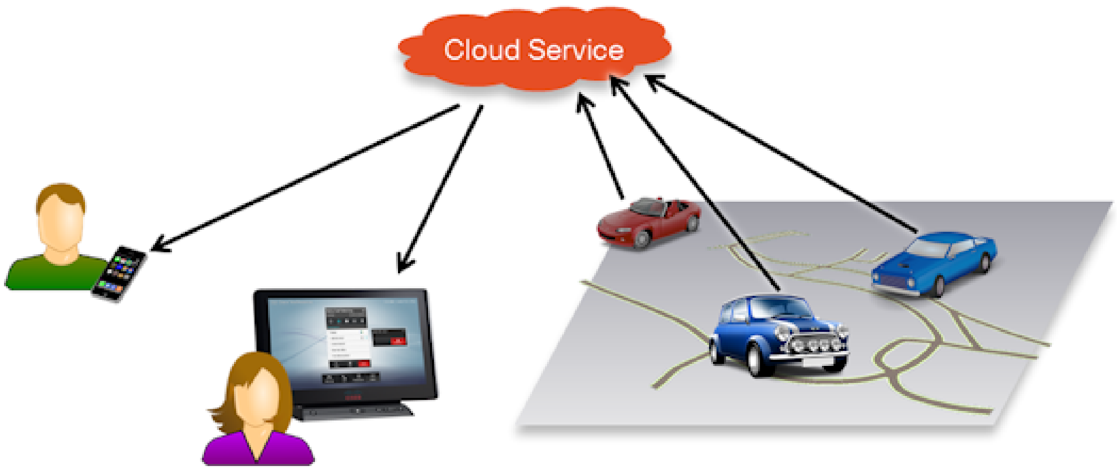
CHALMERS
UNIVERSITY OF TECHNOLOGY



# Resource Adaptive Cloud Services for the Connected Vehicle

A study of auto-scaling cloud for remote diagnosis of vehicles

Master's thesis in Computer Systems and Networks

BETSELOT HAILU ABEBE
GEBRECHERKOS HAYLAY ALEMAYOH

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2016

Master's thesis 2016:NN

# Resource Adaptive Cloud Services for the Connected Vehicle

A study of auto-scaling cloud for remote diagnosis of vehicles

BETSELOT HAILU ABEBE
GEBRECHERKOS HAYLAY ALEMAYOH

Department of Computer Science and Engineering
*Division of Networks and Systems*
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2016

Resource Adaptive Cloud Services for the Connected Vehicle
A study of auto-scaling cloud for remote diagnosis of vehicles
BETSELOT HAILU ABEBE
GEBRECHERKOS HAYLAY ALEMAYOH

Supervisor: KVS Prasad, Department of Computer Science and Engineering
Supervisor: Jonas Lohse, Cybercom Group
Examiner: Anderas Abel, Department of Computer Science and Engineering

Master's Thesis 2016:NN
Department of Computer Science and Engineering
Division of Networks and Systems
Chalmers University of Technology
University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Gothenburg, Sweden 2016

Resource Adaptive Cloud Services for the Connected Vehicle
A study of auto-scaling cloud for remote diagnosis of vehicles
Betselot Hailu Abebe
Gebrecherkos Haylay Alemayoh
Masters Programme in Computer Systems and Networks
Chalmers University of Technology

## Abstract

Cloud-based services are becoming more and more common thanks to the ever-growing cellular networks technology. There are several research works that aim at integrating cloud computing with different industries. The automotive industry is one of those areas that can benefit a lot from connectivity and cloud computing. Vehicular systems and cloud computing can be integrated to provide a safer and better driving experience. One of the areas that connected vehicles can benefit from cloud computing is remote diagnosis. Modern vehicles are made up of thousands of computing devices that work together. Troubleshooting and diagnosing these complex systems is not a trivial task as there are lots of components to diagnose and monitor. Currently, there are modern vehicles that store diagnostic data on a local hard drive and manufacturers use that offline diagnostic data to troubleshoot failures. Connected vehicles can leverage the increasingly fast and cheap mobile wireless networks to log diagnostic data so that it can be accessed and used my the manufacturers on fly.

Cloud services can help manufacturers diagnose and even maintain vehicles remotely without recalling them to automotive workshops. They can monitor the different components while they are working and examine how the components are operating. This help diagnose problems that are not even detectable by the drivers.

In order to realize the concept of connected vehicles, there is a need to have a robust cloud back-end system that adapts to data traffic from these vehicles. Vehicular data traffic is tends to fluctuate from place to place due to the mobile nature of the vehicles. Cloud instances that run in different geographical areas should be able to adapt to this changing nature of vehicular data traffic.

Vehicular cloud service, like the other cloud services, it faces a number of challenges that need to be addressed such as privacy, security, scalability and a lack of standards. This thesis work aims at studying one of the core challenges: scalability. There is a need to have an architecture which can accommodate scaling number of users (connected vehicles, vehicle technicians) and services.

There is extensive research work on autoscaling cloud systems that are not necessarily in the domain of connected vehicles. This work aims at taking advantage of these works to the area of vehicular cloud services. This is done by first building a prototype cloud back-end service and using it as a testbed to study autoscaling.

This work proposes a cloud back-end service design and implementation that is capable of communicating with vehicles and provide access to manufacturers. We also looked into existing algorithmic implementations that enable vehicular cloud back-end services to adjust their resource usage according to the encountered traffic loads. Stress tests enabled us to preliminarily evaluate the implemented algorithm's usability in the case of connected vehicles.

Keywords: Connected Vehicles, Cloud Computing, Remote diagnosis , Autoscaling

## Acknowledgements

We would like to thank our supervision team at Cybercom Group and Chalmers University of Technology for the realization of this project. We also thank the Swedish Institute for providing us with a scholarship opportunity to this Master's program. We also thank our families and friends for their moral support.

Betselot and Gebrecherkos November 2016, Gothenburg

# Contents

# List of Figures

# List of Tables

# 1 Introduction

Cloud technology brought promising advantages in solving re-usability, accessibility, integration, resource provisioning and modularity challenges in various Internet applications. The automotive industry could also be benefited from the new era of cloud computing technology to resolve its challenges. One of the main benefits of cloud technology is its resource provisioning capability. The scalability capability is the main benefit of the new emerging technology. This new feature of cloud computing adds a significant advantage to many Internet applications. With the support of cloud technology, Internet applications can scale up and down their resource usage as the demand of their users or client applications traffic load [1].

The concept of scaling up and down at real time demand is called auto scaling or self adaptiveness. We believe auto scaling property benefits many Internet applications on using their resources usage and accommodation of numerous clients. It supports them to scale up and down dynamically on real time demand. Once applications are deployed to a server in a cloud, the cloud service replicates the application onto more or few server machines according to its demand of resources. However, the cloud service could not provide infinite capacity on demand. The capacity of servers or data centers is also not infinite. But it is supported by virtualization technologies and it has a significant difference with the capacity of local servers. By the help of auto scalability feature, we can achieve high utilization of resources, flexibility, efficiency and reliability in cloud applications.

## 1.1 Background

Cloud-based services are becoming more and more common as network connections grow in size and quality. As a result, there are several research works that aim at providing cloud enabled services [2] to connected vehicles. Vehicular systems and cloud computing can be integrated to have a better and safer driving experience. This will result in increase in the number and type of cloud based vehicular services. However, vehicular cloud systems face a number of challenges that need to be addressed such as privacy, security, scalability and lack of standards. More cloud services also means that there is a need to have a cloud back-end platform to integrate and provide data access to these services. This indicates that there is a need to have an architecture which can scale depending on the number of traffic generated by users and services.

This thesis work aims at studying one major challenge that vehicular cloud systems could potentially face: scalability. We propose a cloud back-end architecture that is capable of communicating with vehicles and providing data to other cloud based services. In order to make the architecture scalable, we look into algorithmic designs and implement features that enable vehicular cloud back-end systems to adjust their resource usage according to the encountered loads. The prototype use case application of this system is a business intelligence data visualization application. This tool is aimed at making use of vehicular data to visually illustrate and summarize diagnostic information.

## 1.2 Project Scope

This thesis work is specifically targeted to play a significant role on the automotive industry and intelligent transport systems. Despite the fact that this project focuses specific sector, the dynamic scalability concept can be applied for various cloud systems due to its significant advantages on management of virtual resources.

The self adaptive cloud system provides various functionalities to the transport and automotive companies. It adds safety and ease of management values to the existing systems. As discussed in previous sections, the state condition and property of vehicles such as speed, temperature, engine and other vehicular data is stored in the cloud system. Therefore, by conducting data analysis in the system, it enables office users to know and diagnose the condition of vehicles.

Most importantly, this system will be used for maintenance and diagnostic of vehicles remotely. After conducting a thorough data analysis of the vehicular data stored in the cloud, it helps to take appropriate measures to fix or enhance the vehicles.

It also plays a significant role in safety control in the transportation system. It enables to identify vehicles' speed and their engine state. If a problem or unexpected behavior is observed in vehicles, then the specific vehicle will be called for maintenance. In addition, it controls drivers activity and makes them to drive safely. Example: if a driver drives with a high speed, which is not allowed in a city then he/she can be caught.
In general, the system brings safety and easy of management and maintenance advantages to the existing automotive and transport systems.

## 1.3 Project Organization

The project documentation starts with elaboration of the problem and the existing related works to make things clear for the rest part of the document content. The project is organized in to chapters, sections and subsections. The project is organized into the following chapters.

**Introduction**: This chapter introduces the reader about what the project accomplishes.

**Background**: This chapter describes basic concepts used in the project.It also clarifies the values that the project adds to the current cloud systems. The gap that the projects fills is explained in this chapter.

**Related work**: In this chapter, a literature review is conducted and relevant points are identified. Previous related projects and research works are discovered. We have also made analysis for some of the related works.

**Problem Formulation**: This chapter discusses the problem that is solved by this. It clarifies the goals of the project. The motivation for the need of the project is explained.

**Requirements, Specifications and Tests**: This chapter details the requirements of the system. The software specifications that are proposed in order to meet the requirements. It also explains the tests that are run to ensure that the system is working properly. Finally the error handling methods and fall-backs are explained.

**System Overview**: This chapter describes the system prototype at high level and its functionalities. It discusses the system components and how they are integrated to solve the stated problem. It also discusses how the system is operated when deployed in cloud environment.

**Prototype Implementation of the Vehicular Cloud Service**:This chapter give detailed explanation and technical details of the cloud back-end system and its components.

**Autoscaling**: This chapter gives an insight of the implemented autoscaling algorithm, the stress test setups, and different assumptions made.

**Results**: We specified test cases for both the components of the system and for end to end of the system. This chapter shows the outcomes of the different test cases of the system and its components.

**Evaluation and Discussion**: This part discusses about the performance of the system and its comparison with other existing related systems. The system is evaluated by its response time performance and its behavior in different traffic patterns. So, the details of these all is discussed in this chapter.

**Conclusion**: Finally, in this chapter the summary of the project is presented.

**Future work**: This chapter highlights the possible continuation tracks of this project.

**Annotations**: This appendix gives explanations of the reference literature cited and their relation with this project.

# 2
# Related Work

The increasing relevance of scalability in business applications over the Internet increased the number of researches in the area. Most of these research works deal with the scalability of applications in the cloud environment, development of cloud services for vehicular data analysis, auto scalable model for cloud applications and so on. These works describe theoretical aspects and suggest about different scalability architectures and scalability levels for various systems. Concepts and future works of these paper works will be used as input to achieve auto scalability in our cloud back-end application. The cloud back-end will be used for vehicular data analysis and decision making purposes in the automotive industry and transportation systems. The related works for this project can be categorized as follows.

