Institute of Architecture of Application Systems
University of Stuttgart
Universitätsstraße 38
D-70569 Stuttgart

Master Thesis No. 3670

A Service-oriented and Cloud-based Statistical Analysis Framework

Sugandha Agrawal

Course of Study:	Infotech
Examiner:	Prof. Dr. Frank Leymann
Supervisor:	Santiago Gómez Sáez
	Johannes Wettinger
Commenced:	May 26, 2014
Completed:	November 25, 2014
CR-Classification:	C.2.4, D.2.11, H.3, H.3.5

Abstract

Cloud Computing has gained popularity among e-Science environments after realizing the propitious use of economical provisions for delivering IT services and the range of resources offered by the cloud for the support, maintenance, and security of running the computation based applications. Cloud Computing being a recently growing technology offers various deployment and service models. In Software as a Service (SaaS) model, the applications and software run on the cloud and are available as 'pay-per-use'. As computing becomes more pervasive within the organization, the increase in complexity to manage the infrastructure of disparate architectures, distributed data and software has made computing very expensive. Cloud offerings promise to deliver all the functionality of existing information technology services at an economical cost. Researchers and scientists use resources provided by the cloud to handle large research datasets and results. The main advantages in Cloud computing are related to dynamic scaling of resources, which are able to adapt to changes based on demand of resources. Another advantage of cloud offering enables the use of multi-tenancy techniques to allow the sharing of resources between different users towards achieving the economy of scale along with considering data isolation as a dominant feature.

Representational State Transfer (REST) based architectural style has gained popularity for designing web service features like statelessness, modifiability, portability and simplicity. REST tends to focus on the components involved and their interactions along with interpretation of the significant data elements. Realising the intricacies of the computation and analysis that e-Science deals with, an attempt to provide a framework for statistical analysis has been made in this Master Thesis. The computational and numerical libraries are made available to the user and its functions provide the user with results in desirable format. Research focuses on providing such libraries can significantly and simultaneously decrease the computation time while decreasing the monetary costs of running such analyses. To enable scalability, Cloudburst technique is used to manage bursting the workload from a private cloud to public at times of capacity spikes and provide more resources on the public cloud to meet the user needs.

Contents

1 Introduction **1**
- 1.1 Problem Statement . . . 2
- 1.2 Scope of work . . . 2
- 1.3 Outline . . . 3
- 1.4 List of Abbreviations . . . 4

2 Fundamentals **7**
- 2.1 Cloud Computing . . . 7
  - 2.1.1 Characteristics . . . 8
  - 2.1.2 Service Models . . . 9
  - 2.1.3 Deployment Models . . . 10
- 2.2 Service-Oriented Architecture . . . 11
- 2.3 Web Services . . . 13
  - 2.3.1 Web Service Description Language (WSDL) . . . 14
  - 2.3.2 SOAP . . . 14
  - 2.3.3 Universal Description, Discovery and Integration (UDDI) . . . 15
- 2.4 RESTful Web Service . . . 15
  - 2.4.1 RESTful API Design . . . 18
- 2.5 Hyper Text Transfer Protocol (HTTP) . . . 20
  - 2.5.1 Status Codes . . . 20
  - 2.5.2 Methods . . . 21
- 2.6 Multi-Tenancy . . . 23
  - 2.6.1 Features of multi-tenancy . . . 24
  - 2.6.2 Challenges . . . 25
- 2.7 eScience . . . 26
  - 2.7.1 Role of different Computing environments on eScience . . . 26
  - 2.7.2 Cloud Burst . . . 27
- 2.8 Computation and Numerical Libraries . . . 29
  - 2.8.1 SciPy . . . 31

3 Related Works **33**
- 3.1 NEWT: A RESTful Service for High Performance Computing Web Application 33
  - 3.1.1 Motivation . . . 33
  - 3.1.2 Approach and Implementation . . . 34
- 3.2 Everest: A Cloud Platform for Computational Web Services . . . 36
  - 3.2.1 Motivation . . . 36
  - 3.2.2 Approach and Implementation . . . 37

3.3 MathCloud: Publication and Reuse of Scientific Applications 38
3.3.1 Motivation 39
3.3.2 Approach and Implementation 39
3.4 e-Science Central: Cloud-based e-Science for chemical property modelling . . 40
3.4.1 Motivation 40
3.4.2 Approach and Implementation 40
3.5 SaaS with Multi-tenancy Support for an e-Contract Management Application 41
3.5.1 Motivation 42
3.5.2 Approach and Implementation 42

4 Concept and Specification **47**
4.1 Requirements 47
4.1.1 Functional Requirements 47
4.1.2 Multi-Tenancy 48
4.1.3 Communication Requirements 50
4.1.4 Storage Requirements 51
4.1.5 SciPy Library 51
4.1.6 Non-Functional Requirements 52
4.2 Use Cases 54
4.3 System Overview 79

5 Design **81**
5.1 Architectural Overview 81
5.2 Resource Model 83
5.3 Databuckets 86
5.4 Database Schemas 86
5.5 Cloudburst Implementation 88
5.5.1 Cloudburst Scheduler 89
5.5.2 Cloudburst Monitor 89
5.5.3 Cloudburst Manager 89
5.6 Cloud interoperability 90
5.7 RESTful API 90
5.7.1 API Design 90

6 Implementation **103**
6.1 Databuckets 103
6.2 Django REST Framework 104
6.3 Resource Model 105

7 Validation **111**
7.1 Computation and databuckets 111
7.2 Validating REST APIs 112

8 Outcome and Future Work **115**

Bibliography **117**

List of Figures

2.1 Illustration for NIST Cloud Definition 8
2.2 SOA Model 12
2.3 SOA Model for Web Services 14
2.4 Conceptual view of cloud burst 28

3.1 Illustration of NEWT – RESTful Services for HPC 35
3.2 Architecture of Everest platform 38
3.3 Science Cloud Platform 41
3.4 Architecture framework of a multi-tenant e-contract management application 43
3.5 A shared database and schema with custom extensions 44

4.1 Use Case diagram for System Administrator 54
4.2 Use Case diagram for Tenant Administrator 55
4.3 Use Case for diagram Tenant Operator 56
4.4 System overview of the Computational System 79

5.1 Architectural overview of the Statistical Analysis Framework 82
5.2 Resource Model representation using ER Diagram 84
5.3 Entity Relationship Diagram for Tenant Registry 87
5.4 Entity Relationship Diagram for Configuration Registry 88

7.1 Request and response using POSTMAN API Client 111
7.2 Request and response using POSTMAN API Client 114

List of Tables

2.1 HTTP Status Codes . . . 21
2.2 HTTP Methods with corresponding CRUD Action . . . 22
2.3 Computation Libraries in C . . . 29
2.4 Computation Libraries in C++ . . . 30
2.5 Computation Libraries in Python . . . 30
2.6 Computation Libraries in Python and Java . . . 31
2.7 SciPy Package At A Glance . . . 32

4.1 Description of Use Case: Add Tenant . . . 56
4.2 Description of Use Case: Remove Tenant . . . 57
4.3 Description of Use Case: Add Tenant user . . . 58
4.4 Description of Use Case: Remove Tenant user . . . 59
4.5 Description of Use Case: Describe databucket size for the tenant . . . 60
4.6 Description of Use Case: Increase Databucket size for the tenant . . . 61
4.7 Description of Use Case: Decrease Databucket size for the tenant . . . 62
4.8 Description of Use Case: Create roles for tenant . . . 63
4.9 Description of Use Case: Define Tenant based roles . . . 64
4.10 Description of Use Case: Assign tenant operator role . . . 65
4.11 Description of Use Case: Change the assigned tenant administrator . . . 66
4.12 Description of Use Case: Delete tenant operator role . . . 67
4.13 Description of Use Case: Describe databucket size for the tenant operators . . . 68
4.14 Description of Use Case: Increase databucket size for the tenant operators . . . 69
4.15 Description of Use Case: Decrease databucket size for the tenant operators . . . 70
4.16 Description of Use Case: Share databucket . . . 71
4.17 Description of Use Case: Login to the service . . . 72
4.18 Description of Use Case: Logout of the service . . . 73
4.19 Description of Use Case: Trigger computation . . . 74
4.20 Description of Use Case: Retrieve result from own databucket . . . 75
4.21 Description of Use Case: Retrieve result . . . 76
4.22 Description of Use Case: Store result . . . 77
4.23 Description of Use Case: View tenant registry . . . 78

5.1 Description of REST API: Display information of available tenants . . . 94
5.2 Description of REST API: Display information of the specified tenant . . . 95
5.3 Description of REST API: Display information of the users for the specified tenant . . . 96
5.4 Description of REST API: Display information of the users for the specified tenant . . . 97

5.5 Description of REST API: Add tenants to the system 98
5.6 Description of REST API: Add user(s) to the tenant specified 99
5.7 Description of REST API: Trigger computation 100
5.8 Description of REST API: Trigger computation 101
5.9 Description of REST API: Retrieve result from databucket 102

List of Listings

6.1 Model definition in Django REST Framework 105
6.2 Serializer definition in Django REST Framework 106
6.3 Class based View definition in Django REST Framework 107
6.4 URL defined for the resource identification in Django REST Framework . . . 108

1 Introduction

Computation in its literal sense refers to the very basic idea for getting a result based on calculations, following a standard procedure or using an algorithm to ensure the authentication of the result that is generated. Computing has been indeed transformed into a model providing utilities or more appropriately called services for example: water, electricity, telecom etc. Service-Oriented Computing is one of the paradigms behind services. Recent advancements in the field of computational paradigm led to the introduction of Cloud Computing which was considered a breakthrough. Cloud computing is built upon the principles of Grid Computing and relies on it for the infrastructural support but basically the evolution has been a result of a shift in focus from an infrastructure that delivers storage and compute resources (such is the case in grids) to one that is economy based aiming to deliver more abstract resources and services (such is the case in clouds) [FZRL09].

Cloud computing has proven to be very efficient in providing coherent computing results, hence storage capabilities, platform and infrastructure independencies, security and more at varied levels. The "pay per use" feature has promptly caught attention in enterprise information technology (IT) environments and has opened more gates for development at all stages. Cloud computing platforms possess characteristics of both clusters and Grids, with its own special attributes and capabilities such strong support for virtualisation, dynamically composable services with Web service interfaces, and strong support for creating third party, value added services by building on Cloud compute, storage, and application services. Thus, Clouds are promising to provide services to users without reference to the infrastructure on which these are hosted [BYV+09].

With the advent of Cloud computing paradigm, it supported hosting services which can be defined as a set of functionalities which are reusable, extendible and can be interoperable. When cloud provisions to host the services, the services communicate over the web as a network and hence, are called as to be Web services. The World Wide Web Consortium (W3C), which has managed the evolution of the Service, SOAP and Web Service Description Language (WSDL) specifications, defines Web services as follows: A software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using Hyper Text Transfer Protocol(HTTP) with Extensible Markup Language(XML) serialization in conjunction with other Web-related standards [ACL+05]. Though, today there are other languages like JSON, YAML available in the web for interaction and are supported by Web services. The Web services provides the capability for self-contained business functions to operate over the internet.

Since the introduction of Web Services as an approach towards the Service-Oriented Architecture, many have shown interest in the use of Web services for simulations and statistical analysis related to scientific simulations and experimentations. Henceforth, in this thesis the use of tools available for statistical analysis of data are considered for exposing as Web service along with providing the multi-tenant architecture support.

1.1 Problem Statement

With the increasing need of faster accessibility to the results generated via simulations and use of optimization techniques in the field of e-Science, it has led the scientists and researchers to ponder upon the need to save time required in order to use the available functionalities and enhance the techniques to get result to their fingertips with least resource exploitation subsequently decreasing the computing cost for the analyses. There are various proprietary tools available to allow the computation to be performed but it sometimes essentially requires considerable amount of computational resources in order to process the given data in a short time frame, e.g.for building statistical models. Such tools also are meant to be run on-premise and to be shipped as packaged programs or libraries. This diminishes the possibility and freedom to use such tools everywhere without being built and installed on each machine. Since, the result thus generated is not less but is a vast generated dataset, it further becomes more difficult and cumbersome to share the result and perform analysis freely for such a large dataset as its fairly distributed at various locations. To involve data centralization techniques, it becomes very inefficient to handle various resources. The need for such huge number of resources increases the monetary cost of operations to perform such analysis drastically and also as stated before, degrades the computation time. Many open source libraries though available need on-premise installation requirements to be fulfilled making the use further restrictive to realize a distributed environment. The purpose of this work is to provide a set of the functionalities from open source libraries offered by various languages, as a Web Service accessible to multiple users and enabling sharing the calculated results amongst them using standardized messaging protocols. The data transfer to support the functionalities offered by many open source libraries for computation in varied formats e.g: JavaScript Object Notation(JSON), XML are to be provided along with enabling the deployment and execution of the application rendering diverse statistical and optimization techniques. This thesis is focused on open source mathematical and experimental ecosystem *SciPy*, a library implemented in Python which enables statistical analysis.

1.2 Scope of work

This web service would enable users to operate on a set of functionalities offered by SciPy library in order to save the effort to install and build the library at each physical entity where the library functions are intended to be used. Also, it deals with saving the resources by optimizing the need to use only the required functions rather than building the whole library by running them on Cloud. The multi-tenant architecture supports data isolation

techniques to maintain data security and allow role based access control over the resources available. Cloud computing characteristics relating to scalability and availability are put into consideration for the on-demand pulling of resources for data storage to support dynamic scaling. Considering cloud burst technique the additional resources can be provided on an as-needed basis. The designed application uses REST architectural design style and hence uses HTTP for the interaction over the network which in our case refers to the web. Conforming to the REST design guidelines, the application maintains a stateless behaviour over the client-server model. The statistical analysis supported in this work limits only to the use of the SciPy libraries for computation and does not support using any other library from any other programming language. Also, only some packages from the SciPy library are supported which are described in the coming chapters.

1.3 Outline

The remaining document is structured in the following way:

Chapter 2 - Fundamentals : Key concepts and their basic description to support the rest of the report with building the elementary knowledge base for the respective topics.

Chapter 3 - Related Works : Similar works assisting the concepts used in context to the application including the research approaches and the challenges faced while deployments and implementations.

Chapter 4 - Concepts and Specifications : To acknowledge the requirements namely functional and non-functional in order to allow perform computational result calculations as a service. Specifications in terms of the set of functionalities exposed and the multiple user data sharing approach.

Chapter 5 - Design : Detailed overview of the architecture and model supported for the application. Using RESTful architectural design and applying design rules for efficient model generation.

Chapter 6 - Implementation : Well defined information including challenges while coding for the application and the configuration details are mentioned.

Chapter 7 - Validation and Evaluation : Validating the implementation and evaluating the application for the different user scenarios.

Chapter 8 - Conclusion and Future Works: Summarizing the work done while passing through various precise detail in each chapter and describing possible scope for enhancements and extensions for the application attempting to expose SciPy functions.

1.4 List of Abbreviations

The following list contains abbreviations which are used in this document.

AJAX Asynchronous JavaScript and XML

API Application Program Interface

ASP Application Service Providers

B2B Business to Business

CRUD Create, Read, Update and Delete

CWS Computational Web Service

e-SC e-Science Central

EC2 Elastic Cloud Compute

GCC GNU Compiler Collection

HATEOAS Hypermedia as the engine of application state

HPC High Performance Computing)

HTTP Hyper Text Transfer Protocol

IaaS Infrastructure as a Service

ITHA Isolated Tenancy Hosted Applications

JSON JavaScript Object Notation

MTA Multi-tenant applications

MTEA Multi-tenant enabled application

MVC Model-View Controller

NERSC National Energy Research Scientific Computing Center

NEWT National Energy Research Scientific Computing Center web toolkit

NIST National Institute of Standards and Technology

PaaS Platform as a Service

QoS Quality of Service

RbAC Role based control Access

REST Representational State Transfer Protocol

ROI Return of Investment

RPC Remote Procedural Call

S3 Simple Storage Service

SaaS Software as a Service

SLA Service Level Agreement

SOA Service-oriented Architecture

SOC Service-oriented Computing

UDDI Universal Description, Discovery, and Integration

UID Unique Identifier

URI Uniform Resource Identifier

URL Uniform Resource Locator

UUID Universally Unique Identifier

VO Virtual Organisation

WSDL Web Service Description Language

XML Extensible Markup Language

2 Fundamentals

This chapter focuses on the fundamental concepts needed to build a foundation for the purpose of the thesis. Acting as the base to the understanding of the key components of the application, many references have been mentioned for better clarity of the main topics.

2.1 Cloud Computing

Cloud computing comes into focus to fulfil the business or enterprise needs to have a flexible way to build powerful and up-to-date IT infrastructure, have access to great computing power in a way to increase capacity or add capabilities on the fly without investing in new infrastructure, training new personnel, or licensing new software. Cloud computing encompasses any subscription-based or pay-per-use service that, in real time over the Internet, extends the existing capabilities.

The National Institute of Standards and Technology (NIST) has very well defined cloud computing as "a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction." [MG11]. This cloud model is composed of five essential characteristics, three service models, and four deployment models [MG11].

In the context of the internet, cloud computing refers to assign computing tasks to a “cloud”—a combination of compute, storage, and application resources accessed over a network. It is not of concern of the business entity to realize that where is the data physically stored or where the servers are physically located, the major concern is only about usability and security. The cloud service providers deliver applications through the internet that are accessed via the web browsers, while the business software and data are stored on servers at a remote location. Most cloud computing infrastructures consist of services delivered through shared data centres. The cloud appears as a single point of access for consumer’s computing needs [JOP11].

As suggested by Fehling [FEL+12], cloud applications have to follow three fundamental architectural principles:

1. Componentisation – The functionality of a cloud application is divided into loosely coupled components.
2. Distribution and Redundancy – Multiple instances of application components are distributed among a set of redundant cloud resources.

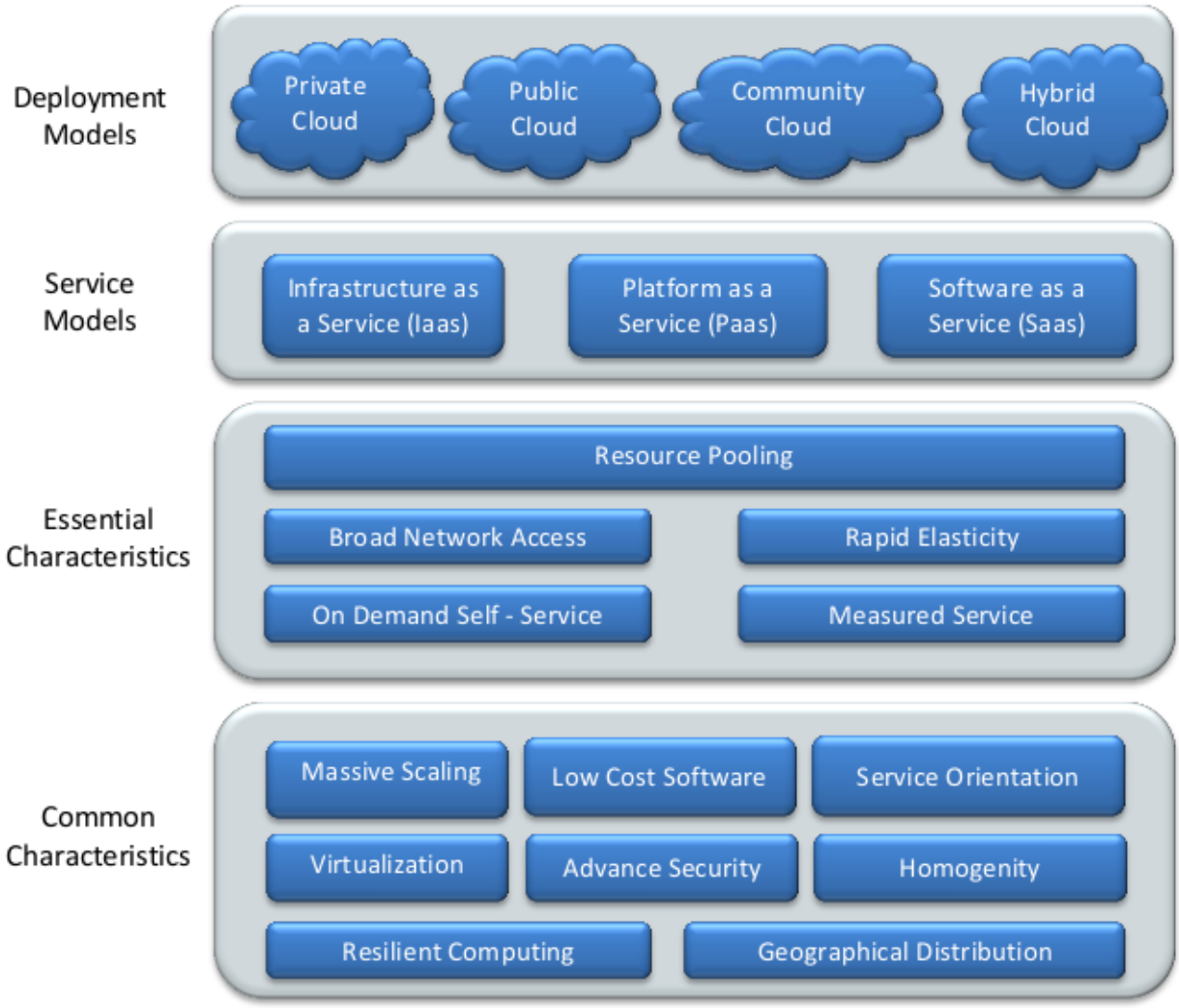

Figure 2.1: Illustration for NIST Cloud Definition [MG11]

3. Automated Management – Resource provisioning and de-provisioning is automated to increase beneficial effects of the cloud's elasticity [FEL+12].

Understanding the aspects of cloud computing and to deeply dive into its capabilities and field of application, characteristics have to be dealt in detail. Oracle CEO L. Ellison thinks that cloud computing is nothing more than "everything that we currently do" [Far08]. This statement is not misleading but misses fractions of information and therefore the characteristics of the cloud computing paradigm are described in the next section.

2.1.1 Characteristics

Characteristics of cloud computing as explained by Gong et al. [GLZ+10] are:

1. Service Oriented : This is a conceptual characteristic with the two major keys as abstraction and accessibility. Through vitalization, abstraction is achievable by hiding the underlying infrastructure from the user and at the same time the key elements within the hidden infrastructure are availed to the user. Cloud users can consume all the capacity easily by exploring system parameters such as processing performance and storage capacity [W3C04]. In general, according to the type of provided capability, the services of cloud computing are broadly divided into three Service models which are described later in the section 2.1.2.

2. Loose Coupling : Even though with virtualisation, the physical and logical separation of the infrastructure is attained, the components tend to affect each other behaviourally. The clients or cloud users connect loosely with servers or cloud providers. All the users have almost no data or control dependence. This is one of the important technical characteristic

3. Strong Fault Tolerance : Fault tolerance has to be taken in account considering the wide spread field of application of Cloud computing. Realizing the fact that application can be prone to faults, for e.g: fault occurring among provider and users then it could be due to one of the many reasons as network congestion, browser collapse, request time out, provider busy and hacker attack then to insure the integrity and security of the system appropriate fault tolerant algorithms are to be put into action.

4. Business Model : Being considered as an economical characteristic there are many business models especially how-to-pay models in cloud computing. Pay-per-use may be the favourite one in many cases. This is almost the same as the concept of utility computing. The capacity of processing, storage and network in cloud computing is utility service as water, electricity and gas in society [BYV+09]. Users pay service providers based on their usage of these utility services.

5. High Security : High security of cloud computing is achieved mainly through three ways, loose coupling makes cloud computing system run well even when part of it is destroyed. Abstraction, virtualisation and privation of cloud provider avoid exposing the details of corresponding implementations. Along with these above mentioned points cooperating it with law guards of cloud computing.

2.1.2 Service Models

There are three service models that are available for cloud systems as explained by NIST [MG11]:

1. Infrastructure as a Service (IaaS) - IPs manage a large set of computing resources, such as storing and processing capacity. Through virtualization, they are able to split, assign and dynamically re-size these re-sources to build ad-hoc systems as demanded by the customers. They deploy the software stacks that run their services. Some of the

prominent IaaS providers currently existing are : Amazon AWS[1], Windows Azure[2], Google Compute Engine[3] etc.

2. Platform as a Service (PaaS) - Instead of supplying a virtualized infrastructure, they can provide the software platform where systems run on. The sizing of the hardware resources demanded by the execution of the services is made in a transparent manner. This is denoted as Platform as a Service (PaaS). Some of the prominent PaaS providers currently existing are : Heroku[4], Google AppEngine[5], Red Hat Openshift[6] etc.

3. Software as a Service (SaaS) - There are applications of potential interest to a wide variety of users hosted in Cloud systems. This is an alternative to locally run applications [RMCL09]. The customer accesses the provider's application running on the provider's servers [JOP11]. The SaaS model delivers the complete application as a service. Some of the prominent SaaS providers currently existing are : SalesForce[7], Google Docs[8] etc.

