Rental Based Autonomous Vehicle Cloud (AVCloud) -Building A Cloud-Based Service Platform for Rental Autonomous Vehicles

CMPE 281 Cloud Technologies, Fall 2021
Instructor: Dr Jerry Gao
Computer Engineering Department
San Jose State University, San Jose, CA

Monica Lakshmi Mandapati
monicalakshmi.mandapati@sjsu.edu
015219711

Annapurna Ananya Annadatha
annapurnaananya.annadatha@sjsu.edu
015218385

Vineeth Reddy Govind
vineethreddy.govind@sjsu.edu
015363556

Indhu Priya Reddem
indhupriya.reddem@sjsu.edu
015930148

Guruvardhan Reddy
guruvardhanreddy.ranjanala@sjsu.edu
015905149

Abstract-- **The AVRental is a Software as a Service (SaaS) application deployed in Amazon Web Service (AWS) cloud platform. The application can be used for booking Autonomous vehicles (AV) to commute from one place to the other. The services provided by our application are booking ride, enrolling to subscription plan, status tracking and sensor data monitoring of the vehicle. We have integrated CARLA simulator with our application to derive real time sensor data of the AV. This application is highly available, scalable, and capable of meeting any requirements posed by huge datasets generated by users and vehicle sensors as we have deployed our entire application onto the AWS cloud service through auto-scaling.**

I. Introduction

The advancements in the vehicular industry have immensely benefited different related industries and served the humanity by increasing the efficiency of our routine operations. Every year, over 34,000 individuals die because of motor-vehicles, traffic-related injuries, and it is the top cause of mortality among those aged 4 to 34. Over 1.20 million people die every year because of road traffic accidents globally, with an additional 20 to 50 million people suffering from non-fatal injuries such as physical disability. Human mistake is at blame for more than 90% of these mishaps. As a result, there is a need for technology that is constantly focused on it, never gets sidetracked, and requires minimum human intervention. Autonomous cars can help attain these objectives.

Autonomous vehicles are a hot topic in study with a long history. The autonomous driving research conducted in the early 1980s and 1990s proved the feasibility of producing cars that autonomously regulate their motions in complicated surroundings. Autonomous car prototypes were first restricted to indoor use. With digital microprocessors, a totally onboard autonomous vision system for cars was created and implemented in the early 1980s. Many firms, such as Tesla, Amazon, Audi, and BMW, are now attempting to make a completely self-driving automobile a reality.

A. Motivation

The popularity of autonomous cars is growing by the day because of all the benefits they provide, such as road safety, environmental friendliness, speed, and accessibility to all, including the specially abled and children who will not require any type of assistance to commute. The automotive cloud market is predicted to expand to USD 9-9.6 billion by 2025, making it a fast-growing sector. There are also several open-source tools available now that enable individuals to create applications based on autonomous vehicles. With the emergence of self-driving cars, there is a demand for apps designed specifically for them.

B. Objective

Our project's main goal is to create a cloud-based autonomous vehicle rental service that helps customers to easily travel from one place to another. Our application will provide the following components:

i) **Carla-based simulator:**
A simulated AV which supports the following functions:
(a) A user interface – It provides the interactions between a client and a simulated AV as an online interface.
(b) A simulated AV with a state-based process which supports its AV configuration, control operation, and state tracking.
(c) Communications between a simulated AV and the back-end server in a cloud.

ii) **AV-based database management component:**
This is an AV database management component, which stores, updates, and managements diverse registered and connected AV status information,

selected emergency data, and service records. Two types of databases must be used here:
a) SQL-based DB of user accounts and registered AVs - MYSQL
b) NOSQL DB supporting diverse service data and records from AVs - MongoDB

iii) **Remote online AV tracking and controlling:** This provides remote online tracking and controlling the status of simulated AVs, including its moving states (Moved/Moving/idle), service states (active, connected, inactive), location information, road service records, and sensor states.

iv) **AV service connectivity supporting rental AVs on a cloud:**
This provides AV service connectivity for simulated AVs. There are two types of connectivity:
a) AV cloud management connectivity protocol between AV simulators and the back-end AV cloud management server, including AV cloud management communications.
b) AV service management connectivity protocol between AV simulators and the back-end AV rental service server. This connectivity refers to AV cloud rental service communications, including moving states (Moved/Moving/idle), service states (active, connected, inactive), location information, road service records, and schedules.

v) User service management:
This component supports user service management, including user account, service account, schedule management, and billing.

