly classify the data in storage and then classify any new data uploaded.

The purpose of this document has been to identify our potential technical challenges and identify solutions to those challenges. The following table outlines are solutions and how confident we are they will work for our challenges scaled from one to five where five is the best.

Technical Challenge	**Proposed Solution**	**Confidence Level**
Ingestion of Meta-Data	Apache Spark	5
Model Training	Tensorflow	4
Learning Type	Supervised Learning	3

Table 4. The confidence level for solutions

We will have an application that will use machine learning to correctly categorize data into its respective tier. We are confident we will be able to produce a product that will help IBM designate data into correct tiers. This will decrease the number of unnecessary costs accrued by IBM and allow IBM to use fewer work hours fixing miscategorizations of data.

The next major milestones of our work will be initializing our pipeline setup, getting our data, and getting our application to produce a well-formed model. We do anticipate there may be discrepancies with the expectations of our data and our data's actual characteristics. We have researched accordingly and have versatile technologies to allow us to switch our machine learning methodology without changing our application infrastructure if we run into this challenge. We currently have a plan of getting our data from our sponsor, Daniel

Boros, on November 29th, 2018 during a face-to-face meeting. Because we will not have our data until later in this month, we will build up the infrastructure of our application before to allow us to immediately begin refining our prototype when we receive the data. After extensive research, this pipeline of technologies will produce the best solution even with the anticipated upcoming challenges; resulting in a well-designed end product that will be able to predict appropriate storage tiers of data during data ingestion.