GFR20 Data Management Report

Database Development

Dylan Albertazzi - Author

Bradley Anderson - Project Advisor
Kyle O'Brien - Project Advisor
Dr. Paasch - Faculty Advisor

Figure I: 2019 Vehicle [Sub-system Name]

Table of Contents

0. Abstract/Design Specification

Write this section after you have completed your project. Give a 1 page concise summary of the following: project description, project goals, major changes/highlights, rationale for changes, major issues encounters, performance, and outlook for next year/what to build upon. Use the checklist below to make sure you are meeting all of the requirements for the Abstract.

Abstract shall include a one to a few sentences about each of:

- ❑ **Team goal**
- ❑ **Team design philosophy**
- ❑ **Sub-team goal**
- ❑ **Sub-system goals**
- ❑ **System design features**
- ❑ **Most important results / systems performance (Compared to baseline)**
- ❑ **Test results and agreement to calculations**

Figure 1: Subsystem CAD Image w/ Parts Labeled

Tables can be made in Google Sheets and imported into the document so long as they are appropriately formatted.

The GFR Data Management team exists to provide data to GFR members that they can leverage to make informed decisions to build a better race car. The data management team is one year old this coming Winter of 2020 and is making significant strides towards a web application that provides data access to all members of GFR, no matter where they are in the world.

Our team's design philosophy is heavily focused on the long term success of this project and the GFR team as a whole. Instead of focusing on what will work fastest, we recognized that our greatest contribution to GFR is what we can contribute to future teams. With that in mind we focused on solutions that will not need to be rebuilt as the team and database grows, are cost effective, and can be built on top of.

Our sub-team goal in the beginning was to get all of the unorganized vehicle log files the team had, put them in a database, and build a UI to access, upload, and download data from. We got the old log files into a database running on the OSU stig server. However, over the course of the project we realized that in order to build a UI we needed to consider the computing architecture the UI is built on. The UI part of the project shifted from building the UI to designing a computing architecture that can support the teams long term needs. These needs include access to the web app anywhere in the world, scalability, and the ability to be built on.

The vehicle log database we populated is currently hosted on the OSU stig servers and running on PostgreSQL. It has been populated with the available vector format vehicle log files from past runs. Each entry is a run from one of GFR's vehicles and holds all the data that the on board computer was able to log. Each log is separated into metadata, and channel data. Metadata consists of information about the car letter, car year, driver, track, etc. Channel data consists of numerical sensor readings, acceleration, velocity, etc.

We achieved GFR's computing architecture needs by designing a hybrid serverless microservice architecture. Hybrid means the front end and computing are done in AWS and the database is hosted on the OSU stig servers. The justification for this option is discussed in detail later in the paper.

Copy and paste abstract from your report to your Google sites project page and link the report.

0.1 Specification Sheet (NOT APPLICABLE)

Detailed specification sheet of final system (Abstract and specification sheet should fit on same page)

Specific requirements for this subsection:

- ❑ **Table spec. Sheet of system**
  - ❑ **All major performance parameters**
  - ❑ **Details on major design features**

0.2 Weight Specification (NOT APPLICABLE)

Detailed weight Specification of component and system totals (Start with a BOM list)

Specific requirements for this subsection:

- ❑ **Table of weights and size of components**
  - ❑ **Weight of every component in your system**
  - ❑ **Weight savings of each component + or -**

0.2 Manufacturing and Vendors

Table summarizing manufacturing technique used on components and vendor used
(Start with a BOM list)

Service	**Vendor**
Database	OSU Stig Server
Lambda	AWS
S3	AWS
API Gateway	AWS
Customer Gateway	AWS

Specific requirements for this subsection:

- ❑ **Table of part manufacturing and vendors**
  - ❑ **Part number and description**
  - ❑ **Materials**
  - ❑ **Vendors**
  - ❑ **Time to produce**

0.2 Summary of Results

High level overview of results

Specific requirements for this subsection:

- ❏ **High level overview of design features**
- ❏ **% Increase or decrease in performance vs Baseline**
- ❏ **High level overview of simulation or calculated performance**
- ❏ **Overview of tests completed and results versus simulation**

This project contributed to the team in three main ways. First, by writing the code that takes vehicle log files and inputs them into the database. Second, filling the database with the log files the team currently has. Lastly, we designed the optimal cloud architecture for the GFR team. Like many projects, in the beginning there are factors that get overlooked, as was the case with my capstone. In the beginning the goal was to fill the database and make a UI, however we totally neglected the infrastructure that the database and UI would sit on. So instead of focusing on the UI fall term, the focus was on designing an architecture to support the UI. We successfully designed architecture so now the foundation has been laid for the next group to build out the UI with confidence knowing it can handle heavy workloads and be accessible across the world, including on the race track. The foundation has also been laid to continue leveraging AWS services to improve the car. The AWS organization has been set up and configured, and because of its serverless microservice architecture the web app is easily extensible to any computing use case GFR has.

1. Project Description

1.1 Introduction

Each year GFR collects a large number of data points on its vehicles. Until recently the data has been kept in its original files and not put into a database. This makes looking for relevant data like finding a needle in a haystack.

The purpose of this capstone project is to make the log data accessible to everyone in GFR regardless of their background. This will be accomplished by creating a user interface (UI) with the vehicle log database that was created in the winter term of 2020. The three main functions of the UI are log file uploading, data filtering, and data downloading. The new UI will allow OSU's GFR team to find the information they need to make design choices that will contribute to a winning car.

1.2 Requirements

The scope of this project is unique in that it is not directly related to the car. Therefore there are no SAE rules outlining the requirements of a database UI that provides data for design choices. For that matter, this section will outline internally produced requirements that will be essential in a successful UI design.

The project description outlines three deliverables. The UI accepts log files, data is accessible and easy to find, and selected data can be downloaded. This project will require strong communication between sub-team members which includes attendance at team meetings and prompt responses to messages. Collaboration with the team at large will provide important insights into how the data is used. This project will be done in close collaboration with the winter 2020 data management team.