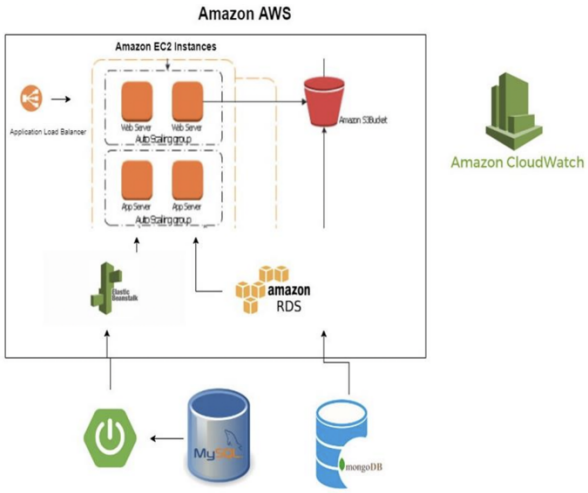
C. *Special Advantages*

We have built an application for renting autonomous vehicles as part of this project. The user must first log in and register with the program. The user may then book a trip to any place, and the trip details will be recorded in the MySQL database, which will be updated by the vehicle management. Sensor data will be recorded in the MongoDB database and the user can track the sensor data of the vehicle. The admin user can view the statistics of number of users registered and number of active and inactive Autonomous vehicles.

II. Related Work:

Traditional rental services use non-cloud-based services as a communication tool to exchange messages between servers and clients. We defined a novel approach in which communication between all the services of the application happens via cloud-based communication protocols. However, this approach brings to light the various technical issues like synchronization messages between all the services on the cloud and the request/response resolution both on the client side and server side. We used several techniques like message queues and load balancers to make sure even distribution of requests to servers and equal distribution of resources to all the services. We used CARLA simulator to simulate the real-world driving experience virtually.

III. Cloud-Based System Infrastructure and Components



Amazon Remote Database Service: To access the database from anywhere, we have deployed the database on AWS-RDS which is web service running on cloud designed to simplify the setup and operation. To connect the node application to MySQL database, MySQL middleware is used.

Amazon Elastic Beanstalk: We have used Spring Boot application to configure restful APIs to fetch data from MySQL database. To access the java application remotely we have deployed the jar file onto Elastic Beanstalk.

Amazon EC2: We have used Amazon EC2 instance to deploy NodeJS and react application. We cloned the application and installed the dependencies on EC2 instance by running NPM install command. Once the application cloned, we start both frontend and backend servers. The application will be deployed on the generated publicIP and can be accessed using the same.

Amazon Cloud Watch: CloudWatch collects monitoring and operational data in the form of logs, metrics, and events, and visualizes it using automated dashboards so you can get a unified view of your AWS resources, applications, and services that run on AWS and on premises. It helps us to monitor the AWS applications that are deployed on cloud.

IV. Simulator Design and Implementation

A. *Simulator Design and Implementation*

For our AVRental application, we used the CARLA simulator to model an area and the automobiles we used. CARLA is a self-driving simulator that is free and open source. It was built from the bottom up to be a flexible and scalable API for dealing with a wide range of autonomous driving activities. One of CARLA's main goals is to assist democratize autonomous driving research and

development by offering a platform that customers can use and customize rapidly. To accomplish this, the simulator must be able to meet the demands of a variety of use cases related to driving (for example, learning driving policies, training perception algorithms, and so on). CARLA is based on Unreal Engine and uses the Open DRIVE standard to describe highways and urban landscapes. A Python and C++ API that is developed in concert with the project is used to control the simulation. We developed certain APIs and adjusted them according to our needs to meet the functionality of our program, using the Python API library given by the CARLA community of users.