2.1.3 Deployment Models

To explore the cloud features, the NIST definition of cloud computing [MG11] defines four deployment models. The following models are proposed:

1. Private Cloud - The cloud infrastructure is operated solely within a single organization comprising multiple consumers (e.g., business units) which may be owned, managed, and operated by the organization, a third party, or some combination of them regardless whether it is located within a premise or off premise. This states that a cloud is privately owned by a complete organisation. The private cloud will provide computing power as a service within a virtualised environment using an underlying pool of physical computing resource. However, under the private cloud model, the cloud (the pool of resource) is only accessible by a single organisation providing that organisation with greater control and privacy. The basic advantages of considering this deployment model as described by Dillon et al. [DWC10]:

   a) To maximize and optimize the utilization of existing in-house resources.

   b) Security concerns including data privacy and trust also make this model is popular among business oriented firms.

   c) Organizations always require full control over mission-critical activities that reside behind their firewalls.

[1] http://aws.amazon.com/
[2] http://azure.microsoft.com/
[3] https://cloud.google.com/compute/
[4] https://www.heroku.com/
[5] https://cloud.google.com/appengine/docs/
[6] https://www.openshift.com/
[7] https://www.salesforce.com/saas/
[8] https://www.google.com/docs/about

The above mentioned reasons make this deployment model an ideal selection for academics for research and teaching purposes also. Though there are certain issues that arise due to expensiveness, security issues while integrating with other cloud models thus obstructing the flexibility, reconfiguration on the fly and one of the major challenge being Return of Investment (ROI). The organization implementing the private cloud is responsible for running and managing IT resources instead of passing that responsibility on to a third-party cloud provider.

2. Community Cloud - Several organizations jointly construct and share the same cloud infrastructure and have shared concerns like mission, policies, security requirements and compliance concerns. The cloud community forms into a degree of economic scalability and democratic equilibrium. The cloud could be hosted by a third-party vendor or within one of the organizations in the community or some combination of them, and it may exist on or off premises.

3. Public Cloud - This is the dominant form of current Cloud computing deployment model. The cloud infrastructure is provisioned for open use by the general public. It may be owned, managed, and operated by a business, academic, or government organization, or some combination of them. The public cloud consumers and the cloud service provider have the full ownership of the public cloud with their own policy, value and profit, costing, and charging model. It does exist on the premises of the cloud provider. Many popular cloud services are public clouds including Amazon Elastic Cloud Compute (EC2)[9], S3[10], Google AppEngine[11] and Force.com[12].

4. Hybrid Cloud - The cloud infrastructure is a combination of two or more clouds (private, community, or public) that remain unique entities but are bound together by standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for load-balancing between clouds). Organizations use the hybrid cloud model in order to optimize their resources to increase their core competencies by margining out peripheral business functions onto the cloud while controlling core activities on-premise through private cloud. Hybrid cloud has raised the issues of standardization and cloud interoperability [DWC10].

2.2 Service-Oriented Architecture

A Service-oriented Architecture (SOA) is the underlying structure supporting communications between services. SOA defines how two computing entities, such as programs, interact with each other in such a way so as to enable one entity to perform a unit of work on behalf of another entity. Service interactions can be defined using a description language. Each interaction is self-contained and loosely coupled, so that each interaction is independent of any other interaction hence said that "SOA is a specific architectural style that is concerned

[9]https://aws.amazon.com/ec2/
[10]https://aws.amazon.com/s3/
[11]https://cloud.google.com/appengine/docs/
[12]https://www.salesforce.com/saas/

with loose coupling and dynamic binding between services." [ACL+05]. SOA is a way of re-organizing a portfolio of previously soiled software applications and support infrastructure into an interconnected set of services, each accessible through standard interfaces and messaging protocols [Pap03]. SOA is a specific architectural style that is concerned with loose coupling and dynamic binding between services. Service-oriented Computing (SOC) provides cooperating services loosely connected, creating dynamic business processes and agile applications that span organizations and platforms. As a computing paradigm, it utilizes services as fundamental elements to support rapid, low-cost development of distributed applications in heterogeneous environments. SOA can also be stated as an architectural style to realize SOC. For e.g., "client/server" is an architectural style for realizing distributed computing.

SOA is a relationship of services and service consumers, both software modules large enough to represent a complete business function. Services are software modules that are accessed by name via an interface, typically in a request-reply mode. The basic SOA is not an architecture only about services but also defines a relationship of three kinds of participants:

1. The service provider
2. The service discovery agency
3. The service requester (client)

The interactions amongst them involve the publish, find and bind operations [W3C04]. A service provider as the name suggests hosts a network accessible application or software module in form of a service available for the consumption. For the identification of a service , the service discovery agency provides a consolidated information center for the services in a form of a registry like UDDI explained in the section 2.3.3. The service is published by the service provider in the registry, where it is discoverable by the service requester. The referred service is accessed by the client via binding with the service provider using an artifact named 'service description'. Hence, the service is realizable.

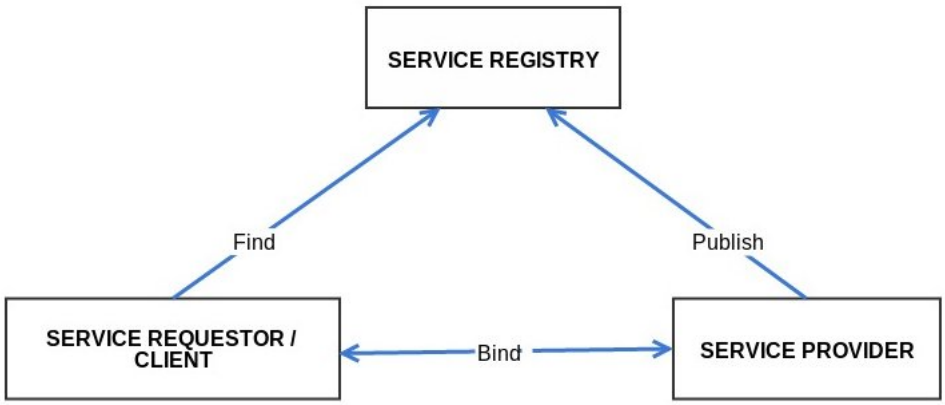

Figure 2.2: SOA Model

The strong impact of SOA and the reason for its business implications involves its loose coupling behaviour, increase in agility over time pertaining to responsiveness and technical

changes in the business environments, collaboration flexibility and well designed services offering more re-usability needless to mention about interoperability, standardization and efficiency. SOA is not a technology but a design philosophy independent of any vendor, product, technology or industry trend. No vendor will ever offer a "complete" SOA "stack" because SOA needs vary from one organization to another [SK09]. SOA intends to provide the architecture which can assist the situation of having varied and non-binding enterprises together by allowing them to use mismatched services into one paradigm.

2.3 Web Services

As stated in the section 2.1.2, a service is hosted on an accessible network and when the network is World Wide Web, it is nothing but a web Service. Web Service Technology (WS) is a standard and technology stack supporting SOA for e.g., "stored procedures", "web browser/server", support the client/server style.

A web service can be available at a particular endpoint in the web or within the private network of a business unit, and it receives and sends messages and exhibits behaviour according to its specification [Pal07]. The service has specific functionality and is deployed with appropriate quality of service at the endpoint. There are interfaces and policies which describe the terms and conditions that govern the use of the service. These are published so that potential users of the service can discover and be given all the information they need to bind (perhaps dynamically) to that service. The information that is published about a service provides details of what the service is and does (its semantics). It further provides all the information that allows the environment hosting a potential user to access the service and successfully interact with it. This can include information about the transport protocols that can be supported and used to send a messages to the service, the wire format of this message expected by the service, whether and how the message has to be encrypted or signed. Although the environment that is hosting needs this information often referred to as a container, must deal with that detail necessary to interact with a particular implementation and the requester doesn't need this kind of access information, and it certainly doesn't need to understand details about the implementation of the service. The requester can think in terms of using a web service without worrying about the underlying technology stack. In that sense, web service technology is virtualisation technology for making use of services, but an implementer of a service can use the technology they are acquainted with to build the service implementation.

Web services are software systems designed to support interoperable machine-to-machine interaction over a network. This interoperability is gained through a set of Extensible Markup Language (XML)-based open standards, such as Web Service Description Language (WSDL) [13], SOAP [14], and Universal Description, Discovery, and Integration (UDDI) [Che11]. These standards support the interaction of a web service requester with a Web service provider and the potential discovery of the Web service description.

[13] http://www.w3.org/TR/wsdl/
[14] http://www.w3.org/TR/soap/

The web services approach implements SOA. A major focus of web services is to make functional building blocks accessible over standard Internet protocols that are independent from platforms and programming languages. These services can be new applications or just wrapped around existing legacy systems to make them network-enabled. A service can rely on another service to achieve its goals.

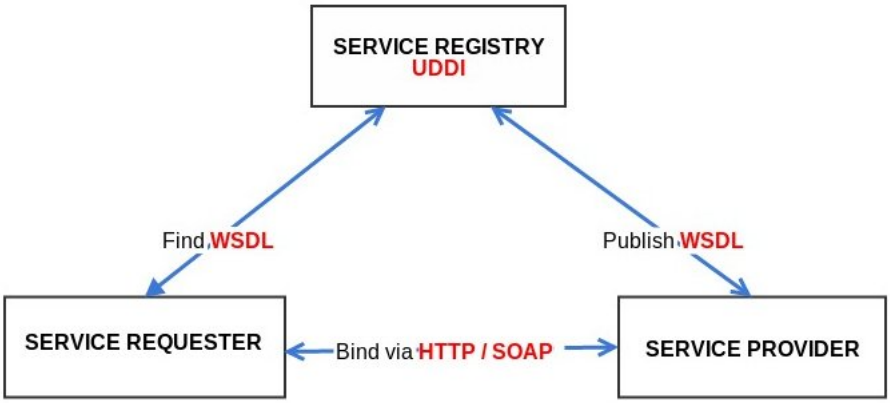

Figure 2.3: SOA Model for Web Services

2.3.1 Web Service Description Language (WSDL)

A WSDL document defines services as collections of network endpoints, or ports [CCMW11]. WSDL is in practice the standard for XML-based service description. It is the minimum standard service description necessary to support interoperable web services. It defines the interface and mechanics of service interaction needed for non-identical services to work together. This description is necessary to specify service-to-service relationships. A complete WSDL service description provides two pieces of information: an application-level service description, or abstract interface, and the specific protocol-dependent details that users must follow to access the service at concrete service end points. This separation accounts for the fact that similar application-level service functionality is often deployed at different end points with slightly different access protocol details. Separating the description of these two aspects helps WDSL represent common functionality between seemingly different end points.

2.3.2 SOAP

SOAP is an XML - based protocol for accessing the available web services. SOAP provides a way to communicate between applications running on different operating systems, with different technologies and programming languages. To support the distributed and heterogeneous nature of the Web, the communication mechanisms have to be platform-independent, international and secure and therefore SOAP as it built using time testing systems like the Hyper Text Transfer Protocol (HTTP) protocol and text mark-up in XML. There are two types of SOAP requests:

- The first is the Remote Procedural Call (RPC) style request similar to other distributed architectures. This is usually synchronous, i.e., the client sends a message and waits to get a response or fault message back from the server. In the RPC style web service, the name of the method and its parameters is used to generate XML structures that represent a method's call stack.
- The second type of SOAP request is the document request. Here in a full XML document is passed to/from the client and server, inside a SOAP message which can be validated against pre-defined XML schema document. [Sud03].

2.3.3 Universal Description, Discovery and Integration (UDDI)

The Universal Description, Discovery, and Integration specifications offer users a unified and systematic way to find service providers through a centralized registry of services that is roughly equivalent to an automated online "phone directory" of Web services [CDK+02]. UDDI is the registry that has the cumulative description of the potential services which can be exploited by the service requester. The entry is made by the service provider in order to be identified and used. In terms of Business to Business (B2B) application of web services, UDDI not only provides a description of the business and its services, but also what the technical programming interfaces (or API's) are that the business has available for doing B2B trade. This allows compatible business partners to be discovered automatically. These registries are run by multiple operators, are freely available and can be used by anyone. UDDI provides a single access point that enables service requester to discover the available services using one standard. It continues to build on already available standards (such as HTTP, XML and SOAP) supporting the extension to already used services and to understand the compatibility of the various services to be used together.

2.4 RESTful Web Service

Client programs that want to communicate to any available web service use Application Program Interface (API) irrespective of the architectural style used. What an API exposes is the set of data and function to facilitate interactions between computer programs and allow them to exchange information. Hence to style these APIs to enable the communication, Representational State Transfer Protocol (REST) is applied to design such APIs. Well-designed REST APIs can be beneficial for the providers and web service developers to attract more and more service requester. The Representational State Transfer (REST) style is an abstraction of the architectural elements within a distributed hypermedia system [Fie00]. REST ignores the details of component implementation and protocol syntax in order to focus on the roles of components, the constraints upon their interaction with other components, and the interpretation of significant data elements. It encompasses the fundamental constraints upon components, connectors, and data that define the basis of the Web architecture, and thus the essence of its behavior as a network-based application. Web service APIs that adhere

to the REST architectural constraints are called RESTful. The architectural constraints that were referred in [Fie00] are:

Client-server - The web communication system can bind only those entities which follow the established standards to accomplish the interaction among each other. E.g., for Client-Server interaction, even thought the entities are implemented and deployed independently while using different languages or technologies, they have to stick to the standards to interact with one another.

Uniform Interface - The uniformity of the interfaces for interaction between the web components such as- servers, clients and network-based intermediaries is of major concern. If not so, then the communication system fails. For this to be achieved, four constraints identified were [Fie00]:

1. Identification of resources - This can be attained by defining a global addressing space for resource and service discovery. E.g., for a particular home page has to be unique in order to be specific to that website's root resource.

2. Manipulation of resources through representations - The same exact resource can be represented to different clients in different ways. Hence, the representation should be manipulatable but it does not change the resource. This provides the freedom for the varied representation of the same resource.

3. Self-descriptive messages - To enable multiple formats for the resource representation as mentioned above, the messages are complete in itself thus including the meta-data to convey details regarding the resource state, the representation format and size.

4. Hypermedia as the engine of application state (HATEOAS) or Stateful interactions through hyperlinks - A resource's state representation includes links to related resources. To know the resource's current state has to be within the message and not on the server end. This leads to interactions which are stateful as the information is contained in hyperlinks but is not stored at the server side but at the client side. A client interacts with a network application entirely through hypermedia provided dynamically by application servers.

Layered System - For enforcing security, load balancing or response caching, components like proxies and gateways are used. They necessarily need to be transparently deployed. Layered systems reduce coupling across multiple layers by hiding the inner layers from all except the adjacent outer layer, thus improving evolvability and reusability. This layered system style allows an architecture to be composed of hierarchical layers by constraining component behaviour such that each component cannot "see" beyond the immediate layer with which they are interacting. By restricting knowledge of the system to a single layer, we place a bound on the overall system complexity and promote substrate independence.

Cache - Considered to be one of the important constraints, it definitely reduces the overall cost of Web as caching of response data enables increased overall availability of responses, reliability of an application thus controlling a web server's load. The advantage of adding cache constraints is that they have the potential to partially or completely eliminate some interactions, improving efficiency, scalability, and user-perceived performance by reducing the average latency of a series of interactions [Fie00]. Caching supports scalability as and when there occurs the dynamic scaling of the resources, the cached data can be easily referred to and moved on to the newly established resources. The only constraint related to maintaining the recent information in the cache and avoid keeping the stale information to avoid inconsistent data from being moved to the newly scaled resources.

Stateless - This constraint forces client to include all the contextual information relevant in the interaction with web server. This in return relaxes the web server by freeing it from memorizing the state of its client application. The benefit lies in the possible scalability of the web's architectural style.

Code-On-Demand - This constraint tends to establish a technology coupling between web servers and their clients, since the client must be able to understand and execute the code that it downloads on-demand from the server.

The architectural properties offered by REST are:

- High performance - The resources are uniquely identified by Uniform Resource Identifier (URI) thus caching closer to the resource is enabled. This helps in faster identification of the resource. A resource is some chunk of information that can be identified by a Uniform Resource Locator (URL) (it's the R in URL). The most common kind of resource is a file, but a resource may also be for example a dynamically-generated query result or a document.

- Scalability - This property supports the growing demand. Many of the reasons are like - caching would allow for bypassing the original server. All the required state information for the interaction are contained within the request itself. The request has no server affinity (i.e., it can be served by any server with the appropriate cache content) which enables to *spray* the request across cluster of servers hence providing a scaling system.

- Simplicity - For the application to be simple and understandable, the separation of concerns plays a major role. Four generic verbs Create, Read, Update and Delete (CRUD) described in section 2.5.2 allows for the ease of implementation.

- Portability - The application must be capable to be run on heterogeneous environments as the service oriented architecture supports the provision to use services being technology and platform independent. Using any service must be independent of the environment it runs on and must support ease in its usability.

- Data Independency - REST promises to allow the possibility to use the possible format for a resource. Data of a single resource may be in different formats therefore the request may contain client's formatting capabilities (in the header) allowing for content negotiation.

2.4.1 RESTful API Design

An API refers to an application programming interface which lets the client program communicate with the service and as mentioned earlier a web service is the service offered over the web for e.g., Amazon Simple Storage Service (S3) supporting storage, Google AppEngine providing tools for enabling SaaS application hosting etc. In order to use the web services there are APIs for the client program and an API exposes a set of data and functions to facilitate interactions between computer programs and allow them to exchange information [Mas11]. A Web API conforming to the REST architectural style is a REST API. To design a REST API, there are certain practices implicit to the HTTP standard but due to the flexibility of designing the API it becomes easier to develop comprehensible APIs that too according to what a service has to offer. The designing of the API is of significance as for a Client to be able to use a service, the functionalities it offers , how it these functionalities can be used is visible via the API alone. The components to an API are the resources. They are the most fundamental units to a REST API. Hence, to access the resource a URI must be designed to be able to reach out to a resource. The other aspect to it is to perform any action on the resource which can be done via standard methods offered by HTTP.

API Design Rules

For maintaining the consistency at some level and to leverage a standard and clean API for the client usage, certain rules have been defined. These rules not only let the client use the API with clear understanding of what service is offering but also liberates the designer from confusions and provokes them for a careful consideration while designing the API. Many of the rules have become de-facto standard for the design while some rules can be accepted or slightly modified to make the URIs more readable and easier to understand. The design rules for URI format are [Mas11]:

- Forward slash separator (/) must be used to indicate a hierarchical relationship.
- A trailing forward slash (/) should not be included in naming the resources.
- Lowercase letters should be preferred in URI paths. This can be clearly explained in the form of an example as:
  `http://API.EXAMPLE.RESTAPI.ORG/my-service/my-name`
  `http://api.example.restapi.org/My-service/my-name`
  The two URIs are not the same.

- File extensions should not be included in URIs rather REST API clients should be encouraged to utilize HTTP's provided format selection mechanism. The REST API clients should allow the user to explicitly mention the format they expect for the response.

- Since URI path conveys the REST API's resource model, with each forward slash separated path segment must correspond to a unique resource within the model's hierarchy.

- While considering different resource archetypes, a singular noun should be used for document names [15], a plural noun should be used for collection names [16] and a plural noun should be used for store names [17].

- CRUD function names should not be used in URIs.

- The query component of a URI may be used to filter collections or stores. This would help to distinguish between the resources. For e.g., `GET/users` - The response message's state representation contains a listing of all the users in the collection. `GET/users?role=admin` - The response message's state representation contains a filtered list of all the users in the collection with a "role" value of administrator.

- GET and POST must not be used to tunnel other request methods where tunneling refers to incorrectly using the HTTP methods to limit the client with less HTTP vocabulary.

- GET must be used to retrieve a representation of a resource, PUT must be used to both insert and update a stored resource with a a request message having a body to represent the desired changes. POST must be used to create a new resource in a collection and DELETE must be used to remove a resource from its parent.

- The HTTP Location response header must designate the URI of the newly created resource.

- Custom HTTP headers must not be used to change the behaviour of HTTP methods. For e.g., POST /user/john/address HTTP/1.1 X-HTTP-Method-Override: PUT , this must be avoided and instead of using X-HTTP-Method-Override to override POST, a distinct resource must be used to process the same request using POST without that header.

These rules are of importance for designing any REST API therefore have been considered for this thesis work. The use of these rules has been discussed in a elaborative manner in Chapter 5.

[15] A document resource is a singular concept that is akin to an object instance or database record e.g., `http://api.soccer.restapi.org/leagues/seattle`

[16] A collection resource is a server-managed directory of resources e.g., `http://api.soccer.restapi.org/leagues` - has a collection of *leagues*

[17] A store is a client-managed resource repository. On their own, stores can not create new resources; therefore a store never generates new URIs e.g., `PUT/users/1234/favorites/alonso`

2.5 Hyper Text Transfer Protocol (HTTP)

HTTP is a stateless protocol. It is the communication protocol that transfers representations of resources over a network for e.g., web [ACL+05]. According to the architectural constraints referred by Fielding [Fie00] , Statelessness was one of the six points mentioned. What stateless means is to restricting the server to maintain the state of the interaction between the client and server and rather ensure that the relevant information must be transferred within the request from each client to the server. This constraint improves the properties of visibility, reliability, and scalability. This means that proxies and other intermediaries would be able to participate in communication patterns that involve self-descriptive stateless messages, server death and failover will not result in session state synchronisation problems, and it is easy to add new servers to handle client load again without needing to synchronise session state. Reliability is improved because it eases the task of recovering from partial failures [Fie00] [WWWK94]. Scalability is improved because services do not need to persist state and do not consume memory, representing active interactions hence allows the server component to quickly free resources and further simplifies implementation.

HTTP aka Hyper Text Transfer protocol is an underlying network protocol used by World Wide Web (WWW). It very well defines how the message formatting and transmitting has to happen along with explaining what actions Web servers and browsers should take in response to various commands. For example, entering a URL in the browser, actually sends a HTTP command to the web server directing it to fetch and transmit the requested web page. REST is considered a simple way to organize interaction between dissimilar and independent systems. It is not at all confined to the web but when it has to be implemented for the design of a web-based services, it can use HTTP as a protocol for the interactions [Fie00]. REST and HTTP go hand in hand and therefore as mentioned at 2.4, the CRUD - four generic verbs are used along with REST. HTTP is hence a stateless protocol which means that it executes each command independently irrespective of knowing what the previous command or the previous request including the content and metadata like headers etc. was.

2.5.1 Status Codes

HTTP status codes are "standard response codes" given by the servers/clients when an error occurs over the network in communication. These codes help identify the cause of the problem when a web page or other resource do not load properly. The term HTTP status code is actually the common term for the HTTP status line that includes both the HTTP status code and the HTTP reason phrase-reason or relevant message for the same. The Reason-Phrase is intended to give a short textual description of the Status-Code and is intended for the easy readability of the user. HTTP Codes indicate whether a specific HTTP request has been successfully completed. These responses are grouped in five classes: informational responses, successful responses, redirection, client errors, and servers errors. Some of the basic codes are listed below:

Table 2.1: HTTP Status Codes
[W3S]

Status Code	Status Text	Description
1xx:Information		
100	Continue	The server has received the request headers, and the client should proceed to send the request body
103	Checkpoint	Used in the resumable requests proposal to resume aborted PUT or POST requests
2xx:Successful		
200	OK	The request is OK (this is the standard response for successful HTTP requests)
201	Created	The request has been fulfilled, and a new resource is created
204	No Content	There is no content to send for this request, but the headers may be useful.
3xx:Redirection		
303	See Other	The requested page can be found under a different URL
4xx:Client Error		
400	Bad Request	The request cannot be fulfilled due to bad syntax
401	Unauthorized	The request was a legal request, but the server is refusing to respond to it.
404	Not Found	The requested page could not be found but may be available again in the future
5xx:Server Error		
500	Internal Server Error	A generic error message, given when no more specific message is suitable
501	Not Implemented	The server either does not recognize the request method, or it lacks the ability to fulfil the request.

2.5.2 Methods

RESTful web services expose resources (both data and functionality) through web URIs and they use the four main HTTP methods called CRUD actions namely GET , POST , UPDATE and DELETE to create, retrieve, update, and delete respectively in order to manage and modify the resources. The HTTP verbs comprise a major portion of the "uniform interface" constraint. The primary or most-commonly-used HTTP verbs (or methods) are POST, GET, PUT and DELETE. These correspond to create, retrieve , update, and delete (or CRUD) operations, respectively. Furthermore description of these are:

Table 2.2: HTTP Methods with corresponding CRUD Action

HTTP Method	CRUD Action
GET	Retrieve a resource
POST	Create a resource
PUT	Update a resource
DELETE	Delete a resource

GET

The HTTP GET method is used to retrieve (or read) a representation of a resource. In the non-error path, GET returns a representation in XML or JavaScript Object Notation (JSON) and an HTTP response code of 200 (OK) (table 2.1). In an error case, it returns a 404 (NOT FOUND) or 400 (BAD REQUEST) (table 2.1) for client side-errors whereas 500 (INTERNAL SERVER ERROR) for the server-side error. According to the design of the HTTP specification, GET (along with HEAD) requests are used only to read data and not change it. Therefore, when used this way, they are considered safe. That is, they can be called without risk of data modification or corruption—calling it once has the same effect as calling it several times, or none at all which means GET (and HEAD) is idempotent, which means that making multiple identical requests ends up having the same result as a single request. It is recommended not to expose unsafe operations via GET—it should never modify any resources on the server [RR07].