This project has the potential to have massive impacts on the teams' performance at competition. This is because having quick access to large amounts of data opens up many avenues for analysis possibilities. As it stands it is very complicated to pull insights out of data from multiple races at the same time because the data is stored in separate files. With the new system, aggregating such data over multiple files is as simple as analyzing just one file in the old system.

Most importantly, all work from this project must be clearly documented. This will allow this project to be used in a meaningful way by future data management teams and GFR at large.

Requirements (Subteam Rules Analysis)

Requirement	**Description**
Accept log files	Application code will be made that accepts a MoTec or Vector file, parses the data, and enters it properly into the database.
All data in the UI is organized and searchable	Users will have options to filter by one or multiple metadata values(date range, car, driver, track, etc.).
Selected data can be downloaded	Once data is filtered the user will have the option to export the file to .json, or .csv file formats.
Data can be accessed through an API (stretch)	An API that allows the database to be accessed, filtered, and altered programmatically.The type of API will be determined at a later date.
Quick preview of data being selected (stretch)	Users will see a preview of the data they have currently filtered. The preview will show what channels are available as well as relevant max and min values over the filtered data.

2. Current State Analysis and Benchmarking

2.1 Current State Analysis

At the date of this report, the database that stores the vehicle logs is designed and running but missing data. This means that the step of designing the overall structure of how we will store log data (the schema) is completed. The database is currently running on the stig gfr server shown below. The stig server was set up by the COE to be used by GFR for computing.

```
Dylans-MacBook-Pro:~ dylanalbertazzi$ ssh gfr_admin@stig-gpu.engr.oregonstate.edu
```

Figure 1: Stig Server Address

The next step for one of the members on the data management team is to populate the database. The current database schema is shown below. Unfortunately the photo is low resolution, however the important part is how data is split up into tables (yellow boxes) and less about the contents of the table.

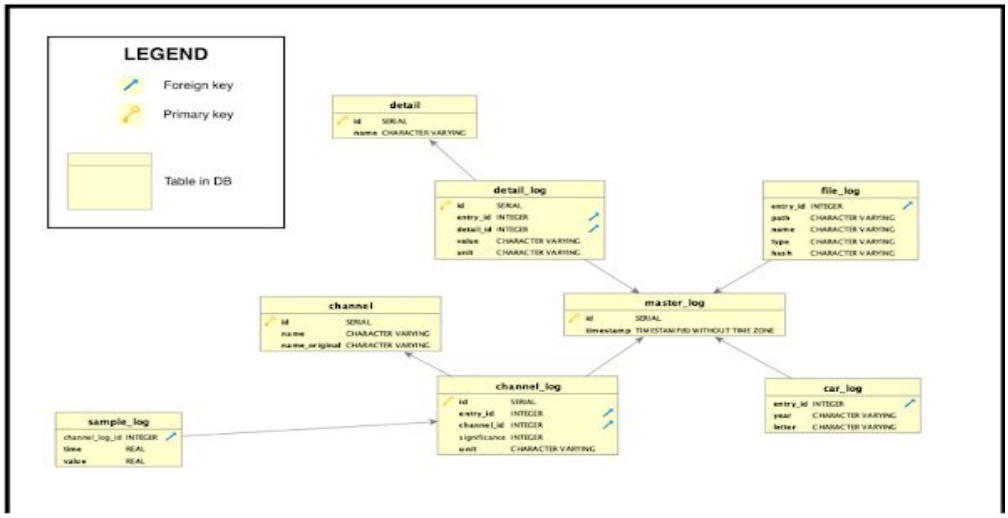


Figure 2: Datamaster v2 Schema

A single log file is uploaded and each piece of information is put into its correct place in the database. The reason this is helpful is because a schema like the one above allows for very fast data retrieval. For example, if a team member wanted the max velocity of a car from every race, the old system would have to open up every stored log file and read through all the unwanted information until it found max velocity. Whereas the new system can go right to the velocity column, filter results by the desired car and return the max velocity by race, only ever looking at velocity data.

The previous years focus was primarily on the structure of the database. This year's team will carry the baton by creating a UI so all of GFR and interact with the database in a meaningful way.

SWOT Analysis

Strengths	• The database schema is already done • GFR has access to servers to host data on • The previous team's code is well documented and written using industry-standard libraries
Weaknesses	• The team is unfamiliar with car data logging systems • There is only one person on this project
Opportunities	• This has never been done, there is a lot of untapped benefits • A robust UI design will be useful for years to come • The data that is stored will be valuable to use year after year • We have years of old data ready to be used
Threats	• It is unclear what the data needs of the other sub-teams are this could lead to a tool that is irrelevant to many sub-teams • There will be a tradeoff between time spent interviewing sub-teams on their data needs and building the tool • Losing our data

2.2 Benchmarking

A detailed analysis of the current state of the process or product in **measured numbers**. Describe the product or process with all **measurable values relevant to the performance of the process or product**. These measurable values can come from previous reports and other sources if they are properly cited. However, in most cases you will need to do some testing or measurement of the current system to be able to thoroughly quantify the current state. This can include surveys of team members using processes, weighing of components, or physical testing on parts of the car. "The information I needed wasn't in the old reports" is not a valid excuse. Make your own measurements where needed.

For Formula: Trevor Takaro's Work to equate design parameters to competition points may be useful here as well as later sections to help relate and evaluate the significance of high level design decisions.

Associate every opportunity you seek to take or weakness you seek to improve from your SWOT analysis into a measurable quantity with a current state value. This is basically developing engineering requirements (like in QFD) for each product or process you seek to improve. **The goal here is to identify how to measure the improvements you make to the system so that you can prove later in the report that your redesign is actually an improvement to the system.**

These are the specific requirements for this section [Sim & Tools]:

- ❑ **Discuss the performance of previous years system & its fulfillment of the system requirements**
- ❑ **Discuss the features and ease-of-use of the previous year's system**
- ❑ **Summarization of use-case analytics, qualitative feedback, etc from previous years**
- ❑ **Identify key areas of improvement**
- ❑ **Develop benchmark to compare new designs to**

Fall term of 2020 the data management team started on a project to host a database of GFR's race log files. Now in Spring of 2020, the first iteration of the system is not completed yet. In its current state the schema is created and the database is hosted on GFR's stig servers but does not hold any data. Nor is there any functionality for uploading or downloading data from it.