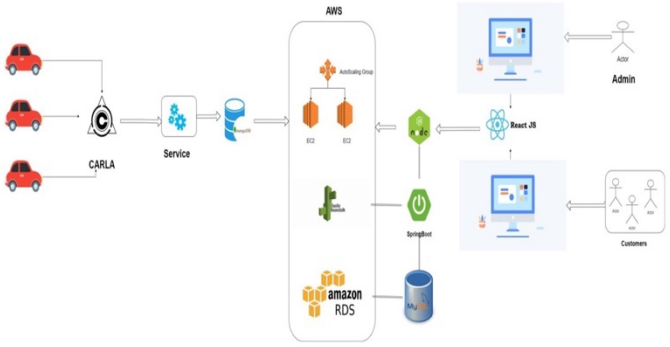
B. Simulation Connectivity Design

The Traffic Manager is the simulation's vehicle operating module. It is based on the CARLA API, which is a C++ library. Its goal is to make the simulation more realistic by including realistic urban traffic situations. Users can alter some behaviors, for example, to define specific learning circumstances. Users have some control over traffic flow by setting parameters that create, force, or enable specific actions. Users can change traffic behavior in both online and offline environments.

The logic can be summarized in the following way:

- Save and reload the current state of the simulation.
- Calculate the movement of each registered vehicle. The traffic manager's major goal, according to the simulation state, is to develop feasible commands for all the cars in the vehicle register. The measurements for each car are done independently. These equations are divided into many phases. The control loop guarantees that all measurements are consistent by establishing synchronization barriers between steps.
- Execute the instructions in the simulation. Finally, the traffic managers identified the next order for each vehicle, and all that's left to do now is put it into action. The command array gathers all the commands and transmits them to the CARLA server in a batch so that they may all be executed simultaneously.

V. CLOUD DATA DESIGN AND IMPLEMENTATION



CARLA Simulator: It is an open-source simulator for Autonomous vehicle driving research. It provides different API's that allows us to retrieve data from the Autonomous Vehicle surroundings.

User Interface: We are using **ReactJS** for the frontend of the application and **NodeJS** for the backend of the application. Customers can register, login, add membership plan, add vehicles, book a ride and track the car from the User Interface and Admins can monitor the Registered Av vehicles data and information.

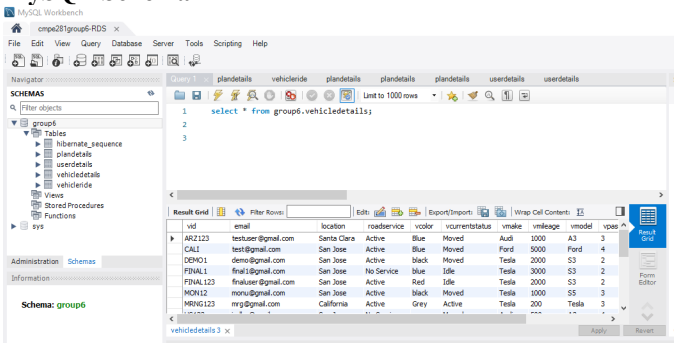
Databases: MongoDB: The CARLA Simulator sends massive amounts of data, so we are using a NoSQL database to handle all the sensor data that is sent by the simulator.

MySQL: We are using MySQL- a Relational database to store the user's data, rides history and the membership plan information.

