Software Requirements Specification

COMP 410/539

February 2016
Contents

1 Abstract 4

2 Problem Statement 4

3 Requirements 4

4 Users 5

4.1 Use-Cases 5

4.1.1 System Administrators 5

4.1.2 Application Users 5

4.1.3 Data Sources 5

4.1.4 Auditors 5

5 Solution 6

5.1 Requirement Solutions 6

5.1.1 Authentication and Permissions 6

5.1.2 Auditability 6

5.1.3 High-volume Data Input 6

5.1.4 Predictions and Real-Time Updates 6

5.1.5 Scheduling 6

5.1.6 Actors 7

5.1.7 Entity Modification 7

5.1.8 Spatio-Temporal Series 7

5.1.9 User Notifications 7

5.2 Architecture 7

5.3 Components 9

5.3.1 Local Data Source Aggregator (LDSA) 9

5.3.2 Data Source Endpoints (DSE) 10

5.3.3 External Data Daemons (EDD) 10

5.3.4 Data Compute Engines (DCE) 10

5.3.5 Database Transaction Layer (DTL) 11

5.3.6 DocumentDB Database (DDB) 11

5.3.7 SQL Database (SDB) 11

5.3.8 External Notification Daemons (END) 12

5.3.9 System Interaction Endpoints (SIE) 12

5.3.10 Authentication Service (AuS) 12

5.3.11 Local User Application (LUA) 13

5.3.12 Internal Logging Framework (ILF) 13

5.3.13 Logging Database (LDB) 13

5.4 Testing and Diagnosability 13

5.5 Development Plan 13

Appendices 16

Appendix A Glossary 16

Appendix B Azure Technologies 18

B.1 Security 18

B.1.1 Definition 18

B.1.2 Concerns 18

B.1.3 Solutions 18

B.2 Sensors / Internet-of-Things Devices 20

B.2.1 Definition 20
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>B.2.2 Concerns</td>
<td>20</td>
</tr>
<tr>
<td>B.2.3 Solutions</td>
<td>20</td>
</tr>
<tr>
<td>B.3 Local User Applications</td>
<td>22</td>
</tr>
<tr>
<td>B.3.1 Definition</td>
<td>22</td>
</tr>
<tr>
<td>B.3.2 Concerns</td>
<td>22</td>
</tr>
<tr>
<td>B.3.3 Solutions</td>
<td>22</td>
</tr>
<tr>
<td>B.4 Data Upload</td>
<td>23</td>
</tr>
<tr>
<td>B.4.1 Definition</td>
<td>23</td>
</tr>
<tr>
<td>B.4.2 Concerns</td>
<td>23</td>
</tr>
<tr>
<td>B.4.3 Solutions</td>
<td>23</td>
</tr>
<tr>
<td>B.5 Data Management</td>
<td>25</td>
</tr>
<tr>
<td>B.5.1 Definition</td>
<td>25</td>
</tr>
<tr>
<td>B.5.2 Concerns</td>
<td>25</td>
</tr>
<tr>
<td>B.5.3 Solutions</td>
<td>25</td>
</tr>
<tr>
<td>B.6 Inter-Component Communication</td>
<td>27</td>
</tr>
<tr>
<td>B.6.1 Definition</td>
<td>27</td>
</tr>
<tr>
<td>B.6.2 Concerns</td>
<td>27</td>
</tr>
<tr>
<td>B.6.3 Solutions</td>
<td>27</td>
</tr>
<tr>
<td>B.7 Networking</td>
<td>29</td>
</tr>
<tr>
<td>B.7.1 Definition</td>
<td>29</td>
</tr>
<tr>
<td>B.7.2 Concerns</td>
<td>29</td>
</tr>
<tr>
<td>B.7.3 Solutions</td>
<td>29</td>
</tr>
<tr>
<td>B.8 Testing and Assessment</td>
<td>31</td>
</tr>
<tr>
<td>B.8.1 Definition</td>
<td>31</td>
</tr>
<tr>
<td>B.8.2 Concerns</td>
<td>31</td>
</tr>
<tr>
<td>B.8.3 Solutions</td>
<td>31</td>
</tr>
</tbody>
</table>
1 Abstract

This document constitutes a description of the requirements of the software system produced by the spring 2016 COMP 410/539 class on behalf of Schlumberger Limited. A glossary of all terms used in this document is provided in Appendix A. An understanding of these terms is assumed throughout this document, and as such, first-time readers of this document are encouraged to view Appendix A before continuing. First, the problem to solve is described in Section 2. Next, the solution requirements for this problem are detailed in Section 3. A description of the users of the system, as well as the use-cases provided for them, is given in Section 4. Finally, the proposed architecture and technologies used to implement the system and a timeline for the development of the system are given in Section 5. Thorough research of the capabilities of Azure products and services and their application to our system is presented in Appendix B.

2 Problem Statement

Schlumberger Limited, the customer, operates in a logistics capacity, and has a limited set of Resources with which to complete many Jobs, organized in Schedules. Currently, there are concerns about the safety and feasibility of proposed Schedules due to problematic weather conditions at Locations where Jobs occur and at Locations en route to a job site. These factors cause inefficiencies in the allocation of Resources as they sit idle and dangerous conditions in which workers or Resources operate.

The customer wants to use past and real-time Data available from sensors and trusted sources to predict future conditions at Locations of interest. These predictions and real-time condition measurements should be used to assist Users in the creation of Schedules in order to optimize the usage of Resources and protect the safety of workers.

3 Requirements

The system shall support these requirements:

- Authentication of Users of the system and the ability provide a set of authorized actions for each.
- Preservation of an auditable record of all data received by, and actions performed in, the various components of the system. This record must be available for at least five years.
- Ability to accept an arbitrarily large volume of input Data from a dynamic, heterogeneous set of Data Sources.
- Ability to make predictions of future conditions based upon Data and provide real-time updates to Users as conditions change.
- Ability to aid in the creation of coherent and efficient Schedules by providing validity checking of Resource allocation and predictions of conditions at specified Locations.
- Interface for the creation and modification of Actors.
- Interface for the creation and modification of entities—including Schedules, Resources, Locations, Alert Criteria, and Jobs—modeled by the system.
- Interface to view analyses of spatio-temporal series of Data stored in the system.
- Notification of Users upon changes of state in the system.

A description of how the system solves these requirements is given in Section 5.1.
4 Users

There are four encompassing types for Users in the system. An individual User may have multiple types.

1. **System Administrators:** These Users are responsible for the administration of other Users and components within the system.

2. **Application Users:** These Users have a set of permissions that allow them to utilize interfaces provided by the system.

3. **Data Sources:** These Users are providers of data to be processed.

4. **Auditors:** These Users have access to the record of events that have occurred in the system.

4.1 Use-Cases

All Users must be authenticated (see Section 5.3.10) before performing any action in the system.

4.1.1 System Administrators

For System Administrators, these groups of use-cases are provided.

- User creation / deletion / modification.
- Permission creation / deletion / modification.
- Organization creation / deletion / modification.
- Actor creation / deletion / modification.
- Data Source creation / deletion / modification.

4.1.2 Application Users

For Application Users, these groups of use-cases are provided.

- Schedule creation / deletion / modification / approval.
- Job creation / deletion / modification.
- Resource creation / deletion / modification.
- Data retrieval from authorized queries.
- Alert Criteria creation / deletion / modification.
- View Data analytics.

4.1.3 Data Sources

For Data Sources, these groups of use-cases are provided.

- Upload Data.

4.1.4 Auditors

For Auditors, these groups of use-cases are provided.

- View system performance.
- View record of logged data received or actions performed in the various components of the system.
5 Solution

The solution is composed of a cloud-based software-as-a-service platform implemented in the .NET framework and deployed to the Microsoft Azure cloud platform. The solutions provided for each of the requirements are detailed in Section 5.1. The architecture of the system is described in Section 5.2. The details of the individual components are described in Section 5.3.

5.1 Requirement Solutions

The system is designed to provide a solution to each of the requirements listed in Section 3. The following sections give a high level overview of how the system accomplishes these requirements.

5.1.1 Authentication and Permissions

The system supports authentication and authorization through the use of Azure Active Directory (see Section B.1). This mechanism is abstracted so that other authentication platforms might be supported in the future by the Authentication Service (AuS) (see Section 5.3.10). Users are split into different roles which can be found in Section 4, along with descriptions of their capabilities. Users may have different sets of Permissions within their domain. Organizations are supported through the creation of groups of Users, which have a set of Permissions, granting access to certain aspects of the system.