POST

The POST verb is most-often utilized for creation of new resources. In particular, it is used to create subordinate resources that is, subordinate to some other (e.g. parent) resource, for e.g., a tenant contains tenant users or a shop contains products etc. Basically, when creating a new resource, POST to the parent and the service takes care of associating the new resource with the parent, assigning an ID (new resource URI) etc. On successful creation, return HTTP status 201, returning a Location header with a link to the newly-created resource with the 201 HTTP status (table 2.1). POST is neither safe nor idempotent. It is therefore recommended for non-idempotent resource requests. Making two identical POST requests will most-likely result in two different resources containing the same information.

DELETE

It is used to delete a resource identified by a URI. On successful deletion, return HTTP status 200 (OK) (2.1) along with a response body, accompanying the representation of the deleted item or a wrapped response. It may either return HTTP status 204 (NO CONTENT) with no response body or a HTTP status 200, these are the recommended responses (table 2.1). DELETE operations are idempotent as per HTTP specifications. A DELETE Method actually removes the resource completely. Therefore repeatedly calling DELETE on that resource ends

up the same: the resource is gone. If calling DELETE say, decrements a counter (within the resource), the DELETE call is no longer idempotent.

PUT

PUT is most-often utilized for update capabilities which means PUT-ing to a known resource URI with the request body which now contains the newly-updated representation of the original resource. However, PUT can also be used to create a resource in the case where the resource ID is chosen by the client instead of by the server. which means that the PUT is to a URI that contains the value of a non-existent resource ID. Again, the request body contains a resource representation. On successful update, return 200 (or 204 if not returning any content in the body) from a PUT (table 2.1). If using PUT for create, return HTTP status 201 (table 2.1) on successful creation. It is not necessary to return a link via a Location header in the creation case since the client already has set the resource ID. PUT is not a safe operation, as it modifies (or creates) state on the server, but it is idempotent. In other words, if a resource is created or updated using PUT and then make that same call again, the resource is still there and still has the same state as it did with the first call. If, for instance, calling PUT on a resource increments a counter within the resource, the call is no longer idempotent. However, it is recommended to keep PUT requests idempotent and this can only be done at the server side. It is strongly recommended to use POST for non-idempotent requests.

2.6 Multi-Tenancy

One of the key features of Cloud is to offer services to independent and heterogeneous users or processes and allow them to share common application and computing resources. Sharing of resources promises high cost savings by reducing one time costs and increasing utilization of the resources. Since the highest level of sharing occurs on the application layer, the highest potential of cost savings lies within the SaaS level, where several users share a single application instance [SHK14]. Multi-tenancy describes the approach where tenants, are served and charged as an individual entity while running on a single application instance with other tenants. A tenant, typically a legal body, is defined as a group of users, which share the same view onto an application [SHK14]. It is an approach to enable multiple tenants share the physical available computing resources while being logically isolated. The explanation to this is that when the application instances run in the physical computing environment partly shared among the tenants which can for e.g., be a user organization, a logical computing environment appears exclusively dedicated to each instance (tenant).

Multi-tenancy shares an application instance between multiple tenants by providing every tenant with a dedicated share of the instance, which is isolated from other shares. This isolation embraces for e.g., tenant specific customizations, tenant specific data and tenant specific Service Level Agreement (SLA) [KSAK14] [Koz11]. It is necessary to make an important, but subtle distinction between the concept of multi-tenancy and multi-user. In a multi-user application it is assumed that all users are using the same application with limited

configuration options whereas, in a multi-tenant application, each tenant has the possibility to heavily configure the application .

However, the challenge is the isolation of the tenants amongst each other from the same resource which they work on. In Multi-tenant applications (MTA), the application provider has the task to isolate the data, configuration and the performance observed by the tenants. SLAs play an important role in here to finely define the performance characteristics offered by the multi-tenancy model and hence the pay-per-use feature is supported. Multi tenancy awareness entails being able to differentiate between tenants, provide an appropriate level of data and performance isolation for each tenant, and allow tenants to be served by individually configured and managed services on demand. In this sense, multi-tenancy awareness of the application components and underlying infrastructure is the mechanism enabling multi-tenancy on the level of application [SASL13]. There are two fundamental aspects of multi-tenancy awareness: communication, i.e. supporting message exchanges isolated per tenant, and administration and management, i.e. allowing each tenant to configure and manage individually their communication endpoints at application or service level. Tenant isolation is further decomposed into data and performance isolation between tenants of the same system [ABLS13].

2.6.1 Features of multi-tenancy

As stated in [Nat12], in order to experience the useful features of multi tenancy, certain characteristics have to offered by the cloud providers irrespective of the cloud service model chosen. Although not all the mentioned characteristics are equally critical in all the situations but these are the expected functionalities to be offered:

1. Isolation of the tenant data. No other tenant must be allowed to peek into the data of the other tenant hence, maintains data security and abstraction despite of being able to use the same service or application.
2. Isolation of the tenant workspace. No sharing relating to any of the tenant user's resource for e.g., in form of memory, disk space or any should be provided among the other tenants. It is mandatory to isolate the different resources among the tenants.
3. Isolation of tenant execution characteristics (in terms of performance and availability). Even if one of the tenant consumes most of the resources or has high workload, it should never effect the performance of the other tenant. Similarly, the availability of resources for one tenant must not be hindered by excessive consumption of resources by the other tenant. This must be managed properly by the cloud service provider.
4. Tenant-aware security, monitoring, management, reporting and self-service administration in case of failures has to be handled with utmost care and has to be taken into consideration with back-up restoration mechanisms .
5. Isolation of tenant customizations and extensions to business logic.
6. Continuous, tenant-aware version control.

7. Tenant-aware error tracking and recovery.
8. Tracking and recording of resources usage per tenant (needed for pricing the appropriate tenant as per the resource exploitation done.)
9. The ability to allocate resources to tenants dynamically, as needed hence providing highly scalable environment.
10. Horizontal scalability to support at runtime enabling addition/removal of tenant resources, tenants or users without interruptions to the running environment.

These characteristics ensure increased business agility, productive, seamless and cost-effective tenant experience and provides better usability of the service among multi-tenant architecture.

2.6.2 Challenges

As mentioned above regarding the challenges faced while the implementation of multi-tenancy, it becomes very complex for such applications more than that for single tenant applications. Some of these challenges are explained in an elaborative way by Bezemer and Zaidman [BZ10]:

1. Performance - The very basic idea of multi-tenancy is to share resources both software and hardware among tenants and it might be possible that if any of the resource is clogged up by a tenant, the performance of other tenants gets obstructed. When compared with the situation where the resources are equally distributed among each instance (or tenant) [LLLC08], this might possibly lead to inefficient utilization of resources and hence is not desirable for pure multi-tenant system.
2. Scalability - In a multi-tenant situation it is bizarre to assume that a tenant may not require more than one application and database server. Based on the geographical position of various tenants it becomes further unrealistic to have only one application and database server, hence scalability has to be acknowledged in conditions with various tenants sustaining at different geographical locations. There may be more constraints such as the requirement to place all data for one tenant on the same server to speed up regularly used database queries. Such constraints strongly influence the way in which an application and its data store can be scaled.
3. Security - In a multi-tenant environment, a security breach can result in the exposure of data to other, possibly competitive, tenants. This makes security issues such as data protection [GSH+07] very important.
4. Zero-downtime - It is desirable and inevitable for new tenants to be added and also the adaptation to the business requirement changes. Therefore, the system constantly evolves and grows. However, adaptations should not interfere with the services provided to the other existing tenants in the ideal situation.

5. Maintenance - In the typical evolutionary cycle of software, a challenge is formed by maintenance for e.g., adapting the software system to changing requirements and its subsequent deployment [JBBvN05]. Despite the advantage of a multi-tenant system to deploy minimum number of application and database instances, maintenance of such system adds complexity whose cost can at times dominate the cost of minimizing the number of instances if there occurs very frequent changing requirements in the software system.

2.7 eScience

The term e-Science is used to represent the increasingly global collaborations – of people and of shared resources – that is needed to solve the problems in the fields of science and engineering [HT03a]. The term e-Science can be further explained as the computationally intensive science using immense datasets and is carried out in highly distributed environment. Implicitly, the e-Science applications define a set of computational and data services that the hardware and middleware infrastructure must deliver to enable the science revolution. The hardware will be of many forms: high-speed networks, super-computers, clusters of workstations and expensive shared experimental facilities [HT03b]. Fostering the multi and interdisciplinary nature of science and research requires synchronization and collaboration within the distributed research projects. To achieve this, the scientists and researchers have to use their scientific expertise, software and resources at an individual level. In order to handle and preserve the on-going researches and their results which eventually turn out to be large datasets, a need for support tool has been of one major concern. From data capture and data curation to data analysis and data visualization, e-Science as a way to unify theory, experiment, and simulate helps the researchers and scientists to process data generated by sensors or simulators etc. by software and store the related information or knowledge in the physical entity so as to enable analyse the database/files using data management and statistics.

2.7.1 Role of different Computing environments on eScience

Realising the constantly demanding need for distributed architecture implementation to provide scope for handling larger datasets and use of computational services, eScience considers using the available computing paradigms worthwhile. The various application requirements pertain to various available computing environments , mostly referring to grid computing and cloud computing. Further the impact of choosing the right computing paradigm is explained. e-Science has significantly used Grid Computing due to its characteristics of providing distributed computing paradigm or infrastructure that spans across multiple Virtual Organisation (VO) where each VO can consist of either physically distributed institutions or logically related projects/groups. The goal of such a paradigm is to enable resource sharing in dynamic, distributed environments [FZRL09]. The advent of Cloud Computing led to growing interest of scientific computing teams to move from Grid to Cloud. The major

differentiating factor between the two is that latter focuses more on economy based aiming to deliver more abstract resources and services unlike the former which dealt majorly with providing an infrastructure that delivers storage and compute resources. Ian Foster specified a three point checklist [Fos02] to help define how Grid is differentiable as:

1. Grid coordinates resources that are not subject to a centralized control.
2. It uses standard, open, general-purpose protocols and interfaces.
3. It delivers non-trivial qualities of service. Quality of Service (QoS) measurement is a key to ensuring an application or a service's performance. Compute intensive applications focus on the strength of processing power and memory usage whereas data base applications measures time per query or queries per second. These factors are essential to be taken care of in the scientific computing environments.

The cloud on the other hand supports the third point mentioned above but leaves point one and two in an ambiguous state. Cloud Computing is provided within a single organisation unlike in Grid, which unifies resources within various organisations. Cloud moreover lays emphasis on a client-server architecture. There is always an uncertainty about the future amount of resources that will be available in infrastructures like Grid, which prevents the researchers from planning their activities to guarantee that deadlines will be met. With the emergence of the cloud computing paradigm came new opportunities with the possibility to run e-Science activities at resources acquired on-demand from cloud providers [CALB10]. Such factors have been appealing to the Scientific Computing groups and find Cloud economically stable solution for the computational purposes.

2.7.2 Cloud Burst

Cloud bursting is an application deployment model in which an application runs in a private cloud or data center and bursts into a public cloud when the demand for the computing capacity spikes. The advantage of such a hybrid cloud deployment is that an organization has to pay only for extra compute resources when needed. Cloud burst is a technique used in tandem with the quality of service (QoS) metric which is used to gauge cloud solution scalability and measure software application capability and performance on hosted cloud platforms. Cloud bursting is the process of moving workloads from the private cloud to a public cloud when there are insufficient resources within the private cloud. This in essence means that resources in an enterprise network are augmented with resources in "the cloud" during times when insufficient resources are available in the enterprise network [dMRF+10]. This can be considered a technique to enable load sharing only if the application deployed in the cloud cannot fulfil the need for resources on demand as also mentioned by NIST [MG11] and in the section 2.1.3.

Considering the popularity gained by the cloud computing paradigm, scientific computing applications have drifted from grid computing to cloud prominently and with options like Hybrid clouds or cloud bursts, e-Science has been able expand in terms of resources as and

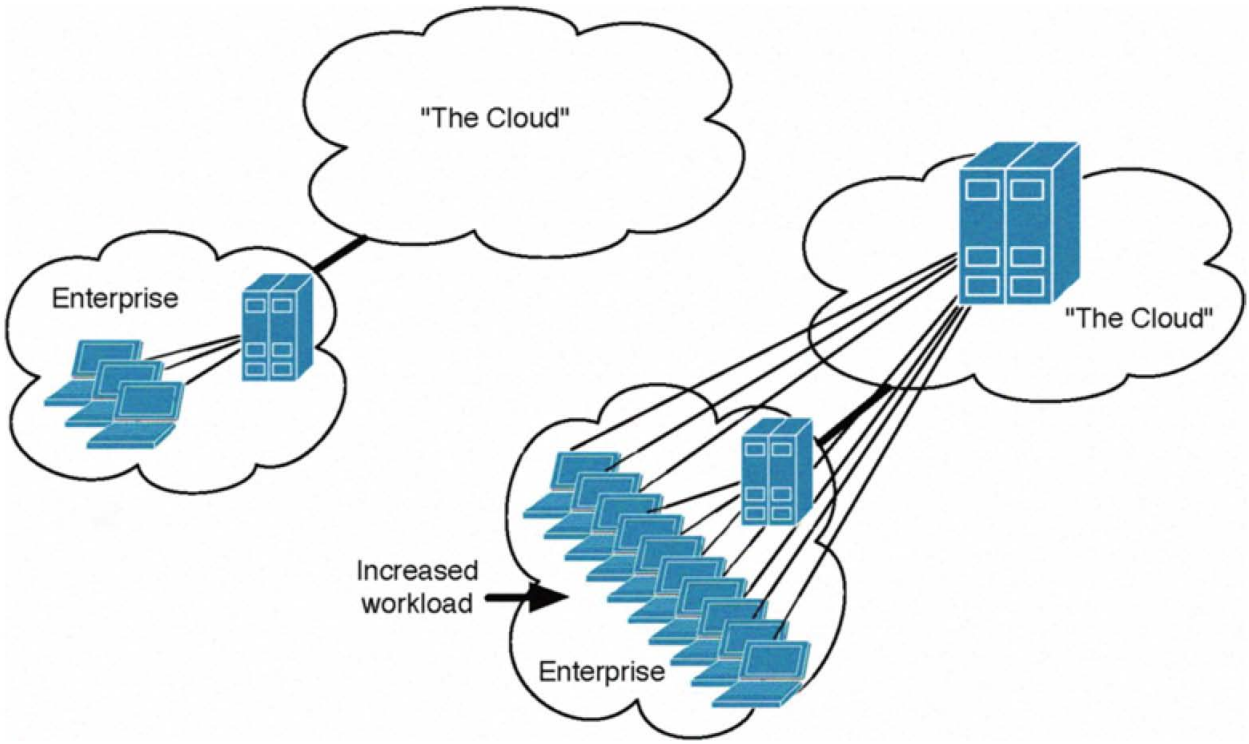

Figure 2.4: Conceptual view of cloud burst
[dMRF+10]

when needed. There are several factors for the increasing need of cloud burst in e-Science, some of them are listed by Dana et al. [Pet13] as:

1. Deal with the peaks in service and resource requests using external ones, on demand basis which solves the issue of uncertainty in expecting the precise amount of resource needed as expressed by Candeia et al.[CALB10] .
2. Optimize costs or improve quality of services.
3. Follow the constraints, like new locations and/or laws associated with the various locations where the application is hosted in the cloud. This eases the concern of scientists at various global locations.
4. Replicate the applications or services consuming resources or services from different Cloud providers to ensure their high availability and needs no involvement of the users.
5. Avoid the dependence on only one external provider.

The cloud burst technique is becoming popular among the business as well as the e-Science environments to provide different methods to dynamically scale the system. Cloud Bursting is a critical way to meet user experience requirements and SLAs without needing to own massive on-premise resources, but this does not mean this process is without challenges all its own. A key consideration with cloud bursting is environment compatibility. Jobs that run in a private cloud may not move easily to a public cloud if there are differences in services,

API support or other inconsistencies between the environments. The challenges also include dealing with differing APIs/policies/UI/tools, troubleshooting load balancing to other VMs, and maintaining control over external cloud resources during and after bursting.

2.8 Computation and Numerical Libraries

Computation libraries would refer to the open source or software libraries provided by various languages to allow numerical calculations and statistical analysis to name a few. These are the libraries used for software developments for performing computational operations. The selection of any of these libraries is dependent on various factors as technology or language chosen for development or is compliant to the other supporting softwares in research purpose(Python, C/C++, .NET, Java), desired features (e.g., for large dimensional linear algebra, parallel computation, operations over complex numbers , vector or large array based calculations , partial differential equations , plotting), commercial/open source nature, portability or platform/compiler dependence (e.g., Linux, Windows, Visual C++, GNU Compiler Collection (GCC)), performance in speed, ease-of-use, continued support from developers, standard compliance, specialized optimization in code for specific application scenarios or even the size of the code-base to be installed. The table below gives a brief overview on the available options consolidated together:

Table 2.3: Computation Libraries in C

Library	**Description**
C	
AmgX	Commercial library of sparse iterative linear solver routines.
BLOPEX	An open-source library for the scalable (parallel) solution of eigenvalue problems.
FFTW	Software library for computing Fourier and related transforms.
GNU Scientific Library	Open source numerical analysis library.

Amongst the mentioned libraries, SciPy has been described further as it is an open-source scientific library, uses general-purpose language(Python) it is relatively easy to connect existing C and Fortran code to the Python interpreter. SciPy is considered in scope for this thesis work. SciPy library contains many fully-featured versions of the linear algebra modules, as well as many other numerical algorithms as compared to the rest. It does include the new features and also has commercial support offered. SciPy uses a variety of methods to generate "wrappers" around the robust scientific algorithms written in C and Fortran which has tremendous difference in the time it takes to solve a problem as compared to using simple algorithm coded in C, hence SciPy is a better choice. SciPy has an active and large community which helps in issues relating to using the library hence, it is actively maintained and higly usable.

Library	Description
C	
GNU Multi-Precision Library	Library for doing arbitrary-precision arithmetic.
hypre	An open-source library of routines for scalable (parallel) solution of linear systems and preconditioning.
IMSL Numerical Libraries	Commercial cross-platform libraries containing a comprehensive set of mathematical and statistical functions.
Lis	A scalable parallel library for solving systems of linear equations and standard eigenvalue problems with real sparse matrices using iterative methods.
SLEPc	A PETSc based open-source library for the scalable (parallel) solution of eigenvalue problems.

Table 2.4: Computation Libraries in C++

Library	Description
C++	
Armadillo	Linear algebra library for matrix and vector maths
Blaze	An open-source, high-performance Math library for dense and sparse arithmetic.
Blitz++	High-performance vector mathematics library.
Boost	C++ libraries for numerical computation.
Dlib	Library for linear algebra and optimization tools which benefit from optimized BLAS and LAPACK libraries.
Eigen	Vector mathematics library.
GMTL	Math library for graphics.
GNU Scientific Library (GSL)	Numerical library with mathematical routines such as random number generators.
IML++	Library for solving linear systems of equations, capable of dealing with dense, sparse, and distributed matrices.
IT++	Library for linear algebra (matrices and vectors), signal processing and communications.

Table 2.5: Computation Libraries in Python

Library	Description
Python	
SCaViS	Scientific libraries based on Jython, Java implementation of the Python language.
matplotlib	MATLAB-like plotting library.

Table 2.6: Computation Libraries in Python and Java

Library	Description
Python	
NumPy	Library for the manipulation of large, multi-dimensional arrays and matrices; it also includes a large collection of high-level mathematical functions. NumPy serves as the backbone for a number of other numerical libraries, notably SciPy.
ScientificPython	Library with a different set of scientific tools
Plotly	Web-based scientific plotting library.
SciPy	Library of scientific tools and also includes NumPy.
Java	
Colt	Set of Open Source Libraries for High Performance Scientific and Technical Computing.
Efficient Java Matrix Library (EJML)	An open-source linear algebra library for manipulating dense matrices.
Parallel Colt	An open source library for scientific computing.
SCaViS	An open-source Java libraries for numerical calculations, data I/O and visualisation of scientific results.
Matrix Toolkit Java	Linear algebra library based on BLAS and LAPACK.
OjAlgo	An open source Java library for mathematics, linear algebra and optimisation.
exp4j	Small Java library for evaluation of mathematical expressions.

2.8.1 SciPy

Python is considered an ideal choice for scientific programming among the other languages as it is a mature, robust, widely-used and very importantly open source. It has a simple, expressive, and accessible syntax [VM08]. Python is often preferred due to its extensibility to all the available operating system platforms and portability. It is hence well suited to a heterogeneous computing environment. Python is also powerful enough to manage the complexity of large applications, supporting functional programming, object-oriented programming, generic programming [RJ05] and meta-programming. Python offers strong support for parallel computing. There is a variety of packages offered by Python to allow the user to span through various fields of computation for e.g., mathematical, statistical , plotting , signal processing and more as show in table 2.6. One of such stable library is SciPy. SciPy is a cross-platform, open source software package for mathematics, science and engineering [GLT09]. It is a collection of mathematical algorithms and convenience functions with additional add-ons for the user with high-level commands and classes for manipulating and visualizing data. It depends on NumPy which also is a fundamental package for scientific computing with Python and provides fast array processing and other significant scientific uses. SciPy is conceptually very similar to Matlab [Mat09], with implementations of many of

Matlab's functions likewise it contains packages for matrix manipulation, statistics, linear algebra as well as signal processing [GLT09]. SciPy also supports Matlab-style plotting and visualization of data through the Matplotlib (which is a plotting library for the Python programming language which produces plots and figures in a variety of hard copy formats and interactive environments across platforms) language extension. The vast library of functions combined with the readability and power of the Python language make SciPy a great tool for quick prototyping as well as for the development of larger applications.

SciPy sub-packages and functions

SciPy is organised into sub-packages covering different scientific computing domains [JOP+01].

Table 2.7: SciPy Package At A Glance

Sub-Package	**Description**
cluster	Clustering algorithms
constants	Physical and mathematical constants
fftpack	Fast Fourier Transform routines
integrate	Integration and ordinary differential equation solvers
interpolate	Interpolation and smoothing splines
io	Input and Output
linalg	Linear algebra
ndimage	N-dimensional image processing
odr	Orthogonal distance regression
optimize	Optimization and root-finding routines
signal	Signal processing
sparse	Sparse matrices and associated routines
spatial	Spatial data structures and algorithms
special	Special functions
stats	Statistical distributions and functions
weave	C/C++ integration

3 Related Works

In this chapter we discuss related works that are relevant to this thesis done whilst using the foundation built from the previous chapter. The use of SciPy library has been widely done for statistical analysis and plotting by researchers. Python being one of the highly acclaimed scripting languages has enabled scientists and researchers to use NumPy and SciPy libraries for computing results and use them for simulations [GLT09]. Similar has been observed by the scientific computing environment at National Energy Research Scientific Computing Center (NERSC)[1] Many other significant scientific computation illustrations have been marked with the need for developing web-based interactions for easy understanding by the scientists and researchers, beneficial for their research works. Further details of the similarities and dissimilarities in the approach towards the design and implementation of such web services for the scientific computation with those accounted in the thesis, is of main concern to this section.

3.1 NEWT: A RESTful Service for High Performance Computing Web Application

National Energy Research Scientific Computing (NERSC) Center is a prominent and renowned flagship high performance scientific computing facility for research sponsored by the U.S. Department of Energy Office of Science. They realized the fact that for ease of understanding and interaction to scientific data and method, web interfaces are very interactive and sophisticated. Though, the jargon and plethora of web decreases the possibility to use such features by the scientists unless they have previous relevant knowledge and skill set for building web applications. As part of its mission to provide high quality resources and services that accelerate scientific discovery through computation, NERSC is helping to build a number of web gateways for science.

3.1.1 Motivation

National Energy Research Scientific Computing Center web toolkit (NEWT) is the NERSC web toolkit which aims at developing a web service to capacitate interaction with High Performance Computing) (HPC) resources via a scientific gateway. This method of interfacing with the HPC center was aimed at enabling rich client side scientific computing applications

[1] National Energy Research Scientific Computing Center is a high performance scientific computing facility for research sponsored by the U.S. Department of Energy Office of Science.

useful for end-to-end user workflows entirely in the web browser [CSB10]. Some of the specific functionalities to be provided to the user through this web tool kit are:

1. Run jobs and enable computing with minimal effort and make it simple and interactive.
2. Access system resources like batch jobs, status, files considering authentication for the access control.
3. Access accounting information from NERSC Information Management (NIM) system.
4. Allows user defined persistent data storage for future references.
5. Run shell commands for enabling computations.

Majority of the HPC still work using command line sessions and shell scripts that run user jobs, post-process output, and perform analysis. Certainly there is also a major section of scientists which have been perceived to be uncomfortable with UNIX command line interfaces hence leaving only experts in command line session handling to run jobs which indeed does limit the cross domain science. This eventually leads to restriction in using already well exposed information from various sources. The rationale focused is to allow users to interact with HPC resources with minimal knowledge of programming the interface. Similar to avoiding the need of installing and building the computational library on each physical entity, requiring minimal knowledge of interacting with language whose library is being used. This abstraction of the technical constraint from the use of method would allow the scientists to use the functions with ease. For example, in this model getting access to a dataset is simply a matter of entering a URI with appropriate parameters. This means that there is no need to understand the XML structure of an application framework, NEWT handles all this automatically on the back-end. The interfaces are more tightly integrated with the scientific applications themselves, thus allowing the users to focus on working directly with their jobs and data. The motivation of this thesis is indeed the same referring to the context of binding the datasets and the other resources to URIs along with appropriate parameters for the computational purpose. Further elaboration of how the URIs are designed as per to locate the resource and parameters needed to deal with the SciPy function that is to be invoked by the user, has been taken care of in the later sections.