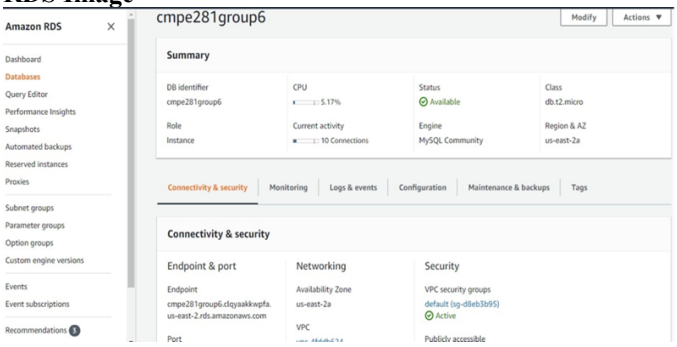
A. Cloud DB Design and Implementation (SQL DB Design)

MySQL is a simple database to work with. It's a relational database management system that's open source. It will be used to record user information, vehicle inventory, and ride history and status. The MySQL database is hosted by Amazon's RDS service, which makes it simple to set up, run, and expand MySQL deployments in the cloud.

MySQL Schema



RDS Image



B. System Big Data Design and Implementation (NOSQL DB Design)

MongoDB Atlas, a cloud-based open-source database server, was chosen for the NoSQL DB implementation. It combines all of the benefits of classic MongoDB with unrivaled data distribution and mobility across a variety of cloud platforms. It primarily supports three major cloud providers: Amazon Web Services, Microsoft Azure, and Google Cloud. We have an AWS cloud platform as part of our project. The data from the GNSS, collision, and lane incursion sensors is stored in MongoDB. All of the sensors are gathered at the end of the trip and saved in MongoDB. For identification reasons, all sensor data is linked to the trip number.

MongoDB Schema


```
_id: ObjectId("61b0002a686a0386deb99d11")
time: 1638924330353200800
Model: "Tesla Model 3"
traffic: "Medium"
passengers: 2
condition: "Good"
fuel: "NA"
battery: "46%"
door: "locked"
state: "moving"
temperature: "20C"
weather: "sunny"
estimated arrival: "2 mins"
vehicle: Array
    0: "Server:                  0 FPS"
    1: "Client:                  0 FPS"
    2: ""
    3: "Vehicle:          Tesla Model3"
    4: "Map:              Town10HD_Opt"
    5: "Simulation time:       0:06:40"
    6: ""
    7: "Speed:                  0 km/h"
    8: "Heading:               91° SE"
    9: "Location:     (-114.2,  43.8)"
    10: "GNSS:  (-0.000403, -0.001026)"
    11: "Height:                  -0 m"
    12: ""
city: "Santa Clara"
```


VI. CLOUD SYSTEM DESIGN AND SERVICES

A. System Function Component Services

Similar to the spatial separable convolution, a depth wise separable convolution splits a kernel into 2 separate kernels that do two convolutions: the depth wise convolution and the pointwise convolution. But first, let's see how a normal convolution works.

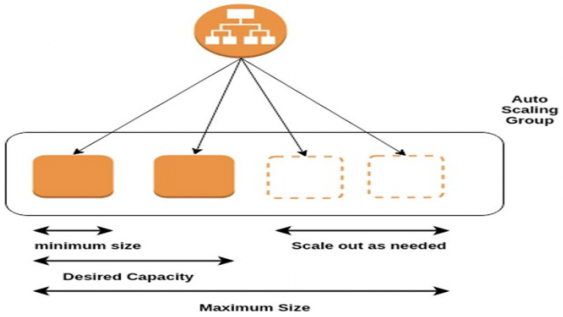
System Function Component Services (such as Billing, Connectivity,)