5.1.2 Auditability

A record of all changes made to entities within the system is stored in the Logging Database, and can be queried by a User with the Auditor type (see Section 4). These Users will be able to issue queries over the records to recreate the state of the system at various points in time. The Internal Logging Framework (see Section 5.3.12) handles all requests from various system components to log events that occur in the system.

5.1.3 High-volume Data Input

The incoming endpoints that accept Data are provided by the Data Source Endpoints (DSE) (see Section 5.3.2). Requests are distributed to different instances of the DSE by an Azure Load-Balancer (see Appendix B.7) in order to provide scalability. Messages are then sent to the Data Compute Engines (DCE), a scalable Actor-based microservice provider described by Section 5.3.4.

5.1.4 Predictions and Real-Time Updates

Predictive Actors in the Data Compute Engines (DCE) (see Section 5.3.4) make predictions and real-time evaluations based on the current system model of conditions at that Location. These Actors then store their predictions and evaluations in the DocumentDB Database (DDB) to be accessed by other components. The DDB is a high-frequency database which contains incoming and processed data from data sources (see Section 5.3.6 for additional details). As Data enters the system through Actors in the DCE, Users are updated based on the changing conditions that affect their relevant Schedules through the External Notification Daemons (END) (see Section 5.3.8).

5.1.5 Scheduling

Users will have the ability to create Schedules using the System Interaction Endpoints (see Section 5.3.9). When a Schedule is created, an Actor in the Data Compute Engines validates the Schedule and sends notifications regarding the newly created schedule to the User through the External Notification Daemons (see Section 5.3.8).
5.1.6 Actors

A microservice framework is provided through a set of Actors inside of the Data Compute Engines (DCE) and Database Transaction Layer (DTL) (see Section 5.3.4 and Section 5.3.5). Users can request information from these services through the System Interaction Endpoints (SIE). System Administrators can create new Actors in the DCE that can receive messages from the SIE. They also may remove or modify existing Actors.

5.1.7 Entity Modification

An Entity is a modelable object stored in the system. These are stored inside of the SQL Database (SDB), a relational database which can be accessed through the Database Transaction Layer (DTL). The System Interaction Endpoints (SIE), the set of endpoints exposed to users that can query the database, can request data that they are authorized to view from the SDB. New data can be stored into the system by messaging data storage Actors in the Data Compute Engines.

5.1.8 Spatio-Temporal Series

The Database Transaction Layer (DTL) (see Section 5.3.5) is a set of Actors that arbitrate access to the set of available databases in the system. This layer supports authorized queries over the Data, including the capability to query based upon the time, location, and/or Data Source of the Data. Analytics are provided to the User through the streaming of new information via Actors in the Data Compute Engines that push events to Actors in the External Notification Daemons.

5.1.9 User Notifications

Users of the system have the ability to specify Alert Criteria, which will trigger an alert when met. The user is able to specify the notification method such as e-mail or SMS. The External Notification Daemons (END) (see Section 5.3.8) is a set of daemon processes that utilize the specified Alert Criteria to send notifications to Users.

5.2 Architecture

The system’s architecture is structured as shown in Figure 1. This section will provide a brief overview of how data flows through the system, while a more detailed treatment of each of the components mentioned will be given in Section 5.3.

Incoming data is generated by a large, heterogeneous set of data sources such as sensors paired with Internet-of-Things (IoT) devices located on-site. The sensors/IoT devices send this data via locally supported protocols to a set of aggregation applications, the Local Data Source Aggregators (LDSA) (see Section 5.3.1). The incoming data is sent from the LDSA through TCP/HTTP to endpoints provided by the Data Source Endpoints (DSE) (see Section 5.3.2). Instances of LDSA need to authenticate via the Authentication Service (AuS) before they can communicate with the system. Data is also retrieved from external trusted sources through a set of daemon processes, the External Data Daemons (EDD) (see Section 5.3.3). New instances of EDD must be added by the software engineers. From here, data is submitted to a microservice framework that will handle the storage of the Data tagged with its Data Source, the Data Compute Engines (DCE) (see Section 5.3.4).

The DCE is the primary workhorse of the system, and it is composed of a set of Actors. These Actors can receive messages from the DSE and EDD, query other Actors in the DCE, and interact with the Database Transaction Layer (DTL) (see Section 5.3.5). The ability of the DCE Actor to query and store information is granted by the Permissions specified by its creator. Actor instances can also publish events to describe changes in state in the system, such as dangerous weather conditions or invalid schedules. These events are pushed to another microservice framework, the External Notification Daemons (END) (see Section 5.3.8).

The DTL provides access to the three database sets in the system, the DocumentDB Database (DDB), SQL Database (SDB), and the Logging Database (LDB). The DDB is an Azure DocumentDB Database.
Azure Virtual Network
Azure Active Directory
Azure Remote App
Azure Service Fabric
Azure VM
Sensors /
IoT
Data Source
Endpoints
Azure Append-
Only Blob
Dynamic Client Hosted
Web Portal
Users
Logging
Database
System Interaction
Endpoints
External Data
Daemons
DocumentDB
Database
SQL Database
Local Data Source
Aggregator
Local User
Application
External Notification Daemons
System Interaction Endpoints
Azure VM
Azure Service Fabric
Azure Service Fabric
Data Compute Engine
Database Transaction Layer
Logging Database
Azure SQL Database
Azure Document DB
Azure Append-Only Blob
Internal Logging Framework
External Notification Daemons
Azure Remote App
Trusted External Sources
Sensors / IoT Devices
User Mailing Services
Figure 1: High-level System Architecture Diagram
that provides high-frequency access to tagged, variable composition data submitted by the various data sources (see Section 5.3.6). The SDB provides a relational database to model lower frequency but more structured data such as the Schedules, Jobs, and Resources in the system (see Section 5.3.7). The LDB is an Azure Append-Only Blob Storage Database that stores a log of all events in the system, only written to by the Internal Logging Framework (ILF) (see Sections 5.3.12 and 5.3.13).

Users interact with the system through Local User Application (LUA) which query the System Interaction Endpoints (SIE), a set of processes that expose an API to interact with the system using REST API over TCP/HTTP (see Sections 5.3.9 and 5.3.11). Users must be authenticated via the Authentication Service (AuS) before any further interaction of the system takes place (see Section 5.3.10). When a user authenticates within the system, an event notification daemon is created in the External Notification Daemons (END) to forward events that occur during their session to them (see Section 5.3.8). The END hosts a set of processes that listen for events and notify users through various messaging services and through direct TCP/HTTP connections to push instantaneous event alerts. SIE processes can only query the DTL, and not write any new information. Writing of data is only provisioned to the DCE. For low-latency editing of information displayed to the user, optimistic concurrency control is used. As an instance in the DCE finishes the asynchronous request, an event is pushed to a daemon in the END, and finally forwarded to the user’s LUA.

The END, SIE, AuS, DSE, and EDD are the only components that are exposed to the Internet. All other components are hosted within a private virtual network to secure communications and improve network fidelity. The Internal Logging Framework (ILF) is provided as a service for all components in the system to record the actions that they have taken.

5.3 Components

The individual components listed in the description given in Section 5.2 are described in detail in the following sections. The risk-assessment of the development of each component is provided.

5.3.1 Local Data Source Aggregator (LDSA)

The LDSA is a locally running application (see Appendix B.3) that aggregates incoming sensor information from various sensors and Internet-of-Things (IoT) devices over the local network (see Appendix B.2). A diagram of the structure of these applications can be seen in Figure 2. Data is aggregated and stored indefinitely until a request is made to the DSE over TCP/HTTP (see Appendix B.4) to submit information to the system.

Current Risk Assessment The key risks for this component are providing a means for the sensors and IoT to communicate with the application so it can aggregate data and building a robust and secure communication protocol (including authentication) between the application and the DSE.
5.3.2 Data Source Endpoints (DSE)

The DSE provide a means for the system to receive data from external sources to be considered, stored, and evaluated. The data sources communicate with this component using a TCP/HTTP connection (see Appendix B.4). An Azure Load-Balancer (see Appendix B.7) is placed in front of the DSE instances in order to distribute traffic and provide scalability as throughput increases. Connecting data sources must first be authenticated through the AuS, relying on configuration by a System Administrator to authorize their connection and identity (see Section 5.3.10, as well as Appendix B.1). This identification is used to tag data submitted by the data source as it is sent to data storage Actors in the Section 5.3.4. This component is Internet facing. Instances of this component exist as web-roles inside of an Azure Virtual Machine alongside instances of the EDD. Auto-scaling is provided through Azure to scale the number of DSE instances as needs change.