The last data management team used PostgreSQL and Sqlalchemy to design the database. This is important and helpful because it is an architecture that will scale to meet future large data needs. PostgreSQL and Sqlalchemy are both open source which means there is comprehensive documentation and large communities to use as a problem solving resource.

In our team's initial assessment of the project, it was found that uploading a .mf4 log file to the server took ~40 minutes. It is to be noted that upload times are much faster when ssh'd into the server. This is an area to assess if the database is going to be used widely by the rest of the GFR team. The two reasons are that many students who do not have a computer science background will have a hard time using ssh and most importantly the team does not want to handle the security behind giving everyone access to the whole server.

Improvements

The primary focus of this project is a UI that accepts log files, makes data accessible and easy to find, and able to be selected and downloaded. Because this is a new feature, the team has made its own benchmarks.
The table below mirrors the teams requirements for the term but has been revised to make it clear whether the requirement has been reached.

Requirement	**Description**
Accept log files	Accepts Vector files, enters data properly into the database.
All data in the UI is organized and searchable	Users can filter by one or multiple metadata values(date range, car, driver, track, etc.).
Selected data can be downloaded	User can export selected data in .json, or .csv file formats.
Data can be accessed through an API (stretch)	An API that allows the database to be accessed, filtered, and altered.
Quick preview of data being selected (stretch)	Preview that shows all data that can be selected when only a daterage is specified.

3. Design Analysis

The goal of this section is to generate distinctly different design iterations that are relevant and meet the customer and engineering requirements of the project. First, restate the specific goals of your design and include the assumptions and boundary conditions that will dictate your concepts. Generate free body diagrams to understand loads. Next, draw hand sketches of multiple (5) concepts. Hand sketches that best meet the requirements should be modeled using CAD. Use analysis tools (where necessary) to confirm designs meets requirements. Use a decision matrix to weigh advantages and disadvantages given for each relative to the current state. A good example of a thorough design comparison and engineering justification can be found here. (From Nick Lampert's 2010 design report) Most projects should include at least three complete future state designs.
Keep in mind, the first idea you come up with probably isn't the best.

3.1 Design Concepts

This section shall include 3 - 5 of the best design concepts. These images should be clear and well annotated. If required you may need to redraw your concept and annotate in MS paint or similar to make the images clear. Each concept shall include a clear description of the concept, advantages and disadvantages, risks, and a simple FMEA analysis to identify key areas where you could need more time to develop the design.

These are the specific requirements for this sub-section:

- ❑ **Clear image of each concept**
- ❑ **Clear description of each concept**
- ❑ **List/Table of advantages and disadvantages**
- ❑ **List or discussion of risks**
- ❑ **Discussion of possible failure modes**
- ❑ **No section extends past one page (Preferably each section fills one page)**

3.1.1 Concept 1 - [Descriptor]

Different types of databases

There are multiple types of databases, each with their own strengths. Over the last few decades, many types of databases have been developed to fit a myriad of use cases.

The choice of database type has a large impact on the types of tasks the system can efficiently perform and the features available. The following section provides an explanation for choosing a PostgreSQL database for GFR's vehicle log data. The choice is made clear though explaining the strengths and trade offs present in the most popular database types.

Relational Databases

Relational databases are the most traditional and comprise the majority of databases in production. They are built in the idea of tables. Tables hold columns and rows. Columns have a name and a data type while a row is an individual entry. To be entered successfully, each entry must match the data type described by the column.

This allows for a high level of data integrity and predictability. The drawbacks of a relational database are that they are hard to restructure, unlike their “NoSQL” counterparts.

Examples:

- MySQL
- MariaDB
- PostgreSQL
- SQLibe

Figure 3: Relational Database ERD

NoSQL Databases

NoSQL refers to a category of databases that does not require a consistent schema in order to make entries. The term NoSQL is a misnomer because some databases can still be queried in a SQL-like fashion. A better name would have been “non-consistent schema databases”.

Key-Value

Key value databases store arbitrary information that is accessible through a key. Similar to storing a JSON object. These types of databases are often used to store state information and data represented by a dictionary or hash. The perks of key-value are fast retrieval time and low-complexity data access. The user is responsible for the naming scheme of keys and that the data is in the correct format.

Examples:

- Redis
- Memcached
- Etcd

Figure 4: Key-Value ERD

Document Databases

These are the next step up in complexity from key value databases. They are similar in that they use keys to identify data in the database. However, instead of storing arbitrary data they store it in structured formats called *documents.* JSON and XML are common ways to implement the structure of a document.

There is no prescribed schema, allowing each document to have a different internal structure. Content can be queried and analyzed. This structure is well suited for rapid development where the final data structure is unknown. It should be noted however that keeping a consistent structure can be hard to maintain and is a drawback
to a non relational model.

Examples:

- MongoDB
- Rethink DB

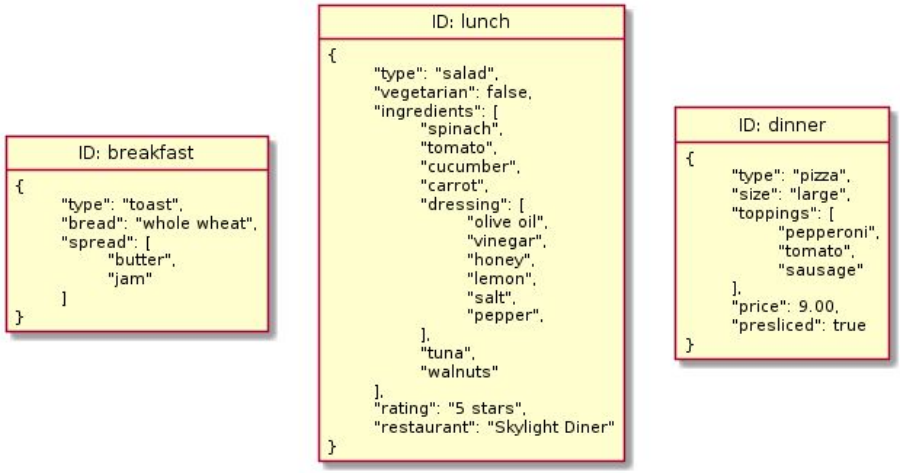


Figure 5: Document Database ERD

Time Series

Time series databases are designed to handle incoming flow of time series data. They are heavily write oriented and are typically append-only. Common uses are for monitoring IOT devices and system performance.