We have included the following functional components in our application

- Simulator: For simulating the real-world scenario we have used CARLA simulator which is an open-source simulator. This simulator provides APIs which expose various functionalities that we need to perform. We used these APIs to interact with the CARLA world.
- Connectivity: For connectivity between database and the application we used 2 types of interfaces for each kind of database. For interacting with the application, we used MySQL database hosted on Amazon RDS to push and fetch the data as required with APIs provided by AWS. For connectivity between NoSQL and the simulator we used APIs provided by MongoDB atlas to fetch and push data when required.
- Database: We used 2 types of databases for this application, the first one in RDBMS based MySQL to store the user data, booking data etc. And the second one is NoSQL database to store real-time sensor information provided by CARLA simulator.
- UI component: To provide a clean user interface for using the AV rental application we used JavaScript frameworks ReactJS and Redux. All the UI components were designed using this framework.
- Backend component: As a middleware between the UI and database we used NodeJS, Spring Framework to create custom APIs to interact with the databases.

B. System Scalability Design and Implementation

A good web application should be able to handle development quickly and seamlessly, and it should be built to scale. A scalable web application can handle an increase in users and load without giving users difficulties. The deployed application's AWS auto scaling group is created automatically by AWS Elastic Beanstalk. AWS Auto Scaling is a feature that allows you to modify the number of computing resources necessary for an application's execution on the fly. In a single-instance environment, the Auto Scaling group guarantees that only one instance is active at any one moment. The Auto Scaling group is configured with a variety of scenarios to run in a load-balanced environment, and it adds or removes instances as needed based on load. We're using AWS auto-scaling groups to manage our load on the AWS EC2 instance for our project, and we've set the minimum EC2 instance to one and the maximum EC2 instance to two



C. System Load Balance Design and Implementation

The AWS Elastic Load Balancer is a basic AWS cloud service that distributes inbound traffic to multiple destinations across one or more Availability Zones. Because AWS Elastic load Balancer lies in front of the EC2 instances, the targets in our scenario are EC2 instances. It monitors the health of its registered targets and only sends traffic to people who are in excellent health. As your incoming traffic changes, Elastic Load Balancing adjusts the load balancer. Elastic Load Balancers are available from Amazon Web Services in two different versions.

- Application load balancer – It's suitable for determining routing decisions at the application level, where HTTP and HTTPS traffic is handled. It also handles sophisticated request routing for microservices and containers, which are common in current application designs.
- Network load balancer – It's excellent for routing choices at the network level, where TCP, TLS, and TDP traffic are handled.

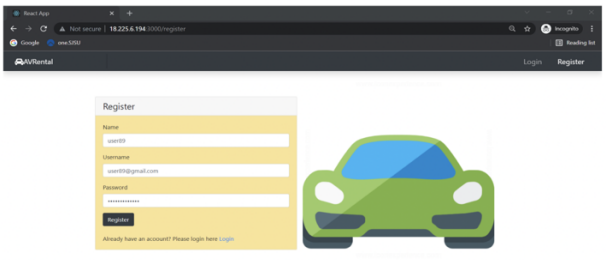
VII. SYSTEM GUI DESIGN AND IMPLEMENTATION

Users may utilize the GUI to book rides, manage their billing, and track users and automobiles. Multiple users can access the program, and these users are divided into two groups.

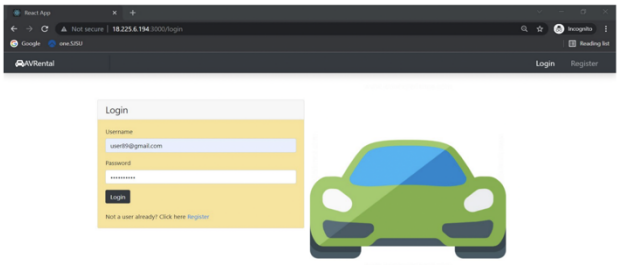
1. Admin User: Admin can track various parameters of the system. Admin can able view the number of registered users, number of registered vehicles. Admin can get a clear idea of how many cars of have an active subscription plan and admin can also know the status of the car linked to the system.
2. End User. End User is an user who can subscribe to the system and use the services offered by the system. End user can add vehicle, schedule a ride. End user can also view the rides history, vehicles added by him.