Current Risk Assessment  There are two large unknowns for this component that could prove problematic. The first is the implementation of the data submission protocol, including the verification of the source. The second is scalability concerns and consistent load-balancing of high throughput scenarios.

5.3.3 External Data Daemons (EDD)

The EDD is a set of processes defined by the software engineers of the system that continuously request data from externally hosted services, such as Aeris Weather, AccuWeather, or WeatherBug. Data retrieved is then entered into the system by querying data storage Actors in the DCE. This component is Internet facing. This component implements a common framework to request information from a service on a timer. New instances must be instantiated by software engineers. These instances are hosted inside of an Azure Virtual Machine alongside instances of DSE.

Current Risk Assessment  The current risk for this component is the accountability of the retrieved data if it is responsible for a poor prediction or decision made in the system.

5.3.4 Data Compute Engines (DCE)

The DCE constitutes the primary component of the system, which is a set of Actors that provide services that are available for query over TCP/HTTP. An Azure Load-Balancer is placed to delegate requests to a set of scalable Azure Virtual Machine instances that host the DCE (see Section B.7). DCE Actors can send notifications to Users by querying the microservice framework, the END. New methods can be instantiated through dynamic dispatch, handled by Actors that host a framework for new methods to run. Instances in the DCE can query the Database Transaction Layer (DTL), a special set of Actors in the same Virtual Machine as the DCE that arbitrate access to all databases.

This component is planned to be implemented using the scalable Azure Service Fabric framework (see Appendix B.6).

Special Actors within the DCE are:

- **Prediction Services** These are Actors that provide the capability of predicting the conditions of a Location when queried. The services they provide are utilized by the SIE.

- **Scheduler Services** These are Actors that validate and attempt to provide solutions to unresolved dependencies and concerns within a Schedule.

- **Data Source Data Storage** These are Actors that consume incoming data from Data Sources and store it within the DDB.

Current Risk Assessment  There is a risk associated with the ability to dynamically create and destroy processes in the engine, as well as the fidelity and responsiveness of the message passing system utilized to perform heavy computational tasks on the behalf of users, such as schedule validation and predictions.
5.3.5 Database Transaction Layer (DTL)

The DTL is a set of processes that arbitrate communications with the various databases in the system. The database services contained in the system are the DocumentDB Database (DDB), the SQL Database (SDB), and the Logging Database (LDB). The DTL can be sent messages via TCP/HTTP, allowing other processes to query or write to the database. Only instances of the DCE can write to the database. The DCE, SIE, and END all can perform read operations through the DTL. This layer will also provide an extra level of authentication by requiring credentials from connecting client processes within the system (see Appendix B.1). A set of pooled connections to each of the database services is stored to provided faster latencies for connecting queries.

The query format uses an extensible filter object that can be constructed to express queries over the relational structure of the entities in the system that each of the databases understands. The entities in the system are modeled by the diagram shown in Figure 3. The definitions of each of these entities is provided in Appendix A.

This component is planned to be implemented using the scalable Azure Service Fabric framework (see Appendix B.6). It is hosted alongside the DCE.

Current Risk Assessment  The highest risk associated with this component is the filter construct by which various other components can query data. The construction of this abstraction is required for security and extensibility. The use of TCP/HTTP, a reliable but slower protocol, also raises concerns regarding latency.

5.3.6 DocumentDB Database (DDB)

The DDB is an Azure No-SQL DocumentDB Database (see Appendix B.5) which provides high-throughput, flexible solution to storage of the incoming data from data sources.

Current Risk Assessment  The storage limitations of the DocumentDB are concerning given the volume of theoretical data flowing through the system.

5.3.7 SQL Database (SDB)

The SDB is an Azure SQL Database (see Appendix B.5) which provides a high-volume, relational data storage for information dense entities in the system. The SDB expresses a schema that contains the relations expressed in Figure 3. The contents of this database are composed of the user created and managed entities in the system, such as Schedules, Jobs, and Resources, which all have a highly relational structure.
**Current Risk Assessment**  The highest risk in the design of the SDB is the schema to store all information losslessly from the representation input by the user and utilized by the rest of the system. Speed of transactions is also a concern as usage of the system increases.

### 5.3.8 External Notification Daemons (END)

The END is a set of Actors that handle external notification of Users. These Actors have the primary purpose of forwarding these events to some external component in the system, such as a messaging service like SMS or e-mail or a LUA. When a user authenticates within the system, they can connect to the END with their credentials to create an Actor that pushes user-relevant events from processes they trigger within the system through the SIE. Users can also create alert criteria through the SIE which are instantiated as Actor processes in the END that forward events through supported messaging services.

This component is planned to be implemented using the scalable Azure Service Fabric framework (see Appendix B.6). It is Internet facing.

**Current Risk Assessment**  Implementing the means by which users can receive events during their entire session through their LUA is currently undefined in terms of the exact protocol that would be used to set up the connection.

### 5.3.9 System Interaction Endpoints (SIE)

The SIE provides a set of external API calls through a REST API over TCP/HTTP (see Appendix B.4) hosted in an Azure Virtual Machine along side the instances of the END. This component is Internet facing. Incoming traffic is routed through an Azure Load-Balancer (see Appendix B.7) in order to provide scalability of service as load on the system increases. Unauthorized users are redirected towards the Authentication Service (AuS) to obtain credentials. The SIE uses these credentials to allow users access to actions and data for which they have matching permissions. The SIE is not allowed to write data to the DTL, although it is allowed to read data. In order to make changes to the state of the system, a call must be made to Actors in the DCE. Those Actors may then notify the user’s session through a message sent to the END. Optimistic concurrency control should be utilized by the LUA in order to preserve the illusion of immediate transaction execution.

The set of API calls provided by the SIE are directly related to the use-cases listed in Section 4.1.

**Current Risk Assessment**  The highest risk aspect of this component is the design of an API expressive enough to capture all use-cases. Scalability is also a concern as user demand grows.

### 5.3.10 Authentication Service (AuS)

The AuS provides user authentication and enables permission-based data access across the system. This services is a set of processes that runs on an Azure Virtual Machine. It registers with the Azure Active Directory (AAD) to manage user identities. A web-based API is provided, and other components in the system will be redirected toward the AuS to obtain credentials upon any access request. Using AAD, the AuS will have access to a full suite of identity management capabilities including multi-factor authentication, device registration, privileged account management, role based access control, and security monitoring and alerting. AAD also allows integration with on-premise Windows Server Active Directory (discussed in Appendix B.1) to utilize any existing identity management investments and manage access to the cloud based application. The application will be able to scale-up by spawning up multiple instances and distribute load by batch in a simple round-robin fashion [1].

**Current Risk Assessment**  The basic version is low risk in terms of implementation, however the design decision between using a single Azure AD instance and several instances to allow further separation of control and scalability needs to be addressed at the early stage of development.
5.3.11 Local User Application (LUA)

The LUA is an application that connects Users’ actions to SIE and forwards notifications from END back to the users. It requires users to authenticate via the AuS before allowing access to any part of the system. It is planned to be implemented as a web portal hosted on Azure (discussed in Appendix B.3).

Current Risk Assessment  Because this component is one of the exposed endpoints in our system, security and the requirement of authentication before actions may be taken is a major concern. Another risk area is the responsiveness of the application and its ability to provide the Users with all of the possible system actions that they are allowed to take.

5.3.12 Internal Logging Framework (ILF)

The ILF is a mechanism by which all other components in the system can log events that occur in a unified way. Log events are stored in a central database, the Logging Database (LDB). It handles debug logging (used to help developers who cannot attach a debugger to cloud processes), which can be turned off for release builds, error logging (any exceptions or invalid API calls, for example), tracing (detailed, low-level control flow), and event logging (a higher-level system level control flow). All of these types of logging are planned to be implemented using Azure Diagnostics, which is built on Event Tracing for Windows (see Appendix B.8).

Current Risk Assessment  Event Tracing for Windows integrates well with the .NET framework, and is incredibly well documented and widely used.

5.3.13 Logging Database (LDB)

The LDB is the database where logging information is stored and retrieved by the Internal Logging Framework (ILF). For insertions and updates, the LDB should follow append-only paradigms. It is planned to be implemented using an Azure Append-Only Blob Storage Database (see Appendix B.5).