## 2.1 Dynamic Scaling of Applications in the Cloud

Chieu et al. [3], investigates how dynamic scaling can be achieved on the web in a virtualized cloud environment. It describes the advantages of scalability such as higher resource utilization, resource consolidation, lower power usage, lower cost and storage saving to business applications in the cloud. It uses number of concurrent users, number of active connections, number of requests per second, average response times per request as scaling metrics. The values of these scaling indicators are collected in real time and a scaling decision is made. It suggests an effective and scalable architecture for scaling web applications in the cloud. Currently, web services are changing to cloud computing and hence resource scalability become vital[4]. In most cases, the cloud computing enables scalability by provision of virtual machines.

Cloud scalability has a great benefit for users and cloud providers in terms of efficiency,cost reduction and flexibility[5]. The purpose of cloud scalability is for provision of truly demanded resources[6].
Cloud computing plays a great role in providing a computational model that enables users to access various resources on demand [3]. The paper describes architecture used for scalability of web applications, based on thresholds in a cloud environment. It uses a front-end load balancer for balancing requests to a web application hosted in the cloud. This work also suggests the use of a virtual machine instances for the deployment of the web applications in the cloud. For automated provision of virtual machines, it recommends threshold based scaling algorithms based on the number of active user sessions.

The use of a scaling algorithm enables the cloud system provide users with resources resources on demand and also maintains its resource utilization. Cloud computing uses efficient virtualized resources that can be scaled up and down on depending on the number of users. The paper also discusses cloud resources such as storage, applications, business process and data protection. Cloud computing has changed the software industry in many aspects. It changed the way applications are purchased, licensed and deployed. All these operations are done over the network unlike traditional user desktop applications. The model of cloud computing has been classified into Infrastructure as a service (IaaS), Platform as a service (PaaS), and Software as a service (SaaS). These are the models of cloud computing services which need to be scalable to provide service for many users simultaneously.

## 2.2 Development of cloud services for vehicular data

The emerging Internet of Things (IoT) and cloud computing technologies have shown a promising opportunity in resolving the challenges in today's transportation systems.

The work by He et al. [2] proposes a novel multi-layered vehicular data cloud platform using IoT and cloud computing technologies. In this work, the authors developed two vehicular cloud services: a vehicular data mining cloud service and an intelligent parking cloud service. It also presents vehicular warranty analysis in the IoT environment. Data mining models for vehicular data mining cloud services such as the Naive Bayes model and a Logistic Regression model are also discussed. Modern vehicles are getting digitalized and equipped with smart computer systems. Today's vehicles have the capability of sensing, networking with other devices, communication with vehicles or other devices and data processing. The combination of these capabilities with the new emerging technologies caused a rapid advance in the automotive industry.

It is suggested that vehicular cloud development is feasible with the current technology and have a big impact on the automotive industry[2]. Software architecture for an intelligent parking cloud service is also described in detail in the paper. It uses transceivers and sensors for vacancy detection in a parking spot.

Paper by Akihito et al. [7] deals with how to build automotive cloud services based on various architectures. The details of the both (Service Oriented Architecture) and AUTOSAR (Automotive Open System Architecture) are described in the paper. It recommends developers to use these architectures as a development platform when they develop automotive software systems.

Data mining cloud services can also be used for accessing behavior and performance of drivers to figure our problems in advance. In addition, it quickly detects dangerous road situations, send warning messages and assist drivers to make decisions in advance [2]. The system also helps for vehicle warranty analysis. It explains that cloud services play a great role in developing vehicular data cloud services.

The main contribution of the paper is the proposal of software architecture for the vehicular data clouds in IoT environments. The architecture has the capabilities to integrate numerous components of a vehicle and other devices that can be integrated with a vehicle. From such works we observed that IoT and cloud based vehicular data clouds will be a backbone for future Intelligent Transport systems. This adds safety and enjoyment values to the transport systems. However, the integration of IoT with the vehicular data cloud is in its infant stage, it may take sometime for researchers to make it mature. Finally, the paper put scalability and technology integrations, performance, reliability and quality of services, security, privacy and global standard of service integration, communications, and architecture as future work.

## 2.3 Automatic Scalability

Since the emergence of cloud technology, development and deployment of software applications have changed entirely. As discussed in section 2.1, cloud systems rely on virtualization technologies for allocating resources on demand. Therefore, scalability plays a significant role in the success of business applications deployed in the cloud. [8] describes a virtual cluster architecture for dynamic scaling of application deployed in a virtualized cloud environment. The paper deals with an auto scaling algorithm that can be used for automated provisioning and load balance of virtual machines. It suggests an auto scaling algorithm that uses active application sessions as a scaling parameter. It also considers the cost of energy in the scaling up or down process. The suggested algorithm is supposed to handle sudden loads to the system, maintain high resource utilization and reduce energy consumption. Though the exiting cloud technologies can provide a basic scalability, it is not enough for enterprise business applications in the cloud. It is stated as many Internet applications are benefited from dynamic scaling service provided by cloud service providers [1]. This work mentions that the dynamic scaling functionality also provides fault isolation in addition to scaling resources. The focus of this paper was in addressing dynamic scalability on application tier though the storage servery may also be overloaded. They used virtual machines to to encapsulate instances of their applications.

Some scaling tools depend on control theory which also work with sensors. The sensors feed data for decision making modules to scale the application instances [9]. In this paper, user defined rules are used as a policy to control the scalability of system. It mainly focuses on scalability at application-level and it also describes the importance of network, container and storage scaling. It describes container scalability can be achieved by multi-tenant containers. Multi-tenant containers are containers which have the ability to run components which belong to different users.

Scalability, in many ways, is critical for the success of the cloud technology. Dynamic scaling allows resizing the number of server machines or other resources needed in the system [8]. Dynamic resizing is used to overcome under-provisioning or over-provisioning of resources in the cloud environment. The paper compares the dynamic scaling algorithm with the Amazon EC2 auto-scaling technology. Unlike the dynamic resizing, Amazon EC2 works for both horizontal and vertical scaling. The paper mainly addresses a dynamic-scaling algorithm with a design of applications deployed in clustered virtual machines. The architecture of the auto-scaling proposed by this paper is composed of the following components: Front-end load balancer, Virtual cluster monitor system and auto-provisioning system with an auto scaling algorithm. These components enable the system to achieve automatic scalability. The auto-provisioning system is used for dynamically provisioning virtual machines based on the active sessions in a virtual cluster. By destroying idle virtual machines, it reduces the energy cost of the system. Generally, the proposed algorithm is capable of handling sudden loads, high resource efficiency and achieves energy consumption reduction.

# 3

# Background

This chapter elaborates the basic concepts used in realization of this project, a brief of auto-scaling algorithms and methodologies applied in the project. Technologies used in realizing the project work are also briefly discussed. The project involves various interrelated concepts such as Cloud services, Automotive industry and Auto-scaling. This chapter is organized into the following sections.

## 3.1 Cloud Services

Cloud services provide computing services to users and applications while being hosted on a remote server as opposed to traditional systems that run ubiquitously on personal computers. This brings the advantage of scalability because of the potential of using a large pool of remote resources as stated in a research paper by Dua et al. [10].

There are three types of cloud service models in the cloud computing.

1. Infrastructure as a Service (IaaS): In this cloud model, virtualized resources are provided over the Internet.
2. Storage as a Service (SaaS): In this model, applications are hosted on the Internet and users get service from these applications through Internet.
3. Platform as a Service (PaaS): This model delivers software or hardware tools needed for application development via the Internet.

Recently, more and more companies are moving to the cloud technology to operate and scale their business applications. Cloud systems heavily use virtualization to allocate, use and isolate remote resources. However, in this thesis work, we will set up the cloud system prototype using light weight containers instead of the usual virtual machines.

The paper work by Containers are means of providing isolation, way of deployment and resource management while sharing the same base kernel in Linux [11]. Docker is a widely used daemon that enables Linux containers to be managed as self contained images. This is advantageous for our prototype because the action of auto-scaling and management of container images will be performed by communicating with Docker's APIs. Hence, using docker and containers in the project has a dual advantage for scaling and deployment purposes.

## 3.2 Connected Vehicles

The realization of reliable and low latency mobile connections attract interesting research on vehicular cloud computing. The introduction of 5G networks will bring about major changes in advancement of vehicular cloud services and intelligent transport services (ITS). The connectedness and digitization technologies are getting involved in various sectors. Vehicle products from various automotive companies are supporting embedded systems and getting connected to the Internet. Therefore, in this project we make use of these advantages to the automotive industry and use a cloud service to diagnose and maintain vehicles remotely.

The work by He et al. [2] proposes a novel architecture for vehicular data cloud services along with some possible cloud-based services. Vehicular cloud systems can be offered based on the three aforementioned cloud service models. This work solves and suggests the scalability challenge in vehicular cloud services. The mobility nature of vehicles makes the challenge hard. The cloud back-end system that supports remote vehicles need to handle a highly dynamic change in data traffic patterns.

## 3.3 Scalability of Cloud Systems

Scalability enables applications to release and acquire resources on demand. Auto-scaling simply automates this process because human intervention is impractical as the system grows in size.

In cloud systems, scalability can be classified as horizontal and vertical.

1. Horizontal scaling: This way of scaling works by replicating server machines. It adds more machine servers to the system according to the requirement and demand of the users or applications.
2. Vertical scaling: It achieves scalability by increasing the capacity of a server machine [10]. It does not add new machines rather it increases the capacity of the machines in use in the system.

In some use cases, the trivial scalability service provided by some cloud service providers is not adequate. Cloud service tenants want to consume the right amount of resources based on the need of their end users or the demand of their applications. Therefore, to achieve highly efficient use of resources, there is a need to apply dynamic scalability algorithms to applications and their back-end systems. The survey paper by Lorido-Botran et al. [10] reviews several auto-scaling techniques used in the cloud environment. According to this paper, auto-scaling algorithms are classified into five different categories. Each category of auto-scaling algorithm has got its own level of maturity, suitable use-case, metric specification and several other properties.

1. Threshold-based rules (Rules)
2. Control theory (CT)
3. Reinforcement learning (RL)
4. Queuing theory (QT)

5. Time series analysis based (TS)

Commonly used autoscaling techniques fall into two broad categories: reactive and proactive [10]. Reactive autoscaling has the capability to handle system loads by triggering scaling action when a condition satisfied. In proactive autoscaling,it has the capability of learning the trend of traffic loads overtime and forecasting about future loads on the system. Therefore, it adjusts the system according to what it has learnt previously.

These are the basic concepts applied in the project work. One can argue that auto-scaling cloud services are extensively implemented by commercial cloud service providers like Amazon. However, there are several reasons to study scalability and develop a prototype of a scalable system in a private cloud setup. One of the major reasons is the sensitive nature of vehicular data. Personal information and automotive manufactures should be careful about privacy which restricts the usage of public cloud solutions for this type of data. In our project, we use and modify open source tools in order to implement the cloud back-end and the prototype business intelligence application. Unlike public cloud systems, the system only considers vehicular cloud services, which adds a performance advantage to future production systems.

Literature studies such as [3] and [8] investigate on resource adaptiveness and scalability of cloud services. However, none of these works consider auto-scaling of a back-end system that provides service for vehicular systems. In this work, we worked on a vehicular cloud back-end architecture and study the specific feature of auto-scalability. Prior works do not take into account the layer of cloud management services, which are becoming basic building blocks of a cloud back-end system.

## 3.4 Containerization vs Virtualization

Both technologies are used for efficient provision of resources on a cloud environment. They add a significant role to the cloud technology in hosting applications.
A container is a very light weight operating system that runs on a host system. There are various container implementations such as Linux Containers, Docker, Warden Container, and OpenVZ. These containers implementations differ in the way they handle process, file system and namespace isolation. The various container implementations have their own pros and cons [11].