The application's user interface was created using multiple frameworks that were sewn together. HTML, CSS, and JavaScript are used to create the front end (ReactJS). The application's back-end is made up of NodeJS and the Spring framework, which are used to manage and expose data from the database to the front end. This program may be used on mobile web browsers as well as displays with greater resolutions, such as a laptop or a desktop. Below are the GUI pages:

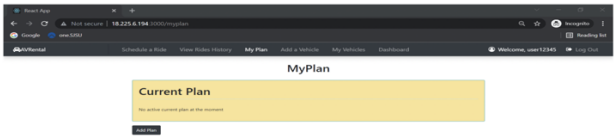
Registration Page: An admin user or an end user should register to the application to use the services of the application. To register with the application user needs to give the details like name, username, and password.



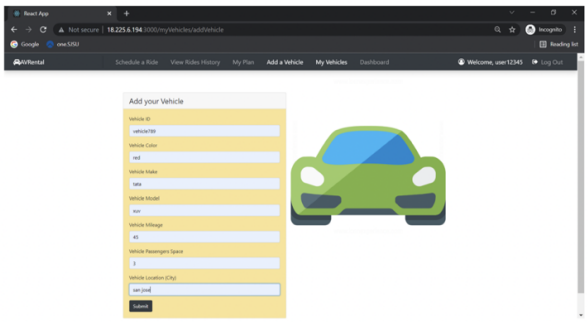
Login Page: If user is an existing user of the application, he/she can just login to the application by entering username and password.



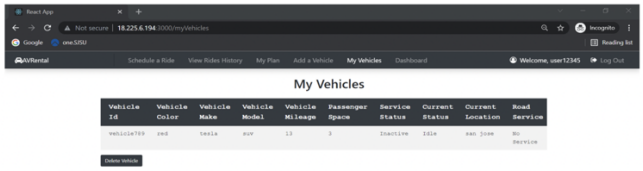
My Plan Page: User needs to subscribe to the application in order to user the services offered, for that user needs to add a plan by navigating to My plan page.



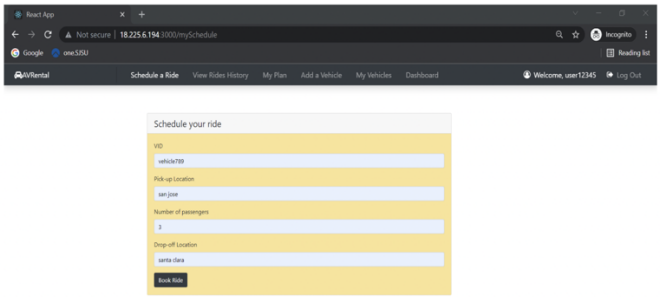
Add Vehicle: User can add vehicle by clicking on Add Vehicle button in the navbar. In order to add the vehicle to the application user needs enter the vehicle details like vehicle ID, vehicle color, vehicle model, number of passengers.



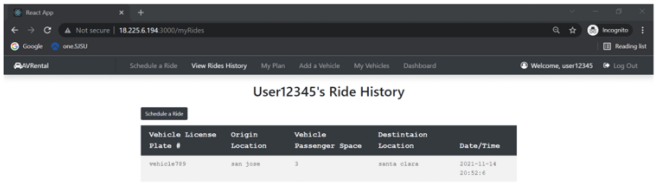
My Vehicles Page: My vehicles page shows all vehicles details that are added by the user.



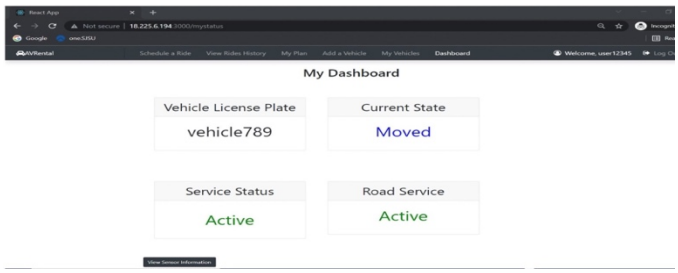
Schedule a ride page: User can schedule a ride by navigating to schedule a ride page. In this page user needs to give details like vehicle id, pick up location and drop off location.