Current Risk Assessment  The aforementioned database solution is used by the Azure Diagnostics framework. Given its adoption in this well-utilized logging framework, the risk in using this solution for the LDB is low. The main concern involves structuring the logs so that system state may be recreated.

5.4 Testing and Diagnosability

Logging  In our first development period, we will create a web application to view the data logged by the Internal Logging Framework (ILF). It will allow the user to query logs based on when they are made and what component of the system made the log. It will then later be extended to be able to follow a data point through the entire system to make sure that sensor data correctly influences predictions made. It would not perform this verification for every log, as it would be far too costly, but could be used on occasion to ensure that the system is still working.

Spoofing  This web would also provide the ability to spoof a sensor, constantly adjust what weather data it’s sending to the system, and show how it affects the predictions made at the other end of the system. It would then confirm that the proper users are notified if the schedules change.

Load and Performance Testing  These tests are described in the Azure Stress Testing subsection of the Security Appendix (B.1).

5.5 Development Plan

Overview  The semester is broken down into four main development periods, scoped into two week segments, each punctuated by a presentation to Schlumberger. Each of the development periods will result in a concrete deliverable, with a specific purpose.
Limitations  The development plan moves from most to least specific. The first prototype is scoped for our first deliverable date (3/19) with the task detail that is necessary to complete it. The second deliverable's components are predicted with the assumption that the architecture is working as expected. Deliverables three and four are probable developments of the project as we see them now, which may change based on the outcome of the first two development periods.

Schedule and Deliverables

1. Prototype (2/24 - 3/18)
   
   (a) **Purpose** Proof of Architecture
       In this stage, we will build out the minimum working component for each part of our network architecture and a small number of use-cases for the user facing system components.

   (b) **Components**
       
       i. Application User Actions
           Create schedules and validate feasibility
           Create jobs and resources
           Send filters to the database (using known schema)
           Authenticate users before providing access to the system

       ii. Admin User Actions
           Create User (including login permissions)

       iii. Data Source Application
           Add data source

       iv. Data Compute Engine
           Evaluate the feasibility of a schedule
           Setup up processing framework (tentatively actors)
           Determine safety of an entered location using input weather data source

       v. External Notification Daemons
           Push to outside communication (e-mail, slack, or database for GUI retrieval)

       vi. Database
           Semester API accurate database wrapper (common interface for requesting all UML objects)
           Single shared database for all objects
           Logging database is not included

       vii. Azure Communications/Networking
           Common interface to actor implementation, usable by outside components even if implementation changes

       viii. Tools
           Common log interface for all components
           Logging pushes to its own database
           UI for structured querying of logs
           Gated check-in
           Enforced Code Review Process

2. MVP (3/19 - 4/1)
   
   (a) **Purpose** Prove our understanding of an adequate product
       Components at this stage work in a manner as to satisfy the most basic constraints of the project as presented by the customer. The system should be usable for simple scheduling examples. The deliverable’s starting point is the prototype above.

   (b) **Components**
       
       i. Application User Actions
           Users can only view widgets that they have access to
           Ability to add a new processing algorithm to the system (using known schema)
Users can build *widgets* or components of the GUI that repeatedly perform an API action and display the results
Create alert notifications

ii. Admin User Actions
Create Organization
View system statistics (uptime, throughput, permissions)

iii. Data Source Application
Provision a new sensor
Begin streaming data from said sensor

iv. Data Compute Engine
Expand API for computations to allow for more technical analysis (framework for different algorithms, etc.)
Predict using all available system data

v. External Notification Daemons
Push directly to GUI
Trigger workflows with notifications

vi. Database
Separate databases optimized for different tasks (tentatively for structured and unstructured data, respectively)
Authentication on User queries

vii. Azure Communications/Networking
Networking optimizations including, but not limited to, sandboxing, virtual networks between components, and a security focus

viii. Tools
Single-script deployment
Development and Release builds (including branching and deployment)

3. Refinement Step (4/2 - 4/15)

(a) **Purpose** Prioritize feature roadmap, refine product towards customer goals
Following the MVP, all of the major components of the system will be created and (minimally) working. At this stage, we’d like to test it with actual users, gather feedback and begin to refine our solution to further conform to customer goals
Furthermore, at this point, extensive testing of our infrastructure will increase with pace, including validation of *extensibility* and *performance*.


(a) **Purpose** Complete the prioritized features, prepare solution for hand-off, and present
This stage ends with the final client presentation, but is focused on moving our features to a state of completion, as well as ensuring that the product can be transferred to Schlumberger (database migration, permission migration, etc.).
Appendix A  Glossary

AAD Azure Active Directory. 6, 12

Actor A process that has a defined interface for the messages it receives and executes some processing algorithm based on the contents of the message. An Actor can asynchronously return a value to the message source that queried it. Actors are utilized in the Data Compute Engines (DCE), Database Transaction Layer (DTL), and External Notification Daemons (END). 4–7, 10, 12, 16

Alert Criteria A conditional query operating over entities in the system that sends a notification through a supported means to a User. See Section 5.3.8. 4, 5, 7

AuS Authentication Service. 6, 7, 9, 10, 12, 13

DCE Data Compute Engines. 6, 7, 9–12, 16

DDB DocumentDB Database. 6, 7, 10, 11, 16

DSE Data Source Endpoints. 6, 7, 9, 10

DTL Database Transaction Layer. 7, 9–12, 16, 25

Data A piece of information provided by a Data Source. Can be of variable schema, but composed of key-value pairs and is timestamped with its creation datetime. Data is stored within the DocumentDB Database (DDB) (see Section 5.3.6). 4–7, 16

Data Source An representation of an Entity that provides Data to the system. Both Users and Actors can be Data Sources. Sensors / Internet-of-Things (IoT) devices are the primary features represented by Data Sources. 4, 5, 7, 10, 16

EDD External Data Daemons. 7, 9, 10

END External Notification Daemons. 6, 7, 9–13, 16

Entity A physical object or being with a distinct existence. We model these within our system. 7, 16, 17

ILF Internal Logging Framework. 6, 9, 13

IoT Internet-of-Things. 7, 9, 16

Job A representation of a task that needs to be accomplished. A Job may require many Resources in order to accomplish its task. A Job may occur at many Locations. A Job has a status that describes the current state of the Job which may be updated through its execution. Jobs are stored within the SQL Database (SDB) (see Section 5.3.7). 4, 5, 9, 11, 16, 17

LDB Logging Database. 6, 7, 9, 11, 13

LDSA Local Data Source Aggregators. 7, 9

LUA Local User Application. 9, 12, 13

Location A representation of a physical location. Is a specific instantiation of a Resource. 4, 6, 10, 16

Organization An Entity composed of many Users that utilizes the system. 5, 6

Permission A provision that provides a User the capability to perform some action or use some data in the system. This is a mechanism for access control so components can be secured against Users who should not have the ability to interact with said component or data. 5–7
**Resource** A model of some **Entity** in the world that a **Job** is dependent on. **Resources** exist in a heirarchy, and as such a resource might hold reference to many other **Resources**. A **Resource** also holds a status. **Resources** are tagged with a searchable description of their **Entity** type, such as electrician, location, or truck. **Resources** are stored within the SQL Database (**SDB**) (see Section 5.3.7). 4, 5, 9, 11, 16, 17

**SDB** SQL Database. 7, 9, 11, 12, 16, 17

**SIE** System Interaction Endpoints. 6, 7, 9–13

**Schedule** A representation of a set of **Jobs** that need to be executed in some order. A **Schedule** can have many **Jobs**. **Schedules** must be approved by authorized **Users** before they can be executed. The system provided a means for **Schedules** to be verified (see Section 5.3.4). The **Jobs** within a schedule are structure as a directed, acyclic graph (DAG), where edges are dependencies between **Resource** utilizations. **Schedules** are stored within the SQL Database (**SDB**) (see Section 5.3.7). 4–6, 9–11, 17

**User** A representation of an external user of the system. May be a human being, an application, or other process. See Section 4 for a description of the various types of **Users**. 4–7, 9, 10, 12, 13, 16, 17
Appendix B  Azure Technologies

B.1  Security

B.1.1  Definition

Security concerns how to achieve safe and reliable data transmission along the whole system. This includes application user authentication and authorization, external data injection from devices/APIs, internal data transmission between system components, and result data display for user interface.

B.1.2  Concerns

The system should support user authentication and authorization in order to verify who a user is and what a user can do. Also, external data source should be verified via device authentication and/or public APIs verification. Moreover, data transmission among every system components should be authenticated. Finally, the system should ensure that only authenticated users with proper authorization can access and manipulate the segments of data storage that are designated to such users.