Virtual machines support in allowing multiple guest operating systems to run together on a single machine.
The two technologies are compared in terms of performance, isolation security, storage and networking factors. Based on the listed comparison factors, container technology has a significant advantage over virtualization in terms of resource consumption and less start-up time [11].

After carefully looking the two technologies, we decided to continue with container technology in our project. From the various containers implementations, we used

Docker technology due its easy resource management, network, CPU, file system and isolation capability.

## 3.5 NATS Messaging System

NATS is an open source messaging system which is mostly used in cloud and distributed systems. It has a client and server components that facilitate the messaging system of distributed applications. NATS' light weight protocol, performance, and support of various messaging models make it preferred over other messaging systems.

NATS supports various messaging models. Publish-Subscribe, Request-Reply and Queuing are the commonly used messaging models. The NATS Reference manual gives detailed information about this messaging system and its protocol[12].

# 4 Problem Formulation

Auto-scaling cloud services are being studied extensively in various literatures [10]. The problem this thesis tries to solve is not to come up with a brand new auto-scaling technique or algorithm; rather it is to investigate the existing auto-scaling algorithms and techniques and test one or more algorithms that are relevant to vehicular cloud back-end systems.

## 4.1 Scalability for Vehicular Cloud Services

It is obvious that the vehicular ecosystem comprises millions of vehicles. The number of modern cars also increases with time which means that we will have large number of vehicles that will be connected to the Internet. A vehicle is a complex machine that contains a number of computers, sensors, mechanical elements...etc. A vehicle's diagnostic data may become complex because of several components that need to be monitored. Sending this diagnostic data to the cloud needs two basic enhancements in the infrastructure: the mobile network infrastructure and the cloud back-end system. This thesis aims at solving the latter problem by building a prototype of the system and studying existing auto-scaling algorithms that fit the scenario.

## 4.2 Auto-Scaling Event Detection

There is a need to effectively detect when the cloud system should be scaled up or down. In order to sense the need accurately, it is important to have an accurate method of sensing and learning the current state of the system. This can include using several metrics and setting realistic thresholds.

Flawed auto-scaling detection mechanisms can lead to these undesirable results: over-provisioning or under-provisioning of resources. Both of these results cost the cloud company and the tenants. Over-provisioning of resources can affect the cloud company because large resources are allocated for a relatively small amount of load. Under-provisioning of resources affects the quality of the cloud service by creating congestion and latency. This thesis work tries to solve the problems of both under and over provisioning of resources. This is achieved by adopting an efficient auto-scaling system. An auto-scaling algorithm is the heart of the auto-scaling system as scaling decisions are made by it.

# 5
# Requirements, Specifications and Tests

This chapter briefly discusses requirements, specifications and tests of the system in general and the respective sub systems. Sometimes here is an overlap in clearly separating use cases, requirements, and specifications of the system and it is probable that a statement can serve as a requirement and specification. However, we make our best effort to distinguish and summarize them separately in the following section.

## 5.1 Requirements

The cloud back-end system is a collection of different subsystems that work together. There are a set of functional and non-functional requirements that each subsystem should satisfy. In addition to that, there are requirements that require one or more subsystems to interact with another, creating inter dependency among subsystems. Therefore a requirement can be belong to one or more of the following categories: functional, non-functional, subsystem level or system level. Requirements of the system in general followed by requirements of the subsystems are described below.

#### Vehicle Simulator

The vehicle simulator subsystem is the subsystem that generates and creates data traffic to the cloud application. The simulator is required to satisfy the following functional and non-functional requirements:

1. Vehicle simulator should generate random vehicular data in a well defined data model.
2. Vehicle simulator should serialize and publish the generated data to a desired topic on the *NATS server*.
3. Vehicle simulator should be able to run concurrent in multiple instances to imitate data traffic from several vehicles.
4. Vehicle simulator should be able to receive CAN commands from the cloud application.

#### Cloud Application

The cloud application is the subsystem that is responsible for processing, storing and accessing vehicular data logged remotely. It also has a unit that enables back office users to visualize the data. The following requirements must be satisfied by this subsystem:

1. The cloud back-end application should be able to handle multiple data traffic from several vehicles.
2. The cloud application should be able to maintain an acceptable amount of delay despite increase of traffic from users and vehicles.
3. The remote diagnosis tool should consist of a unit which provides visual presentation of one or more attributes of vehicular data from the database. (E.g., speed or engine temperature in graphic representation.)
4. The cloud back-end application should enable back-office users to retrieve vehicular data.
5. The cloud back-end application should consist of a database that stores remotely logged vehicular data.
6. Cloud back-end application should be able to run in multiple stateless instances. That is, all instances must be independent and should process the data in exactly identical way.

### NATS Server

The NATS server is the central broker which routes data back and forth between the *vehicle simulator* and the *cloud application.* It is a key unit that is essential for data traffic. The following requirements are expected to be satisfied by this subsystem:

1. It should route data traffic from vehicular simulator to the cloud application and vice-versa.
2. It should distribute a single data submission from a simulator to only **one of** the cloud application instances.
3. It should be able to handle maximum specified amount of concurrent data traffic without being a bottleneck.

### Container Manager

The cloud application in a multiple instances of independent units called containers. The container manager subsystem orchestrates any tasks related with the life-cycle of the underlying containers. The following requirements are expected to be satisfied by the container manager.

1. Container manager should be able to access container usage metric statistics like CPU usage, memory and number of concurrent connections.
2. The Container manager should add and remove containers via a remote command from the sensor-actuator subsystem.

### HTTP Load Balancer

The HTTP load balancer plays a role in evenly distributing HTTP traffic among the containers. HTTP traffic mainly originates from back-office users that query and modify vehicular data. The HTTP load balancer is expected to meet the following requirements.

1. HTTP load balancer should distribute traffic evenly among containers.

2. HTTP load balancer should be able to update configuration about the underlying containers dynamically.

**Sensor-Actuator**

This is the core subsystem responsible to the auto-scalability of the system. This subsystem communicates with other subsystems to gather data and sense the state of the system. The auto-scaling algorithm computes and determines the number of containers needed to accommodate the load. This subsystem carries out actuation by initiating scaling actions. The following requirements describe the tasks expected from this subsystem:

1. Sensor-actuator subsystem should be able to regularly collect the metrics for resource usage, load from vehicular data traffic, load from HTTP traffic.
2. Sensor-actuator subsystem should implement an algorithm that calculates the number of containers needed based on the different metrics collected from the other sub systems.
3. Sensor-actuator subsystem should be able to pass scaling decision to the container manager subsystem when necessary.

## 5.2 High Level Specifications

This section describes the high level functional specifications of the system. Functional specification illustrates how the system works from the user's perspective. It describes how the components must behave in order to satisfy the requirements stated in the previous section. Technical level and implementation specifications are found in the implementation chapter of this document. The more general system level specifications can also be decomposed into detailed subsystem level specifications. This section starts by describing the system level specifications then proceeds with stating subsystem level specifications. It also includes flowcharts that describe the possible workflows and with in the system.

### 5.2.1 System Level Specifications

The system level specifications give a general overview how the system works and what attributes it should have in order to meet the requirements.

The vehicle cloud service, at its maximum capacity, can support up to approximately 10,000 vehicles concurrently logging vehicular data remotely over the Internet. A single instance of 'vehicular data' refers to a well defined JSON structure that has fields to accommodate one time reading of current technical status of the vehicle's sensors time-stamped with the time of reading. Vehicles follow this model when formatting the data to send to the back-end cloud system. Back office users can access and visualize this data logged by vehicles remotely. The system provides RESTful API that gives read and write access to vehicular data.

The system can sense the density of vehicular data traffic at any given time. It can then use this input along with the resource usage statistics to decide the number of back-end containers that are required to accommodate the load.

This system runs on a single machine. The cloud application subsystem, which is responsible for processing vehicular data runs in multiple instances of containers. The number of required containers grows as the data traffic increases. The system shall have one common database for storing vehicular data. There will also be one instance of NATS server that routes all data between vehicle (simulators) and the cloud application.

### 5.2.2 Subsystem Level Specifications

This section decomposes the above system level specifications into more specific subsystem specifications. This clarifies the function of each subsystem and the role it plays in fulfilling the system requirements.

#### Vehicle Simulator

The vehicle simulator generates a vehicular data based on a configured pattern. The generated data is then serialized in the JSON based data model that is specified to represent vehicular data. The vehicle simulator can spawn multiple instances of data generators and senders in order to initiate multiple vehicles sending data simultaneously. The simulator uses the NATS protocol to send and receive messages with the cloud back-end system. Each instance of the simulator subscribes to a topic unique to itself in order to take CAN commands from remote back office users (via the back-end cloud application). In addition each instance of the simulator publishes vehicular data to a common NATS subject.

#### NATS Server

The NATS server is the center of data exchange between the vehicle simulator and the cloud application subsystems. When the NATS server receives vehicular data publication from an instance of the vehicular simulator, it forwards it to subscribers. Subscribers are instances of the cloud back-end application that process the data. Subscribers should register in the same queue group and subscribe to the same subject. When forwarding the publication, NATS server randomly chooses one subscriber among the queue group.

The NATS server also forwards CAN commands from the cloud back-end application to vehicle simulator instance. Here publisher is an instance of the cloud back-end application and subscribers are instances within the vehicle simulator. Every instance within the vehicle simulator subscribes to an individual topic and the message from cloud back-end application is targeted at exactly one instance. Since this is a one-to-one communication no queue group is required from the subscribers side.

The NATS server runs in an non-clustered single instance for simplicity. Thus one instance should be able to handle the maximum number of vehicles that the system is desired to support. It is worth to note that the performance also depends on several factors like the hardware capabilities of the system that runs this testbed.

#### Remote Diagnosis Back-end Application

As described in the requirements section, The remote diagnosis back-end application is responsible for processing and presenting vehicular data. This subsystem consists of several components that enables it to communicate with the back office end user and the NATS server subsystem.

#### Container Manager

The container manager is an application that is responsible for managing the containers throughout their life cycle. It is responsible for being the source of data for the usage metrics statistics of each container. It should also be to have command line interface commands or API's for automated creation of container. The creation or destruction of containers is initiated from the sensor-actuator subsystem. The container manager is also responsible for managing the memory, storage and network resources of each container.

#### Sensor-Actuator

This subsystem is where the auto-scaling decision is made. It runs one or more auto-scaling algorithms to carry out the decision. This subsystem polls resource usage metrics statistics and uses this data as input for the auto scaling algorithm which, in turn, computes estimates or even forecasts the number of containers needed to accommodate the traffic load. It then communicates with the container manager subsystem to initiate scaling actions when necessary. It is also responsible for updating the configuration of the HTTP load balancer when the scaling action affects the number of back-end containers.

#### HTTP Load Balancer

The HTTP load balancer is used to distribute HTTP traffic among the back-end containers that run the cloud application. It serves as a point of contact for the cloud application's HTTP service which is accessed by back office users. The load balancer updates configuration (data about the back-end containers) on-the-fly without service interruption.

### 5.2.3 Workflows

The three flowcharts in figure 5.1 depict the major workflows followed by the system. The left flowchart shows the process of end-to-end data exchange originating from vehicle simulator system. The second flowchart explains the interaction of the system with the back-office user. Note that the scope of each flowchart is system wide meaning that the processes in the chart can happen/execute in different subsystems.