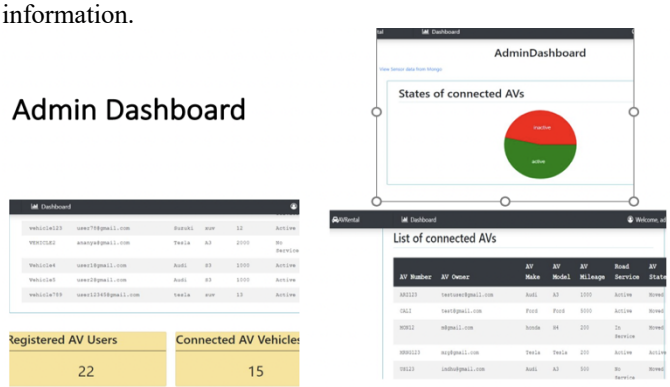
Rides History Page: User can view all his/her rides history by navigating to Rides History page.



Dashboard: Dashboard section shows information about vehicle license plate, vehicle status, Service Status, Road Service.



Admin Dashboard: It consists of information about the active and registered cars along with the user's information.



VIII. Edge Station Design and Simulation

The automobile is the main actor in our application. Using the Python API library offered by the CARLA environment, we attached numerous sensors to this actor, including a 360-degree RGB camera, a LIDAR sensor, a GNSS sensor, a Radar sensor, a Collision Detector, and a Lane Invasion sensor. This actor takes data from all of these sensors and transmits it to the car's internal computer, which controls the car's real hardware. It may be anything from directing the automobile to accelerating or decelerating using the accelerator or brake, as well as a variety of other car controls.

GNSS Sensor: The Global Navigation Satellite System is the most widely used equipment for vehicle placement on land, sea, and air (GNSS). The GNSS system, which is made up of a constellation of satellites orbiting roughly 20,000 kilometers above the earth's surface, offers a stable reference for the absolute location of a receiver. These transmit data about the spacecraft, such as its position and orbital characteristics. This framework comprises signal receiving systems that capture data like position, speed, and precise time.

LIDAR: LiDAR devices were initially created in the 1970s to quantify components in the water or on land using satellites or airplanes. It was created by the US Navy primarily for the purpose of identifying submarines. The time it takes for a pulsed light emitted by a laser diode to travel from the laser diode to an emitter is measured by LiDAR devices. The wavelength of the radiation is in the infrared range (905 nm or 1550 nm). Emissions at 905 nm require less energy than those at 1550 nm because water in the atmosphere begins to absorb energy around 1400 nm. Vehicle lasers are categorized as Class 1 and are completely safe under normal working circumstances.

IX. System Application Examples

The following is the user narrative for the application. All users have the ability to register and use the application through sign-up and login features. The JWT token library in NodeJS is used to encrypt the password field. A typical user can use our application by first logging in to the application by providing username and password. The user next can add a subscription plan and he/she can add vehicle to the account. User can schedule the ride by providing pick-up and drop-off location. When the user schedules a ride, the car that corresponds detects this and begins the journey. Once the ride starts. Information such as GPS location, temperature, etc. about the car can be monitored by the admin user on the web portal. Finally, a system administrator has access to all the automobiles in the system and can handle all of the users' contact information in the event of an emergency.

The automobile's status displays whether it is now being utilized by the car user or whether it is idle. The automobile might be in one of three different statuses.