B.1.3  Solutions

1. Azure Active Directory

   **Pros** Azure Active Directory (Azure AD) provides a centralized administration mechanism over the whole application that includes many desired capabilities; resources are protected with user identity verification and authorization of data access; it supports multi-factor authentication and third party sign in [2], provides flexibility in organizational model and object management; Azure AD is able to interact with diverse database systems.

   **Cons** Azure Active Directory is difficult to integrate into existing systems. It has little support on Macintosh or Unix, and can only manage Windows clients. Active Directory free and basic service limit users to 10 single sign-on (SSO) applications [3], we will need to start with Premium tier at early stage of development.

   **Justifications** Azure Active Directory provides authentication and authorization to applications and resources, it’s a relatively easy-setup way to manage application resources based on user permissions.

   **Risk Assessment** Azure Active Directory relies on DNS to function, some existing DNS systems may need to be upgraded or replaced to support it [4]. Active Directory Connect synchronizes user passwords by default and the authentication process happens within Azure AD rather than the user’s credentials being validated against the corporate AD [5].

2. Azure Device Registration

   **Pros** Azure Active Directory Device Registration is a built-in service available in Azure Active Directory, it allows user-defined additional access rules based on requirements of security. Devices are registered individually, need both device and password for access.

   **Cons** Device Registration in Active Directory only supports iOS, Android, Windows devices [6]. With all devices registered in one service, it can be difficult to manage them.

   **Justifications** Azure Device Registration provides standard device authentication services with device-based access to application resources.

   **Risk Assessment** The device based conditional access policies require device object write-back to Active Directory from Azure Active Directory. It can take up to 3 hours for device objects to be written-back to Active Directory [7].

3. Pathway and Protocol: TCP + HTTPS + REST

   Upon collecting data from devices and APIs, TCP socket + HTTPS + REST is the recommended protocol and pathway for data transmission between system components.
Pros Azure Active Directory was specifically designed to support web-based services that use RESTful interfaces [8]; Azure Storage Service provides easy access with API(Rest); TCP is a well-established data transfer protocol that guarantees packet deliveries, there’s added security when combined with HTTPS; C# offers high performance socket server libraries.

Cons TCP with HTTPS could potentially increase the size of data packets; TCP without the inclusion of HTTPS can send smaller data packets but may potentially cause issues with formatting and readability.

Justifications TCP is known to be fast, secure, and reliable. It is a well-established data path that can be used both over ip and satellite connections that guarantees packet deliveries or timeout.

Risk Assessment TCP/HTTPS are well-known and widely used, they can be a target for hackers. It also trades reliability for speed, so if speed is the higher priority, alternate solutions such as UDP may be preferred.

4. Azure Storage Service

Top database choices are No-SQL DocumentDB, Azure SQL, and Cassandra (more information on their pros and cons can be found in section B.5). Security concerns for data management mainly fall in: Role-based read/write access to database instances; Data segregation among different organizations; Possible data encryption for client-server data interaction.

Pros Azure Storage Service is easy to use, it has good community support for C#. For database implementation, both role-based read/write access and data segregation among organizations can be handled by user authorization (Azure Active Directory here) and appropriate database wrapper/adapter. Secure Sockets Layer (SSL) can be used integratedly for encrypted data transmission between clients and servers to add on security.

Cons Azure Storage Service relies on Azure specific platform and can be difficult to switch to other cloud storage system; Needs premium storage service to achieve high performances [9]; Can be incredibly expensive for large scale to store data in structured way.

Justifications Azure Storage Service is used for data management. Security features for data management can largely be handled by higher level authentication and authorization, which should be decoupled from the underlying choices in the database layer.

Risk Assessment Backup challenges exist for cloud storage system like Azure Storage Service, along with risks of network failure, memory failure and data loss.

5. Azure Stress Testing

Pros Azure offers built-in stress testing suites for performance benchmark and load balancing.

Cons It requires decent amount of expertise and work to design reliable security tests. How much we need to customize the Azure test components to better reflect our most concerned threats may be challenging. There are professional cyber-security companies whose major business is offering security assessment for other websites, e.g., offensive-security.com.

Justifications Stress tests can be combined with aspects of security by evaluating how the system responds to malicious actions under large pressure. High memory/CPU usage or large number of busy threads can potentially expose security leaks that may not present otherwise. Along with stress tests, other positive/negative tests can be applied to examine the primary security mechanisms of the system. Positive tests cover whether the system secures network connections, encrypts data transmissions, ensures user authorizations, handles system failures, etc., while negative tests check whether the system properly rejects/handles any attempt to break any secured component mentioned above.

Risk Assessment The tests themselves should not be very risky, but lack of comprehensive test coverage can lead to security risks in production.
B.2 Sensors / Internet-of-Things Devices

B.2.1 Definition

There exist many types of sensors and IoT devices that provide weather data that would be useful to the system. The IoT devices can be roughly categorized into microcontrollers, application processors and FPGA devices, each of which differ based on available input peripherals, memory size, processor, reliability, security, and power source. Each sensor can additionally be categorized as a simple or compound sensor according to whether it supplies a single type of data (such as air temperature) or multiple types of weather information. The sensors need to be paired with a board which is programmed to authenticate the sensor and relay the data to the front-end API.

B.2.2 Concerns

Sensors need to be reliable, cost-effective, and accurate in the data they collect.

B.2.3 Solutions

1. Arduino with Dyacon TPH-1 or TPH-2

   **Pros** The Arduino microcontroller is lightweight, accepts a wide variety of inputs, has a collaborative community, and is affordable. Dyacon compound module sensors (TPH-1 / TPH-2) measure temperature, pressure and humidity, and are backed by a 1-year warranty[10].

   **Cons** The max program size is 32KB.

   **Justifications** Arduino’s extensive documentation means developer effort is minimized, and tutorials exist for connecting sensors and writing programs to read and use the data[11]. It is a very cost-effective product and it can be hooked up to a Dyacon TPH-1 or TPH-2 using the Modbus or SDI-12 protocol to communicate.

   **Risk Assessment** Using an Arduino may not be flexible enough if the sensor needs grow, and since the boards are not backed by a warranty, their use in an industrial setting may not be justified. Although we know what communication protocols Dyacon TPH-1 or TPH-2 use, there are no known tutorials on how they will be connected to Arduino.

2. Raspberry Pi with Dyacon TPH-1 or TPH-2

   **Pros** The Raspberry Pi, a single-board computer, has 4 USB-ports and a 100-mbps ethernet port, has extensive documentation, is low cost, and features a processor with sufficient processing power for high-throughput relaying of sensor data.

   **Cons** Raspberry Pi only has a 90-day warranty[12].

   **Justifications** With the Raspberry Pi, compatibility with other Windows programs would be a moot issue since it can run any operating system. The plethora of ports allow for interfacing with multiple other devices, and the extensive documentation would reduce programming difficulty.

   **Risk Assessment** While the Raspberry Pi might be overkill, it is cost-effective and is flexible enough to run large programs. Like the Arduino, however, it may not be reliable enough for commercial use. In order to use Modbus protocol with Raspberry Pi (for TPH-1), one of the recommended solutions is to use a shield that is developed for Arduino and use a Raspberry Pi to Arduino shields connection bridge[13]. We do not yet know whether this bridge makes the shields fully functional.

3. Raspberry Pi with various simple sensors

   **Pros** The Raspberry Pi can collect a number of types of data via analog connections, such as temperature or barometric pressure data from a thermistor. New sensors can easily be installed to collect other types of data. The sensors costs when purchased standalone may be significantly lower than in a combination product such as Dyacon’s.
Cons To convert analog-to-digital, an external analog-to-digital converter (ADC) must be installed, unlike the Arduino which features a built-in 10-bit ADC[14].

Justifications Relying on individual sensors for each specific type of data allows the system to be built for less money—up to a factor of 10. The generous 1 GB of RAM of the Raspberry Pi means that sensor data can be relayed with a high-throughput.

Risk Assessment Each sensor must be evaluated separately for reliability, cost, and accuracy. As the simple sensors are not production-ready, there are certain mechanical steps that must be done to install the sensors such as mounting radiation shields.