Examples:

- OpenTSDB
- Prometheus
- InfluxDB

Time	CPU Temp	System Load	Memory Usage %
2019-10-31T03:48:05+00:00	37	0.85	92
2019-10-31T03:48:10+00:00	42	0.87	90
2019-10-31T03:48:15+00:00	33	0.74	87
2019-10-31T03:48:20+00:00	34	0.72	77
2019-10-31T03:48:25+00:00	40	0.88	81
2019-10-31T03:48:30+00:00	42	0.89	82
2019-10-31T03:48:35+00:00	41	0.88	82

Figure 6: Time Series ERD

NewSQL

NewSQL databases were designed in the 2010's to combine the scalability of NoSQL with the consistency of relational databases. They address the tradeoff between consistency and availability. NewSQL databases are well suited for high voulumns of relational data in distributed, cloud environments.

Examples:

- MemSQL
- VoltDB
- Spanner

PostgreSQL Database Choice Justification

PostgeSQL is the best database choice to meet GFR's needs for a log database that will be consistent with high member turnover, scalable for the future, and reliable. It meets all these requirements and more. PostgreSQL is the most widely used database for production. It's open source which means it's free and there is a large community of developers to reach out to when members run into problems.

A document database like MongoDB would have been another viable option. This is what datamaster uses to store motec files. The reason for not choosing the old way is because it doesn't impose the structure of a relational database. Giving each user access to structure a file opens doors to a potential nightmare in tracking the data structure of log files across the organization. Especially with so many students new to development and GFR.

Figure [#]: Concept 1 - [Description]

3.2 Design Iterations ---NOT APPLICABLE ALL WORK IS IN SECTION ABOVE

This section shall include a minimum of 3 CAD'd design concepts. There shall be appropriate images that should be clear and well annotated. If required you may need to redraw your concept and annotate in MS paint or similar to make the images clear. Each concept shall include a clear description of the concept, advantages and disadvantages, risks, and a simple FMEA analysis to identify key areas where you could need more time to develop the design.

These are the specific requirements for this sub-section:

- ❑ **Minimum 3 CAD'd iterations**
  - ❑ **Design criteria**
  - ❑ **Listed assumptions**
  - ❑ **Required factors of safety**
  - ❑ **Design inputs: dimensions, loads, flow rates, cycle time, etc.**
  - ❑ **FBD of each part**
  - ❑ **All accompanying simulations or calculations**
    (This means CFD for aero, FEA for drivetrain/suspension/cPow, Kinematics suspension/aero, etc.)

There must be simulations or calculations associated with your design!

- ❑ **Discussion of benefits of each concept with minimum:**
  - ❑ **Detail description of design**
  - ❑ **words, graphs, pictures, tables, decision matrix, etc.**

4. Design Selected

A complete specification of the design & components selected.

4.1 Technical Specification

Create, and refer to, detailed engineering drawings, process diagrams, simulation results, facility layouts, or other appropriate tools to completely specify all aspects of the future state. Equations and sample calculations included and explained.

These are the specific requirements for this sub-section:

- ❑ **Schema of database with explanation; Description of master views**
- ❑ **Description or functional diagrams for application code**
- ❑ **API documentation (can be linked)**

The vehicle log database schema is shown above. Note, google docs makes photos blurry when resized. Make the diagram larger to read it. In order to make sense of it, first *database normalization* must be understood. Database normalization divides the data into separate, independent logical entities and relates them using a common key. Consider the non-normalized database in figure 7 describing computer science(CSE) students enrolled at a university.

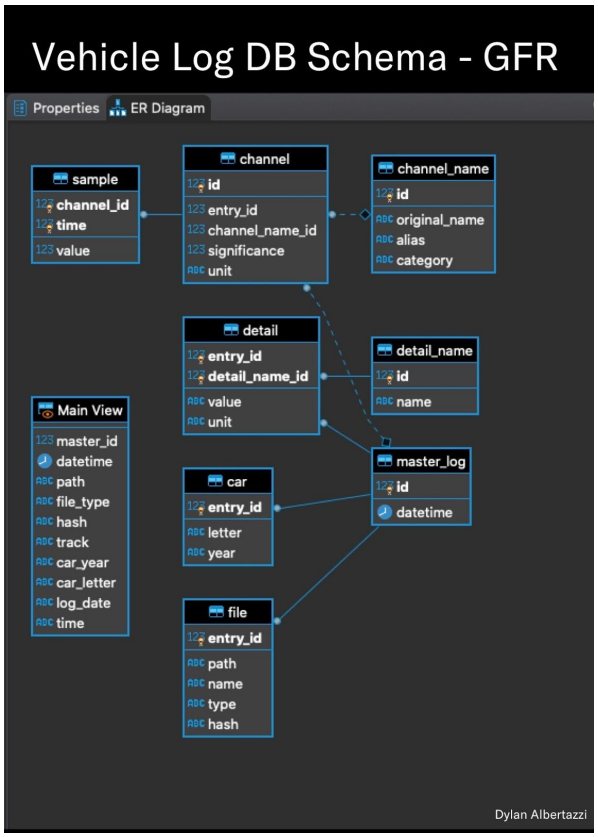
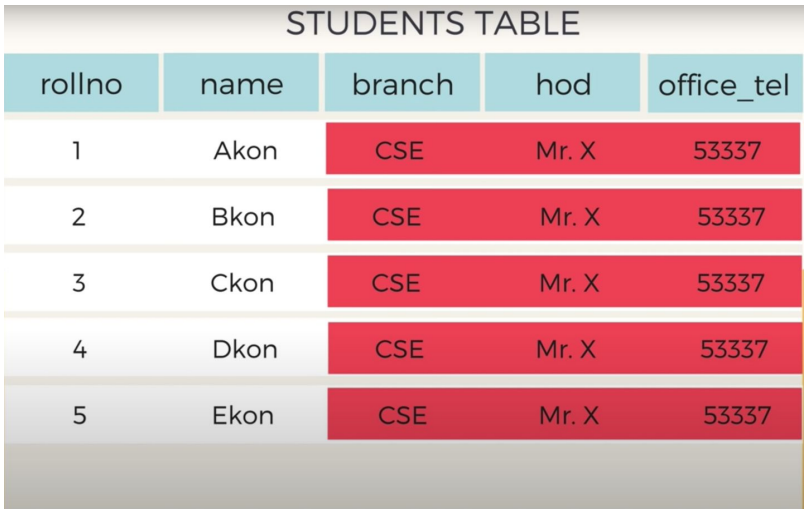