- Active
- Moving

- Moved

X. System Performance Evaluation and Experiments

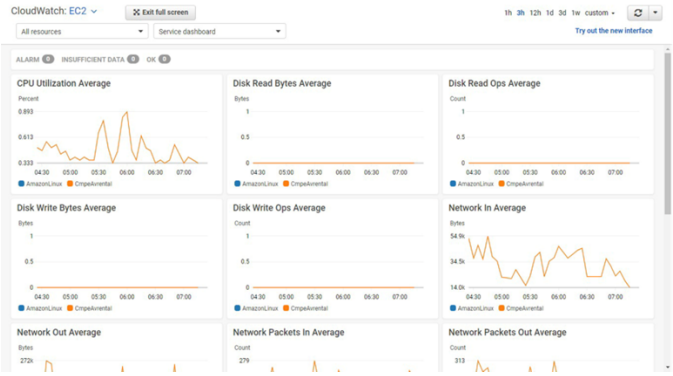
Software testing and quality assurance are critical components of the software development process. Both methods optimize the whole process to guarantee that the end output is of the highest possible quality. It also reduces repair costs and improves use and accessibility.

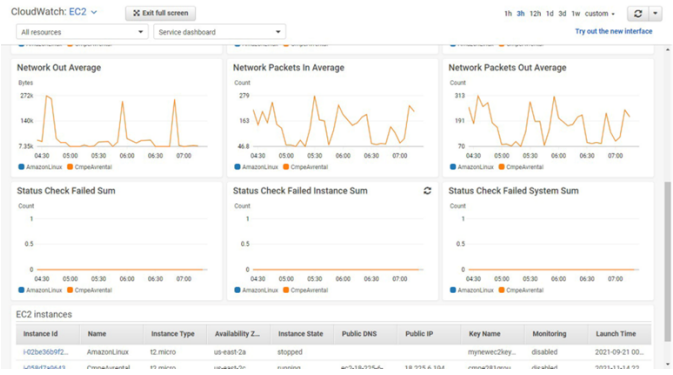
Functionality Testing: Although not complete, the following are some of the tests that are carried out: We have tested UI pages include registration, login, schedule ride pages and backend APIs and Databases by giving respective inputs.

Configuration verification: Once the application configuration is setup, we have verified that all the data is being passed correctly from the frontend to backend databases.

Performance testing: This is used to test the server's response time, throughput, CPU utilization and memory utilization under various load circumstances. We utilized Amazon's CloudWatch service to keep track of the above-mentioned tests and maintain the application's health.

AWS Cloud Watch Graphs





XI. Conclusion

Finally, our goal for this project was to create a system that allows a user to take advantage of a vehicle's idle time by scheduling trips for other family members and maximizing the vehicle's utilization. When a car is self-driving, it may travel from one location to another without the need for a driver. We learnt how to create an application to execute these activities using cloud technologies such as AWS for deployment in this project. We learnt how to design an AV cloud application, deploy it on the cloud, and use cloud technologies like load balancing and auto scaling to enable for scalability by executing this AV Cloud project. Our team learnt how to communicate with numerous databases, including MySQL and Mongodb, and how to integrate our frontend and backend together. We honed our skills in constructing Carla simulator. We learned how to set up Carla simulator to give a car and riding simulation. We also learnt how to add business logic to our application so that it could be utilized as a service. Finally, we learned how to set up an ec2 instance and configure it to meet our application's requirements.

XII. References

I. CARLA: An Open Urban Driving Simulator. *Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, Vladlen Koltun Proceedings of the 1st Annual Conference on Robot Learning*, PMLR 78:1-16, 2017.

II. Chong, B. Qin, T. Bandyopadhyay, T. Wongpiromsarn, E. Rankin, M. Ang, E. Frazzoli, D. Rus, D. Hsu, and K. Low. Autonomous personal vehicle for the first- and last-mile transportation services. In CIS 2011, 2011

III. Campbell, M. Egerstedt, J. P. How, and R. M. Murray. Autonomous driving in urban environments: approaches, lessons and challenges. Philosophical Transactions of the Royal Society Series A, 368:4649--4672, 2010