4. BeagleBone with various simple sensors

Pros Using the BeagleBone as opposed to Raspberry Pi or Arduino board has the advantage of being able to draw power from micro-USB or a 5VDC connection. For security, it supports additional modules, or capes, to add encryption and authentication options. The plethora of input types accepted and number of input pins means it can easily be connected to various sensors without additional mounts. The BeagleBone includes 6 ADCs corresponding to 6 input ports.

Cons BeagleBone has a fairly big community; however, compared to Raspberry Pi’s it’s small, and has fewer tutorials and sample projects. BeagleBone offers only a 90-day warranty.

Justifications The BeagleBone, while slightly more expensive than Arduino and Raspberry Pi, is a low-cost computer with a range of inputs that can be used to connect various sensors.

Risk Assessment BeagleBone’s longevity in an industrial setting is not clearly defined, and research needs to be done to determine whether it is sufficiently reliable.
B.3 Local User Applications

B.3.1 Definition

Data collected generated by both sensors are collected by / input into local applications before being forwarded to our system in the cloud.

B.3.2 Concerns

These local applications need to have secure login features as well as high reliability.

B.3.3 Solutions

1. Azure Application Hosting (web / mobile)

   **Pros** Offers support for both mobile and web-based applications backed by Microsoft hosting and support. Azure hosting also allows for corporate sign on. Starting at $55 per month, this is a very reasonably priced option. Additional features include offline sync, push notifications, and auto-scaling.[15][16]

   **Cons** It’s SLA is credit based instead of guaranteed, where an SLA of less than 99.95% receives service credits. This solution also requires the most custom coding and developer time.

   **Justifications** The security and flexibility of web hosting make it a top choice for our local application.

   **Risk Assessment** This solution relies on the developers to use the framework correctly to create a secure and efficient application. Microsoft-backed support and hosting once the application has been created provides a solid and reliable foundation for the application.

2. Microsoft RemoteApp

   **Pros** Features include remote login that works with corporate credentials and a 99.9% monthly SLA.[17]

   **Cons** With a price tag starting at $17 per month per user, this option gets expensive extremely quickly. In addition, the requirement of initializing a remote connection to a virtual machine whenever data needs to be sent could cause problems when trying to automatically forward data to the cloud or when trying to view information offline.

   **Justifications** The remote running of the application sequesters the application and its data from attack while ensuring that deployment is consistent across all users.

   **Risk Assessment** Since the application is hosted remotely, there is the risk that it is not flexible if offline functionality becomes important.

3. Microsoft Web Apps

   **Pros** Microsoft promises a monthly SLA of 99.95% with this Azure-hosted solution, with pricing starting at $55 per month. It supports both SSL and TLS Mutual authentication as security options, and can additionally support both auto-scaling and controlled deployment.[18]

   **Cons** Requires an internet connection, limiting offline sync and push notifications without a companion application.

   **Justifications** At a reasonable price and with the support of Microsoft, this is a good option for web-only applications.

   **Risk Assessment** It would most likely require a companion application for situations when persistant online connections are not possible.
B.4 Data Upload

B.4.1 Definition

Data Upload concerns uploading data from local applications to the cloud.

B.4.2 Concerns

Data uploading needs to be secure, reliable, and fast. It should also have a reasonable price.

B.4.3 Solutions

1. **Azure Event Hubs** is a service that processes large amounts of event data from connected devices and applications. [19]

   **Pros** The Event Hubs security model is based on a combination of Shared Access Signature (SAS) tokens and event publishers. Event Hubs can connect disparate data sources while handling the scale of the aggregate stream. Support for Advanced Message Queuing Protocol (AMQP) and HTTP allow many platforms to work with Event Hubs. For BASIC version, ingress events cost $0.028 per million events and throughput unit (1 MB/s ingress, 2MB/s egress) costs $0.015/hr (~$11/mo)[20].

   **Cons** Although the price seems low ($0.028 per million events), at this point we may not fully understand at which speed events will be generated, and what exactly an event is. If we are generating millions of events per second, Azure Event Hubs can be expensive.

   **Justifications** Event Hubs is a well-maintained and reliable service by Microsoft. It has both scalability and flexibility.

   **Risk Assessment** Event Hubs is a complicated system that has way more features than we actually need. Therefore, it may be hard to learn and use.

2. **Azure Service Bus** is a generic, cloud-based messaging system for connecting just about anything[21].

   **Pros** Applications can authenticate to Azure Service Bus using either Shared Access Signature (SAS) authentication, or through Azure Active Directory Access Control (also known as Access Control Service or ACS). Azure Service Bus can run anywhere, and connect nearly anything. It builds robust cloud solutions that scale to meet demand. It connects on-premises applications to the cloud. Queues offer simple first in, first out guaranteed message delivery and support a range of standard protocols (REST, AMQP, WS*) and APIs. For BASIC version, operations cost $0.05 per million operations. For STANDARD version, the base charge is $10/mo[22].

   **Cons** Similar to Event hubs, Azure Service Bus may be expensive. Although the price seems low ($0.05 per million operations), at this point we may not fully understand at which speed do we need to ”operate”, and what exactly an operation is. If we are operating millions of times per second, Azure Service Bus can be expensive.

   **Justifications** Like Event Hubs, Azure Service Bus is also a well-maintained and reliable service by Microsoft. It has both scalability and flexibility.

   **Risk Assessment** Like Event Hubs, Azure Service Bus is a complicated system that has way more features than we actually need. Therefore, it may be hard to learn and use.

3. **AzCopy** is a popular command-line utility designed for high-performance uploading, downloading, and copying data to and from Microsoft Azure Blob Storage.

   **Pros** AzCopy is a free tool with which the user can migrate data from the file system to Azure Storage, or vice versa, using simple commands and with optimal performance.[23]

   **Cons** AzCopy is not as popular as Event Hubs and Service Bus. As a result, AzCopy may be insecure in an unknown way: its vulnerability may not have been discovered and fixed yet.
**Justifications** It is a simple tool to transfer data. Since it is not as big and fancy as Event Hubs and Service Bus, it is probably easy to learn and use.

**Risk Assessment** As mentioned in cons, there may be some security attacks against AzCopy.

4. **Azure Import/Export Service** Sending hard drives to an Azure data center.

**Pros** You can use the Microsoft Azure Import/Export service to transfer large amounts of file data to Azure Blob storage in situations where uploading over the network is prohibitively expensive or not feasible.[24]. Let $n$ denote the amount of data to transfer. While normal service takes $O(n)$ time, Azure Import/Export Service takes $O(1)$ time. The price is also reasonable: $80$ for device handling [25]

**Cons** The user needs to physically send hard drives to the data center. Therefore, this service is not appropriate for transferring real-time data.

**Justifications** The user encrypts data before sending the drive. Microsoft also encrypts data before shipping the drive back.

**Risk Assessment** The physical shipment may not be as reliable as we want. For example, it is not uncommon that a package can be few days late.
B.5 Data Management

B.5.1 Definition

Data Management concerns the short and long term storage of data in the cloud. Specifically this concerns the underlying databases within the DTL.

B.5.2 Concerns

The system needs to store large volumes of low and high frequency data. Furthermore, any storage solutions must scale effectively in storage volume, throughput, and cost. Additionally the storage format needs to be flexible so that new data may be added from heterogenous sources. The customer also has strong security concerns which encompass both generally ensuring that only appropriate users have rights to read and access given data and more specifically that data from different companies will need to be segregated. Another customer concern is that raw data obtained from sensors should be stored for at least five years. To accomplish this we must scale to store potentially nontrivially large volumes of data.

B.5.3 Solutions

1. Azure SQL

Pros

An Azure Platform as a Service (PaaS) solution which provides functionality very similar to SQL Server, including support for Transact-SQL. Azure SQL can scale cost-effectively for increased storage requirements[26]. SQL databases also guarantee transactions with ACID consistency. It will also integrate easily with other Azure services, such as Azure Active Directory[27], Machine Learning[28], Stream Analytics[29].

Cons

Obtaining high throughput for large volumes of high frequency data may be difficult and costly[26]. Furthermore, some types of data such as inventory data and sensor data may be highly heterogenous. For such data, a more flexible storage structure than the traditional relational database schema may be more intuitive and useful.

Risk Assessment

Because it is a PaaS solution, there is increased infrastructure reliability and easier deployment. Additionally, SQL is common and well-known which reduces implementation risk. However a significant source of risk would be a failure to cost-effectively scale for high data throughput. This risk may be alleviated by segregating higher frequency data into a separate storage option.