Figure 6: Vehicle Log DB Schema



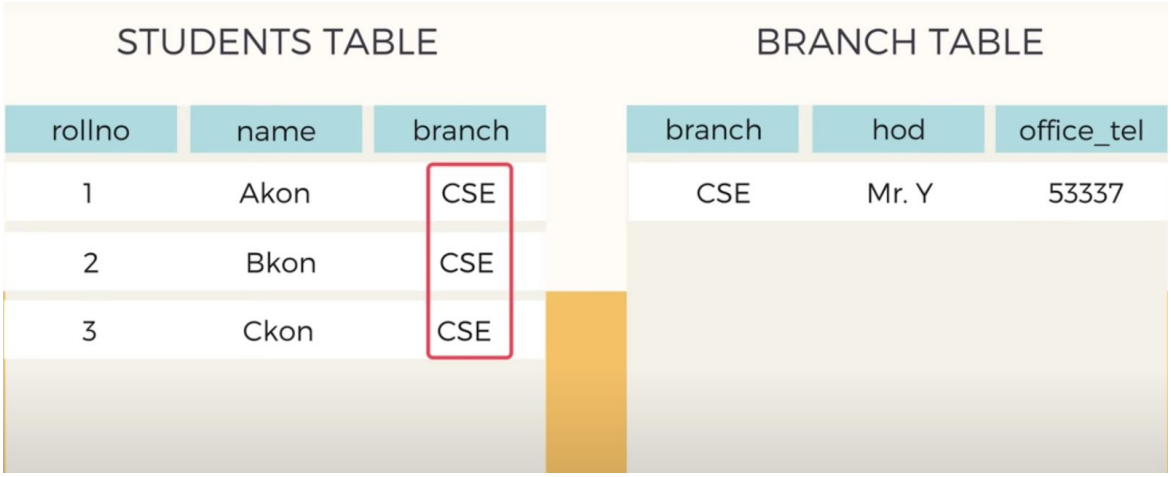
STUDENTS TABLE

rollno	name	branch	hod	office_tel
1	Akon	CSE	Mr. X	53337
2	Bkon	CSE	Mr. X	53337
3	Ckon	CSE	Mr. X	53337
4	Dkon	CSE	Mr. X	53337
5	Ekon	CSE	Mr. X	53337

Figure 7: Non-normalized database

Everytime a student is enrolled, hod and office_tel are entered into the database again. Even though they are the same for every CSE student. This is bad. In order to update the hod or office_tel, the user would have to update every single row that contains the old value. This is a lot of extra work and in the event of an update getting interrupted, could lead to data inconsistency.

Now consider the normalized version of the previous example.



STUDENTS TABLE

rollno	name	branch
1	Akon	CSE
2	Bkon	CSE
3	Ckon	CSE

BRANCH TABLE

branch	hod	office_tel
CSE	Mr. Y	53337

Figure 8: Normalized database

What was one is now two tables of independent logical entities, student and branch. Notice that redundancy is not eliminated but reduced, branch name still needs to be repeated for every student. Now however if the office_tel was updated, it would only be changed once in its row on the branch table.

Second normal form database normalization was used as the standard of normalization across the vehicle log database. Each table on the schema diagram above represents an independent logical entity. The root of the database is the master_log table where each row represents a different entry into the database.

Database Contents

Each entry is a run from one of GFR's vehicles and holds all the data that the on board computer was able to log. In some cases the .dbc files are not known so the data cannot be interpreted. A dbc is a text file that

contains information for decoding raw CAN data. Each log is separated into metadata, and channel data. Metadata consists of information about the car letter, car year, driver, track, etc. Channel data consists of numerical sensor readings, acceleration, velocity, etc. Channels are stored as numpy arrays nested inside of a dictionary.

Master View

The masterview pulls together relevant data from many normalized tables to present a clear picture of what the database contains. In GFR's case that is track, car_year, car_letter, log_date, etc. This view is created by left-joining columns from the master_log and detail_name tables on the master_log.id. The current masterview is shown on the schema diagram above. All GFR members will likely refer to the master view when selecting data for their analysis.

Application Code (Accessing the DB)

This section provides an overview of the vehicle log database's application code. For documentation refer to the Logfile Querying Tutorial. Application code allows a user to access the database with Python instead of SQL. The data-management team chose to use the SQLAlchemy library to write application code to interact with the db. Note that if a GFR member is familiar with SQL they do not need the application code to access the db. The application code has two main sections, viewing and querying. Many would consider the SQLAlchemy library sufficient to interact with the database. However, many of GFR's members have little coding experience and would benefit from an easier way. The narrow use case of viewing and filtering made it easy to build functions on top of SQL alchemy that allow for very straight forward use. The function specifics are in the tutorial linked above.