3.1.2 Approach and Implementation

NEWT requests consist of a combination of HTTP methods with URIs, parameters, and HTTP headers. NEWT responses are typically comprised of HTTP status codes, headers and JSON (JavaScript Object Notation) objects. As suggested by Cholia et al. [CSB10], the format chosen for encoding the responses was JSON for their application. Asynchronous JavaScript, XML, Asynchronous JavaScript and XML (AJAX) and HTML5 technologies provide all the tools to interface with the HPC application. NEWT is based on the idea of encapsulating the building blocks in the form of HTTP URIs. The set of these URIs and the actions that can be performed on them, give the API. This enables group of scientists to build complete web gateways by using client-side technologies alone. The RESTful architecture style supports using many URIs, often infinite resources, each addressable through at least one URI and exposed through

the uniform interface [RR07]. The features that outweigh REST architecture over others has been explained in section 2.4. The approach to expose the offerings as a web service using RESTful API by requesting resources via URIs is the proposal for this thesis. Though, the responses would be irrespective of the format be it JSON or XML. They would be as per the user's choice to expect a response of a particular format unlike the one explained by Cholia et al. [CSB10]. The data storage managed in NEWT varies from that proposed in this thesis in a way that we have considered storing not just the user information on the internal memory but also the requested computational results by the user are stored on the server. This data storage is in form of data buckets. The access to this data storage is understood by the assigned user roles. This resource handling can be done via the HTTP methods. The other resources relevant to the user for computation are accessed via HTTP methods as well. The desired SciPy function call has to be mapped to the URI and the input parameters to the same are passed as a query string. This maintains the RESTful API design structure.

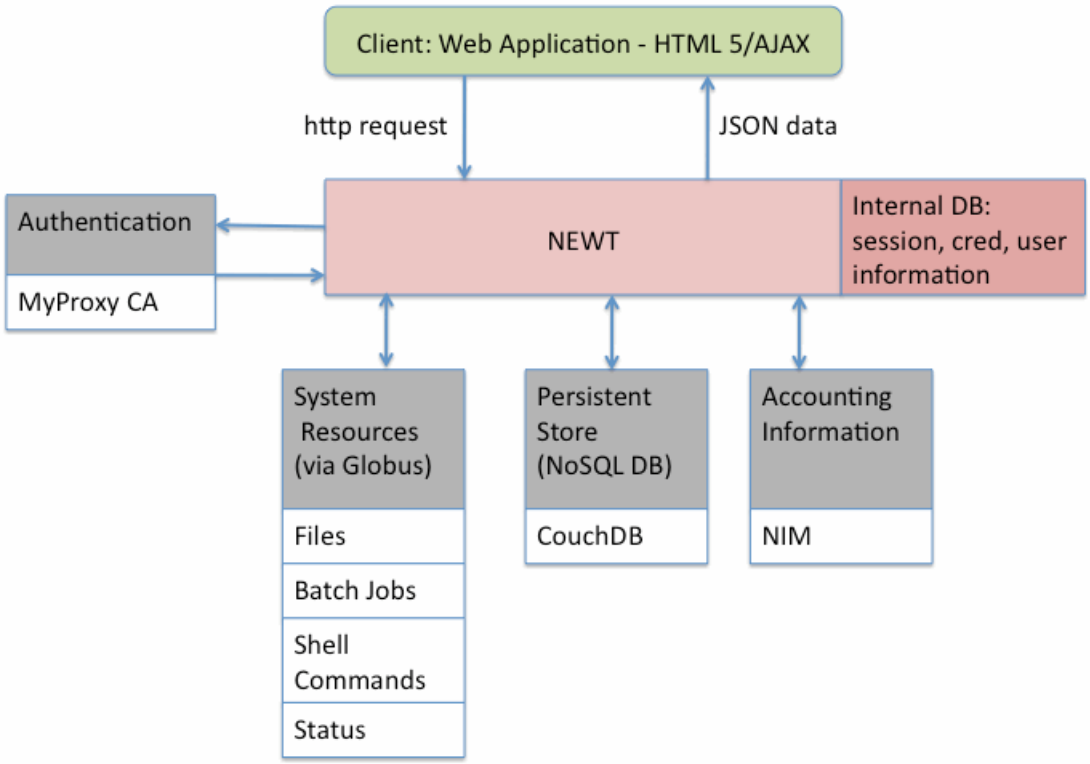

Figure 3.1: Illustration of NEWT – RESTful Services for HPC [CSB10]

The architectural design for NEWT has resources pertaining to the computational functions under the scope of the framework developed along with the data storage and authentication as other resources. Since, the requirement is similar to our work, the approach will be similar too for this thesis.

The REST API is built using the Django Framework for NEWT and also for this thesis. Django offers us a flexible web framework based on the Model-View Controller (MVC) pattern,

which allows us a clean separation between the data models, business logic and front-end [GHJV95][KP88]. It also has a powerful middleware stack that allows us to easily integrate authentication, user sessions and header manipulation into our application. Django does much of the heavy lifting when it comes to URI routing and request/response handling. This framework being one of the most powerful MVC framework in Python and also most widely adopted and documented Python framework helped in structuring the resource model and map them to URIs with ease [HKM09]. Since Django applications are written in Python, it becomes easier to access the rich eco-system of Python based tools and libraries for scientific computation. This thesis clearly focuses on using Python's SciPy library for scientific computation and its functions for statistical analysis.

3.2 Everest: A Cloud Platform for Computational Web Services

Modern scientific researches ask for using applications that implement computational algorithms, methods and models. Nevertheless, the major challenge still remains to handle the complexity and constant need of HPC resources at each step in the research. The ability to reuse existing computational tools becomes a critical factor to influence research productivity. This does also bring in an important issue of application composition so as to allow combination of multiple tools to solve complex problems. Sakharov et al. [SA14] claims that the existing web based scientific environment largely ignore the aspect of providing programming interfaces in the applications and hence is in stark contrast to Web 2.0 applications and cloud computing services. Everest, a cloud platform for Computational Web Services thus spans through the possibilities to combine the approaches to allow programmatic access via web service based APIs and therefore proliferate the scope for sharing workflows for extending the presented application on the level of computational jobs and provide tools for application composition. It is also focused to provide convenient user interfaces to allow scientists for service discovery, invocation and composition while taking in account the security requirements.

3.2.1 Motivation

There have been many successful approaches which focused on provisioning remote access to scientific tools and running computational methods through web interfaces but have limited the opportunities to reuse the application, compose them on the fly and integrate them with external applications [SA14]. Everest's Service Model proposed to focus on supporting management of long-running jobs and transfer of job data. In contrast to generic web service interfaces to computing infrastructures, such as grids, Computational Web Service (CWS) are specialized in running specific applications, i.e., solving specific classes of problems. Therefore a request to CWS normally doesn't contain an executable, but instead represents a set of input parameters describing a problem to be solved and such requests are referred as service-level jobs or just jobs. The job results can be represented as a set of output parameters in the same fashion. The approach in this thesis is the very same except that the result of the

job as coined by Sukhoroslov et al. [SA14], in our thesis is of the requested format as per the user. The main focus of Everest in terms of functionalities provided lists four operations:

1. Job submission (as a set of input parameters)
2. Retrieval of job state and results (as a set of output parameters)
3. Job cancellation
4. Retrieval of service description (including description of input and output parameters).

3.2.2 Approach and Implementation

For Everest, all functionality of the platform is provided remotely using the PaaS model unlike this thesis which at this level is SaaS offering and does not deal with providing the environment needed for computing. Our thesis aims at highlighting the use of web services to implement computational methods at large and avoids providing specific environment for the computation. In contrast to stateful web services typically found in enterprise systems, CWS process each incoming job in isolation. The only state managed by CWS is the state of processed jobs, so all data needed for a job should be provided in a request. While such restriction leaves out interactive session-based applications, this restriction contributes to scalability properties of CWS. Sukhoroslov et al. [SA14] explicitly mentions using REST architectural style over SOAP-based web services in order to avoid service reuse complexities due to SOAP-based web services. Though SOAP style does encourage creation of specialized interfaces and operations which provides a greater flexibility but it hinders the reusability. The described uniform interface follows an approach used by the HTTP protocol to indicate the desired action to be performed on the web resource identified by URI. The service resource in Everest supports the following HTTP methods:

1. GET, which returns service description and the job state and result if available.
2. POST, which performs job submission and returns a job URL.
3. PUT, which enables changing of the job state (e.g., job cancellation).
4. DELETE, which destroys the job resource and deletes its data.

An additional file resource can be introduced to identify files passed to or returned by a service via its parameters. In such case parameter value contains a file URL. This enables passing large amount of data, which is particularly important for scientific computing, via appropriate data transfer mechanisms, such as HTTP, FTP or GridFTP [ABK+05][OI06]. Our thesis allows using data buckets for reusability of the computational results and the parameters are passed for the method to be invoked in the form of a query string. The data representation format for our thesis is both JSON and XML and other arbitrary formats like YAML, MD etc. unlike for the Everest which supports only JSON. This architecture is similar to the our thesis, in terms of how asynchronous job requests are handled and computed and also the data storage. Though specific to our thesis, the data storage is in form of data buckets associated to each user. Sukhoroslov et al. [SA14] have very explicitly mentioned about enhancement of the

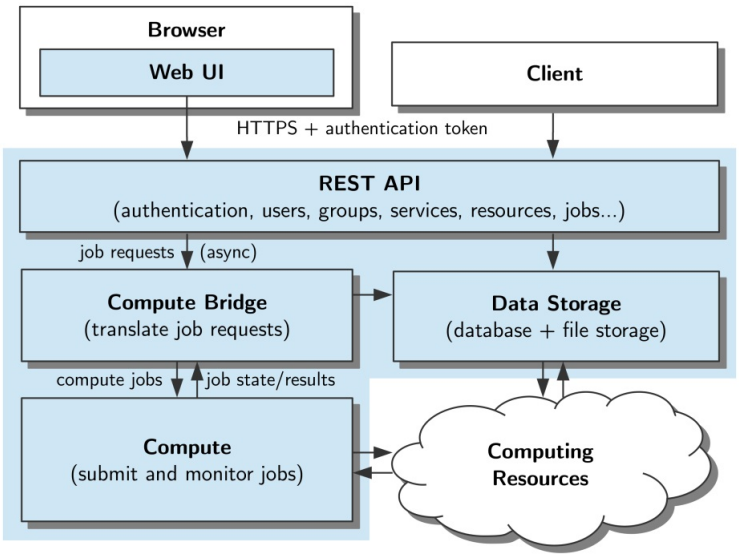

Figure 3.2: Architecture of Everest platform
[SA14]

web UI to ease the use of the application for the researchers and scientists but currently this is out of the scope for our thesis. Also, since there is no need for explicit translation of the format of input parameters in our application, the architectural component of bridge is not considerable for our work.

3.3 MathCloud: Publication and Reuse of Scientific Applications

MathCloud platform enables wide-scale sharing, publication and reuse of scientific applications as RESTful web services. Main components of MathCloud platform include service container, service catalogue, workflow management system, and security mechanism. Many of the problems faced by the scientists and researchers relate to the collaboration within distributed research projects including coordinated use of scientific expertise, software and resources. This requires simultaneous use of several computational codes and computing resources which in turn leads to an increased complexity of applications and computing infrastructures. The reuse of existing computational software can help in influencing the research productivity by volumes. Using the available computing paradigms like grid provide researchers with access to high performance computing resources. But, one of the major problems faced by a modern computational scientist is related to the need to combine multiple applications such as models or solvers in order to solve a complex problem. Therefore to tame the complexity and enable reuse of applications, MathCloud provides a SaaS based application as a service for wide-scale sharing, publication and reuse of available scientific applications.

3.3.1 Motivation

It is very similar in approach to the previously mentioned works but this proposes the cloud offering to be of SaaS due to its features such as ability to run software without installation using a web browser, centralized maintenance and accelerated feature delivery. The ubiquity of SaaS applications and the ability to access these applications via programmable APIs have spawned development of mash-ups that combine data, presentation and functionality from multiple services, creating a composite service. REST has been chosen over SOAP based web services so as to avoid their excessive complexity and incorrect use of core principles of the Web architecture [ASV13]. The objective of our thesis is indeed to provide the services offered by computational library (in this case SciPy) without the need of installing and building it on each physical entity from where the functions would be used.

3.3.2 Approach and Implementation

Afanasiev et al. [ASV13] deemed in their work that service-oriented scientific environments should also emphasize ease-of-use and ad hoc integration of services, therefore implemented computational services as RESTful web services. This REST API is based on the proposed abstract model of a computational service and the design of the API of this thesis is built upon with the identical approach. In our thesis, a client's request is successfully processed when a parameterised description of the SciPy function is provided by a set of input parameters. The result is then represented as a set of output parameters to the client in the desired data format. The interface takes into account features of computational services by supporting asynchronous request processing and passing large data parameters. MathCloud supports three of the HTTP methods unlike in our approach which considers all the four methods. MathCloud does not support PUT method as it restricts the change in any of the resource to maintain the data security. The described interface for MathCloud supports job processing in both synchronous and asynchronous modes whereas our thesis supports only the asynchronous mode for processing the requests. Undoubtedly in both the approaches, if the result can be immediately returned to the client, then it is transmitted inside the returned resource representation along with the indication of DONE state. Thus, a DONE state implies that the user request is successfully processed. If, however for MathCloud, the processing of request takes time, it is stated in the returned job resource representation by specifying the appropriate job state (WAITING or RUNNING). A response contains the state as WAITING if the result cannot be generated as the computing resource is busy managing other previous requests. This can be related as a system busy state. The RUNNING state is responded when due to long calculation, the result is still to be computed and is in the process of completion. In this case, the client uses the obtained job resource identifier for further checking of job state and obtaining its results. Likewise in our structure, the result if not computed in the time out frame, the response to the client refers to the location URI from where the result can later by requested by polling. The result would also be stored into the data bucket of the user and its can also be requested by others if the requester has the permission to share their data.

3.4 e-Science Central: Cloud-based e-Science for chemical property modelling

e-Science Central (e-SC) through its science software as service provides both SaaS and PaaS for scientific data management, analysis and collaboration. e-SC can be deployed both on private and on public cloud. The overall functionality of e-SC is exposed through its Science Software as Service web interface [WoNuTCS10]. e-Science Central is a Science-as-a-Service platform that combines three technologies — SaaS, social networking (to encourage to interact and create communities) and cloud computing (to provide with storage and computational power). Using only a browser, data can be uploaded, shared in a controlled way, and analysis of the data using either a set of pre-defined services, or by uploading tools for execution and sharing. The tools in form of services can be developed for e.g., data analysis and then can packaged and uploaded using desktop service development tool. This service can be included in the workflow. These services can be developed using Java, R [IG96] and Octave [com14], The progress can be recorded in notebooks and the work can be published online or conventionally. Moreover, e-Science Central also gives option for workflow editing and enactment tool to allow automation of analysis through the browser, though this is not an aspect of our work.

3.4.1 Motivation

The SaaS application allows scientists to upload data, edit and run workflows, and share results in the cloud. The wrokflow is used to scalably execute code in e-SC. This feature lets the user decide the services useful for their application and hence, makes it customizable. For enhancing the user experience for scalability, instances of the workflow can be deployed on the cloud nodes. It is underpinned by a scalable cloud platform consisting of a set of services designed to support the needs of scientists. The platform is exposed to developers so that they can easily upload their own analysis services into the system and make these available to other users [HWWL10]. The Software as a Service (SaaS) interface that allows users to access the cloud platform's services entirely through a web browser which is also the paramount factor for our work. Further Watson et al. [WoNuTCS10] explained the browser-only approach suits the new way of working for many scientists, who wish to do their work wherever they are, on a mixture of systems ranging from mobile devices, laptops and desktop PCs, both at home and at work. This provides the user flexibility to acquire the computational results with least effort needed to understand the complexity of the services. The social networking feature offered by e-SC enables users to connect so as to exchange results, our work focuses on doing the same by the use data buckets.

3.4.2 Approach and Implementation

The basic architecture of e-SC has components as data storage, security, services, workflow enactment and provenance. While relating this work to our thesis, the modules data storage,

security or rather roles and privileges with authentication and Services are the adaptations. Though the figure 3.3 shows the use of cloud infrastructure for processing and storage displays

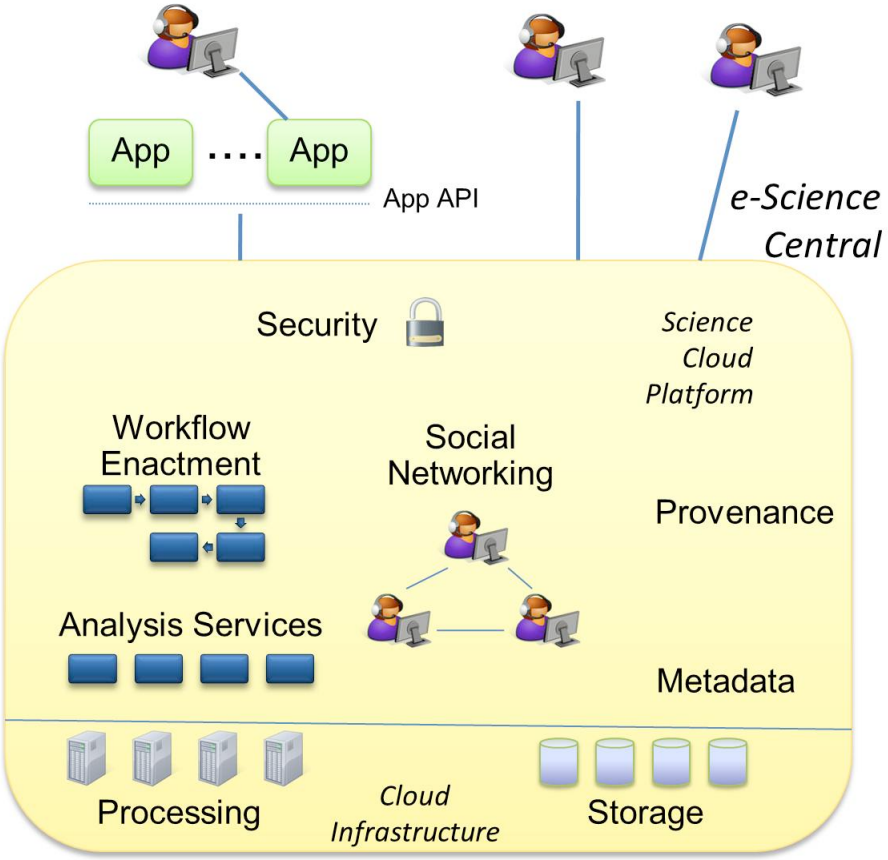

Figure 3.3: Science Cloud Platform [WoNuTCS10]

and the workflow enactment as IaaS and PaaS respectively, in our work service, security and authentication are dealt as SaaS. The data storage in e-SC conceptualizes versioning to restore the previous data as well whereas as a scope for our work, we store the result for the time mentioned by the owner of the data bucket else keep it for a default period of time and then discard it. We do not consider versioning the data in this piece of work, but can be a valid argument for the future work.

3.5 SaaS with Multi-tenancy Support for an e-Contract Management Application

Kwok et al. [KNL08] described the need for exploring SaaS with multi-tenancy support for electronic contract management applications as the commercially viable applications demanded different customized code base has to be developed, deployed and operated to support each tenant. Though few advanced applications did use a single code base with

configuration options to support multi-tenants still, a separate instance of the code base was to be deployed and operated for each tenant even in these applications. These applications were anyways outnumbered by the former application type. Maintaining individual code bases for each tenant is a costly and high maintenance task. Therefore Kwok et al. [KNL08] proposed a new multi-tenancy SaaS model that promises to reduce the application hosting cost and make the application more affordable to the tenants because of its capabilities in customization and scalability while continuing to support an increasing number of tenants. The model also supports our work by providing realistic approach to allow multi-tenancy features for using the same application for various tenants.

3.5.1 Motivation

SaaS can be very well associated with business software applications as it lowers the cost of development, customization, deployment and operation of a software application with feature to support multiple tenants over the internet and contract management applications creates a substantial value for any company. A few characteristics of a multi-tenancy SaaS application include multi-users supports and installation of management accessibility along with maintaining data isolation and security. It benefits the Application Service Providers (ASP) by reducing hosting cost due to customization and scalability with the increasing number of tenants or customers. With a single application code base to support several tenants, the total deployment time is shorter. In addition, updating of application features is simpler and centralized because there is only one instance of a single source code base. The cost to use the application is a per user basis and pays as it goes. However, there are initial setup, configuration and maintenance steps that have to be carried out first in order for the application to support multi-tenants in a SaaS operational structure [KNL08]. Although tenants are using the same building blocks in their configuration, the appearance or workflow of the application may be different for two tenants [BZ10]. Our work emphasizes in this aspect by considering the appearance of the application varying on the basis of shared database. The architectural approach in which the tenants (or users) make use of a single application and database instance is the differentiating factor between multi-user and multi-tenancy and is also explained in the section 2.6.

3.5.2 Approach and Implementation

The architectural framework of the multi-tenancy SaaS electronic contract management application provides the capability of hosting multi-tenants using a single or multiple application instances supported by the same set of computing backend servers. It consists of a web server, a DB2 server and an application server.

The presentation and customization modules of the application are deployed in a web server. Though, in our work the customization of the application is not availed to the tenant users. The security, directory, metadata and shared services, the tenant extensions module as well as the core modules of the application are deployed in the application server and this architecture is similar for our work too. The DB2 server contains a central user account database, a central

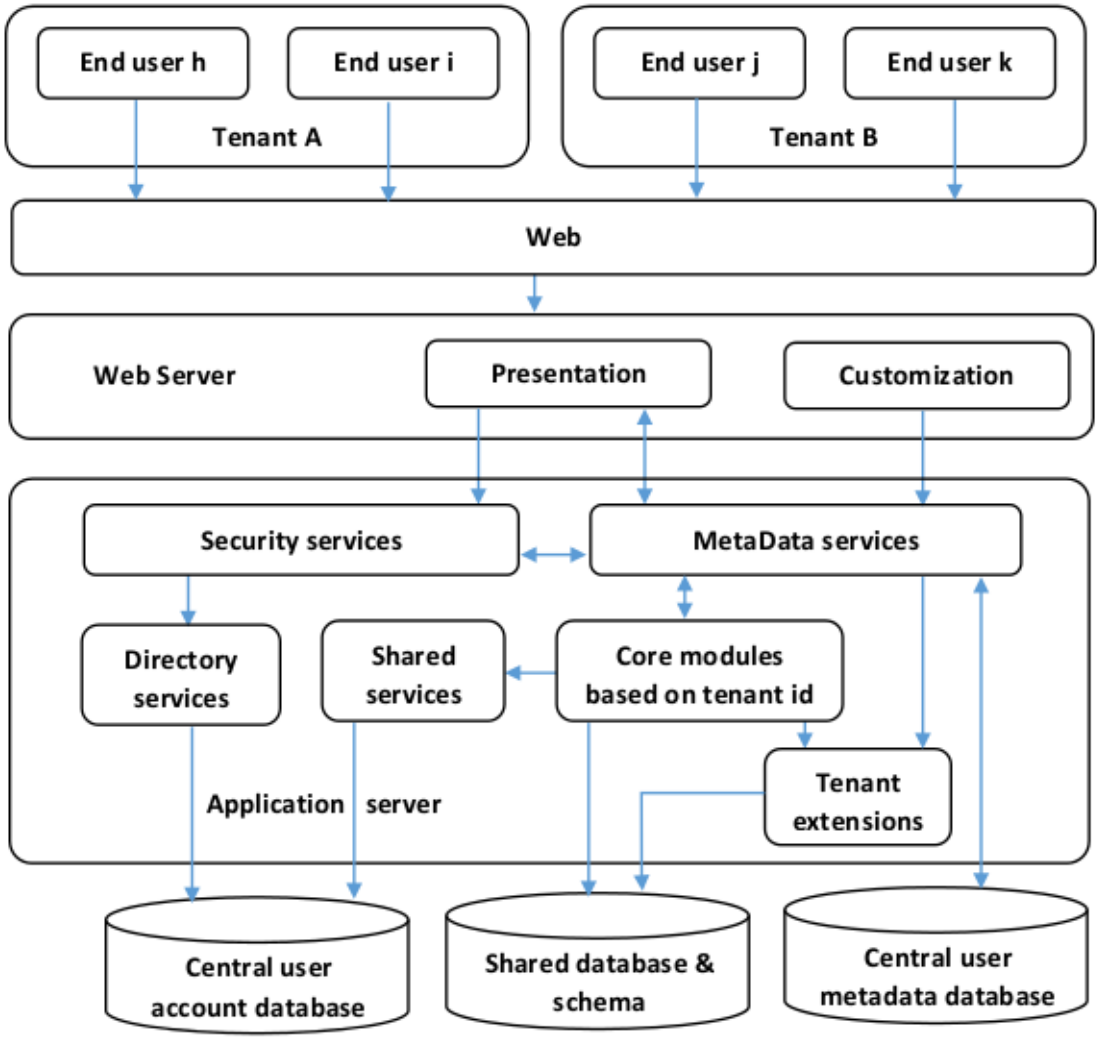

Figure 3.4: Architecture framework of a multi-tenant e-contract management application [KNL08]

tenant metadata database, and a shared database along with the schema. The server contains all the tenant related information, tenant user information and the electronic contracts and their related documents. The server also contains the instances of the workflow designed by the user which can be used for later references. The structure of each component in the above shown architecture is described as:

1. For each tenant the application provider registers an administrator and assign a unique tenant identification (ID) using the security services.
2. The tenant's administrator can create user accounts and register the users.
3. The tenant information is stored on a central tenant metadata database through the metadata services while the end user information are stored on a central user account database through the directory services.