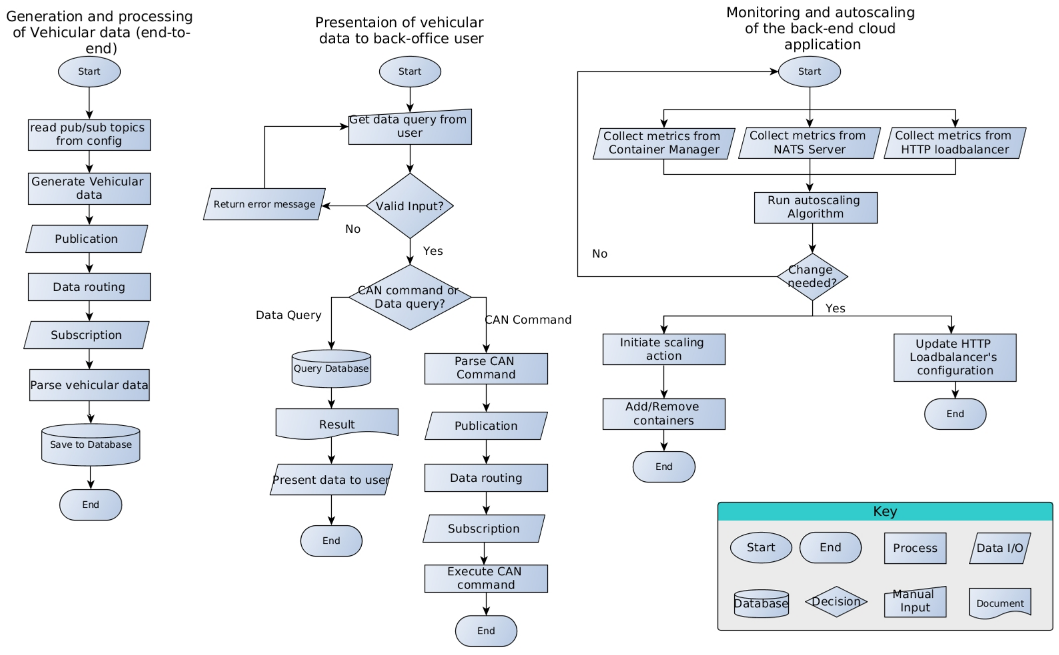


**Figure 5.1:** Flowcharts illustrating the different workflows followed by the system.

## 5.3 Tests

The system has defined features and behaviors expected after its successful deployment. To make sure that the system operates according to its specifications, we specified three test cases.

### 5.3.1 Generation and Storage of Vehicular Data

This is an end to end system test that shows the integration, performance and behaviors of the system. In this test case, the vehicle simulator is supposed to publish vehicular data to a topic on the NATS server as specified in a configuration file. The cloud app container instance subscribes to the same topic as the vehicular simulator and reads vehicular data. Then, the application parses the data and saves it to a NoSql database.

Initially, we planned to conduct this test case by using ten thousand vehicle simulator instances and ten application container instances on the back-end. This test measures the reliability of the end to end system. In addition, it helps to see the impact of the auto scaling feature in the system by handling thousands of concurrent traffics.

The specific objective of this test case is to check if the vehicular data generated by vehicular simulator is reliably stored in the database. The vehicular data is routed via NATS server and finally parsed by the logic application.

### 5.3.2 Data Presentation to Back-office Users

The system has also a feature that lets back-office users to query data from the application and get access to data according to their request.

In this test case, back-office users send an input data that can be query or CAN command on an user interface. The application checks the validity of the input data. If it is a valid query then data will be retrieved and presented to the user according to the query. If the request is a CAN command, then the application parses it and sends it to NATS server to a specific topic. Finally, the command is read and executed on a vehicle simulator container instance subscribed to the same topic. Otherwise if the user request contains invalid input, the application generates appropriate error message to users.

Hence, this test case is conducted by letting Back-end office users write a query or command on user interface. After sending a request, if it is a valid query, then it retrieves vehicular data from the back end system and present to the user in JSON format. If it is a command, it is sent to vehicular simulator. Otherwise, the system generates error to users. It tests the reliability and correctness of the data presented to back-end office users. Back-end office users should access a real data according to their request. If the request is not valid, appropriate error message is presented to users.

### 5.3.3 Monitoring and Autoscaling of the Cloud back-end Application

Auto scalability is a feature that makes the system to be self-adaptive as explained in the previous chapters. So, here we have a test case to observe the impact and reliability of the auto scaling feature in the system.

As explained in the specification, the auto scalability component starts by reading values of metrics to be used for scaling up or down from container manager, NATS server and HTTP load balancer. These values are feed to the auto-scaling algorithm. When the algorithm is executed, it decides if there is a need to scale up or down according to an initially set up scaling requirements or not. After a decision, if there is a change, it also updates the configuration of load balancer.

If there is a need of scaling, it performs the action of adding or removing container instances. Otherwise, it goes back and starts a new metrics reading to check if there is a need of scaling. This is done in an iterative way in a given small gap time.

So, this test case is achieved as subsystem test when we conduct the generation and storage of vehicular data test case. This specifically enables to measure the impact of the auto scalability feature on the system. We look at the response time and the capacity of handling thousands of concurrent traffics to the system.

The response and processing time of the system should be somehow stable regardless of a dramatic increase in traffic. Therefore, the system is supposed to operate with an almost constant response time to users due to the auto scalability effect.

## 5.4 Failures and Error Handling

In order for the system to carry out its desired workflows, all of the subsystems that involve in the workflow must function properly. However failures can happen to one or more of the subsystems thus, affecting the expected workflows. Some failures has far reaching system wide consequences while others affect only a certain functionality. In some cases proper error handling prevents a subsystem level failure not to propagate to other subsystems. Thus, it is important to point out some of the possible failures and the possible error handling actions. However this does not mean that this section covers all the failures that can happen to the system as the failures have different scopes. The following sections detail failures, their effects and the proposed handling methods.

**Vehicle Simulator**

**Possible Failure:** Generation of vehicular data in wrong data format.
**Effects:** Back-end cloud application subsystem will ignore the data and display an appropriate error message.
**Error handling:** None

**Remote Diagnosis Back-end Application**

1. **Possible failure:** Slow consumption of published data.
   **Effects:** NATS server subsystem detects slow consumers and discards them.
   **Error handling** Provision of more containers if resources allow.

2. **Possible failure:** Latency in shared database I/O (bottleneck) **Effects:** slow consumption/processing of data.
   **Error handling:** apply caching mechanisms to cloud application subsystem reduce to reduce frequency of disk I/O.

3. **Possible failure:** Badly formatted query from end users.
   **Effects:** The bad query is ignored.
   **Error handling:** Return appropriate error message and HTTP status code.

**Container Manager**

**Possible failure:** Failure in creating or removing containers.
**Effects:** Failure of auto-scaling system, over or under provisioning of resources.
**Error handling:**

### NATS Server

**Possible failure:** Intermittent or slow data routing.
**Effects:** Congestion, latency in transport of vehicular data.
**Error handling:** NATS server clustering or change in configuration.

### Sensor-Actuator

1. **Possible failure:** Miscalculation of number of required back-end container instances.
   **Effects:** Over or under provisioning of resources.
   **Error handling:** Modification of auto-scaling algorithm, change in the input metrics.

2. **Possible failure:** Failure to communicate with load-balancer
   **Effects:**over or under provisioning of resources
   **Error handling:** temporarily limit HTTP service to a fixed number of containers resulting in no auto-scaling of the data visualization service.

3. **Possible failure:** Failure to communicate with container manager.
   **Effects:** Failure of auto scaling, over or under provisioning of resources.
   **Error handling:** Critical to auto scaling system, requires troubleshooting of the sensor-actuator and container manager subsystems.

### HTTP Load Balancer

1. **Possible failure:** Failure to acquire correct or recent configuration about the back-end contaners.
   **Effects:** Complete or partial Interruption of HTTP service.
   **Error handling:** critical to HTTP service, requires troubleshooting of sensor-actuator and HTTP load balancer subsystems.

# 6
# System Overview

This chapter describes components of the system and their interaction to perform the functionalities of the system. The section starts with the explanation of system components and later it discusses the main functionalities of the system.

As vehicular cloud services advance, scaling up and scaling out back-end resources become increasingly challenging [2]. As it has been mentioned in the previous chapters, this work addresses the challenge of scalability in a vehicular cloud back-end system. It enables the system to handle numerous concurrent traffic connections.

## 6.1 System Architecture

In this work, we designed and developed a prototype a cloud back-end system that is capable of auto scaling dynamically as per the load requirement of the users or clients of the system. As it can be seen in the system architecture in Figure 6.1, the system is composed of several components. The following are the main components.

1. Vehicle simulator: We use this component to simulate a vehicle. It generates a vehicular data to the back-end cloud system. After reading papers related to vehicular data, we defined a data model to be used in the system. Hence, the data generated by the vehicle simulator is expected to comply with the defined vehicular data model. Otherwise, the system discards the data if it is not according to the defined data model.

   This application is run after both the NATS and the back-end cloud service application are ready to receive the generated data. The NATS server and the back-end cloud service application should be run before generating data otherwise the data will be lost. The data generator doesn't guarantee about the arrival of the generated data. NATS is made to behave like that to decrease the communication time and to keep it as light as possible.

2. Visualization application: This component interacts with the cloud back-end service. It enables back-office users to view logged vehicular data in the cloud system. Back-office users may be interested in specific type of vehicular data, this component presents that data to them. It is simple and easily accessible for them.

The back-end office users will visualize and analyse the vehicular data from the cloud system. Analysis of the vehicular data can be used for decision making, safety and maintenance purposes. The system adds values to the existing automotive industry and transportation system.

3. NATS Broker: Nats is a messaging system which is designed to be used for distributed and cloud systems. It supports messaging models such as Publish - Subscribe, Request - Reply, and Queuing. In this project, we use the Publish - Subscribe messaging model. The vehicular simulator publishes to the NATS server to a specific topic and generates vehicular data to that topic. The cloud back-end application subscribes to one of the topics in the NATS server to receive vehicular data. When the Vehicle simulator generates vehicular data, it is routed via NATS server to the back-end application.

4. Cloud back-end system: This is the core of the system that logs the vehicular data generated by the vehicle simulator. This system serves for enormous vehicles and back-office users. Back-office users use this component to view vehicular data.

5. HA-proxy Load balancer: This technology is used for making a fair distribution of the incoming data traffic to the cloud back-end container instances. It brings efficiency and flexibility to the system. It is a standard used by many cloud applications.

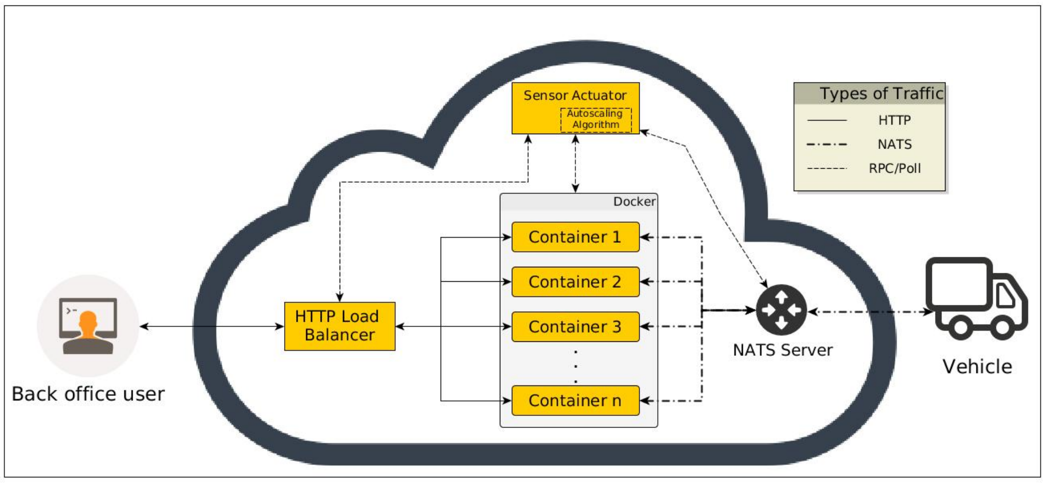


**Figure 6.1:** System Architecture

## 6.2 Sensor and Actuator Module Architecture

In the cloud back-end system, an auto-scalability algorithm is implemented and integrated to make it self-adaptive as per the resource requirement of the system clients. In addition to building the prototype application, the work also involves implementing auto-scaling algorithm in the Sensor and Actuator module. The major benefits of this component are achieving low latency in data transmission between the vehicle simulator and the cloud back-end system, making the system efficient, flexible and reducing cost.