API documentation

An API provides an alternative way of accessing the database programmatically. Instead of giving users access to the whole database, an API exposes only specific data. The API is best practice because only certain actions are available to users. The GFR API will give access to viewing and querying but not to file insertion or deletion. Insertion and deletion is possible with the application code and is a security risk that should be taken into account. Documentation will be linked here when the API is built.

4.2 Project Timeline

Specific requirements for this subsection:

- ❑ **Detailed plan outlining:**
  - ❑ **Major tasks**
  - ❑ **Deadlines**
  - ❑ **Time required to complete the tasks**
  - ❑ **Number of people required to complete tasks**

Major tasks

1. Write application code to perform CRUD operations on the vehicle log database. This includes accepting Vector files to the database.
2. Design and deploy an API that can query the database.
3. Deploy a UI that displays log file data in an organized and searchable manner.

Deadlines

1. Task 1 - Tuesday of week 10 Spring 2020.

2. Task 2 - Monday of week 5 Fall 2020.
3. Task 3 - Monday of week 10 Fall 2020.

Time and number of people required
Time estimates are based on a GFR member who is completing the task at hand for the first time and with no prior experience. Most of the time accounted for will be spent learning the concepts needed to complete a task.

1. Task one will take one person roughly 60 hours. This time includes the learning curve to get familiar with GFR's codebase. This also includes the subtask of putting data into the database.
2. Task two will take one member roughly 70 hours. On top of building it, proper consideration must be given to the correct API architecture, deployment method, and endpoints to expose. This will require collaboration with others with experience.
3. Depending on the progress of the UI a DHBW team is working on, the time taken for task three will vary. Hopefully our team will be able to take the UI they have built and simply connect our API. The other case is if it isn't completed, the UI will have to be built from scratch and require significantly more time.

5. Implementation

A detailed manufacturing/implementation record. Description of construction issues encountered and design changes made. Includes pictures and engineering analysis to support and describe all design changes made during the manufacturing / testing process. Engineering methodology justifies selection and sizing of tools used to manufacture or implement the design. Equations and sample calculations used included and explained. Assembly and process tutorials included in appendices when appropriate.An example of a good visual process flow chart for the manufacturing of the Formula chassis can be found HERE. (From Formula Chassis 2012 Report)

This section is intended to be an actual record of the manufacturing processes to match the outline above plans.

5.1 Shifting Project Scope

When building out the web application and UI the team quickly ran into problems with accessing the OSU hosted database. For security reasons, a database cannot be accessed directly from the internet and must be connected to a server which acts as a bridge between the client and the database. This forced the team to take a step back and assess the architecture that the web app will sit on. This was unforeseen and shifted the course of the project from building the UI to designing the architecture that will allow the web application to work with the database. In the architecting process, the team was also able to plan for access to real time data at the race track and long term cost which were not considerations before.

5.1.1 Computing Platform Selection

Our team weighed three options to build the architecture on, Amazon Web Services, Google Cloud Platform, and hosting locally on the OSU servers. We ultimately determined that a hybrid solution of hosting the database on the OSU servers and the web app on AWS is optimal for cost, availability, and scalability. By using our own server, GFR saves the cost of hosting a database in the cloud, which would have been the most expensive part of the setup. By using AWS, GFR gains access to the largest cloud computing infrastructure in the world, which allows for scalability as the team grows, opportunities to integrate other AWS services into the app, and is a highly marketable experience for GFR members to showcase to future employers.

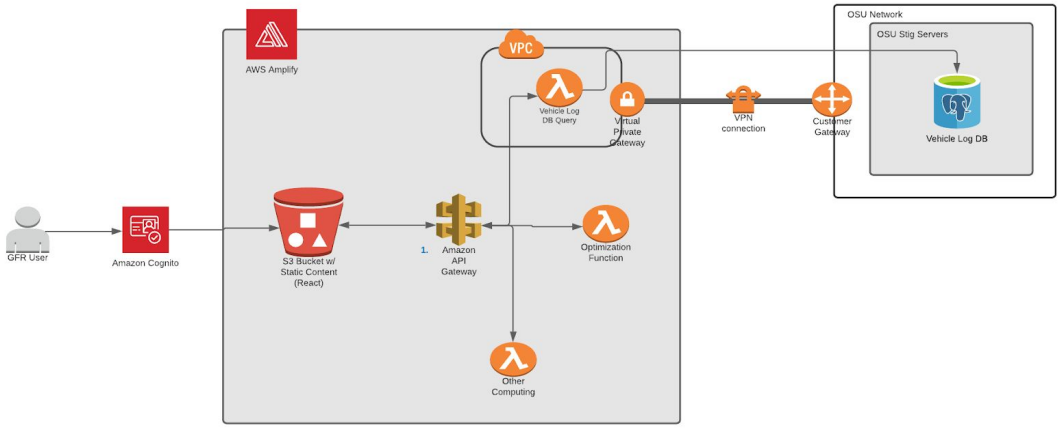


Figure 9: Final Web App Architecture Diagram

5.2 Architecture Design

Our team iterated through multiple architectures before landing on the hybrid option with AWS. Early on we knew that it would be cost prohibitive to host a PostgreSQL database in the cloud. There are other cloud database options that are much cheaper but because we already determined PostgreSQL to be best for our use case and because it is already running on the OSU servers, we kept it. Future teams may want to assess other database options. Possible improvements could be made in speed, logging, availability, and reliability if migrated to the cloud. All architectures below have the database hosted on the OSU servers.