2. No-SQL DocumentDB

Pros

DocumentDB is more flexible and extensible than a relational database such as SQL. There is no requirement to define a schema, so JSON data of any format can be easily inserted into the database without any downtime. In addition, DocumentDB supports SQL queries, which is a very common querying language for databases that many people have experience with. In addition, Device Sensor Data and Cataloging Data are two use cases given for DocumentDB.[30] In choosing this over Cassandra, the biggest factor is that it is a fully-managed Azure service. There is no need for virtual machines or deploying and configuring software. It will also integrate easily with other Azure services, such as Machine Learning or Stream Analytics.

Cons

The first con is that given that a collection is 10 GB, the current projection of the amount of data being received would result in these being filled up very quickly. While it does support SQL queries, it does not support complex queries or the ability to query multiple collections at one time. [31] Finally, since there is no schema it cannot guarantee data consistency.

Risk Assessment

The largest risk with DocumentDB is the amount of data we will potentially be ingesting. With 500,000 wells and about 100 sensors per well, we would be receiving 79 petabytes a year assuming 10 bytes per message. Therefore, a collection would be filled up very quickly. [32] Ideally, we would be able to aggregate some of this data as it would be hard to store in any system in its raw state. In addition, there is no guarantee of data consistency that you would get with a SQL database.
3. **Cassandra**

Cassandra is a horizontally scalable NoSQL solution that is designed for huge throughput while maintaining data integrity. It has a masterless node setup, rather than master-slave. It also has a SQL like query language.

**Pros**

- Cassandra can increase its throughput linearly by adding more nodes into the system, essentially guaranteeing as high throughput as needed. By using a wide column store, updates in the schedule won't lock the entire row for an update but just a single column which is must faster. It is also extremely fault tolerant, as when a node goes down the data is repartitioned such that there are always however many specified copies of the data are needed. As a speed tradeoff, the data can be eventually consistent, or can be immediately consistent as long as writes and reads are at the quorum \((n/2 +1)\) level. There is also support for a caching layer. Querying the database can be done either through drivers for .NET, or using CQL via command line. By using masterless architecture, there is a reduced cost for infrastructure since every node can be read or written to, instead of the typical master-slave architecture. Security is handled in the form of 3 things: authentication, object permissions, and data encryption.

**Cons**

- Because caching isn’t direct access to the data, but rather a cache to the data location on disk, it may be of more use to actually utilize a different caching application still. Another issue is that Cassandra isn’t natively hosted on Azure, so additional infrastructure would be needed. This entails having virtual machines that are dedicated to having cassandra nodes. For the same reason, security protocols would have to be externally done, although there are security protocols in place.

**Justifications**

As far as NoSQL solutions goes this a great choice for handling the high throughput while maintaining data integrity and ignoring any in-memory solutions. The choice for a NoSQL solution is that a pure relational database is just too slow for any consistently high throughput DB, and also would not scale well. As far as additional infrastructure costs go, it is better than most NoSQL DB’s as it is based off of Google’s HBase paper which utilizes a masterless architecture, reducing the number of nodes needed by half, saving a lot of money. Enterprise support would be required to be able to get the help needed with deployment.

**Risk Assessment**

Because it is external to Microsoft Azure’s platform, there would need to be people involved with integrating it into the platform and handling the security as well as communication between the two services. This increases deployment difficulty, as there are many people unfamiliar with Cassandra’s environment. There would have protocols in place as well to move archived data into external storage since the size of the data for 5 years is too large, regardless of how many nodes the architecture chooses to go through with.
B.6 Inter-Component Communication

B.6.1 Definition

Inter-Component Communication concerns how different components within the system effectively communicate with each other and transfer data to each other.

B.6.2 Concerns

The highest concern is that data movement between components must be done efficiently and securely for data entering the system at different frequencies. Data flows from the Data Source Endpoints (DSEs) at a high frequency (1 data packet per second per location), so the entire flow of that data must also be high-frequency. Notifications sent from the External Notification Daemons (ENDs) are sent at a much smaller frequency. Additionally, all data must be pushed to and received by the appropriate user with minimal or no data loss. Although DSEs can handle a much lower frequency of data, they require much more reliability than ENDs. A lost data packet from a DSE can be replaced by a new one a second later, but a lost notification would be a serious problem. (Notifications might be sent multiple times within the system to ensure that a notification reaches the end user.)

B.6.3 Solutions

1. **Redis Cache** is an on-disk, distributed cache that allows clients to publish to channels on the cache and for subscribers to be pushed messages from those channels. Channels are essentially regular expressions.

   **Pros** Redis Cache offers a high throughput of up to 250,000 messages per second. Pricing is based on storage rather than channel, so an arbitrary number of channels can be created. Redis cache is hosted and monitored by Microsoft. Redis cache clients are available in many languages including .NET/C#.

   **Cons** Weak but reasonable forms of data safety and availability; writes can be lost within small windows of time. Redis cache is more expensive than Service Bus Topics for a small number of topics.

   **Justifications** For data coming from high frequency devices, we need a service with high throughput. Service Bus does not scale to the frequencies we are dealing with. We will also likely want a large number of topics which Service Bus does not support.

   **Risk Assessment** Redis Cache clients are open source and may not be well-maintained or documented. We have no experience with Redis Cache and don’t know how well it will actually work for what we’re trying to do. Not having a guarantee against data loss is a problem.

2. **Service Bus Topics** is a queue that is divided into a number of topics users may publish to or subscribe to. For subscribers, additional filters can be added to subscriptions beyond topics.

   **Pros** Service Bus Topics is low cost and familiar.

   **Cons** Service Bus Topics does not scale to a high frequency of data (serves <2,000 requests per second) or a large number of topics. Filtering could allow a reduction in the number of topics, but at the cost of throughput.

   **Justifications** Service Bus Topics are an effective solution to situations where high frequency of data and large number of topics are not required.

   **Risk Assessment** We have experimented with Service Bus Topics in the warmup project, but have not stressed tested it.

3. **Azure Service Fabric and Reliable Actors** Service Fabric manages services by solving problems such as failures, upgrades, utilizing resources efficiently. It offers full application lifecycle management through development, deployment, and runtime. Reliable Actors is an API provided by Service Fabric that allows you to package actors which use the Actor Model in services that can be deployed by Service Fabric.
Pros Service Fabric is reliable and self-healing; it recovers from failures and manages service state so that it is not lost. Many services run inside a container and many containers run on a single machine, allowing hundreds of thousands of instances of a service to be running on a single machine. A resource balancer distributes services evenly across a cluster. Each service scales independently. Each service can be deployed independently. There is not incremental charge for Service Fabric itself, you pay for the compute instances you use and how much you use them. Using Service Fabric would allow us to turn other components of our system into services and have a system of microservices.

Cons Harder to set up than Service Bus Topics.

Justifications Service Fabric is a good solution to help build a service from microservices. Each of the microservices is run efficiently and reliably, allowing the entire service to scale. Using an actor pattern for publish subscribe communication between components should offer higher throughput than Redis Cache or Service Bus Topics and lower cost.

Risk Assessment We don’t know how to use Service Fabric or Reliable Actors. We need to experiment with the throughput of Reliable Actors.

4. Azure Stream Analytics receives streamed data as input from one Azure component, such as an Event Hub, performs operations on the data defined by an SQL-like language, and outputs the operated data to another Azure component like a storage component or another Event Hub. Azure Stream Analytics is designed for processing data arriving at high frequencies to IoT applications.

Pros Can process 1GB of data or millions of messages per second. Input and output components are ones we will likely use (Event Hubs, DocumentDB, SQL). Operations are easy to define. Input can also come from lower frequency or historical sources through Azure Blobs. Pricing by the amount of data processed and compute time used. Stream Analytics is designed to process sensor data, which is exactly our use case.

Cons Data must be input and output in specific formats (JSON, CSV, UTF-8 encoding) and custom connectors to unsupported Azure components cannot be written.

Justifications Azure Stream Analytics is a good way to receive very high frequency data, perform simple operations on it, and quickly store it. It ensures that all the data coming in makes it to the right place.

Risk Assessment We need to look into the security of this option. We also assess what the actual data demand on our system will be to know if we need Stream Analytics. If we think we do, we need to try it out because we have no experience with it.

5. Remote Procedure Calls (RPCs) allow one component to call a defined function on another component and receive that function’s result.

Pros Flexible, efficient for retrieving data directly from a component and using it in the calling thread

Cons Blocks the calling thread, so it is a poor choice in cases where no useful result is returned

Justifications RPC calls will be most useful in the system for connecting components to the Database Transaction Layer (DBTL), as all transactions initiated by a component will have a necessary result to process, either a read result or a write confirmation.