When the vehicle simulator and back-end users interact with the cloud system, its response time is slightly independent of their number. The auto scalability algorithm plays a significant role in reducing the transmission delay by scaling up or scaling down the computing resources in the cloud back-end system on demand. In the system, behavior, performance and impact of the algorithm is investigated under different traffic load patterns.

In the operation of the system, scaling decisions are carried out by the auto scaling algorithm. Generally, the scaling events include *creating a new instance of virtual machine*, *provisioning newly created virtual machine instances*, *destroying idle instances of virtual machines* and *updating load balancer configuration*. The preliminary metrics that are used to determine the resource usage of the virtual machines are *CPU usage* and *the number of active connections*. After the scaling event is done, same logic is repeated again to monitor the system's well being. This method of auto scaling is based on on-line decision making because the algorithm may fire up scaling events as the system encounters load changes.

In addition to on-line decision of scheduling events, we implement an algorithm that works based on a decision that is made offline. This is good for preventive maintenance of the system in scenarios where the traffic change is known in advance. This method of scalability is time-based auto scaling. In this mode, the algorithm does not monitor resource usage of the virtual machines that run the back-end application. Rather, it fires up scaling events based on a specific schedule which follows a distribution function or human input. For instance, the schedule may dictate the algorithm to scale down the system by a factor of 2 during weekends. In offline mode, the algorithm will calculate the number of virtual machines to add or remove and carry out the tasks accordingly. In general, this testbed is expected to scale up to ca. 10,000 (depending on the hardware limits of the system) maximum concurrent connection without affecting the message transmission delay. Stress testing is conducted in a condition where the system is saturated.

We make use of Docker, which is an open platform that supports to build, run and deploy cloud applications. In this work, Docker and its components plays a significant role in the deployment of the back-end application. Instead of virtual machines, the Docker uses Containers. Containers are light and portable software images and their dependencies. The Sensor and Actuator module interacts with the Docker to collect resource usage metrics from Containers. This module also interacts with Docker, which orchestrates the creation, provisioning and destruction of containers.

## 6.3 Cloud Application

This prototype contains a cloud application that is capable of remote diagnosis of vehicles and vehicular data presentation. The software architecture of the prototype diagnosis application is shown in Figure 6.2. This application demonstrates collection of diagnostic data from vehicles and serves as a testbed to test the auto-scaling environment. The development of this application demonstrates an example use-case of a cloud service for connected vehicle. This prototype application's main functionalities are categorized based on the following use-cases.

- Handle connections from vehicles
- Serve back office users with data visualization
- Simulate multiple vehicles, generate syntactic data
- Store and access logged data
- Generate different data traffic patterns, test and measure system performance

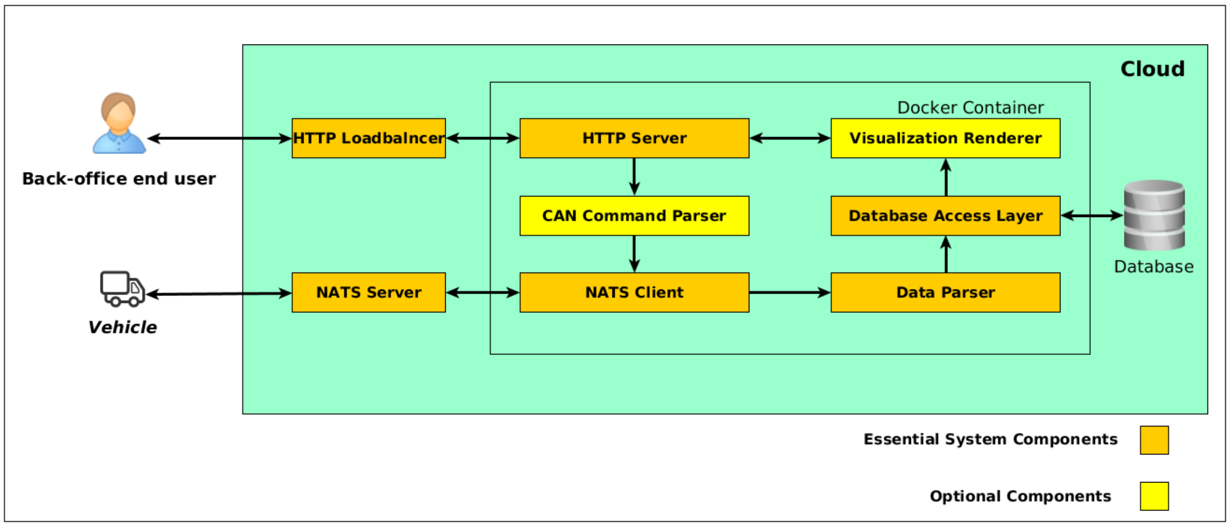


**Figure 6.2:** Back-end cloud system design

As per the need and the practical events, different components of the application may be required to run on different virtual machine instances in order to achieve better scalability. This application prototype must also be designed in such a way that it runs on clustered scalable environment.

# 7 Prototype Implementation of the Vehicular Cloud Service

This chapter describes in detail the development of the testbed. As described in the previous chapters, this prototype is a composed of different subsystems that work together in order to meet the requirements.

At a high level, the system the client-server architecture where the clients are the vehicles and the back office users. Vehicles continually send their technical status. Back office users can access and modify vehicular data via the provided APIs. The cloud service, in general, plays a role as a server.

The following sections explain the technical details, the assumptions made and the technologies used in this implementation of the prototype system. For better understanding of the system, the sections are organized based on client and server roles of the system.

## 7.1 The Clients: Vehicle Simulator and the REST Client

The clients of this system are mainly involved in production and consumption of vehicular data.

The Vehicular subsystem generates vehicular data while the REST Client (back-office user consumes vehicular data and produces CAN commands. The following section describes the technical details behind the vehicular simulator and the REST Clients.

### 7.1.1 Vehicule Simulator

The vehicule simulator is a vital part of this prototype. As discussed in the previous chapters, this subsystem is responsible for generation of synthetic vehicular data. This data is then passed to the cloud back-end system and becomes stored persistent database.

The modifications made to the open source tool are enumerated below:

**Table 7.1:** Vehicular data model

| Category | Attribute | Data Type | Description |
|---|---|---|---|
| Main | Vin | string | Vehicle identifier, must be unique for a vehicle |
| | time stamp | date | time stamp of the reading |
| Telematics | Speed | double | the current speed of the vehicle |
| | latitude | double | GPS location reading |
| | longitude | double | GPS location reading |
| | heading | string | The direction pointed by the front of the vehicle |
| Engine | rpm | double | the rotation speed of the engine shaft |
| | temperature | double | reading of the temperature sensor inside the engine |

1. **NATS broker support:** The data generator did not support logging to a NATS server. Because NATS is the chosen messaging protocol between the vehicle and the cloud back-end system, NATS client is incorporated into the tool. The coming sections discuss more about the NATS messaging system.
2. **Support for simulating multiple vehicle instances:** The generator supports multiple instances that can be programmed in a certain way, with an analog yof vehicles having different data sets. The draw back of this approach was that each instance's (vehicle's) configuration needs to be defined in a configuration file even though the instances have similar data sets and generation patterns. This is not scalable as it is impractical to write thousands of individual configuration files. We modified the source code so that it supports multiple instances from one configuration.

The following sections discuss about the vehicular data model and data generation.

#### 7.1.1.1 Vehicular Data

In this project, the term *vehicular data* refers to set of well defined attributes that represent readings of the different sensors available in the vehicle. In this prototype, a minimal set of attributes are chosen for simplicity. A single reading contains the vehicle identification (VIN) number and one or more of the attributes belonging to sensor readings. The attributes are organized into categories for better understanding.

There are data types and acceptable ranges for each attribute. For example, a speed reading is not expected to contain Strings but floating point numbers. The structure or order of these attributes do not matter as the data adopts a NoSQL Document model. Table 7.1 shows the data attributes and their description.

#### 7.1.1.2 Data Generation

The vehicular data is modeled in JSON format. We customized an open source JSON-based data generator tool [1] to fit the requirements of the system. The generator is generic and supports different data types and data generation patterns (Data generation patterns are one of the factors that we use to evaluate the autoscaling

[1]https://github.com/acesinc/json-data-generator

```json
[{
    "timeStamp": 1470776087939,
    "vin": "00004",
    "telematics": {
        "speed": 3.7867,
        "heading": "SW",
        "latitude": 57.3468,
        "longitude": 11.9076
    },
    "engine": {
        "rpm": 1157.1274,
        "temperature": 45.0186
    }
}]
```

**Listing 1:** A sample vehicle state update with all attributes present.

techniques).

The simulator is configured to generate synthetic sensor readings with the attributes' respective data type and ranges of the attributes. The generated data usually contains random values of the attributes within the acceptable range. The tool can also be configured to generate vehicular data that can resemble real readings such as a gradual increase in speed of the vehicle and the vehicle following the GPS coordinates of a particular road. The frequency of data updates is also configurable. The actual frequencies of update are stated and justified in the next chapters. Listing 1 shows a sample data generated by the simulator.

## 7.2 The Server: Cloud Service

The cloud service is the backbone which supports the clients described in the previous sections. The vehicles connect to this service to log their data and the back office users connect to this service to retrieve and update vehicular data. At a high level, the cloud system consists of a distributed cloud application, a messaging system server, an HTTP server and an autoscaling unit. The following sections describes the implementation details of these components.

### 7.2.1 Messaging System

Messaging system plays a big role in applications that involve communication among distributed components. Cloud and Internet of Things applications usually contain messaging systems at their heart. The NATS messaging system is part of this cloud application. The messaging system plays a big role in communication of the vehicle simulator with the cloud service.

As stated in Chapter 3, NATS protocol implements the publish-subscribe and message queuing models of communication. In this project, we leveraged these models for bidirectional communication between vehicles and the back-end system.

All instances of the cloud application subscribe to the same subject and register a queue name. As the vehicles publish data to that particular subject, the NATS server randomly chooses one member of the group and forwards the message. This way, the NATS server acts as a load balancer on the vehicle back-end line of communication.

Every vehicle instance is represented by a thread that generates a unique synthetic data.

### 7.2.2 Back-end Cloud Setup

The cloud service that supports the connected vehicles consists of an array of Docker containers running a Spring application. The concept of containerization is discussed in Chapter 3. Docker containers are light weight instances that provide isolated environment to run application. In this setup, each container runs only one service which is the vehicular cloud prototype application. It is also worth noting that the application is a spring java application and the container includes all the dependencies and libraries needed to run the application in isolation. This makes containers light weight and faster to provision compared to virtual machines. Fast provisioning is an important property for autoscaling systems which brings usage of docker containers to an advantage.

#### Imposing Limits on Resource Usage

There is a need to effectively limit and monitor the system resources accessible by each container in order to manage multiple containers running on a single host. Otherwise all containers assume to have remaining free system resources for them and this creates a congestion when the load is high. The Docker engine has different mechanisms to allocate a limited amount of system resources for each container.

In this implementation, we imposed limits on CPU quotas and memory usages for each container. This is because these two are the resources that are relatively scarce in our test machine compared to other metrics, for example, disk usage and network bandwidth. These metrics are used in monitoring how the containers are loaded and thus, are crucial inputs for the autoscaling algorithm.