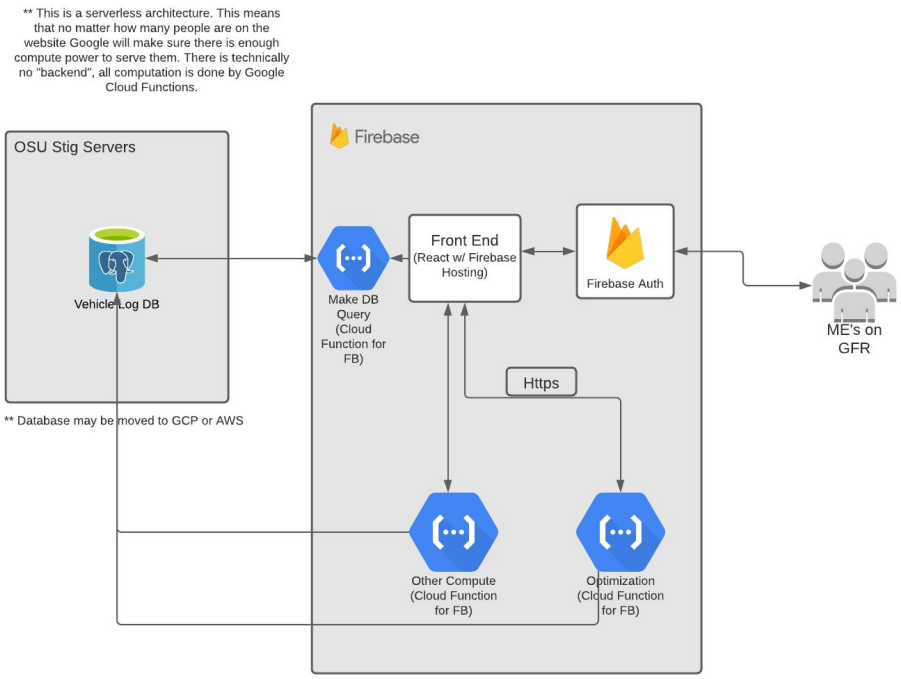


Figure 10: GCP Web App Architecture Diagram Iteration

Above is a serverless architecture hosted on GCP. The front end React app is stored in firebase and the computing is done by cloud functions. Everytime the database is called, or a computation needs to be made a cloud function runs via an HTTP request and returns the response to the front end.

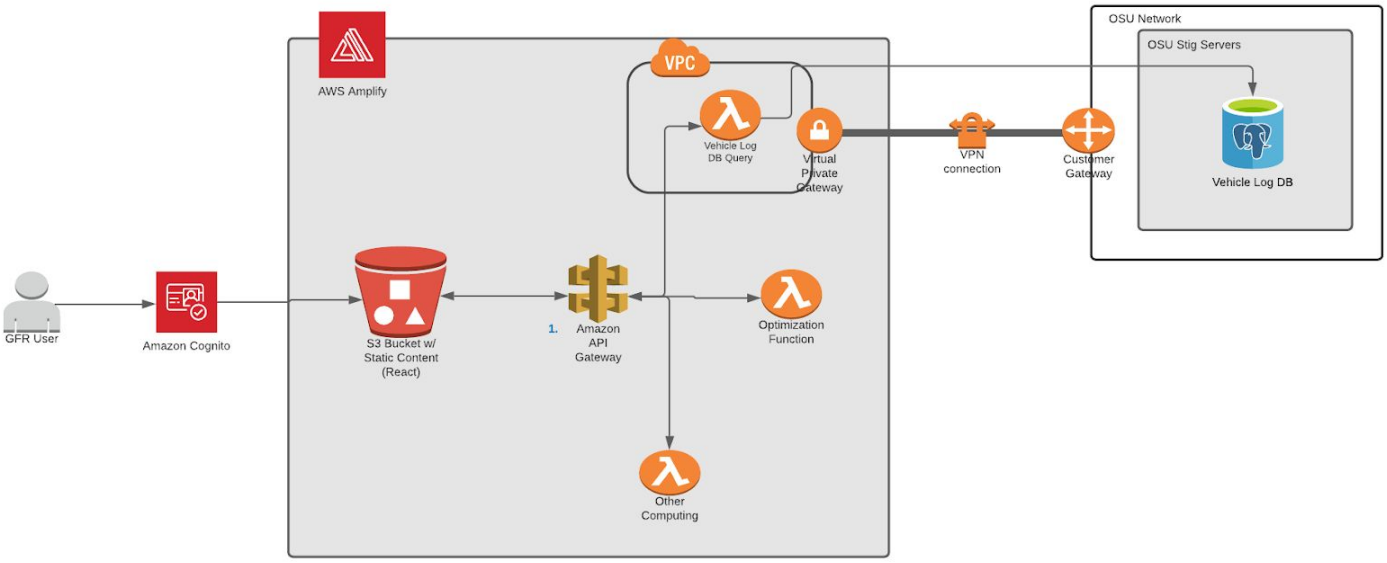


Figure 9: Final Web App Architecture Diagram

Above is our final solution built on AWS with the database hosted by OSU. The architecture is serverless just like the GCP one and utilizes lambda functions for computing. Connection to the database is made though Customer Gateway which is a service that will connects the OSU VPN to GFR's network in the cloud. While GCP and AWS will both get the job done. AWS is much larger than GCP and offers more services to serve GFR in the future like SageMaker for machine learning and Kinesis Streams for real time log ingestion.

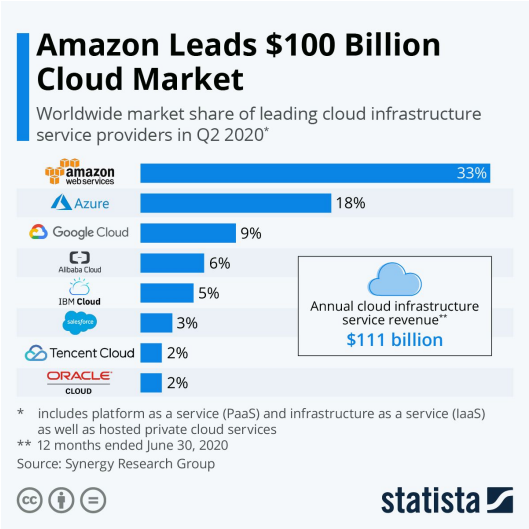


Figure 11: [Cloud Market share Q2 2020 - Statista]

Within AWS the GFR organization will be set up with one root user, Bradley Anderson. The root user is in charge of billing, and delegating permissions to GFR members. GFR developers are then only given access to what they need to complete their project. You'll notice that normal GFR members aren't on this diagram. That is because they won't have access to the backend and will log in to the web app with their email via Amazon Cognito as shown in the diagram above. This follows the principle of least permissions and only gives team members access to the resources they need.