4. When the end user from different tenants access the application through HTTP or HTTPS communications, the security and directory services in the application server have to authenticate and authorize their accesses to the application. The users are tagged with their tenant ID numbers. The identification for an already existing tenant user is done by the authentication service and for the newer members, the registration is done and related user roles are assigned. The users are indeed tagged with their tenant ID numbers.

5. The shared services in the application server provide the metering and billing for each tenant. The shared database and schema in the DB2 server is used as a repository for all the electronic contracts and their related documents.

The proposed data model by Kwok et al. [KNL08], has a shared database with separate schema. Since a multi-tenancy SaaS application shares computing resources and software codes among all the tenants on the same set of servers, the data of the tenants are also hosted in the same database with the same set of tables in this approach. This is the condition for our work too hence, it is taken into consideration. This approach uses a tenant ID in a table column to associate every record with the appropriate tenant. Each record has a number of pre-allocated custom fields (typed or un-typed) that tenants can use for their own purpose. The tenant specific data is stored in a fixed extension set with these pre-allocated fields.

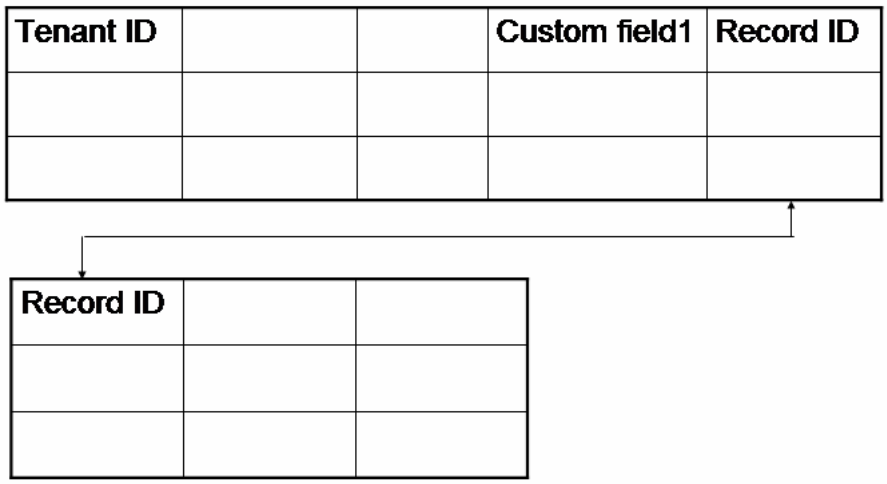

Figure 3.5: A shared database and schema with custom extensions [KNL08]

The multi-tenancy SaaS electronic contract management application also uses an enhanced approach to manage multi-tenant data approach for shared database and schema with custom extension where the tenant specific data is stored as name-value pairs in a separate table. The tenant record with tenant data is assigned a unique record ID that matches one or more rows in a separate extension table. It allows tenants to extend the data model arbitrarily. A data type identifier is needed in the third column, as data in the tenant specific data table cannot be typed. However, this enhanced approach has an added level of complexity for

database functions, such as searching, indexing, query, and updating records. Since in our approach, we consider implementing data bucket then the content or data stored by any user are recorded and is available to the user via the assigned bucket. There are some challenges even with the implementation of multi-tenancy architecture which have been explained in the section 2.6.2. In this particular approach, database security and data isolation, such as permissions, trusted database connections, secure database tables, tenant view filter and data encryption are the majorly faced challenges [KNL08]. Features like scalability, such as scaled up by moving the application to a larger, more powerful server, scaled out by running the application on more servers and scaling the data by partitioning the data into smaller segments in order to improve the efficiency of queries and updates or by replication [KNL08] have been stated as the main areas of investigation for the betterment of the application.

4 Concept and Specification

This chapter details about the requirements for the system design. Categorized into functional and non-functional requirements, we later dive deeper into the system overview with brief explanation to the components involved. The use cases are listed some of which are extensions to the use cases described in [Muh12].

4.1 Requirements

This work aims to provide a set of its offered functionalities as Web services that can be accessed by multiple users using standardized messaging protocols. To accomplish the usability of the application, we focus in this thesis performing computations using standard libraries and generating results, mechanism to store data in the Cloud for retrieval or sharing and isolating the data between different users. This is indeed in scope of the application to enable data sharing for computational analysis or referencing, suitable data storage model has to be designed. The main focus for this thesis study is to allow users to run computational libraries in a Cloud premise rather than on-premise to significantly decrease the computational time and simultaneously decrease the cost for running such analyses. Hence, for the application to run and access statistical analysis tools in the Cloud, the requirements are defined in the further detailed sections.

4.1.1 Functional Requirements

The system is built to provide service to perform statistical analysis useful for generating computational results, hence there exists some functional requirements ensured by the system for the user. The following are:

- Possible Statistical functions provided - The functions provided in the SciPy library are available for the user for computation. The input parameters for the function need to be specified by the user in order to enable the calculation.
- Temporal storage of the result - In order to provide results for the calculations beyond a time out limit, the user is provided with a location address in the response so as to be available for accessing it after the calculation is completed. The user needs to request the mentioned resource address to get the computed result.
- Resource sharing - The provided resources in the system are made available to the user for complete or partial sharing if demanded. This is an approach to ensure resource

availability to all the users. Though the sharing is restricted, in a sense that if the user permissions do not include sharing the resources then this is not achievable.

- Result in specific format - The system allows the user to receive the response for the computation to be in a specific format, be it JSON or XML. The expected format has to be proposed as a part of the request to ensure the response is in the specified format.
- Multi-tenant architecture support - The system has to support multi-tenancy that is, the users from various tenant must be able to use the service and store their data in the databuckets. The provision to share the databucket and the result must be provided as well. The requirements based on multi-tenant architecture are discussed in detail in the section 4.1.2.
- Monitoring the data store - The user of the service is provided some data storage for keeping the computed results. The limitation on the resource consumption is maintained by the provisioning of the resource. But to ensure that a user must be well informed about the consumption of the resource before time, suitable monitoring over the data store must be provided. Though this is not in scope of this work but it is a significant factor to maintain stability. The user must be sent out triggers to inform them about the resource utilization.

The other functional requirements specific to the multi-tenant architecture are described in detail in the next section.

4.1.2 Multi-Tenancy

In this section, we detail the multi-tenant requirements the system must fulfil in order to ensure tenant isolation at data storage level. From the user experience, Quality of service (QoS) and administration perspectives, the tenant naturally desires to access and use the service as if there were dedicated ones. Therefore, isolation should be carefully considered in almost all parts of architecture design, from both non-functional and functional level, such as security, performance, availability, administration etc. [GSH+07]. Multi-tenancy supports that all the tenants share a single application instance in a single hosting environment. The system must ensure a multi-tenant aware transparent access to data hosted in the Cloud. Compared to the Isolated Tenancy Hosted Applications (ITHA), Multi-tenant enabled application (MTEA) has to ensure not only the stored data to be kept apart between the operators within a single tenant but also between the different tenants. Since, MTEA has to safeguard against intrusion among tenants, tenant authentication is indispensable in MTEA. The storage support in the service for hosting the tenant's data and also sharing the stored data but only to the authenticated tenant users, is implemented via a multi-tenant storage model. This relates to the significance of authentication and access control at both the tenant and at the tenant user levels. There are two distinct layers of roles participating in the system [Muh12]:

- The system role differentiates between system administrators and tenant users, with each tenant user belonging to one tenant. Both system roles are mutually exclusive and

users of one system role may have a completely separate resource allocation from users of the other system role.

- The tenant role classifies the tenant users into tenant administrators and tenant operators.

User Roles

In the application, there are three actors namely: system administrator, tenant administrator and tenant operator. System administrator has the authority to create tenants and hence provide access to the members for respective tenant. The system administrator has the control over the resources and hence assigns the resource which in our work is databucket to the tenant. The tenant administrator has the privilege to create tenant operator for their own tenant and further more distribute the tenant resources within the group of users. The tenant operator only then can register for the service and use it for computational purpose. The detailed description of each role is the following:

1. System Administrator - This role does not belong to any tenant but is responsible to manage the resources that are to be consumed by the tenants. The right to add a tenant and remove it lies with System Administrator. The tenant is identified in the resource model via a unique identifier called the tenant ID which also is used to identify the member to any tenant. The registration of the tenant is for authentication and control to enable isolation of data among various tenants. As a scope for this thesis, it is allowed to share data within tenant users of the same tenant and also with other tenant, but as per the design model it is required to identify the tenant from which the data is to be requested. A system administrator can also perform the operation of a tenant administrator and a tenant operator which is to access resources. This role has the maximum privileges and permissions compared to other roles. The system administrator has unlimited permissions and consequently is allowed to interfere in the actions of the tenant users [Muh12]. Another important task that handled by the system administrator is to keep the tenant data together.

2. Tenant Administrator - This role belongs to a tenant in particular. As the name suggests, the tenant administrator provides access to the resources sanctioned to the tenant by the system administrator further to the tenant operators. The creation of the tenant operator and also the removal of a tenant is managed by the tenant administrator. This role is obligated to distribute the resources among the members of their tenant and has the permission to increase or decrease the quota of resources for an individual tenant operator. The definition for the role of a tenant administrator and tenant operator and therefore, subsequent assigning of that role to any other tenant operator is also done by the tenant administrator. A tenant can have only one tenant administrator but multiple tenant operators are considered in our work. The tenant administrator can also assign this role to some other tenant operator within the tenant, if wishes to. A tenant administrator has access to all the services available to a tenant operator.

3. Tenant Operator - This role is lowest level in the hierarchy of roles participating in our system. The Tenant Operator as assigned by the Tenant Administrator can register the service so as to start using it. The application as is aimed for performing computational calculations in the Cloud environment, allows the Tenant Operator to request for the result of a computational library function by specifying the function and providing the input parameters for the same. The Tenant Operator can also request to save the computed result for future reference or for sharing with the rest users. Similarly, the Tenant Operator can request an already computed data from other operator from within the same tenant or from another tenant.

4.1.3 Communication Requirements

The system in our work deals with the request-response operations only, there is no other mode of communication between the service and tenants specifically. The functional requirement associated for the communication between the tenants and data store and also among the tenants are as follows:

- Tenant-aware correlation -In the request-response operations, the response obtained from the backend data store must be correlated with the tenant's request to the system, and ensure that one tenant does not receive responses from another tenant's request [Sá12].

- Tenant configuration isolation - The organizational data of the tenant users existing in the system should be isolated between tenants [Sá12]. A tenant's configuration data contains sensible information which identifies and allows access to the tenant's backend data stores. Each tenant user is provided with a data storage resource named as databucket for storing their results. Since each tenant has a set of databuckets assigned, accessing data from other tenant's databucket is a functional requirement, the tenant users have the read access to all the data of other tenants as well. This permission allows to read the specific tenant operator's ID so as to request the data from databucket. Since this data is visible to all, proper encryption of user credentials has to be done for authentication and security purposes.

- Transparent tenant-aware creation - Since the system administrator has the permission to add or delete a tenant and provide resources to them, this data must be maintained with read access to the tenant users necessarily because they need to be aware of the number of tenants using the service. This information is stored in the tenant registry. Hence, the tenant registry is available to all the users for viewing. This functionality gives a transparent view to the system to share the data among all the tenants on requesting. The information regarding each tenant user, their role, their permission and the databucket information is stored in the configuration registry. Only system administrator and tenant administrator have the privilege for both read and write in the configuration registry but the tenant operator must only be given the right to read the information.

4.1.4 Storage Requirements

A very significant purpose of work for this thesis is to enable storing the result of the computation performed and enabling sharing the results. Maintaining multiple database servers and each server multiple database instances must contain the necessary meta-data to provide tenant-aware routing in the system, and ensure that only one tenant can access the information in their own database instance. For accessing the other tenant information is only for the purpose of knowing the tenants who are using the service to be able to request for the computed results by them. In our system, we have not considered maintaining different database servers over Cloud but the implementation does incorporate multiple database instances to enable dedicated data stores for the tenants. The functional requirements to implement and render data storage for the multi-tenant architecture are:

- Resource provisioning for data storage - System administrator is entitled to distribute the resources among the tenants. For storing data, a databucket is assigned to each tenant by the system administrator by defining its capacity. This databucket is further divided between other tenant operators by the tenant administrator. The feasibility for complete or partial sharing of the databucket is to be enabled considered as a non-functional requirement fulfilment. This is provided by the Tenant Administrator between the Tenant Operators only within same tenant. This sharing of databucket can only be done among users within same Tenant to ensure secured data within a tenant.
- Tenant data isolation - In a multi-tenant architecture, it is of utmost priority to ensure no data peeking among different tenants. The data model considered for our work is Shared DB, Separated Schema [XBXW10] and [KNL08] hence, due to shared DB the data isolation has to be dealt with importance. Therefore, the data among different tenants has to be secured and should be made available only within tenant in general. The data can be shared among the tenants only when requested. It is of prime importance to provide the tenant ID from which the data is requested.

4.1.5 SciPy Library

The available functions to the user for the computational purposes are formed into groups and are clubbed under one defined package and sub-package. The SciPy library offers multiple packages which are divided on the basis of functionality they offer. The packages included are listed in the table 2.7. These packages furthermore contain sub-packages which enclose the functions like integrate, file read and write, interpolate, fourier transform and etc. within. The user for this service is restricted to use only for four packages mentioned in the section 5.2 and includes all the sub-packages and functions inside them as the scope of this thesis. These packages have been chosen as they include the basic functionalities for the numerical calculations and provide the functions for statistical analysis. It is also assumed that the user is well aware of the function a user wants to use and hence, provides the correct input parameters for the function. The categorization of the packages and sub-packages has been done depending on the functions offered and/or on the mathematical

calculations they perform. Packages like *Stats*[1] provide all the possible function relating to plotting and statistical analysis whereas "Linear Algebra" provides basic arithmetic properties and calculations.

4.1.6 Non-Functional Requirements

MTEA are considered difficult to design as it is of prime importance to provide application level isolation among different tenants. As stated earlier, at both functional and non-functional levels in the architectural design isolations relating to Security, Performance, Availability and Administration are to be maintained. The following are the non-functional requirements:

- Security - This is of paramount interest when the system is multi-tenant that means other tenants must not be able to access other tenant's data. The tenant context contains not only tenant information but also sensitive data such as access credentials, backend data source names. It is a necessity to secure this data from third party access or any malicious attack also.
- Performance - Performance in MTEA can be hindered greatly with the increase in number of tenants accessing the same application. If there are large number of tenants accessing SaaS at the same time then it is necessary to implement isolation of processes to sustain performance for an application. Some of the tenants require to have higher priority than other tenants. To manage some static priority and isolation among tenants, web server application pooling can be used [ACK12]. SaaS application would be accessed simultaneously by different tenant users, leading to concurrency issues while reading the tenant configuration. Extreme load on the system can result in late responses, hence effecting the performance. Load Balancers should be provided and appropriate caching mechanisms can be put in action to avoid performance degradations.
- Availability - The application must be up and running for all the users. The availability is percentage of the time a user can access the service [GVB11]. The application availability is irrespective of the user's geographical location and as per SLA, the resources should made available to the users. The system is bound to adhere to the SLA.
- Scalability and Elasticity - Both considered as the feature of Cloud refers to flexible extension of the resources as and when needed to meet the user needs. Scaling up and down the resources if the requirement arise should be accompanied with automated data back-ups, monitoring and error reporting mechanisms on the fly. When the need increase, more resources must be provided and the application must be available to the user to balance the load without them even knowing. When the load level is down, the extra resources must be scaled down to serve economic satisfaction among the users. Elasticity is defined in terms of how much a Cloud service can be scaled during peak times. This is defined by two attributes: mean time taken to expand or contract the service capacity, and maximum capacity of service. The capacity is the maximum number of compute units which can be provided at peak times [GVB11]. The key point

[1] http://docs.scipy.org/doc/scipy/reference/stats.html

about elasticity is that scaling up and down occurs automatically depending on the load therefore, the architecture of the application must allow this.

- Resource Management - The resource provided to the user are defined in the SLA. To realize the 'pay-per-use' feature for the Cloud based applications, the resources must not outnumber the value defined in the agreement to track the billing for the resource consumed by the user.

The functional and non-functional requirements have been mentioned already therefore, now the relevant use cases for the system are described in the next section.

4.2 Use Cases

As described in the section 4.1.2, there are three roles identified namely: system administrator, tenant administrator and tenant operator. The use case depicts all three roles with the possible actions each can perform. System Administrator being the most privileged role can perform all the actions of the other two roles as well. Tenant Administrator defines permissions for itself and for Tenant Operator and can perform actions of Tenant Operator as well. A Tenant Operator has the least privilege and can perform the actions shown in the use case. The permissions are set by Tenant Administrator for them.

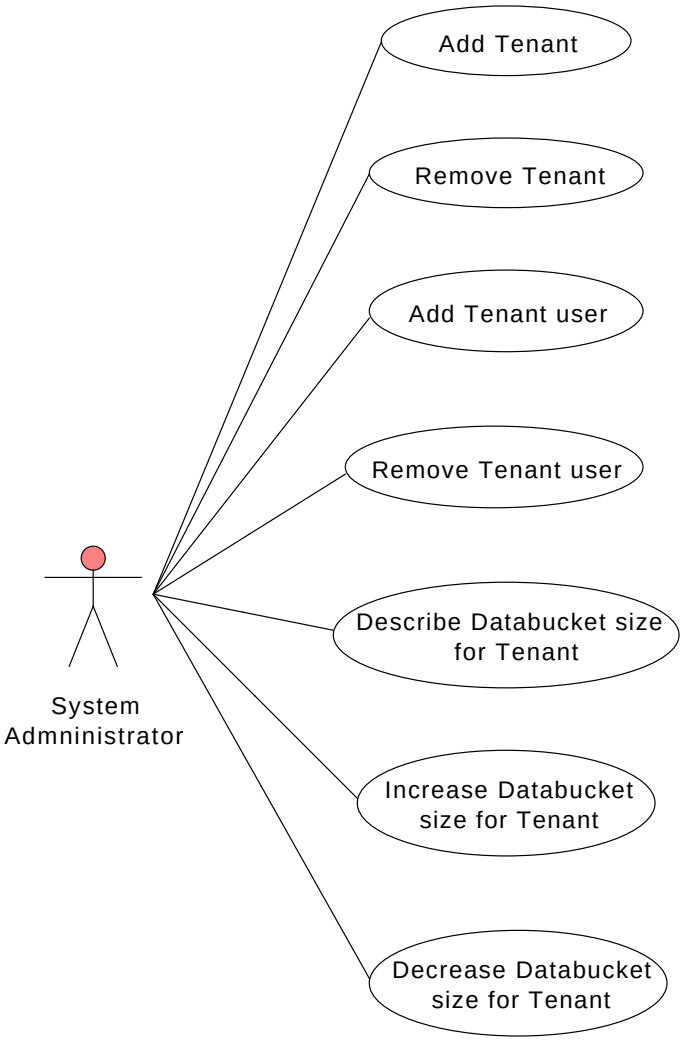

Figure 4.1: Use Case diagram for System Administrator

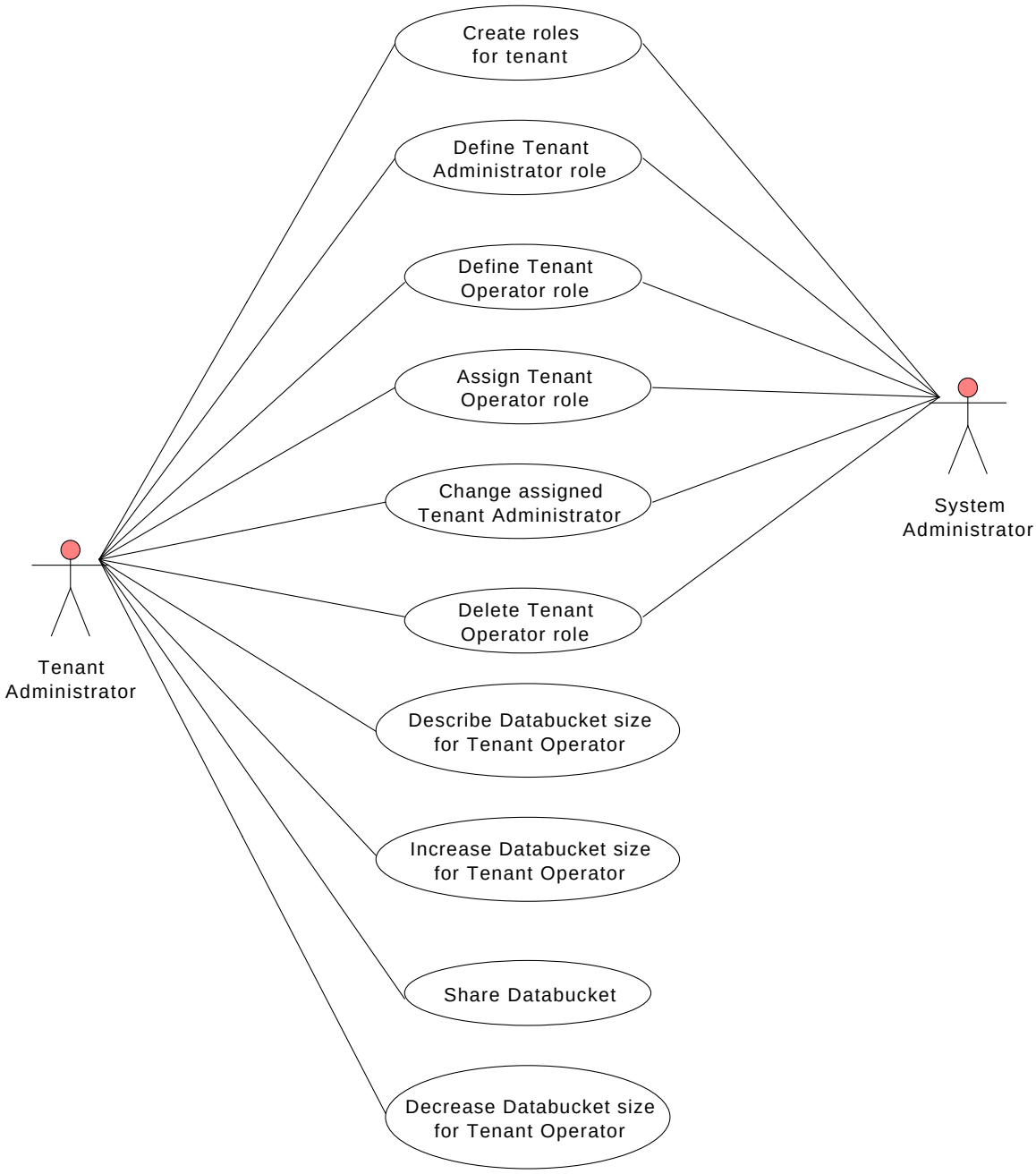

Figure 4.2: Use Case diagram for Tenant Administrator

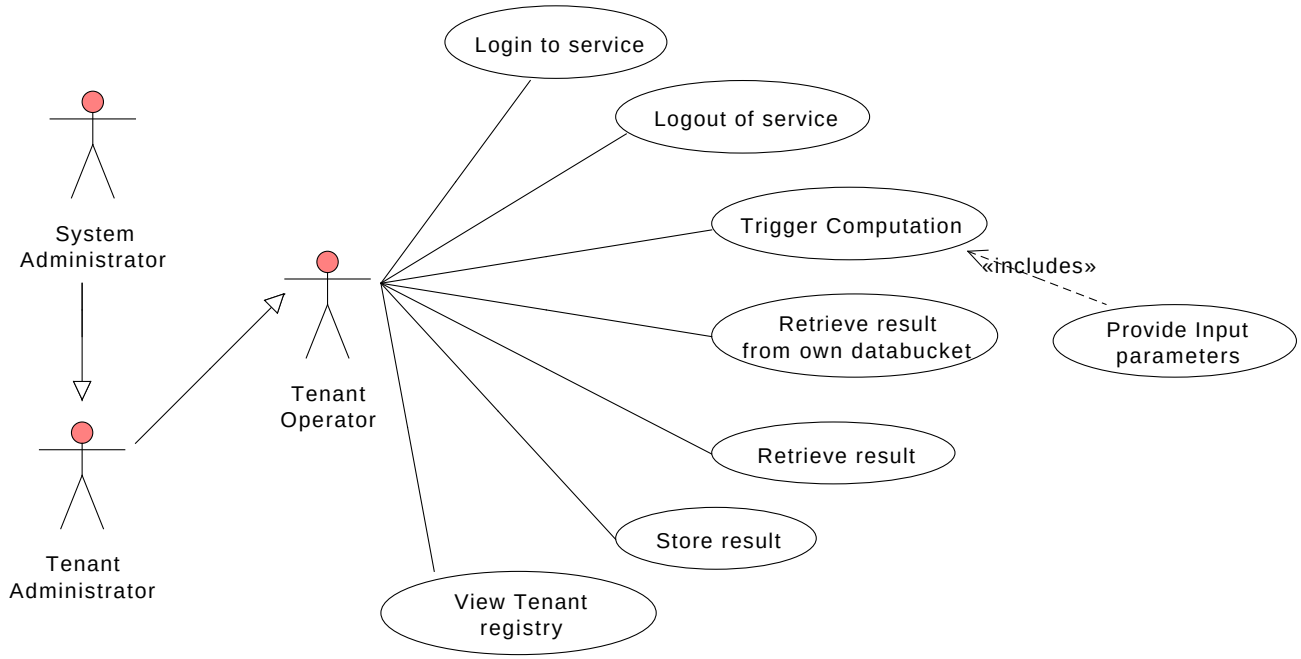

Figure 4.3: Use Case for diagram Tenant Operator

Name	**Add Tenant**
Goal	Adding the new tenant to the system.
Actor	System Administrator.
Pre-Condition	The tenant does not already exist in the system.
Post-Condition	The new tenant is added to the tenant registry and unique identifier as tenant ID is provided to the tenant successfully.
Post-Condition in Special Case	The tenant registry is not updated and no new tenant is added.
Normal Case	1. The system administrator adds a new tenant to use the application. 2. The tenant is given a new identifier as tenant ID in the registry. 3. The tenant registry is updated with new tenant details for keeping the record.
Special Cases	1a. The System Administrator is unable to add the tenant. a) The system shows an error message and aborts. 3a. The registry is not updated with any new data. a) The tenant is already a part of registry.