The limits on CPU is imposed as Docker engine's `cpu-quota` feature [13]. The Linux kernel uses Completely Fair Scheduler (CFS) to allocate CPU time quotas to all processes. CFS implements CPU bandwidth control for process groups or hierarchies. This is achieved by changing the values of the `cpu.cfs_period_us` and `cpu.cfs_quota` attributes. The CPU quota attribute represents the available runtime within a scheduling period while the CPU period attribute indicates the length of a single scheduling period [14].

We specified a CPU-quota of 33ms while leaving the CPU period to the default 100ms. We imposed these limits after repeated trials to tune the resource allocated to the containers. Each container running the back-end application is allowed to run for a maximum of one third of the scheduling period. We decided to specify this quota limit by using stress tests and overloading the system to its limits and fine tuning the CPU quotas such that the application containers do not take all the server's resources.

### 7.2.3 Back-end Application

The back-end cloud application is responsible for providing service for the vehicles and back office users. This prototype application is developed on Java's Spring Boot Framework. The Model-View-Controller architecture was followed when implementing the application.
The application is designed to be identical or session independent. This means that a client (vehicle or back office user) can connect to any of the running instances of the cloud back-end application and it gets the same service regardless of previous states or sessions.
The application roughly comprises of the following parts:

1. NATS Client: The NATS client for Java is used to enable NATS connections for Java applications. In this application the NATS Client is responsible for exchanging data with the NATS Server subsystem, which handles all connections with the client application.
2. HTTP Service: The prototype application incorporates an embedded Tomcat HTTP server. The HTTP service gives access to vehicular data access and modification via the designated REST APIs endpoints.
3. Data Service: No vehicular data is stored inside the Docker container running the application as it creates inconsistency and containers running the cloud application can be added or removed via auto scaling actions causing data inconsistency. Thus, it was important to setup a database server which is accessible from every container instance. MongoDB, a document-based NoSQL database management system. NoSQL is chosen because the data from the vehicles may not always be structured and contain all the attributes needed. It is also more important to have speed than data accuracy when logging diagnostic data for vehicles.

This Chapter described the technical implementation details, assumptions and design choices made during the development of this prototype.

# 8
# Autoscaling

It is becoming increasingly common for cloud companies that to make solutions elastic [1] and scalable. This is because of constant changes in the amount of resources demanded from these systems. Manual tuning can be practical in small application contexts but it is not practical in systems that have large number of users over a geographically distributed area. This elasticity or auto scalability feature enables customers to only use resources that they need and to pay for what they have used. More importantly, autoscaling adjusts resources demand according to incoming traffic and it brings efficiency, flexibility and cost reduction. So, both customers and cloud service provider get benefit from it.

In this project, the autoscaling algorithm is used for increasing and decreasing the number of docker containers running the cloud back-end application. The application is deployed in a private cloud setup with limited resources. Making this service self adaptive or autoscaling, promotes flexibility and efficiency for use in building private clouds setups that connect to numerous vehicles.

The implemented algorithm is a slightly modified version of the autoscaling algorithm by Chieu et al. [3]. The algorithm uses threshold rule-based theory for initiating scaling actions. The algorithm is modified to fit into this use case's scaling indicators and thresholds.

The implemented algorithm uses reactive autoscaling technique, which means that action is taken after a change is sensed in thee scaling metrics. However, we tuned the thresholds and scaling metrics generous margins so that the data traffic will not affect the system before the algorithm acts. The upper threshold of 80% and lower threshold of 20% are taken as points to initiate scaling up and scaling down actions respectively.
The algorithm periodically polls the traffic load on the system and the resource usage of the system and matches them. Algorithm 1 shows the main work flow of polling and scaling actions along with the different metrics used, it is described in the remaining sections of this chapter.

## 8.1 Stress Test Setup

As discussed in Chapter 6, the prototype system is mainly composed of a vehicle simulator (client) that generates vehicular data, message broker (NATS server) and a back-end application that receives and processes and stores data.

In this context, the vehicular simulator and back office users are considered as clients. However, the traffic load from back office users is too small compared to the one from vehicles. Thus, only the NATS traffic from vehicular simulators is considered as scaling metrics for this algorithm.
As discussed in Chapter 7, limits are imposed on containers running the back-end cloud application. Taking the physical machine's system specifications and our goal of incorporating thousands of messages per second, we set the memory usage limit of 1.5 GB and CPU quota of 33ms CPU time for each container that runs the cloud back-end application. This enabled us to know the capacity of each container in the specified limited resources.

The script that runs the autoscaling algorithm polls these scaling indicators from the monitoring built in APIs of the message broker and the container manager. The imposed limits on system resources clearly tell us how much resource the containers can user at maximum allowable limit. Therefore it is possible to take a ratio of the current usage to the maximum amount of usage and calculate the scaling factors. These ratios are used as inputs to the autoscaling algorithm.

The stress test involved generating synthetic traffic from machines that run the vehicular simulator and record how the cloud back-end system reacts to different amounts and patterns of vehicular data traffic.

In the NATS subsystem, the client (vehicle simulator) is considered as publisher and the cloud application container a subscriber. Different instances of the application form a queue group and the NATS server distributes published message to one of the subscribes in the group.
The vehicle simulator is configured to publish one message per second per vehicle. This assumption is taken considering worst case scenario in case frequent data logging is required from the vehicles. This rate of message generation fails when ever the system is overloaded. So, when the system understands that rate is not as per the initial setup, it scales up system. After scale up, the rate of message generation goes back as per the initial setup and the system stays in its safe state.

## 8.2 Scaling Indicator Metrics

Autoscaling algorithms use different scaling metrics to as indicators of the system's state. The autoscaling algorithm implemented in this project uses *container CPU usage* and *number of messages per second* as scaling metrics. Different error messages from the message broker and overall CPU usage were also used to determine he limits of the system. The algorithm takes the values of these metrics to initiate the events of scaling up or down. The values of these two metrics vary in accordance with the data traffic load on the system.

Container memory usage is not used as scaling indicator for this particular implementation because the memory usage of the spring applications barely varies with varying traffic load. This could be due to the Java Virtual Machine's and Spring framework's internal architecture. The physical machine used for this setup also has ample memory to accommodate a number of application containers running the cloud back-end application. However, we ran the containers with an imposed memory limitation for the containers to make sure that they are isolated systems with limited resources.

## 8.3 Algorithm Operation

Autoscaling algorithms make decision for allocation of system resources for virtual machines or containers.The algorithm implemented for this project is threshold rule based which is capable of handling unexpected load in the system. The algorithm starts with running two application containers. The reason why it starts with minimum two containers is for redundancy and fault tolerance purposes. Then, the algorithm checks the resource usage of each container and triggers the scaling action.

Here is the modified autoscaling algorithm used in the project.

```
input  : $S_{mm}$ = maximum messages per second per container
         $S_{cm}$ = maximum container CPU Usage
         $N_{instance}$ = number of running containers
         $N_{max}$ = Maximum number of containers
         $A_c$ = container_cpu_usage
         $A_m$ = number_of_messages_per_second
         $T_{upper}$: The upper threshold of use of resources
         $T_{low}$: The low threshold of use of resources
output: Scaling Action
while True do
    for $i \in N_{instance}$ do
        if $max((A_{ic}/S_{cm}), (A_{im}/S_{mm})) > T_{upper}$ then
            Increment $N_{\text{Exceed}}$
        if $min((A_{ic}/S_{cm}), (A_{im}/S_{mm})) < T_{lower}$ then
            Increment $N_{\text{Below}}$
    end
    if $N_{Exceed} \geq N_{Instance}/2$ then
        ScaleUp ()
    if $N_{below} \geq N_{instance}/2$ then
        ScaleDown ()
    end
```

**Algorithm 1:** Autoscaling algorithm

The pseudo code snippet in Algorithm 1 describes the details of the autoscaling algorithm used in the system. The steps taken during scaling events are described in Algorithm 2 and Algorithm 3. The maximum and minimum thresholds of messages

```
input  : $N_{\text{instance}}$ = number of running containers
output: Scaling up action
Scaling methods
Function ScaleUp($i \in N_{instance}$):
    if $N_{instance} \geq N_{max}$ then
        Return "physical limitation" ;
    end
    else
        Start (i) ;
        Update load balancer(i) ;
        Increment $N_{instance}$ ;
    end
```

**Algorithm 2:** Scaling up action

```
input  : $N_{\text{instance}}$ = number of running containers
output: Scaling down action
Function ScaleDown($i \in N_{instance}$):
    if $N_{instance} \leq 2$ then
        Return “min container limit” ;
    end
    else
        Stop (i) ;
        Update load balancer(i) ;
        Decrement $N_{instance}$ ;
    end
```

**Algorithm 3:** Scaling down action

per second per container and CPU usage should be set before the algorithm runs. These initial values are set after various repeated tests in the system. The system has a set of ready-to-run containers from which the algorithm can pick and run when scaling up is triggered.

The algorithm regularly polls the values of scaling indicators, CPU usage and messages per second, of every running container. Then it calculates the ratio of the collected CPU usage and messages per second values of each container with maximum messages per second and maximum CPU quota respectively. If any of the ratio values of both metrics is greater than the maximum threshold set in initially, then the scale up action is triggered.

The maximum number of messages per second ($S_{mm}$)for a container running on these constraints is about 2500 messages per second. The maximum CPU usage at this maximum working capacity ($S_{cm}$ is about 32%. These are the denominators to calculate the ratio values for the scaling indicator metrics, as described in the snippet in Algorithm 1. The next chapter has more details about these numbers.

When any of the ratio values of both metrics is less than the minimum threshold, then scaling down action is triggered in the system. Otherwise, the system is considered to be in a stable state and it does not need scaling. When the scaling up action is triggered in the system, the algorithm adds a new container to the system. The newly added container joins and shares the traffic load of the running containers. When the algorithm triggers a scale down action, one of the running containers is stopped. The load of the system is shared among the remaining running containers.

When the algorithm triggers a scaling action, it always updates and restarts the HTTP load balancer. In scaling up action, a new running container is added to the list of running containers in the load balancer. If scaling down action is triggered, a container is removed from the load balancer's list and the load balancer is restarted. When the system does not require scaling, the load balancer stays intact. It operates with the containers which are already running in the system.

# 9
# Results

## 9.1 Single Container Response

As described in the previous sections, containerization is used to create multiple instances of the cloud back-end application. The system needs two or more container instances to provide the expected service. There is a need to know the capacity of a single container since it enables to estimate required number of containers for a given size of vehicular traffic.

A single container is tested against various sizes of vehicular traffic. As plotted in Figure 9.1, a single container handles a maximum of two thousand five hundred concurrent vehicular traffic. As shown in graph one, for that maximum vehicular traffic, the CPU usage of a single container is thirty five percent. The reason behind this maximum CPU usage is, each container has limited CPU quota. This enables to estimate the resource usage of every running container in the system.

## 9.2 Response to Different Traffic Patterns

Tests under this section enable to know the behavior of the system under various traffic patterns. The system is tested under three kinds of traffic patterns.

### 9.2.1 Sudden Traffic Pattern

The first test case is sudden traffic pattern which suddenly increases or decreases the vehicular traffic in the system. The aim of this test case is to know the number of needed containers in the system in sudden change of the vehicular traffic (message per second). The required number of containers for a given size of vehicular traffic is shown in Figure 9.2 The number of required containers in the system varies with the size of the traffic. A single container has the capability to serve for around two thousand five hundred size of vehicular traffic.