Risk Assessment RPCs are well-supported on Azure, but also more complicated to set up than other protocols [33], and also don’t have an intermediate component with a convenient debugging interface as others (such as the SBQ) do. Setting up RPCs and ensuring that they work should be a priority anywhere they are used.
B.7 Networking

B.7.1 Definition

Networking concerns about services running at the network application layer.

B.7.2 Concerns

Network services should provide a secure, fast, and reliable way for other services to connect to each other. It also needs to load balance traffic from both the open internet and internal data transmission.

B.7.3 Solutions

1. **Virtual Network**
   Azure Virtual Network (VNet) is a logical isolation of the Azure cloud dedicated to your subscription. It enables users to fully control the IP address blocks, DNS settings, security policies, and route tables within this network.

   **Pros**
   - VNet provides enhanced security and isolation because only virtual machines and services that are part of the same network can access each other.
   - It provides extended Trust and security boundary by the trust boundary from a single service to the virtual network boundary.
   - Third, VNet supports a hybrid cloud solution (i.e., use both Paas and Iaas) such that Paas and Iaas Instances in different cloud services are automatically connected with each other within the VNet.
   - The communication between these different services do not need to go through public Internet.

   **Cons**
   - Have to use a VPN Gateway or ExpressRoute to connect securely to the Virtual Network, which will add up the cost and complexity of IT configurations.

   **Justifications**
   Aside from the extra security provided by VNet, we should seriously consider using VNet because adding existing services to a virtual network postcreation is difficult and very time consuming. Also, if later we decide to adopt a hybrid cloud solution that use both Paas and Iaas or make use of multiple cloud services, they can all be added to the same VNet so that the communication between different services do not need to go through public Internet. Last, the VNet can provide some isolation for data from different companies.

   **Risk Assessment**
   The IT configuration for VPN Gateway and VPN devices and network security settings can be tricky.

2. **ExpressRoute**
   ExpressRoute is an Azure service that lets you create private connections between Microsoft datacenters and infrastructure that’s on your premises or in a colocation facility.

   **Pros**
   - ExpressRoute connections do not go over the public Internet, and offer higher security, reliability and speeds with lower latencies than typical connections over the Internet.

   **Cons**
   - ExpressRoute can be costly. Also, a single virtual network can link with up to 4 ExpressRoute circuits.

   **Justifications**
   The usage of ExpressRoute is not necessary. If the customer can alway add them later without affecting other parts of system if they decide that the current speed or security level are not enough.

3. **VPN Gateway**
   VPN Gateways are used to send network traffic between virtual networks and on-premises locations. They are also used to send traffic between multiple virtual networks within Azure.

4. **Traffic Manager**

   **Pros**
   - Traffic Manager provides three traffic routing profile: failover, round robin, and performance.[34]
   - It also provides automatic failover capabilities when an endpoint goes down. The endpoint could be an Azure cloud service, Azure website, or other location.

   **Cons**
   - Do not support sticky routing. Therefore a user might get a new host after his/her TTL cache expires.
Justifications The service can route users to the closest end-point in terms of latency. This would provide help provide the best possible user experience should the product be deployed globally. Additionally, Traffic manager would help allow for continuous uptime while upgrading endpoints as all traffic is redirected to other endpoints. Lastly, it is possible to nest Traffic Manager profiles to optimize performance and distribution for larger, more complex deployments.

Risk Assessment The throughput of this service is not listed in its documentation. The service also adds an extra layer of redirection.

5. Load Balancer

Pros The software load balancer provides both internet facing load balancing and internal load balancing by a hash-based distribution algorithm. It can automatically reconfigure itself when we add or remove instances of our services/VMs. It also provides service monitoring for its endpoints. It supports multiple load-balanced IP addresses for a set of VMs.[35]

Cons Load balancer will not notify you if it found a failed node. Need additional health probe on each node.

Justifications The service is reliable. The internal load balancing is also free of charge.

Risk Assessment This is the only option Azure provides for load balancing among VMs.


Pros High throughput, each DNS query is answered by the closest available DNS server.[36]

Cons Cannot purchase domain names on Azure.

Justifications Our services needs a web front-end and we need to use this service to delegate domains we purchased elsewhere.

Risk Assessment Almost none.

7. Application Gateway Application Gateway provides application-level routing and HTTP load balancing for web front end. It has cookie-based session affinity, SSL offload, and URL based content routing.

Pros The service provides health monitoring. The service is very scalable because you can create up to 50 application gateways per subscription, and each application gateway can have up to 10 instances each. It can also be configured to terminate the Secure Sockets Layer (SSL) session at the gateway to avoid costly SSL decryption tasks to happen at the web farm.[37]

Cons Cannot weight servers in the backend pool. So the traffic always splits evenly (bad for A-B test).

Justifications Since works, administrators, and auditors will interact with our web front end very often, this service is necessary to provide a consistent service.

Risk Assessment For higher throughput option, the price is about $238/month.[38]
B.8 Testing and Assessment

B.8.1 Definition

Testing and Assessment entails having a good code coverage of the entire system (at least 80%) and having thorough unit tests, stress tests, and logging.

B.8.2 Concerns

Throughout the entire system, there needs to be extensive testing to ensure that everything is working properly, and logging for both auditability and testing.

B.8.3 Solutions

1. Solution MSUnit testing

   Pros It has extensive support for load tests and integration tests. It can also be used with gated check-in, by installing Visual Studio on the build server.
   Cons It appears to have not been updated since 2005. It is also somewhat slow.
   Justifications It should be used for all the tests aside from unit tests, which NUnit can handle.
   Risk Assessment It is incorporated into Visual Studio, so one would not expect it to have many problems.

2. Solution NUnit testing

   Pros Appears to be an industry standard for unit testing. It also has plenty of documentation online. The build server can be configured to run NUnit tests for gated check-in.
   Cons Also has not been updated in years. It also requires other tools to get code coverage analysis.
   Justifications Being the industry standard, it makes sense for us to use it for Unit tests.
   Risk Assessment It does incorporate more outside code than MSUnit does, which would be more likely to break with updates to Visual Studio.

3. Solution Azure Diagnostics

   Pros Built on and integrates well with Event Tracing for Windows, which is an industry standard that is very well documented and widely used. Logs events to a file in real time. This could be used in the ILF. Also buffers data locally and sends to cloud storage in batches, decreasing the cost of transactions.
   Cons Documentation is auto-generated (unhelpful) or nonexistent, and help articles are often outdated or misleading. Azure Diagnostics changed a lot between versions 1.0 and 1.3 (the latest version for which documentation is available) and the current version is 2.8.
   Justifications Azure Diagnostics is built into Azure and is very easy to integrate with other testing frameworks, such as Apache’s log4net or Microsoft’s Enterprise Library.[39] You can use Azure’s DiagnosticMonitorTraceListener to listen for traces generated by any framework. So beginning with this frameworking and transitioning to another one will be manageable.
   Risk Assessment Though the documentation is not ideal, using an older version that has better documentation should be safe, and Event Tracing for Windows is a very reliable fallback. And integrating other frameworks or switching frameworks entirely is easy and thus mitigates risk.

4. Solution Enterprise Library

   Pros Assists with many development cross-cutting concerns (logging, validation, data access, exception handling, and more). [40]
   Cons Last update was in 2013, and Azure changes often and rapidly.
**Justifications** Many developers who have come before us have already solved these cross-cutting problems that affect similar large-scale projects. Following their examples and integrating their logging code into our logging framework could be incredibly useful and save us time.

**Risk Assessment** Enterprise Library is just a collection of code templates provided by Microsoft. The snippets are trusted by many developers. However, there is a risk that they are outdated and could break when we attempt to use them.

5. **Solution** Application Insights

**Pros** Makes viewing and analyzing logging data incredibly easy. Can collect and analyze data from many disparate azure services.[41]

**Cons** It’s not mandatory for logging, so we’re essentially adding overhead and developer time to get better visualization and analysis. But might save us time and prevent us from making a GUI to view logging data.

**Justifications** We want to store data for at least 5 years, and this software makes it easy to send the logging data to a separate table, or even account. It also makes it easy for developers to visualize the potentially overwhelming amounts of logging data, minimizing developer time in the long run.

**Risk Assessment** It’s still in preview, so it could change at any moment. But since it’s not a necessary tool, if it changes and we can’t use it anymore we can still generate and save logging data using Azure Diagnostics.
References