Table 4.1: Description of Use Case *Add Tenant*.

Name	**Remove Tenant**
Goal	Removing an existing tenant from the system.
Actor	System Administrator.
Pre-Condition	The tenant must be registered in the system.
Post-Condition	The tenant is removed from the tenant registry and the tenant users for that tenant are also removed successfully.
Post-Condition in Special Case	The tenant registry is not changed and no tenant is removed.
Normal Case	1. System searches for the tenant to be removed. 2. Each tenant user related to that tenant are removed as well from the system by the System Administrator. 3. The resources provided to the tenant are freed. 4. The tenant registry is updated with the tenant details removed for keeping the record.
Special Cases	1a. System is unable to find the tenant. a) The system shows an error message and aborts. 2a. The System Administrator is unable to remove the tenant users. a) The system shows an error message and aborts. 3a. The resources can not be freed. a) The system shows an error message and aborts. 3b. The resources can not be freed. a) The resources are still occupied by the tenant users. Appropriate error message must be shown. 4a. The system is not updated with any tenant removed. a) The system shows an error message and the tenant is still in the registry.

Table 4.2: Description of Use Case *Remove Tenant*.

Name	**Add Tenant user**
Goal	Adding the tenant user in the system.
Actor	System Administrator.
Pre-Condition	The tenant user does not exist in the system already.
Post-Condition	The new tenant user is added to the system and an identifier is provided to the tenant user for a tenant successfully.
Post-Condition in Special Case	The tenant registry is not updated and no new tenant user is added.
Normal Case	1. The System Administrator adds a new tenant user for the provided tenant ID. 2. Each tenant user is given a new identifier in the registry for a tenant. 3. The tenant registry is updated with new tenant user details for keeping the record.
Special Cases	1a. The System Administrator is unable to add the tenant user. a) The system shows an error message and aborts. 1b. The provided tenant ID does not exist. a) The system shows an error message and aborts. 3a. The registry can not be updated. a) The tenant user is already a part of registry.

Table 4.3: Description of Use Case *Add Tenant user*.

Name	**Remove Tenant user**
Goal	Removing existing tenant user from the system.
Actor	System Administrator.
Pre-Condition	The tenant user must be registered in the system.
Post-Condition	The tenant user is removed successfully.
Post-Condition in Special Case	The tenant user is not removed from the system.
Normal Case	1. System searches for the tenant ID and tenant user ID to be removed. 2. The resources provided to the tenant user are freed. 3. The system is updated with the tenant user details removed for keeping the record.
Special Cases	1a. System is unable to find the tenant user. a) The system shows an error message and aborts. 1b. System is unable to find the tenant. a) The system shows an error message and aborts. 1c. The system administrator is unable to remove the tenant user. a) The system shows an error message and aborts. 2a. The resources cannot be freed. a) The resources are still occupied by the tenant user. Appropriate error message must be shown and aborts. 4a. The registry is not updated with the specified tenant user removed. a) The system shows an error message and the tenant user remains in the registry.

Table 4.4: Description of Use Case *Remove Tenant user*.

Name	**Describe databucket size for the tenant**
Goal	Resource allocation for data storage to all the tenants.
Actor	System Administrator.
Pre-Condition	The tenant must be registered in the system.
Post-Condition	The size of the bucket for data storage per tenant is defined.
Post-Condition in Special Case	The size of the bucket is not defined and no value is specified in the system for the same.
Normal Case	1. System checks for tenant entry whether existing or not. 2. The size of the databucket is defined for the particular tenant. 3. The assigned size is updated in the system for reference.
Special Cases	1a. The tenant entry does not exist in the system. a) The system shows an error message and aborts. 3a. The entry in the system could not be updated. a) The system shows an error message and aborts.

Table 4.5: Description of Use Case *Describe databucket size for the tenant*.

Name	**Increase Databucket size for the tenant**
Goal	Change in resource allocated to the tenant for data storage.
Actor	System Administrator.
Pre-Condition	The tenant must be registered in the system.
Post-Condition	The size of the bucket for data storage for the provided tenant ID is increased.
Post-Condition in Special Case	The size of the bucket remains same as before in the system for the given tenant ID.
Normal Case	1. System checks for tenant entry. 2. The size of the databucket is increased for the particular tenant. 3. The increased size is updated for reference in the system.
Special Cases	1a. The tenant entry does not exist in the system. a) The system shows an error message and aborts. 2a. No extra resource can be provided to the user. a) The system shows an error message and aborts. 3a. The entry in the tenant registry could not be updated. a) The system shows an error message and aborts.

Table 4.6: Description of Use Case *Increase Databucket size for the tenant*.

Name	**Decrease Databucket size for the tenant**
Goal	Change in resource allocated to the tenant for data storage.
Actor	System Administrator.
Pre-Condition	The tenant must exist in the tenant registry.
Post-Condition	The size of the bucket for data storage for the provided tenant ID is decreased.
Post-Condition in Special Case	The size of the bucket remains same as before in the registry for the given tenant ID.
Normal Case	1. The System Administrator checks for tenant entry in the tenant registry. 2. The size of the Databucket is decreased for the particular tenant. 3. The changed size is updated in the tenant registry for reference.
Special Cases	1a. The tenant entry does not exist in tenant registry. a) The system shows an error message and aborts. 3a. The entry in the tenant registry could not be updated. a) The system shows an error message and aborts.

Table 4.7: Description of Use Case *Decrease Databucket size for the tenant*.

Name	**Create roles for tenant**
Goal	Create roles named Tenant Administrator and Tenant Operator for the tenant.
Actor	Tenant Administrator.
Pre-Condition	The tenant must be registered in the tenant registry and the tenant must have an assigned Tenant Administrator already
Post-Condition	The Tenant Administrator and Tenant Operator roles are created.
Post-Condition in Special Case	The roles are not created.
Normal Case	1. The Tenant Administrator creates two roles named Tenant Administrator and Tenant Operator. 2. The roles are created for assignment to users in the Configuration registry.
Special Cases	1a. The role as Tenant Administrator not created. a) The system sends an error message and aborts. 1b. The role as Tenant Operator not created. a) The system sends an error message and aborts. 1c. None of the role created. a) The system sends an error message and aborts. 1d. Roles other than Tenant Administrator and Tenant Operator also created. a) The other roles are discarded and are not added in the Configuration registry 2a. The roles not added to the Configuration registry. a) The system sends an error message and aborts.

Table 4.8: Description of Use Case *Create roles for tenant*.

Name	**Define Tenant based roles**
Goal	Create permissions for the role of Tenant Administrator and Tenant Operator for the tenant.
Actor	Tenant Administrator.
Pre-Condition	The tenant must be registered in the system and the tenant must have an assigned Tenant Administrator already.
Post-Condition	Tenant Administrator and Operator role is defined.
Post-Condition in Special Case	Tenant Administrator and Operator role is not defined.
Normal Case	1. The Tenant Administrator defines role named Tenant Administrator and Operator. 2. The permissions are added to the Configuration registry.
Special Cases	1a. No role as Tenant Administrator found. a) The system sends an error message and aborts. 1b. No role as Tenant Operator found. a) The system sends an error message and aborts. 2a. The permissions are not added to the Configuration registry. a) The Configuration registry is not updated.

Table 4.9: Description of Use Case *Define Tenant based roles*.

Name	**Assign tenant operator role**
Goal	Provide a user the role of tenant operator for the tenant.
Actor	Tenant administrator.
Pre-Condition	The user must belong to a tenant.
Post-Condition	Tenant operator role is assigned.
Post-Condition in Special Case	Tenant operator role is not assigned.
Normal Case	1. The tenant administrator assigns a role named tenant operator to the tenant user. 2. The tenant has tenant operator added to the configuration registry. 3. The permissions for the tenant operators are updated in the configuration registry.
Special Cases	1a. No tenant users found. a) The system sends an error message and aborts. 3a. No permissions for the tenant operator found in the configuration registry. a) The system sends an error message and aborts.

Table 4.10: Description of Use Case *Assign tenant operator role*.

Name	**Change the assigned tenant administrator**
Goal	Change the current tenant administrator for the tenant.
Actor	Tenant administrator.
Pre-Condition	The tenant must be registered in the tenant registry and the tenant must have an assigned tenant administrator already
Post-Condition	Tenant administrator is changed for the tenant.
Post-Condition in Special Case	Tenant administrator is the same as before.
Normal Case	1. The tenant administrator finds another user suitable for the role of tenant administrator for the tenant. 2. The previous tenant administrator's role changes from tenant administrator to tenant operator. 3. A new tenant operator is assigned to the role of tenant administrator in the configuration registry. 4. The permissions for the new tenant administrator change accordingly.
Special Cases	1a. No other tenant user found. No users in the tenant. a) The system sends an error message and aborts. 2a. The previous tenant administrator role does not changes in the configuration registry. a) The configuration registry is updated and previous tenant administrator role is changed. 3a. No tenant operator role changes in the Configuration registry. a) The Configuration registry is updated and a new Tenant Administrator role is assigned. 4a. The permissions for new Tenant Administrator not changed in the Configuration registry. a) The Configuration registry is updated and new permissions assigned to the Tenant Administrator is removed.

Table 4.11: Description of Use Case *Change the assigned tenant administrator*.

Name	**Delete tenant operator role**
Goal	Delete the role of tenant operator for the tenant.
Actor	Tenant administrator.
Pre-Condition	The user must belong to a tenant.
Post-Condition	Tenant operator role is deleted.
Post-Condition in Special Case	Tenant operator role is not deleted.
Normal Case	1. The tenant administrator deletes a role named tenant operator. 2. The configuration registry has no tenant user assigned as tenant operator for a tenant.
Special Cases	1a. No tenant operator role found. a) The system sends an error message and aborts. 2a. Tenant operator still exists in the configuration registry. a) The configuration registry is updated and the tenant operator role is removed for the user.

Table 4.12: Description of Use Case *Delete tenant operator role*.

Name	**Describe databucket size for the tenant operators**
Goal	Resource allocation for data storage to all the tenant operators.
Actor	Tenant administrator.
Pre-Condition	The tenant must exist in the tenant registry with a specified bucket size.
Post-Condition	The size of the bucket for data storage for each tenant operator is defined.
Post-Condition in Special Case	The size of the bucket is not defined for each tenant operator and no value is specified in the configuration registry.
Normal Case	1. The tenant administrator checks for tenant entry in the tenant registry. 2. The size of the databucket available for a particular tenant in the tenant registry is divided for all the tenant operators belonging to that tenant. 3. The assigned size is updated in the configuration registry for reference.
Special Cases	1a. The tenant entry does not exist in tenant registry. a) The system shows an error message and aborts. 2a. The databucket size for all the tenant operator is not provided. a) The system shows an error message for the tenant operators without the databucket size defined and aborts. No entry to the configuration registry is made. 3a. The entry in the configuration registry could not be updated. a) The system shows an error message and aborts.

Table 4.13: Description of Use Case *Describe databucket size for the tenant operators*.

Name	**Increase databucket size for the tenant operators**
Goal	Change in resource allocated to the tenant operators for data storage.
Actor	Tenant administrator.
Pre-Condition	The tenant operator must exist in the configuration registry with a specific bucket size.
Post-Condition	The size of the bucket for data storage for the tenant operator is increased as requested.
Post-Condition in Special Case	The size of the bucket is not increased for the requester Tenant Operator and no change in the Configuration registry.
Normal Case	1. The tenant administrator checks for tenant operator in the configuration registry. 2. The size of the databucket available for the requested tenant operator is increased. 3. The size is increased either by decreasing the size of a bucket from other tenant operator or from extra available resource. 4. The assigned size is updated in the configuration registry for reference.
Special Cases	1a. The tenant operator entry does not exist in the Configuration registry. a) The system shows an error message and aborts. 3a. The databucket size for the requested tenant operator is not increased as no extra resource available. a) Free or idle resource check is performed for other tenant operators. b) The system shows an error message and aborts. 3b. The databucket size for the requested tenant operator is not increased as no other tenant operator has free space in databucket. a) The system shows an error message and aborts.

Table 4.14: Description of Use Case *Increase databucket size for the tenant operators*.

Name	**Decrease databucket size for the tenant operators**
Goal	Change in resource allocated to the tenant operators for data storage.
Actor	Tenant administrator.
Pre-Condition	The tenant operator must exist in the configuration registry with a specific bucket size.
Post-Condition	The size of the bucket for data storage for the tenant operator is decreased to provide space to other.
Post-Condition in Special Case	The size of the bucket is not decreased for tenant operator and no change in the configuration registry.
Normal Case	1. The tenant administrator checks for tenant operator in the configuration registry. 2. The size of the databucket available for a tenant operator which is unused or idle is decreased. The size is decreased to provide it to other tenant operator. 3. The changed size is updated in the configuration registry for reference.
Special Cases	1a. No tenant operator entry exist in the configuration registry for sharing databucket. a) The system shows an error message and aborts. 2a. The databucket size for none tenant operator is decreased as no idle space available. a) Free or idle resource check is performed for other tenant operators. b) The system shows an error message and aborts.

Table 4.15: Description of Use Case *Decrease databucket size for the tenant operators*.

Name	**Share databucket**
Goal	Complete or partial sharing of resource allocated to the tenant operators for data storage.
Actor	Tenant administrator.
Pre-Condition	The tenant operator must exist in the Configuration registry with a specific bucket size and permission to share Databucket.
Post-Condition	The bucket for data storage for the Tenant Operator is shared between multiple Tenant Operators partially or completely successfully.
Post-Condition in Special Case	The bucket sharing between multiple tenant operators fail.
Normal Case	1. The system checks for the tenant operator in the configuration registry. 2. The system checks for the tenant operator permissions in the configuration registry. 3. The bucket is shared between the Tenant Operators. 4. The shared bucket size for the Tenant Operators is saved in the configuration registry for reference.
Special Cases	1a. No Tenant Operator entry exist in the configuration registry for sharing databucket. a) The system shows an error message and aborts. 2a. No Tenant Operator with bucket sharing permissions exist in the configuration registry for sharing databucket. a) The system shows an error message and aborts. 2b. The databucket sharing permission exist only supporting partial sharing. a) The related Tenant Operator's entry for databucket size changes in configuration registry by specifying the databucket size shared and the size privately owned by the Operator. 2c. The databucket sharing permission exist for complete sharing. a) The related tenant operator's entry for databucket size changes in configuration registry by specifying the whole databucket size available to other operator for use.

Table 4.16: Description of Use Case *Share databucket*.

Name	**Login to the service**
Goal	Login to use the service
Actor	Tenant administrator, Tenant operator
Pre-Condition	The credentials must be authenticated to use the service.
Post-Condition	The service is available for use.
Post-Condition in Special Case	The service is not available for the user logged in.
Normal Case	1. The user provides credentials on the login page. 2. The user is authenticated to use the service.
Special Cases	2a. The user credentials entered are invalid. a) The system generates error and login page is shown.

Table 4.17: Description of Use Case *Login to the service*.

Name	**Logout of the service**
Goal	Logout of the service
Actor	Tenant administrator, Tenant operator
Pre-Condition	The user is logged in to use the service.
Post-Condition	The service is not available for use without login.
Post-Condition in Special Case	The service is still being used.
Normal Case	1. The user opts to logout of the service. 2. The login is required for using service again.
Special Cases	1a. The user could not logout. a) The system generates error and needs logout again.

Table 4.18: Description of Use Case *Logout of the service*.

Name	**Trigger computation**
Goal	Enable using service for computation and provide result for the requested function calculation.
Actor	Tenant administrator, Tenant operator
Pre-Condition	The user is logged in to use the service and requests function to perform with input parameters and format for response.
Post-Condition	The response contains the result.
Post-Condition in Special Case	The response contains location address.
Normal Case	1. The user enters function to compute and gives input parameters with format for response in a URI. 2. The response is in the format specified by the user. 3. The response containing the result is received by the user.
Special Cases	1a. The user enters function from package other than SciPy. a) The response contains appropriate message. 1b. The user enters invalid input parameters. a) The response contains appropriate message. 3a. The user receives a response with a location address as the result computation exceeds a given time frame. a) The user must refer the location for getting the result.

Table 4.19: Description of Use Case *Trigger computation*.

Name	**Retrieve result from own databucket**
Goal	Get computed result from own databucket.
Actor	Tenant administrator, Tenant operator
Pre-Condition	The user is logged in to use the service and has assigned databucket.
Post-Condition	The response contains the results from the databucket.
Post-Condition in Special Case	The response contains error message.
Normal Case	1. The user requests for result from their databucket. 2. The response is received by the tenant.
Special Cases	1a. The user has no databucket assigned. a) The response contains appropriate message.

Table 4.20: Description of Use Case *Retrieve result from own databucket*.

Name	**Retrieve result**
Goal	Get computed result from specific tenant.
Actor	Tenant administrator, Tenant operator
Pre-Condition	The user is logged in to use the service and provides tenant ID for retrieving result.
Post-Condition	The response contains the results from tenant's databuckets.
Post-Condition in Special Case	The response contains error message.
Normal Case	1. The user requests for result by providing tenant ID. 2. The response is received by the tenant.
Special Cases	1a. The user requests for an invalid tenant ID. a) The response contains appropriate message. 1b. The provided tenant does not have permissions to share results. a) The response contains appropriate message.

Table 4.21: Description of Use Case *Retrieve result*.

Name	**Store result**
Goal	Store computed result in the databucket.
Actor	Tenant administrator, Tenant operator
Pre-Condition	The user is logged in to use the service and has computed result for storing.
Post-Condition	The response contains the success message and address to access the result.
Post-Condition in Special Case	The response contains error message.
Normal Case	1. The user requests for result by providing tenant ID. 2. The response contains the results from the tenant.
Special Cases	1a. The user requests for an invalid tenant ID. a) The system generates error and aborts. 1b. The provided tenant does not have permissions to share results. a) The response contains appropriate message.

Table 4.22: Description of Use Case *Store result*.

Name	**View tenant registry**
Goal	View the tenant registry to know the tenants using the service.
Actor	Tenant administrator, Tenant operator
Pre-Condition	The user is logged in to use the service.
Post-Condition	The tenant registry is available to the tenant for viewing.
Post-Condition in Special Case	The user cannot view tenant registry.
Normal Case	1. The user requests for viewing the tenant registry. 2. The response is received by the tenant.
Special Cases	2a. The user is unable to view tenant registry. a) The response contains appropriate message.

Table 4.23: Description of Use Case *View tenant registry*.

4.3 System Overview

The system overview depicts the various ways in which the application is accessible for the user. As shown in figure 4.4 multiple tenants with multiple users can access the same application running in the cloud. The application in the private cloud infrastructure is accessed by various tenants with their data stores available on the server or on dedicated data storage servers. The application can be assumed to be running partially in the cloud or completely is run and is accessed via URLs defined to locate the various resources. The REST API designed allows the user to identify the resources and perform HTTP methods on them to

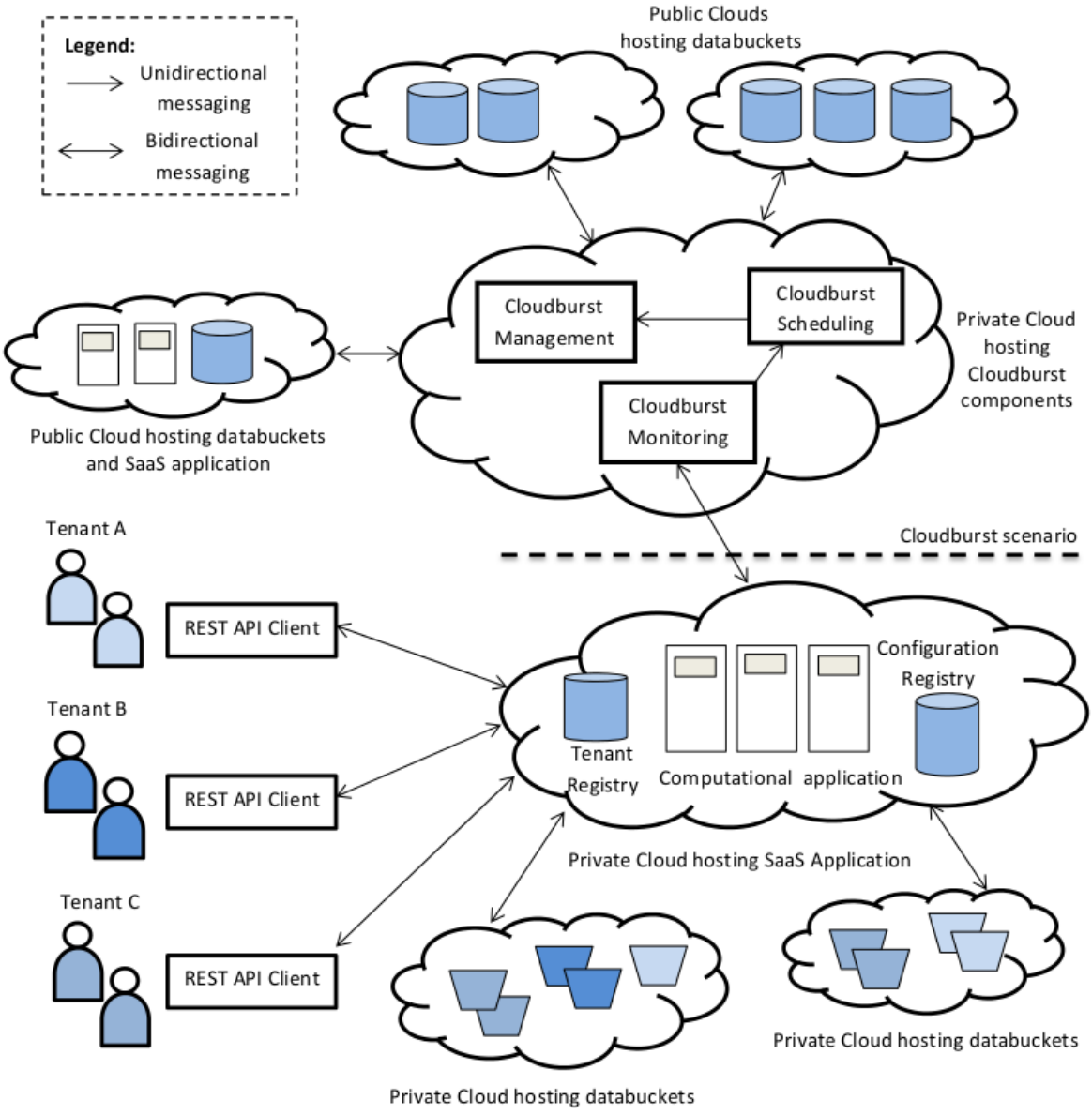

Figure 4.4: System overview of the Computational System

perform operations on the resources. To meet the demands of dynamic scaling, the resources can be assigned to the user via cloud burst and hence, the expansion of the resources happen from a private cloud to a public cloud. The databuckets associated to each user are accessible using the URIs and the application enables the features to share the data. However, in the thesis we consider hosting the application on the cloud partially. The partial hosting refers to migrating the computation on the private cloud of the application and storing the tenant information on-premise. The databuckets are hosted different from the cloud hosting the application. The application is hosted on the private cloud to perform the computations and the tenant related information is hosted on the same cloud in form of tenant and configuration registries. The results hence generated are stored in the databuckets for each tenant which are stored on another private clouds dedicated to host only the databuckets. To retrieve the result, the databuckets are referred using the tenant and the configuration registries. If there is a need to provide more resource for the data storage then the cloud bursts into public cloud for using more resources. The support for the data migration process in case of cloud burst scenarios is provided from an traditional to a cloud data store, or between cloud data stores hosted on different clouds and can be managed by a dedicated Data Bucket Manager. After this the data store configurations maintained in the tenant registry and the configuration registry need to be updated as per the changes. Any further resource scaling must be handled and the appropriate updating of the database schemas should be ensured to maintain data security. To balance the load of the requests on the server with the increasing demand, the requests should be distributed among various servers in the cloud to ensure that all the requests are handled. Same is the scenario in case of any node failures as no state is held by the communication. The computational libraries available to the user are accessed via requesting the function name and its input parameters with appropriate URI, and the computed result is sent to the user in the ideal case. If the calculation is long-running, in the response the location of the databucket storing the result is sent and the user needs to poll to check if the result is computed. To enable cloud bursting, three components namely: Cloudburst Manager, Scheduler and Monitoring are proposed in the system overview. The cloudburst management handles the provisioning of the resources in form of hosting more computational SaaS applications or providing more databuckets. This management is triggered by the cloudburst scheduler. This feature schedules the cloudburst by realising the essential resource interactions and restoring the information related to the resource to host them on the public cloud. This scheduler works when the cloudburst monitor informs about the possible need of dynamically scaling the resources and hence shows about the need of pulling resources from the public cloud.