### 9.2.2 Random Traffic Pattern

In this test case, the aim is to generate a random load of vehicular traffic to the system and observe the behavior of the system. It helps to know the system's requirement of containers under such traffic pattern. Figure 9.3 shows the number

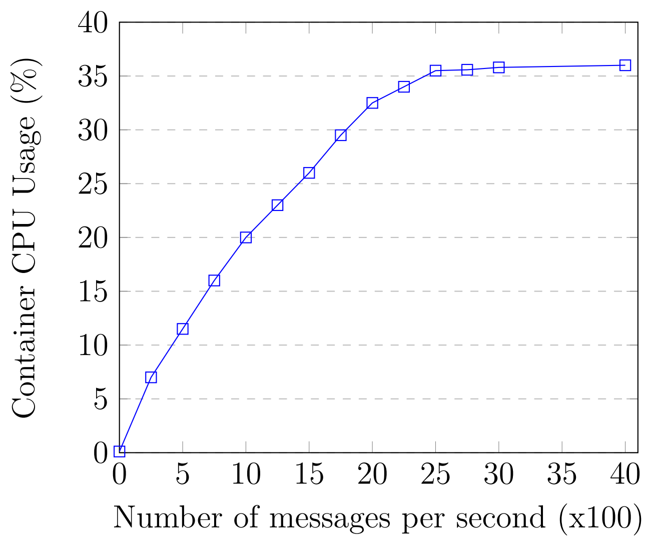


**Figure 9.1:** Single Container CPU usage with increasing load

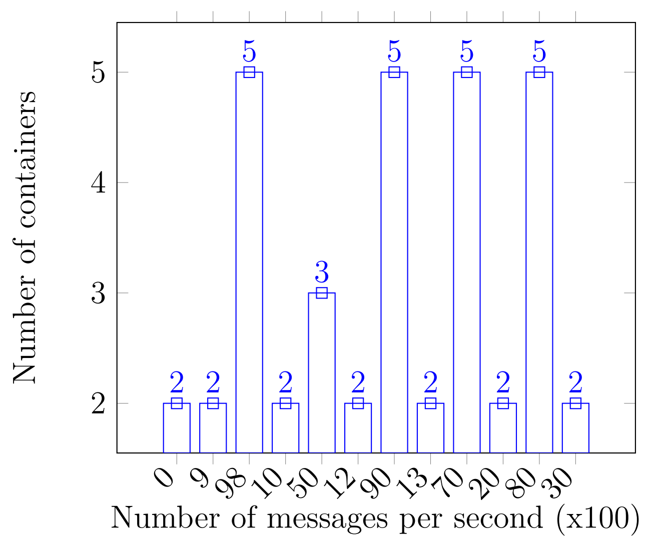


**Figure 9.2:** Sudden Traffic Message per Second vs Number of Containers

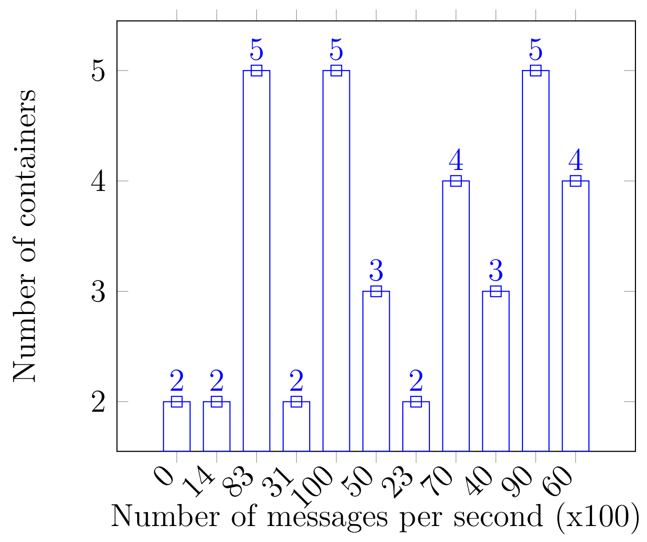


**Figure 9.3:** Random Traffic Pattern Message per Second vs Number of Containers

of containers versus random message per second. The message per second represents vehicular data traffic generated by vehicle simulator.

### 9.2.3 Steadily Increasing Traffic Pattern

This test case aims to observe the behavior of the system in steady increasing vehicular traffic pattern. The vehicular traffic increases in a constant rate and the required number of containers in the system also increases as expected. Figure 9.4 shows the number of containers required by the system versus message per second generated by the client (vehicle simulator).

## 9.3 Overall System Utilization

This test case enables to estimate the total capacity of the system in terms of number of running containers. As the size of vehicular traffic increases, the number of required running containers in the system also increase. explicitly, the test indicates the number of containers that the system can handle. Figure 9.5 indicates the CPU usage of the system versus the number of message per second generated by the vehicle simulator. The CPU usage of the system is dependent on the number of running containers in the system.

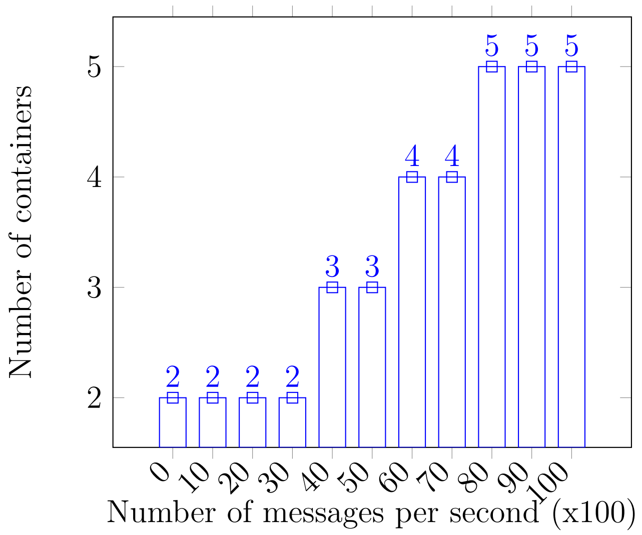


**Figure 9.4:** Steadily Increasing Traffic Pattern Message per Second vs Number of Containers

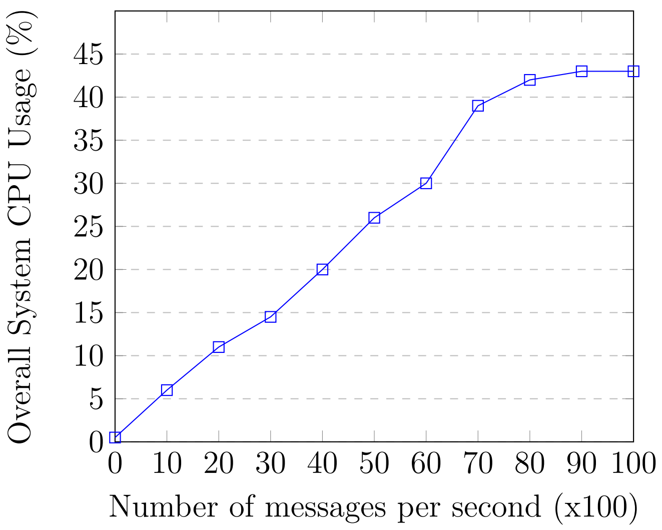


**Figure 9.5:** Overall System's CPU Utilization

# 10 Evaluation of Results and Discussion

The previous chapter described the results we got after running different stress tests on the vehicular cloud back-end system. In this chapter, we give explanations and discussions behind the result figures.

## 10.1 Single Container Stress Test

The single container response versus CPU usage plot in Figure 9.1 shows a steady increase usage with increasing number of messages per second. This test is carried out by subjecting one container to an increasing number of messages per second. The autoscaling algorithm need not to be run in this case because the test aims at finding out the limits of a single container. This test is specifically important in setting the higher and lower thresholds of the autoscaling algorithm. As discussed in Chapter 8, these thresholds are the basis for defining whether the container is over or under-utilized.

The steady increase in Container CPU usage is related to the increase in processor demand to process the increasing number of NATS messages from the vehicle simulator. After reaching 2500 messages per second, the increase in container CPU usage appears to stall. This is because the container is limited to use only a limited amount of CPU and memory resources of the main system. As discussed in Chapter 7, CPU and memory throttling plays important role in maintaining fairer distribution of resources among the Docker containers. The stall in CPU usage at right end of the graph is explains the CPU throttling applied to the container. This way, one can make sure that one busy container is eating up system resources and affecting the stability of the other running processes in the system. From this experiment, we learned that a single container can handle up to 2500 messages per second without being saturated. Above this number, the NATS broker and client could not guarantee delivery of all messages and packet drops will occur. To be on the safe side, the autoscaling algorithm takes 80% of this maximum value as an upper threshold to make scale-up decisions.

## 10.2 Response to Traffic Patterns

Response to traffic patterns is one important property of for cloud systems with irregular changes in traffic patterns. Due to the mobile nature of vehicles, the data traffic from them can affected distributed cloud back-end systems located nearest to the traffic. Thus, such system need to respond to sudden and randomized changes in load. In this work we carried out tests for three different variations in traffic pattern: *steady increase, random* and *sudden spike*. These traffic patterns cover different scenarios that could happen in a production system. For instance, a steady increase in traffic could show general increase in traffic in the morning when people commute to work. Similarly, Sudden traffic surge could happen if a traffic jam happens in a particular area. Random traffic could explain normal traffic flow with occasional increase in traffic.

As illustrated in Figure 9.2, the cloud back-end system is subjected to sudden changes in load from a lower traffic load 1,000 messages per second) to the peak value of 10,000 messages per second within the test time interval of five minutes. The autoscaling algorithm picked up the change soon and the scaling up operation is finished within this interval. The algorithm also responds well in sudden drop in traffic. However, sudden fluctuations in less than five minutes are harder to detect. This is because the CPU usage, one of the scaling indicator, takes time to be stable enough to exhibit the accurate state of the system. The cloud application consumes additional CPU power upon start up and shut down and CPU readings taken at this time may mistakenly assume that the system is overloaded. That is why the test interval is set at five minutes, giving the application more time to stabilize.

The test that involved steady increase of traffic showed more or less expected results. Steady increase in traffic is the most stable traffic pattern. This pattern varies scaling indicator in a more steady and accurate manner, which results in a less varying input to the autoscaling algorithm. The result in Figure 9.4 shows the autoscaling system responding well to the steadily increasing number of messages per second.

Random variation of traffic pattern closely explains real life scenario of a cloud system supporting connected vehicles. This pattern sits in the middle of steadily increasing traffic and suddenly increasing traffic. As plotted in Figure 9.3, the traffic load swings between high, low and medium from time to time. The number of containers changed accordingly, showing that the autoscaling algorithm reacts to randomly varying traffic patterns.

## 10.3 Overall System Utilization

The testbed for the cloud back-end system runs on a one physical machine as explained in Chapter 7. This resource limitation can adversely affect the quality of the tests unless resource usage of individual subsystems is monitored closely. Therefore, some measures are taken to make sure there is actually controlled usage of resources and a space to scale up to our specified goal.

These measures taken to control resource usage lie on design choices and introduction of constraints. The choice of a light weight message broker that works on minimum resources. NATS broker's light weight property helps for the system not to be congested by message routing affecting application containers to scale up. We also decided to containerize the database and NATS broker applications as an additional measure towards fairer distribution of resources. This is important because it is possible to isolate and control containers as opposed to native processes. We also closely monitored the overall CPU and memory utilization of the system while the scaling actions take place and when the system is subjected to the maximum desired load. Figure 9.5 illustrates the CPU usage of the system as the system is scaling up. It shows that even at full load the overall system is not congested and this, in turn, depicts that the system is performing its tasks properly.

## 10.4 Comparison with Other Autoscaling Systems

As described in previous sections, there are existing autoscaling systems, although most do not target supporting connected vehicles in particular. Most commercial cloud solutions are proprietary and difficult to know the exact autoscaling mechanism they used. These factors make direct comparison of results difficult. However, some of the approaches we follow can be compared.