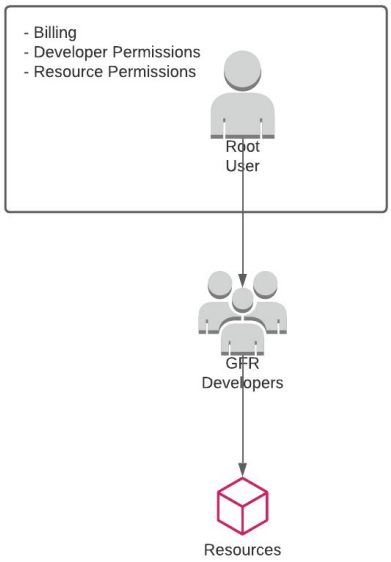


Figure 12: AWS User Permission Structure Diagram

AWS has built in logging to track each developer's use on the platform, allowing for quick diagnosis of problems and revocation of permissions if necessary. AWS also has spending limits and billing alarms that can be set to make sure no one accidentally spins up a service that costs GFR more than is allocated.

By the end of the term, the organization will be set up in AWS, the current code base will be migrated over, and the current capstone members will have the correct permissions for the project.

6. Testing

6.1 Sub-System Validation Design of Experiment

Specific requirements for this subsection:

- ❑ **Detailed list of tests required to validate calculations and simulations**
- ❑ **Detailed plan on how to measure subsystem performance**

6.2 Tests Complete to Date (Not Applicable)

A detailed explanation of the tests performed & analysis of test results. Testing process tutorials included.

6.3 Tests to Complete

A detailed testing plan for the tests that to be performed on your system to validate functionality & design

calculations. Detailed designs for needed testing apparatus or fixture included. Testing process tutorials included when appropriate. Sample calculations of the end analysis & expected values given.

Connection Test

This test verifies connection between the AWS cloud and the OSU hosted database. In order to make the connection, a VPC and Customer Gateway must be set up in AWS to connect the OSU and AWS networks. To test, write a lambda function that calls for the first 10 rows of the master view from the vehicle_log table. See the DatabaseORM and DB Querying Tutorial files for application code and connection string to write the lambda function. If the rows are returned a successful connection has been made.

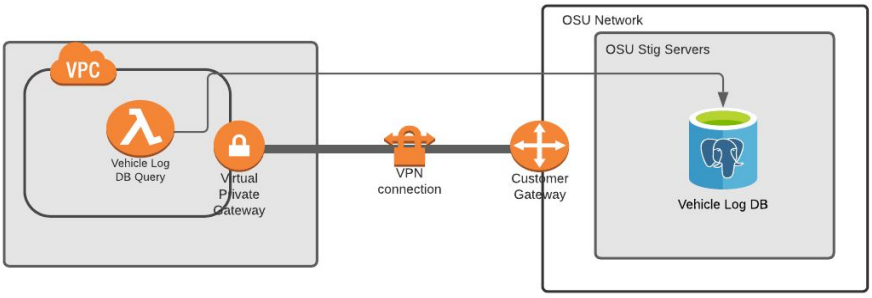


Figure 13: Connection Test Diagram

7. Conclusion

Reflection on entire project and your individual contribution. Suggestions/advice for future team members and management on design, manufacturing, and team aspects.Graded by quality of content; positive & negative critiques are encouraged.

7.1 Reflection and Recommendations

Over the past six months, I learned a lot about what it takes to build a data-driven web application. I contributed to the team by writing the code that takes vehicle log files and inputs them into the database. I used that code to fill the database with the log files the team currently has. I researched and designed the optimal cloud architecture for the GFR team. I will be continuing the project with a 406 next term and am looking forward to implementing the UI that was the goal of the project in the first place. Like many projects, in the beginning there are factors that get overlooked, as was the case with my capstone. In the beginning we said that I would fill the database and make a UI, however we totally neglected the infrastructure that the database and UI would sit on. So instead of focusing on the UI fall term, the focus was on designing an architecture to support the UI. The good news is the architecture is designed so now the foundation has been laid for the next group to build out the UI with confidence knowing it can handle heavy workloads and be accessible across the world, including on the race track. The foundation has also been laid to continue leveraging AWS services to improve the car. The AWS organization has been set up and configured, and because of its serverless microservice architecture the web app is easily extensible to any computing use case GFR has.

In the future I recommend the team looks at kinesis streams for real time vehicle log data ingestion. I also recommend we look at using Amazon Aurora which is a PostgreSQL compatible serverless database. Because it's serverless we will only pay for what we use, possibly making costs significantly lower than a normal PostgreSQL cloud solution. The other reasons for migrating the database to the cloud is we won't have

to pay for data transfer charges if we host in the same Region, we will have highly enhanced logging on the data, and we will have redundancy so if one database goes down we will have a backup.

For new team members on this project, I highly recommend ACloudGuru for learning about AWS. Over the course of this capstone project I used the ACloudGuru courses to get my AWS Cloud Practitioner, and Cloud Solutions Architect certifications. Not only will it help you on this project but the certifications are very marketable when you look for a job after school.

8. Works Cited

Use the Chicago Manual Style to create a full Works Cited page, per SAE International recommendation. For OSU SAE documents that were linked within the body, simply write the title & author(s) ("2011 Chassis Manufacturing Team") is acceptable for author) and create another hyperlink. For external documents, write a proper citation.

Figure 1: Creating a PostgreSQL Database - Justin Mai
Figure 3-6: Comparing Database Types - From comparison of databases article on prisma.io
Figure 7, 8: Basic Concept of Database Normalization - From video created by Studytonight on YouTube
Figure 9, 10: Created by Dylan Albertazzi
Figure 11:Cloud Market Share Chart - Statista
Figure 12, 13: Created by Dylan Albertazzi