5 Design

This chapter explains in detail the system design with focus on the components involved in the application and the resources for the API design. In the previous chapter, the concepts for realizing the service have been proposed, which are referenced for designing the prototype. The aim is to realise the component interactions and relations between them. The design of the API is dealt by redefining the RESTful API design rules relevant to our system and then expressed as per the use in this work.

5.1 Architectural Overview

The overview of the system is expressed in a three layer architectural representation with components within each layer constituting Presentation Layer, Business Layer and Resource Layer.

The presentation layer comprises the functionalities that enable the user to access our system. There are two offered possibilities: using a web GUI or accessing through a web-service API. The user must provide all the required information for using the application which for e.g., in our case is providing function name and input parameters to perform computation. To access the user-related data through the web API, user authentication and authorization must be checked for. The business layer constitutes of the logical implementation for the application. The service is aimed to provide the computational results for the statistical analysis using the open source library SciPy provided by Python. To ensure this, computational logic must be implemented which takes the user request and performs the computation using the function name and the input parameters provided by the user. If the input parameters or the function name is incorrect then the user must be informed about the error with correct and complete error messages. The multi-tenancy behaviour of the system is maintained via various roles assigned to the users and the system handles the basic multi-tenancy features of data isolation, tenant customisation, tenant content routing etc. by managing resources among the tenants. The tenant related information which defines the tenants using the service with the quota of data store available to each is kept in the tenant registry and the tenant registry manager has to take care of managing the information. In case of migrating this information from on-premise private cloud to public or between the private clouds, the tenant registry manager is responsible to migrate the information and maintain the authenticity of the data. The tenant configuration information relating to the users in the tenant, their roles and permissions and resource distribution are maintained in the configuration registry. For the migration of this information and to maintain the data together, the configuration registry manager is responsible. The data bucket manager takes care of the data store resource distributed among

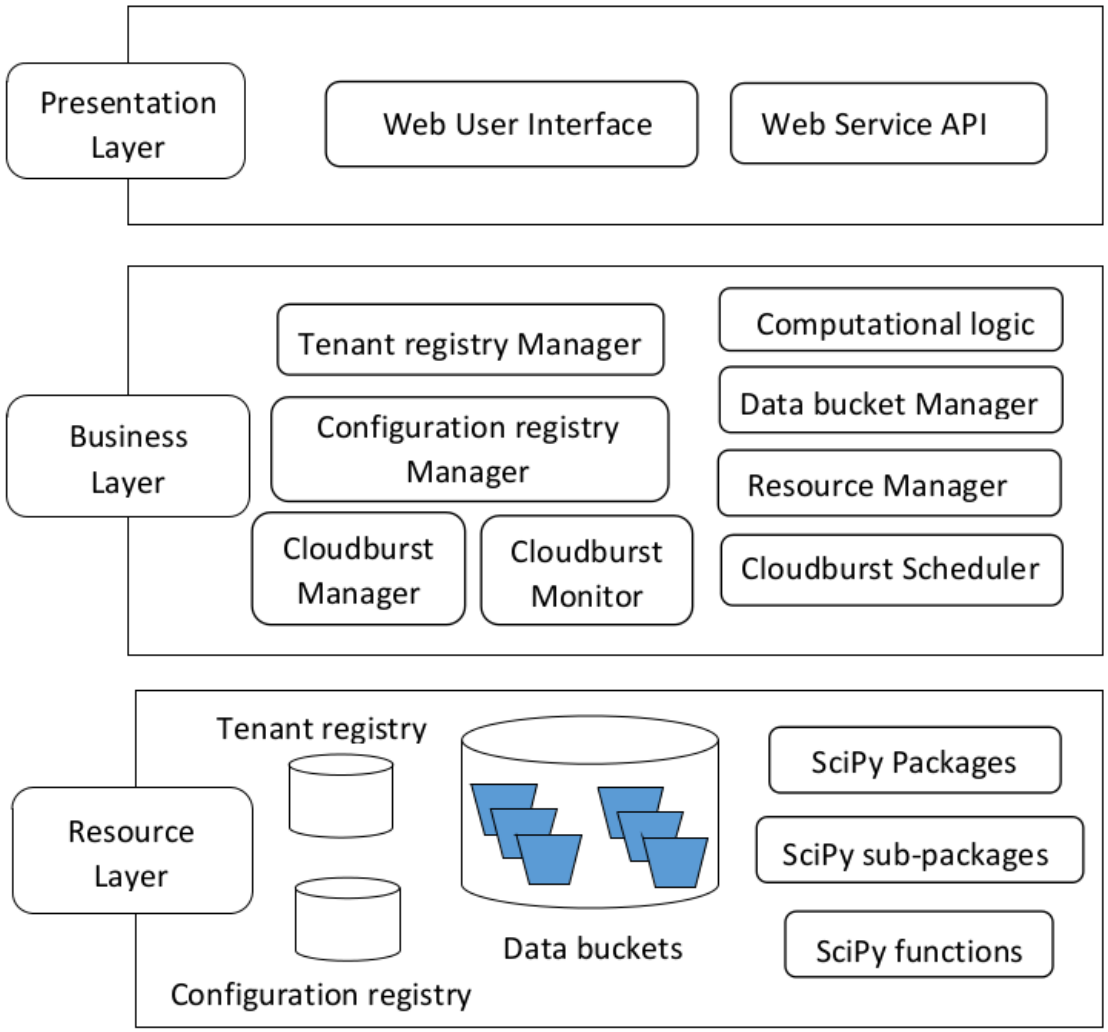

Figure 5.1: Architectural overview of the Statistical Analysis Framework

the tenants. If there is a need to dynamically scale up the resources needed by the system, the data bucket manager accomplishes the data migration from one cloud to the other while maintaining the data security. The data bucket manager monitors the databucket for the tenant user and alerts the individual if the databucket assigned is all occupied. The task is also to manage the data of individual tenant user in case of databucket sharing and let the user access only the databucket assigned to them. Another task handled by the data bucket manager is to allow the data sharing between the tenant users and also enable sharing the data bucket space partially or completely. A resource manager takes care of managing resources if there occurs a situation to cloud burst and migrate the resources between the clouds. The manager provisions resources on the cloud where the data is migrated. To enable cloudburst, the three modules namely cloudburst manager, cloudburst schedule and cloudburst monitor are implemented in the business layer. These three manage provisioning resources on the public cloud and migrate the related data for the newly set-up resource interactions. These modules

work together to adjust the resource management in times of high demands and ensure resource availability when demand spikes occur. The detailed functionality is explained in the section 5.5. The resource layer defines all the resources of the system. The resources available to the user consist of data storage capabilities in form of databuckets which would enable the user to save their computed results for later reference or for sharing purposes. The tenant registry and the configuration registry allow the system administrator to save data for user authentication and providing them with the permissions to enable data sharing and accessing resources. The description of each component in detail would be considered in the later section. The computation of the required function is performed using RESTful APIs which allow the user to request for a function and provide the input parameters for the same. The result is later responded to the user in the file format it is requested into. The response may contain the location address on the Application server where the computed result will be stored if the result calculation is exceeded by a given time period. The user explicitly has to request for the result to maintain the asynchronous behaviour of the system. The SciPy package, sub-package and the functions are the resources used by the computation logic for performing the operation requested by the user for computation. SaaS application thus exposed to the users ensures scalability aspect by providing resources on the fly. Realizing the fact of providing resources on a deployed cloud model economically, the system can further be extended to another model to ensure resource availability. The prototypical implementation of cloud bursting technique is not in scope of the thesis work but the available options for scaling are taken into consideration and explained in brief.

5.2 Resource Model

In the section above, the representation of the system depicts the resources like: tenant registry, configuration registry and databuckets. The other resources depicting the application users like tenant administrators and operators are not explicitly defined but are dealt in detail in the next section. A REST API consists of an assembly of interlinked resources. This set of resources is known as the REST API's Resource model [Mas11]. Fielding explains that "REST uses a resource identifier to identify the particular resource involved in an interaction between components", hence resources for a REST-based service must be defined.

The resource model represents the resources and the relations among them as: tenant, system administrator, tenant administrator, tenant operator, data store, databucket, SciPy packages, SciPy sub-package and SciPy function. The relation between the resources is depicted using an ER diagram. Tenant, SciPy package and data store are the collection of resources whereas tenant administrator and tenant operator are the sub collection of resources within tenant. SciPy sub-packages and functions are the sub collection within SciPy package resource and a data store contains a sub-resource namely databucket. The system shows that a system administrator creates tenants. There exists multiple tenants constituted by multiple tenant users. The distinction made between the tenant users is based on their roles: tenant administrator and tenant operator. Necessarily there is one and only one tenant administrator assigned to each tenant. There are optionally many tenant operators within a tenant. Summing all the roles, a tenant mandatorily has only one tenant administrator and many tenant

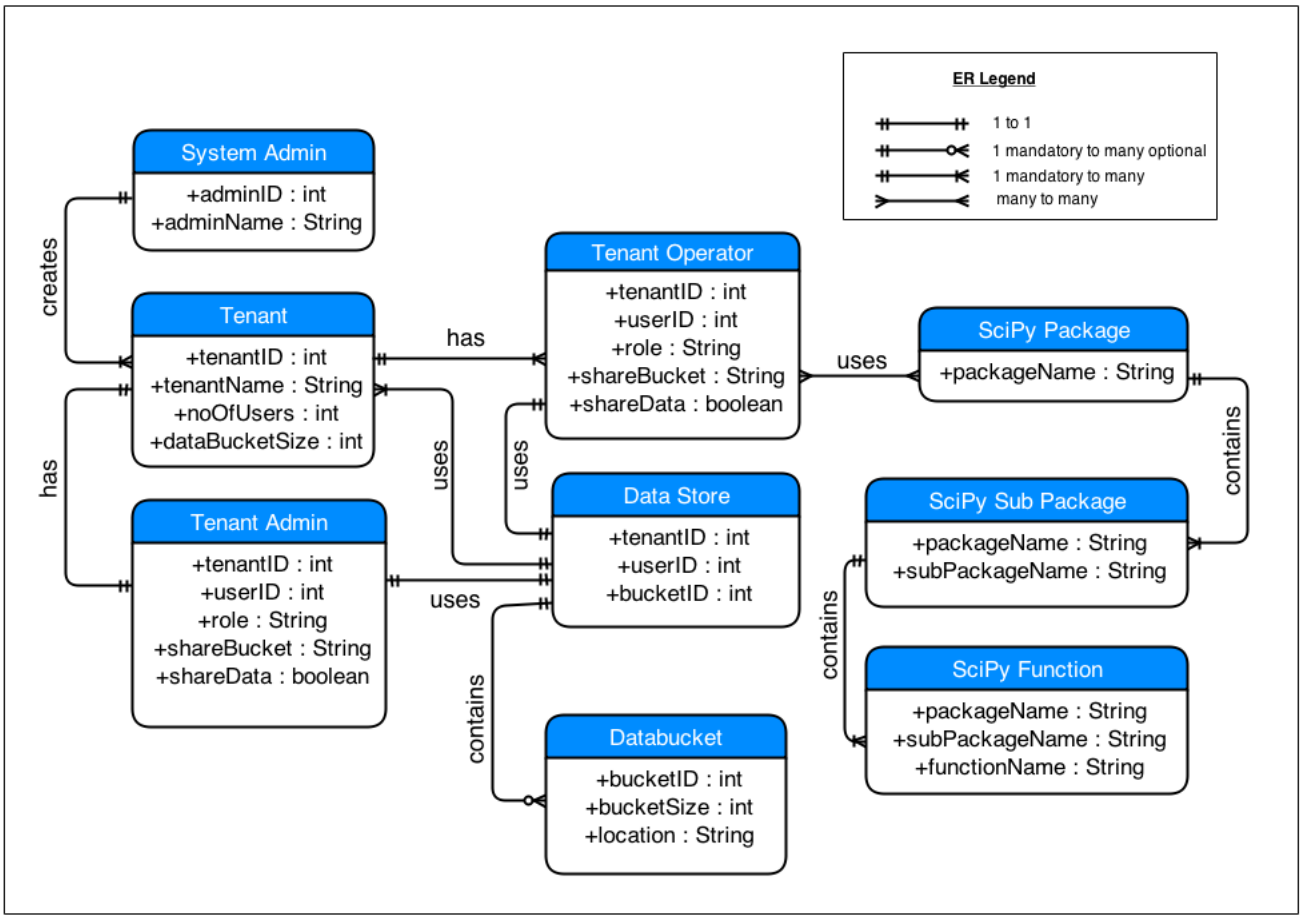

Figure 5.2: Resource Model representation using ER Diagram

operators. Each tenant user has attributes as identifier, name, role and permissions assigned to it. A tenant identifier uniquely identifies the tenant from the other who use the service. Since, within each tenant there exist multiple users in form of tenant administrator and tenant operator, they hold another identifier. This relation is crucial to understand the resource consumption in form of data store and for accessing the owned data resources. The role for the corresponding user defines its type and hence differentiates between the type of user. This impacts the permissions allotted to the user. The permissions for any role are defined by the tenant administrator which previously was proposed in the Section 4.1.2 and is dealt in detail later. Permissions deal with the sharing of the data residing in the data store with other tenants using the service and to share the resource itself partially or completely or not at all for privacy sake which supports the functional requirement of the system to maintain data isolation as described in section 4.1.4. Partial sharing or complete sharing of data buckets refer to an operator being able to share their own bucket space with the other operators of the same tenant. The difference between *shareBucket* and *shareData* attribute refers to sharing space amongst the tenant operators and sharing the results generated from computation respectively. Each operator can share their bucket space with the other by defining some space for sharing hence, making it partial sharing or by allowing the other operator to use

the complete bucket space. Partial sharing refers to sharing the space among the same tenant users and not with the other tenants whereas complete sharing the bucket among multiple tenants. By default each tenant user shares the result within their tenant but to enable sharing the result with other tenant, permissions attached to the same must be checked. Hence, to share the result stored within the bucket with other tenants, a boolean value is used. The value expressing *False* implies not allowing to share the result and *True* for enabling the result sharing. After realizing the permission for sharing the data bucket with another user, the status has to be maintained specifying that the bucket is shared hence, a dictionary as a data structure is maintained. The key refers to the userID who shares the bucket partially and the value refers to the size shared in terms of space for storage e.g MB of space with the other user. Once recognised, the data from the corresponding data store instances are collected and given to the user as a response for the request. Any tenant user irrespective of whether being a tenant operator or a tenant administrator, share their data bucket space with the other users within their tenant. The bucket correspond to only a single user and hence the bucket to be shared is not a set of bucket but an individual bucket. Each tenant has multiple databuckets but is segregated on the basis of an identifier which relates to each user's identification. A unique bucketID is assigned for the identification so that each user can store or retrieve their data in their own databuckets. Considering a fact that there is a definite size defined for each bucket, the monitoring of bucket size has to be considered. This would trigger an appropriate message to the user who owns the bucket about the memory consumed and about no space left situation. This helps the user to demand for more space which is handled in two ways. Either the system administrator provides more resource if the SLA permits or the space management is handled by the tenant administrator. The tenant administrator checks for a user within the tenant whether is there any possibility to completely or partially share the bucket with another user. The *shareBucket* attribute indicates if a bucket is shared amongst one or multiple tenants. Our system enables the partial or complete sharing of buckets. The statistical resources are further defined as SciPy package, SciPy sub-package and SciPy function. There are four packages whose functionalities are offered. These are :

- Statistical
- Linear Algebra
- File I/O
- Integration

The basic functionalities provided by these libraries has been defined already in the section 2.8.1. Further each of these packages have sub-packages which contain functions used for various statistical and arithmetical computations. Hence, the packages are a resource and so are the sub-packages and functions. The input parameters to the functions must be provided by the requester and hence the computed result is sent out to the user in the desired response format.

5.3 Databuckets

A databucket is the logical grouping of physical resources for storing data corresponding to a specific user of a tenant. This data store is intended to store the results from the functions used for analysis and sharing of the information if needed as described in section 4.1.3. The storage is defined in a given size for each user. The databuckets can be present on dedicated database servers or multiple database servers. In our thesis, buckets are hosted on the application server and are accessed by the userID. Hence, for a user to access their bucket, their userId has to be known. The data storage is static in nature that is either is stored in files in databases. For our work, the results are stored in files dedicated for users. Each tenant has a directory to store sub directories for each of the user. The user files are stored in the sub-directory assigned for a user. To enable partial sharing of the databucket, a subdirectory specific to public sharing of space within a tenant is created. If any operator has permission to enable bucket sharing, then the space of the sub-directory changes accordingly. Similarly to allow complete sharing of the data bucket between all the tenants, a specific directory is created. For designing the buckets, the system administrator provides a definite size of data store for the tenant. The tenant administrator further divides it in some defined size for each user within the tenant. A tenant can access their own databucket by authenticating. There is a possibility for sharing or requesting for space on the bucket. Only a system administrator or a tenant administrator can provide the space on the bucket. The sharing of a bucket completely or partially is checked by permissions set for a user and is performed by a tenant administrator. The databucket resource has the following attributes:

- bucketID - This is the identifier which maps the bucket identifier to the data store resource where the tenant data persists. This id is useful for identifying the owner of the bucket.
- bucketSize - This defines the capacity of the bucket to store data. The size of the data bucket is defined by the system administrator for each tenant. The whole size is further divided among the rest of the users belonging to the same tenant. The size may later change for the user as per their role and definition of the size property given by the Tenant Administrator.
- location - This defines the location of the data on the server. For the implementation of the prototype, the data is stored in file on the system storage of the format possibly JSON or XML. This refers to the structure in which each file is stored in a sub-directory belonging to a tenant user. Each tenant has a dedicated directory which has the sub-directory for the user and the user result is stored in a file within this sub-directory.

5.4 Database Schemas

The two registries of concern to the design are Tenant and Configuration registries, respectively. The tenant registry is for storing the information of the tenant and their corresponding users. The attribute *bucketSize* relate to the resource distribution for data storage among the

users belonging to a tenant. Each tenant is provided with an Unique Identifier (UID) in order to be uniquely identified in our system. The registry depicts the number of users for the corresponding tenant, and provides the space allocated to each tenant as provided by the system administrator. The system administrator maintains the tenant registry in order to provide the resources to a tenant. Tenant administrator and tenant operator can not make any change to this tenant registry. This gives the information to the tenant user to realize if there exist any other tenants from whom the data can be requested. The figure below displays the relation between the entities tenant, tenant user and tenant databucket to define the schema.

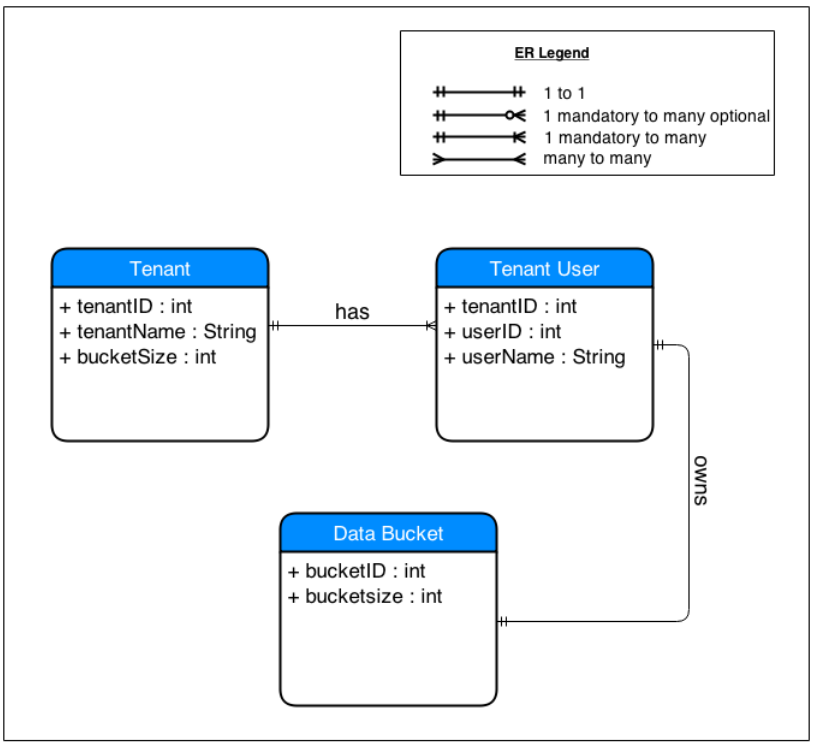

Figure 5.3: Entity Relationship Diagram for Tenant Registry

The configuration registry has the information needed for specifications at the user level for using the service. It gives the information that relates to the role of the user, the permissions and the resource allotted to individual user. As explained in the section 4.1.2, the tenant administrator has the write access to the registry to maintain the user information for a tenant. There are separate schemas for each tenant but the database is shared which means that each tenant has an individual configuration registry but they are all a part of the same database. This promises data isolation. The figure below shows the various attributes related to a tenant specification. This logical model defines the relation between the users and the tenant. However, each tenant data is stored in a different schema as each tenant has a separate configuration registry but the stored information for all the tenant users is about their roles, permissions and databucket size. The design ensures one schema for each tenant defining the role and permission exercised by a user. The attribute *roleName* determines whether the

user is a tenant administrator or a tenant operator. The permission relate to tenant specific permissions and can vary as per the tenant need. For the implementation of the work and for the features relating to the resource sharing, the attribute *shareBucket* specifies whether the user can completely or partially share the bucket with the tenant users. Another valuable attribute *shareResult* checks whether the result within the data store can be shared among other tenants. The sharing among the tenant users within the same tenant is considered true for the system but further the sharing can be finely decided on the basis of permission of each tenant user also. This means that if a tenant user does not want to share their data with anyone even in the same tenant, this can be enabled but as a potential future work.

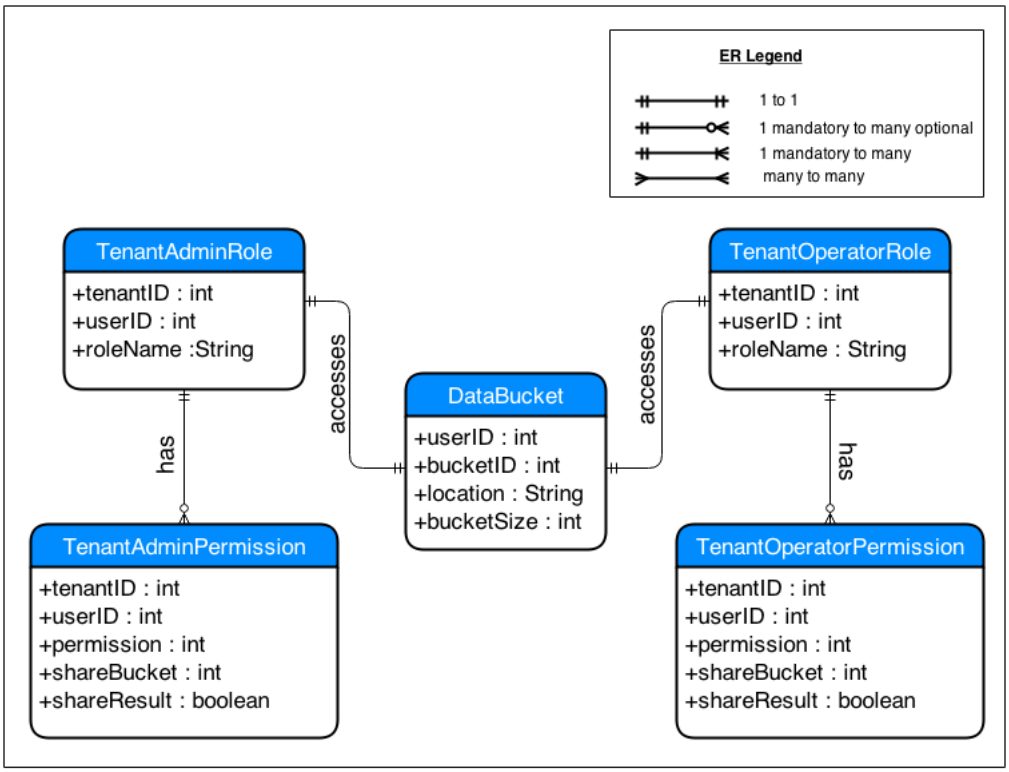

Figure 5.4: Entity Relationship Diagram for Configuration Registry

5.5 Cloudburst Implementation

The cloudburst technique enables bursting the application running in a private cloud into the public cloud when the demand of computing capacity spikes. This has an advantage as it allows the use of extra compute resources only at runtime and hence is an economical solution. An application bursting into a public cloud provisions more resources and hence, more computation power in case when the available resources on the private cloud have exhausted. But to realize the possibility of handling the high demand of resources, the reservation of the computing and storage resources and schedule for the resource utilization has to be estimated to allow on the fly resource demand fulfilment. Many such and other

essential capabilities must be handled to support cloudburst technique. Our work proposes some components in the system responsible for handling the allocation of resources at peak times. The components are described in the next sections:

5.5.1 Cloudburst Scheduler

A cloudburst scheduler manages to schedule the reservation of resources so as to provide the user resources in no time. Some of the replicas of the jobs or specifically the application are maintained on the public clouds to support the high demand of the resources. The dynamic deployment at runtime of the running application and storage resources can be implemented by using algorithms that detect the late computation result generations. These algorithms also maintain the state of the computation running on the replicated applications, within the same cloud so as to reassign the computation to be performed on the idle instances of the application servers. For e.g., if one application instance is idle then the computation can be performed by dividing the computation in batches. The scheduler also monitors the current state of the assigned batch tasks, so as to allot the next set of tasks to the idle application instances. The scheduler algorithm can also simultaneously monitor the private cloud hosting the application for further managing the reservation of more resources. The data store replication has to be handled with utmost concern as the tenant information must be isolated even if the resource expansion occurs on the public cloud. The data must be secured from the other tenants as well as from the third party attacks, considering the deployment model of the cloud to be public. The scheduler can implement a load balancer to distribute the load of computation between the multiple instances hosted in the public cloud.