As stated in research paper by Xiao et al., [1], most existing autoscaling systems use virtualization technology for managing virtual machines and other resources in the system. When a scaling up action is triggered in such systems, a virtual machine instance is added and provisioned to the system and for scaling down action a virtual machine instance is removed from the system. In this project, we used containerization instead of full blown virtualization. When the cloud system scales up, a container instance is added to the system and in scaling down action, a containers is removed from the system. In these two type of creating instances, they have a clear difference in provision time and performance. Containers boot in few seconds compared to virtual machines. In containers, it is possible to set access controls where as in virtual machines it depends on the hypervisor used. In terms of storage, containers take lower storage than virtual machines. Most importantly, containers have greater performance than virtual machines. The prototype implemented in this project uses the advantages of containerization technology.

Different autoscaling systems use different scaling metrics depending on the type of the system[10]. Most autoscaling systems use more than one scaling parameters. In most cases, CPU usage is considered as one of the scaling metrics. So, in terms of the scaling metrics, the implemented prototype uses the same as other systems.

Generally, due to the use of containers and proper choice of the scaling parameters, we believe that the implemented prototype is a good starting ground for study of autoscaling vehicular cloud services.

# 11 Conclusion

This project work enabled close study of two concepts of autoscaling cloud systems and connected vehicles. The concept of applying existing autoscaling algorithm in order to orchestrate a cloud back-end system supporting vehicles is demonstrated.

The implemented prototype cloud system served two purposes. First, it showed that the idea of remote diagnosis of vehicles is viable under the advancement of cheaper and faster mobile networks. Second, it showed that autoscaling systems are important in a system of such nature.

Collection and storage of diagnostic vehicular data helps the vehicle owner and manufacturer greatly. This prototype implementation showed that the design of such autoscaling systems requires careful selection of appropriate messaging protocols and technologies are important in realizing the system. The major design choices that are critical to the prototype system involve: the virtualization technology, the messaging protocol and the cloud application implementation. The prototype system met its goal of demonstrating the possibility of building cloud service to collect data and diagnose a vehicle remotely.

The chosen implemented autoscaling mechanism is a threshold rule based one. The practicality and ease of implementation are the major attributes that made us choose to go with this mechanism. The prototype system achieved its expected outcome of preventing message transmission loss due to congestion of system resources. This happened without over provisioning more resources than necessary. The autoscaling algorithm is crucial in matching the provision of resources with the amount of load the system faces. The traffic pattern tests also showed that the algorithm responds well to varied traffic patterns with some exceptions in detecting very frequent variations. Especially, the steadily increasing traffic pattern showed the complete expected behaviour.

Cloud computing is a building block of the future of connected vehicles. In turn auto-scaling system is a fundamental concept in cloud computing and this project aimed at building and testing a prototype that includes these concepts together.

# 12 Future Work

This thesis work's main focus resides in a relatively new area of connected vehicles. There is lot of room for more research in this area. This chapter highlights some of the most relevant areas that can be studied to expand and compliment this thesis work.

The cloud application demonstrated the possibility of using vehicular data for the purpose of remote monitoring and diagnosis. However, this project's scope is limited to storing the data and making it accessible via REST API. A front-end visualization tool that uses the API to render and summarize vehicular data can make this prototype more strong and demonstrate the use case stronger.

Another important area of future extension is adding a two way communication between the vehicle and the cloud back-end system. This helps in performing remote maintenance works in addition to monitoring and diagnosis. The NATS protocol, which is the currently used as messaging protocol, enables publication-subscription model in two ways. So this feature can easily be introduced in the system and add on the usability of the system.

The autoscaling mechanism that is implemented in this system is threshold-rule based. It is interesting to study other autoscaling techniques that are based on other theories for this use-case. The study and implementation of proactive autoscaling algorithms such as Reinforcement Learning approach could introduce more efficient autoscaling of this vehicular cloud back-end system.

This cloud environment testbed is setup with limited hardware resources. We believe that deploying this system in a real environment could introduce better results and open new questions. For instance, if the mobile network is used between vehicular simulator and the cloud back-end system, it would create a chance to assess other parameters like network delay and cost. Also, if the different subsystems are deployed in separate physical machines or commercial cloud instances, it would create more opportunity to separately look into the resource usages of the back-end application, message broker and the autoscaling subsystems.

# Bibliography

[1] H. L. Zhen Xiao, Qi Chen, “Automatic scaling of internet applications for cloud computing services,” *IEEE Transaction on Computers*, 2013.

[2] L. D. X. Wu He, Gongjun Yan, “Developing vehicular data cloud services in the iot environment,” *IEEE Transactions on Industrial Informatics*, 2014.

[3] A. A. K. A. S. Trieu C. Chieu, Ajay Mohindra, “Dynamic scaling of web applications in a virtualized cloud computing environment,” *IEEE International Conference on e-Business Engineering*, 2009.

[4] P. J. J. B. Jonathan Kupferman, Jeff Silverman, “Scaling into the cloud,” *Journal of Cloud Computing*, 2011.

[5] O. A. B. Maram Mohammed Falatah, “Cloud scalability considerations,” *International Journal of Computer Science Engineering Survey (IJCSES) Vol.5*, 2014.

[6] M. H. Ming Mao, “Auto-scaling to minimize cost and meet application deadlines in cloud workflows,” *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, 2011.

[7] M. A. Akihito Iwai, “Automotive cloud service systems based on service-oriented architecture and its evaluation,” *IEEE International Conference on Cloud Computing*, 2011.

[8] K.-C. L. Che-Lun Hung, Yu-Chen Hu, “Auto-scaling model for cloud computing system,” *International Journal of Hybrid Information Technology*, 2012.

[9] R. B. Luis M. Vaquero, Luis Rodero-Merion, “Dynamic scaling applications in the cloud,” *ACM SIGCOMM Computer Communication Review*, 2011.

[10] T. Lorido-Botran, J. Miguel-Alonso, and J. A. Lozano, “A review of auto-scaling techniques for elastic applications in cloud environments,” *Journal of Grid Computing*, vol. 12, no. 4, pp. 559–592, 2014.

[11] R. Dua, A. R. Raja, and D. Kakadia, “Virtualization vs containerization to support paas,” in *Cloud Engineering (IC2E), 2014 IEEE International Conference on*, pp. 610–614, IEEE, 2014.

[12] “Nats protocol reference.” `http://nats.io/documentation/`. Accessed: 2016-09-30.

[13] “Docker run reference.” `https://docs.docker.com/engine/reference/run/`. Accessed: 2016-05-10.

[14] “Cfs bandwidth control.” `https://www.kernel.org/doc/Documentation/scheduler/sched-bwc.txt`. Accessed: 2016-09-30.

[15] J. A. L. Tania Lorido-Botran, Jose Miguel-Alonso, “Comparison of auto-scaling techniques for cloud environments,” *Conference paper*, 2013.

# A
# Annotations

[1]
The paper presents the importance of scalability on resource usage to different Internet applications. It explains the practical use of virtualization technology in cloud computing. It also discusses about fault isolation in cloud computing. It evaluates the auto scaling algorithm by varying size of traffic in a system. It also uses color set algorithm for load distribution among the existing instances of an application. It used large scale simulations for testing auto scaling algorithms. A possible system architecture for scalable application is proposed in the paper.

[2]
The paper describes about advancement of cloud technology and Internet of things and its advantage to automotive system. It also presents multi-layered vehicular data cloud by using Internet of Things and cloud technology. It discusses about Intelligent parking cloud service to show how the advancement in cloud and IoT technologies can improve the transportation system. It also describes about the usage of cloud service for vehicular data mining. At last, it put performance, security, privacy and reliability as challenges of vehicular data clouds. It clearly shows the importance of cloud service for the automotive industry.

[3]
This paper introduces the importance of scalability for business applications on the web. It also describes about various virtualization techniques such as VMware, Xen, KVM, force.com and Microsoft virtualization. Advantages of virtualization on web application are also discussed. The main advantages described in the paper are higher utilization rates, resource consolidation, low energy usage and space saving. Various metrics that enable scalability of web applications are discussed. The paper considers metrics such as number of concurrent users, number of active connections, number of requests per second, and the average response time per request as scaling parameters. We found it useful as it clarifies the concept of scaling and how virtualization technology supports the cloud technology.

[4]
The paper describes effective scaling of resources in the cloud. It also describes about dynamic scaling algorithms. It compares the algorithms based their performance, availability and cost. The paper concludes as dynamic provisioning has higher performance, availability and cost compared to static provisioning. As explained in the paper, proactive auto scaling algorithms respond better in sharp increase of traffic

load than reactive algorithms.

[5]
The paper explains the importance of scalability in cloud computing. Due to flexibility, cost reduction and other characteristics of cloud computing many organizations are moving to cloud computing. It discusses scalability at different levels such as server scalability, network scalability and platform scalability.

[6]
This paper deals with how resource are allocated in cloud computing. It describes auto scaling algorithms such as schedule and threshold based algorithms which are used in resource allocation. Most auto scaling the algorithms focus on resource utilization. It is also important to consider performance and budget. The paper explains the virtual machine's size,cost,budget, performance and deadline. It also describes dynamic allocation and de allocation of virtual machines and scheduling of tasks with cost consideration.
[7]
This paper presents the improvements that can be made on the next generation of service-oriented architecture (SOA) of automotive cloud service system (ACSS). Due to the rapid development of cloud computing, the automotive industry is also changing to cloud service based systems. It illustrates the feasibility ACSS based on SOA by proposing a system architecture. The proposed system is a reliable SOA based ACSS.

[8]
This paper describes how auto-scalability improves the way cloud applications are built and deployed. It describes as cloud services are based on virtualization technologies for dynamic resource allocation. Specifically, it discusses about the auto-scaling algorithm for automatic provisioning of resources by using active number of sessions as metrics. The algorithm also considers energy efficiency. It also states as the algorithm is capable of handling sudden load in the system and achieves resource efficiency. In this paper, the reason why auto-scaling algorithms are needed in the cloud service is clearly stated.

[9]
The paper states that major advantage brought due to cloud is scalability.It presents the advantage of using cloud technology and its scaling capability. It also discusses about the main initiatives for scalable applications in the cloud technology. The main challenges that should be addressed in the scalable cloud services are also discussed in the paper. In addition, scalability at server level, database level, network level and containers is discussed. The paper deeply describes the need of scalability at different levels of the system.

[10]
The paper reviews five auto scaling algorithms used in the cloud technology and their use-case scenarios. It starts with description of cloud models that are cur-

rently used on the Internet. It also deals the behavior of elastic or scalable cloud applications. Finally, it experiments the performance of auto scaling algorithms by applying multiple metrics such as CPU load, memory, I/O, waiting time, number of active sessions, and so on. The paper shows the behavior of the different auto-scaling algorithms by considering various metrics. Hence, this review paper supports us in selection of the algorithms and the metrics to measure the performance of the algorithms.

[11]
This paper compares and contrasts the performance and use-cases of virtualization and containerization in a platform as a service (PaaS) cloud model. It investigates the various types of container implementations. It also deals about how the container implementations handle the file-system, process and other features. The factors affecting the container implementation choices and their missing features are also discussed in the paper. Finally, it concludes as containers have significant advantage over virtual machines in the PaaS cloud model. This is due to the performance and low latency time behavior of containers. The paper only discusses about the two technologies in a PaaS cloud model. It does not cover the application of virtualization and containerization in Software as a service (SaaS) and Infrastructure as a service (IaaS) cloud models.

[15]
Many auto scaling techniques are discussed and compared in the paper. It compares them in terms of cost and performance. It specifically compares proactive and reactive auto scaling techniques. It confirms that dynamic threshold has a better performance over other threshold based auto scaling techniques.