5.5.2 Cloudburst Monitor

A cloudburst monitor aims at keeping a track of resources consumed to meet the user demands at computing capacity spike. A real-time monitoring implementation is needed to bill the user as per the resource consumption. Since, providing the resource at runtime has to be monitored, its is also useful for provisioning more resources on the cloud if the threshold for the replicated application instance is also reached. The QoS monitoring is an important feature for the application to ensure the functional requirements being fulfilled by the newly set up resources. The scheduler is triggered by the monitoring algorithm and implementation to alert the need of reserving resources in the future times.

5.5.3 Cloudburst Manager

To manage pulling resources at runtime can be accomplished by providing fully integrated software and hardware platform for hosting the SaaS application and maintaining the data stores at the public cloud for fast roll out. The automated provisioning and de-provisioning of the resources has to be managed along with supporting the ability to manage the heterogeneous environment for the system replicated on the public cloud. Resource provisioning

makes the system elastic and allows it to grow and shrink dynamically to meet the QoS requirements of application. Appropriate mechanism for the data backup and recovery has to be used ensuring the data isolation and data security provisioning as well.

5.6 Cloud interoperability

For supporting the cloudburst technique, the interoperability between the private and the public cloud has to be taken into consideration. There are certain implementation details that should be examined which state that it is necessary to maintain a mechanism for an automated acquiring of lost connection between the clouds. The interactions among the clouds should also be considered to realize the data involved between the clouds. Since, our system is built using the REST based architectural style, the statelessness of the interactions minimise the data volumes moved between the application and the cloud. It is also necessary to adhere to the non-functional requirements of the system therefore, standard for defining and exchanging meta information regarding the application and the resources must be implemented. The algorithm can be designed to allow the meta data exchange between the clouds to reconstruct the application on both the clouds. This algorithm will also take care of keeping the data secured and isolated. Uniform processes and tool sets can be implemented to monitor and report the level of services and the compliance/violation of security/privacy policies in remote clouds.

5.7 RESTful API

REST API design rules are dealt in detail in the previous section 2.4.1. The need for following the design rules mainly provides consistency for the design and sticking to standards make it easier for being used. In the design of this API, the resource model mentioned in the figure 5.2 is considered for realizing the resources available to the user and also is used for the URI design which is of major concern in the design.

5.7.1 API Design

The resources for this prototype can be basically explained as one group for performing computation operations, for example using SciPy library functions and producing results. The other mainly deal with accessing the results from other tenants and retrieving or viewing tenant information to request results from other tenants using the application. To address each resource uniquely, an easy to understand URI is assigned to each.

- The first part of the URI shows which resource the client is addressing to. Hence, the first part is going to be /tenants, /datastores or /scipy. These URIs represent set of tenants, tenant databuckets and SciPy Packages respectively.

- The second part of the URI defines the identifier to the resource to which the action is to be applied, for example : /tenant/1 would express the tenant with ID as 1.
- To perform the computation using the SciPy library, it is necessary for the client to specify the function name and the input parameters in the query string format thus the URI would appear as : /scipy/scipy package name/scipy sub package name/?function-name=abc&param=1,2,3. This is an example to depict the URI design to access the functions offered by SciPy library. When the URL limit of parameters is exceeded which is under 2,000 characters, then POST method must be uploaded containing the parameters in the file. The file format can be any and is useful for containing more input parameters than supported by the URL limit.
- Since the authentication is the reuse of the Django (REST framework from Python) administrator login, the login based authentication for Tenant Administrator and Tenant Operator are implemented using custom authentication supported by Django. The session information helps to access the details for providing different permissions to the tenant users as per their roles.
- Each resource is linked to another resource and the tenant has user specific access to the data store using an identifier to their own databucket. The computed result from the SciPy library function can be stored in the databucket and the data store has collection of tenant owned databuckets. Hence, all the resources are connected.

The proper use of HTTP methods GET, POST, PUT and DELETE has been defined for the resources. The description about the effect a HTTP request will have on the resource, are described below:

- /resource
  - GET: shows a limited list of the resources on the system. If the URI contains a query which are defined later in this section, the list of resources returned will meet the criteria specified in the URI. This is accessible to all the users in the prototype we design.
  - POST: adds a new resource of the specific type to the system taking the information embedded in the body of the request. This is Role based control Access Role based control Access (RbAC) hence, not all the users can add a resource. For example: in our system, only a system administrator can add a tenant.
  - PUT: This method is applicable to the tenant, tenant administrator, tenant operator, data store and data bucket resources only as the resources like SciPy package or function cannot be updated. Also, to update any of the above mentioned resources only RbAC is provided.
  - DELETE: This is a RbAC and does not apply to all the resources. It applies only to tenant, tenant administrator, tenant operator, data store and data bucket resources.
- /resource/x (x=resource identifier)

  - GET: returns the information of the resource with identity X in the content type requested by the user.
  - POST: not applicable to all the users as only system administrator and tenant administrator can add new resources. Also resources like tenant, tenant administrator, tenant operator, data store and data bucket resources are the resources which can be added.
  - PUT: updates the information of the resource with identity X taking the information embedded in the body of the request. Not applicable to all the resources and also has RbAC.
  - DELETE: Not applicable to all the resources and also has RbAC.

- /resource/x/?key=value (x=resource identifier)
  - GET: returns the information of the resource with identity X in the content type requested by the user and is matched to the query string. Specific to our work, this is a URI designed for performing calculations using the SciPy library. The function name and the input parameter have to be specified by the user. If the input parameters cross the URL limit of parameters, then POST method must be used for uploading a file containing the parameters.
  - POST: not applicable to all the users and all the resources. This is accessible to all users when the input parameters for the calculations cross the URL limit of parameters. A file containing the parameters must be uploaded.
  - PUT: updates the information of the resource with identity X taking the information embedded in the body of the request. Not applicable to all the resources and also has RbAC.
  - DELETE: not applicable to all the resources and also has RbAC.

- /resource/x/resource
  - GET: returns the information of the resource with the identity X in relation to the other resource. For our system, this is the case when a user requests the result from another tenant. Hence, the tenant ID is the identifier and the relative resource id the databucket.
  - POST: used for updating only a particular sub resource, this is used but with RbAC. For example, to add permissions to the user of a particular tenant.
  - PUT : used to update the sub resource and is a RbAC. For example: to update the databucket size for any tenant or a tenant user in specific. Since, the data of any tenant can only be modified by the system administrator or tenant administrator for the tenant it belongs to, it is a RbAC.
  - DELETE : not applicable.

As per the rules expressed in the section 2.4.1, we make sure to stick to the rules specified to design the URI and use only nouns for the expression. To stick to the guidelines for better readability of the responses by the user, custom response messages are used. The client is given the liberty to propose the content type of the response expected and the response is made available either in JSON or in XML format as per the client request. Some of the URLs with the request response characteristics are put in table below:

Description	**Display information of available tenants**
Access Control	system administrator, tenant administrator, tenant operator
HTTP Request	`GET/tenants/`
URL params	none
Query params	• value : none • criteria : none
Post params	none
Response	Array of all the present tenants.
Error Responses	• 500 Internal Server Error

Table 5.1: Description of REST API *Display information of available tenants*.

Description	**Display information of the specified tenant**
Access Control	`GET/tenants/x/`
HTTP Request	x: TenantID. Must be an integer value
URL params	• value : none • criteria : none
Query params	none
Post params	Tenant details including tenantID, tenantName and the databucket size for the specified tenant ID.
Response	• 500 Internal Server Error • 404 Not Found - If the URL parameter entered is not an existing tenant.
Error Responses	

Table 5.2: Description of REST API *Display information of the specified tenant*.

Description	**Display information of the users for the specified tenant**
Access Control	system administrator, tenant administrator, tenant operator
HTTP Request	`GET/tenants/x/users/`
URL params	x: TenantID. Must be an integer value
Query params	• value : none • criteria : none
Post params	none
Response	User details including user ID, user Name and tenant ID for the users from specific tenant ID.
Error Responses	• 500 Internal Server Error • 404 Not Found - If the URL parameter entered is not an existing tenant. • 404 Not Found - If there are no users for the specified tenant ID. • 401 Unauthorized - The logged in user has no access to view the users of the specified tenant ID.

Table 5.3: Description of REST API *Display information of the users for the specified tenant*.

Description	**Display information of the users for the specified tenant**
Access Control	system administrator, tenant administrator, tenant operator
HTTP Request	`GET/tenants/x/users/y`
URL params	x: TenantID and y: UserID. Both must be integer values
Query params	• value : none • criteria : none
Post params	none
Response	User details including user ID, user Name and tenant ID for the specified user from the specific tenant ID.
Error Responses	• 500 Internal Server Error • 404 Not Found - If the URL parameter entered is not an existing tenant. • 404 Not Found - If there are no such user for the specified tenant ID. • 401 Unauthorized - The logged in user has no access to view the user details of the specified tenant ID.

Table 5.4: Description of REST API *Display information of the users for the specified tenant*.

Description	**Add tenants to the system**
Access Control	system administrator
HTTP Request	`POST/tenants/`
URL params	none
Query params	• value : none • criteria : none
Post params	tenantName - Must be a string and databucket Size - Must be an integer.
Response	Tenant details including tenantID, tenantName and databucket size for the new tenant to be created.
Error Responses	• 500 Internal Server Error • 400 Bad Request - If tenant with the same ID exists. • 400 Bad Request - If the POST parameters are not of the specified data type. • 401 Unauthorized - The logged in user has no access to add tenant to the system. This is allowed only for the System Administrator.

Table 5.5: Description of REST API *Add tenants to the system*.

Description	**Add user(s) to the tenant specified**
Access Control	system administrator
HTTP Request	`POST/tenants/x/users`
URL params	x: TenantID. Must be an integer value.
Query params	• value : none • criteria : none
Post params	userName - Must be a string, tenantID - Must be an integer.
Response	User details including userID, userName and tenantID for the new tenant user are displayed.
Error Responses	• 500 Internal Server Error • 400 Bad Request - If tenant with the same ID exists. • 400 Bad Request - If the POST parameters are not of the specified data type. • 401 Unauthorized - The logged in user has no access to add tenant to the system. This is allowed only for the System Administrator or Tenant Administrator of the specified tenant (here it should be x).

Table 5.6: Description of REST API *Add user(s) to the tenant specified*.

Description	**Trigger computation**
Access Control	system administrator, tenant administrator, tenant operator
HTTP Request	`GET/scipy/packagename/subpackagename?function_name={}&params={}`
URL params	packagename: SciPy package name, subpackagename: SciPy sub-package name. Must be string.
Query params	• value : SciPy function name • criteria : The function should be a valid function name for the specified package and sub-package name. • value : Input parameters for the function • criteria : The values provided must match the expected data types for the function to be computed.
Post params	none
Response	The result hence generated must be displayed in desired format.
Error Responses	• 500 Internal Server Error • 400 Bad Request - If the function is not recognised. • 400 Bad Request - If the input parameters provided are incorrect.

Table 5.7: Description of REST API *Trigger computation*.

Description	**Trigger computation**
Access Control	system administrator, tenant administrator, tenant operator
HTTP Request	`POST/scipy/packagename/subpackagename?function_name={}`
URL params	packagename: SciPy package name, subpackagename: SciPy sub-package name. Must be string.
Query params	• value : SciPy function name • criteria : The function should be a valid function name for the specified package and sub-package name.
Post params	filename - upload the file which contains the input parameters. The file format can be any.
Response	The result hence generated must be displayed in desired format.
Error Responses	• 500 Internal Server Error • 400 Bad Request - If the function is not recognised. • 400 Bad Request - If the input parameters provided are incorrect.

Table 5.8: Description of REST API *Trigger computation*.

Description	**Retrieve result from databucket**
Access Control	tenant administrator, tenant operator
HTTP Request	`GET/result/x`
URL params	x: databucket location. Must be a string. This value is known to the user from the response message when any computation result is returned.
Query params	• value : none • criteria : none
Post params	none
Response	The result from the databucket must be shown
Error Responses	• 500 Internal Server Error • 400 Bad Request - If the location is incorrect.

Table 5.9: Description of REST API *Retrieve result from databucket*.

6 Implementation

The implementation of this work relates to providing SciPy library functions off the premise of the client using RESTful Web services to realise the system providing responses in formats like JSON and XML for the requests by the user. Using REST as an architectural style for the service provide freedom of choosing format for request-response and to maintain the stateless behaviour REST is the obvious choice. The need of not maintaining the state of the communication at the server end ensures high scalability by providing a cluster of nodes that can be used as servers available to handle the multiple HTTP requests in case of high load. It also ensures that in case of node failure the request can be handled by the other available servers. The scalability mechanism is achieved by considering cloud burst which would enable the expansion of the public cloud resources to the private cloud environment in case of on-demand pulling of resources is needed. Considering statelessness, the response caching is provisioned to ensure that if the calculation time exceeds then the result is stored and can be requested by the user later. The statistical analysis supportive library offered by Python is installed and build on the application server end and hence, needs no on-premise installation for the use. The design of the API is REST based as the resources identified for the service are identified using URIs. As explained in the previous chapter 5, the resources are identified using URIs and HTTP methods are performed on the resources based on the user access permissions. The provisioning of the resources aimed for the data storage to be provided to the user on Cloud is managed using databuckets. The main aim of the work is to allow users to use SciPy libraries for statistical analysis and therefore, the framework used was Python supported. It saved the effort to interface a different language build API to deal with Python based SciPy library.

6.1 Databuckets

The implementation of the databuckets is realised by the file structure for this thesis work. Each tenant has a static storage in form of directory which contains sub-directory for each user. Each subdirectory contains the files storing the results of the computations performed by the user. The file format depends on the format in which the result was sent as a response. The location of the file is sent to the user in the response when the computational result is sent to the user. The result can then be retrieved by specifying the location. To ensure that no other tenant can access the result without the permission of the user owning the result, authentication and mapping of the databucket location is checked from the data stored in the configuration registry. The location is also shared with the user as a response when the computation goes long beyond a time frame but the user must keep polling to check whether the result is available or not. For enabling the sharing of the databucket space, special

sub-directories defined as *public* are created and the space within is managed by the attribute defined for the data store, which defines the space shared by a particular user for the tenant. For the implementation of the system, the following has been considered:

- Tenant Directory - This directory belongs to each tenant which has the size as defined by the system administrator as the size of the data bucket. This contains sub directories as user sub-directories and public sun-directory. The public sub-directory may or may not exist as per the sharing of databucket defined for a tenant user. For e.g., the directory path exists as `/pathvariable/tenantID_tenantName`. The directory name is the combination of tenant ID and the tenant name.

- Tenant user sub-directory - This sub-directory belongs to a user and contains the result files as and when generated by the user. The files contained within are retrieved when a user requests for the result by providing the file location. For e.g., the sub-directory path exists as `/pathvariable/tenantID_tenantName/userID`.

- User file - This file contains the result of the computation. The file can be of any format be it JSON or XML. The name of the file is a Universally Unique Identifier (UUID). Each time when a result is generated the result is stored in the file and is stored in the user sub-directory. For e.g., the file path exists as `/pathvariable/tenantID_tenantName/userID/UUID.json`. This address is later also used for retrieving the result.

The implementation of the data bucket can be managed in other form of static storage as well and this can be a prospective future work.

6.2 Django REST Framework

Django is a high-level Python web framework which supports the development of RESTful applications. It works on the MVC architectural pattern. The core Django MVC framework consists of an object-relational mapper which mediates between data models (defined as Python classes) and a relational database ("Model"); a system for processing requests with a web templating system ("View") and a regular-expression-based URL dispatcher ("Controller") [HKM09]. The resource model has been mapped to the relational database using 'Model' and the business logic resides within the 'View'. To access the resources the regular-expression based URLs are defined and are configured to perform a request based functionality. Realizing the use of four HTTP methods, the resource identification and operation to be performed on them are decided on the basis of the request made by the user, and the appropriate functionality is performed by checking the type of request made. We have incorporated also maintaining the session of the user to perform request based operations only for the users with certain roles and therefore, does not perform methods for those without enough permissions to preserve the tenant architecture as proposed.

6.3 Resource Model

The resource model described in the section 5.2, highlights the resources and their interaction. For the implementation, the resources have been defined as collection with resources, sub-collection of resources and singleton resources. A tenant is a collection with resource, containing tenant operator and tenant administrator as the sub collection of resource. Data store is another collection with resource containing databuckets as the resource. The SciPy library being the collection contains packages as the resources within whereas sub-packages are the sub-collection and their functions are the sub resources. Hence, there are three collection with resources and two sub-collection of resources for our system.

```
1  from django.db import models
2  # Model to define Tenant properties
3  class Tenant(models.Model):
4      tenantID = models.IntegerField(primary_key=True) # Unique
          integer value to identify the tenant
5      tenantName = models.CharField(max_length=100, blank=True,
          default='')
6      bucketSize = models.CharField(max_length=6) # databucket size
           assigned to the whole tenant
7      class Meta:
8          ordering = ('tenantID',)
9  # Model to define Tenant User properties
10 class User(models.Model):
11     # Used to restrict the user role to be either tenant
          administrator or tenant operator
12     roleChoices= (
13         ('TA','TenantAdmin'),
14         ('TO','TenantOperator'),
15     )
16     # Unique integer value for identify the user
17     userID = models.IntegerField(primary_key=True)
18     # Foreign key to Tenant model to relate user with the tenant
19     tenantID = models.ForeignKey('Tenant', related_name='Tenant.
          tenantID')
20     userName = models.CharField(max_length=100)
21     # Using the above defined user choices for role
22     userRole = models.CharField(max_length=2,choices=roleChoices,
          default='TO')
23     email = models.EmailField()
24     # Size of the databucket for sharing with other users in the
          tenant
25     shareBucket = models.Integer()
26     # Permission to share result within tenant
```

```
27      shareData = models.BooleanField()
28      bucketSize = models.CharField(max_length=6)
29      class Meta:
30          ordering = ('tenantID',)
31
32  class Databucket(models.Model):
33      tenantID = models.ForeignKey('Tenant',related_name='Tenant.
            tenantID')
34      userID = models.ForeignKey('User',related_name='User.userID')
35      # Unique integer value to identify the databucket
36      bucketID = models.IntegerField(primary_key=True)
37      # Databucket location to store or retrieve results
38      location = models.CharField(max_length=100)
39      # userID and shareBucket for sharing with other users
40      bucketInfo = models.ManyToManyField(User)
```

Listing 6.1: Model definition in Django REST Framework

In Django REST framework, the data models are in form of Python classes which enable inheritance. The model is shown to be consisting of the Tenant and Tenant User resource. They are mapped to a relational database in order to store the relevant details like, creating a tenant and then assigning the users to it. Some of the implemented data models are described in the listing above.

```
1  from model import Tenant, User
2  from rest_framework.renderers import JSONRenderer
3  from rest_framework.parsers import JSONParser
4  from rest_framework import serializers
5  class TenantSerializer(serializers.ModelSerializer):
6     tenantID=serializers.RelatedField(many=True)  # Allow multiple
          Tenant entry for serialisation
7     class Meta:
8          model = Tenant
9          fields = ('tenantID', 'tenantName', 'bucketSize')
10
11 class UserSerializer(serializers.ModelSerializer):
12    userID = serializers.RelatedField(many=True) # Allow multiple
         Tenant User entry for serialisation
13    class Meta:
14         model = User
15         fields = ('userID', 'tenantID', 'userName')
```

Listing 6.2: Serializer definition in Django REST Framework

To provide the support for various data formats for the application, Django "Serializers" are used. Serializers allow complex data such as querysets and model instances to be converted to native Python datatypes that can then be easily rendered into JSON, XML or other content types. Serializers also provide deserialization, allowing parsed data to be converted back into complex types, after first validating the incoming data. A class named "ModelSerializer" supports serilaization in Django REST framework to control the output of the responses to deal with the model instances and querysets.

```
from django.shortcuts import render
from django.http import HttpResponse
from django.views.decorators.csrf import csrf_exempt
from rest_framework.renderers import JSONRenderer
from rest_framework.parsers import JSONParser
from models import Tenant, User
from serializers import TenantSerializer, UserSerializer
class JSONResponse(HttpResponse):
    #An HttpResponse that renders its content into JSON.
    def __init__(self, data, **kwargs):
        content = JSONRenderer().render(data)
        kwargs['content_type'] = 'application/json'
        super(JSONResponse, self).__init__(content, **kwargs)
@csrf_exempt
def tenant_list(request):
    #To get the list of all the tenants
    if request.method == 'GET':
        tenants = Tenant.objects.all()
        serializer = TenantSerializer(tenants, many=True)
        return JSONResponse(serializer.data)
    #To add more tenants to the list
    elif request.method == 'POST':
        data = JSONParser().parse(request)
        serializer = TenantSerializer(data=data)
        if serializer.is_valid():
            serializer.save()
            return JSONResponse(serializer.data, status=201)
        return JSONResponse(serializer.errors, status=400)
@csrf_exempt
def user_list(request):
    #To get the list of all the tenant users
    if request.method == 'GET':
        users = User.objects.all()
        serializer = UserSerializer(users, many=True)
        return JSONResponse(serializer.data)
    #To add more tenant users to the list
```

```
37      elif request.method == 'POST':
38          data = JSONParser().parse(request)
39          serializer = UserSerializer(data=data)
40          if serializer.is_valid():
41              serializer.save()
42              return JSONResponse(serializer.data, status=201)
43          return JSONResponse(serializer.errors, status=400)
```

Listing 6.3: Class based View definition in Django REST Framework

A view is a Python function that takes a Web request and returns a Web response. The view contains the necessary logic to return the response for the request made by the user. A configured URL is attached to a particular view using URL dispatcher as shown in the Listing 6.3. The request is captured according to the HTTP method it uses and the particular function is invoked. The response can be in the desired format for example JSON , XML or even a standard HttpResponse. To enable JSON or XML response, appropriate serializer has to be defined and then called to support the format. The shown listing deals with providing JSON format response to the user. For the response to be in JSON and XML formats, Django offers renders and parsers for both the formats. To include the format in the URL for the resource identification, use of format suffix patterns is provided by Django to allow multiple format requests.

```
1  from django.conf.urls import patterns, include, url
2  from rest_framework.urlpatterns import format_suffix_patterns
3  from django.contrib import admin
4  from app_name import views
5
6  urlpatterns = patterns('',
7      url(r'^admin/', include(admin.site.urls)),
8      url(r'^tenants/$',views.tenant_list),
9      url(r'^tenants/(?P<pk>[0-9]+)/$',views.tenant_detail),
10     url(r'^tenants/users/$',views.user_list),
11     url(r'^tenants/(?P<pk>[0-9]+)/users/$',views.
          user_detail_unknown),
12     url(r'^tenants/(?P<pk>[0-9]+)/users/(?P<uk>[0-9]+)/$',views.
          user_detail)
13 )
14 urlpatterns = format_suffix_patterns(urlpatterns, suffix_required
     =False,allowed=['json', 'xml'])
```

Listing 6.4: URL defined for the resource identification in Django REST Framework

The urls are configured using various regular expressions to identify the resource. Basically the urls defined are mapped to the Python functions defined in the Views along with the

instance of HttpRequest. If the matched regular expression returned no named groups, then the matches from the regular expression are provided as positional arguments. If no regex matches, or if an exception is raised during any point in this process, Django invokes an appropriate error-handling view.

7 Validation

In this chapter, the prototype is validated as per the conditions mentioned in the use cases in section 4.2. It must be ensured that the requirements specified in the chapter 4 must be fulfilled in the design and implementation sections and hence, the requirements must be validated. For the validation, the responses for the requests made are checked to ensure whether they provide the expected response. The use case validation is performed in the next sections.

7.1 Computation and databuckets

The system allows using the SciPy library packages to perform computations using the statistical functions. The application supports providing the function name and the input parameters by the user and computing the results. If the input parameters are invalid then appropriate error message is generated. The function name provided must belong to the sub-packages defined within the four packages listed in the section 5.2. Another functions are not included in the scope of this work. The packages considered for our work are shown as a response in figure 7.1 using POSTMAN API Client[1]. The databuckets are implemented as a

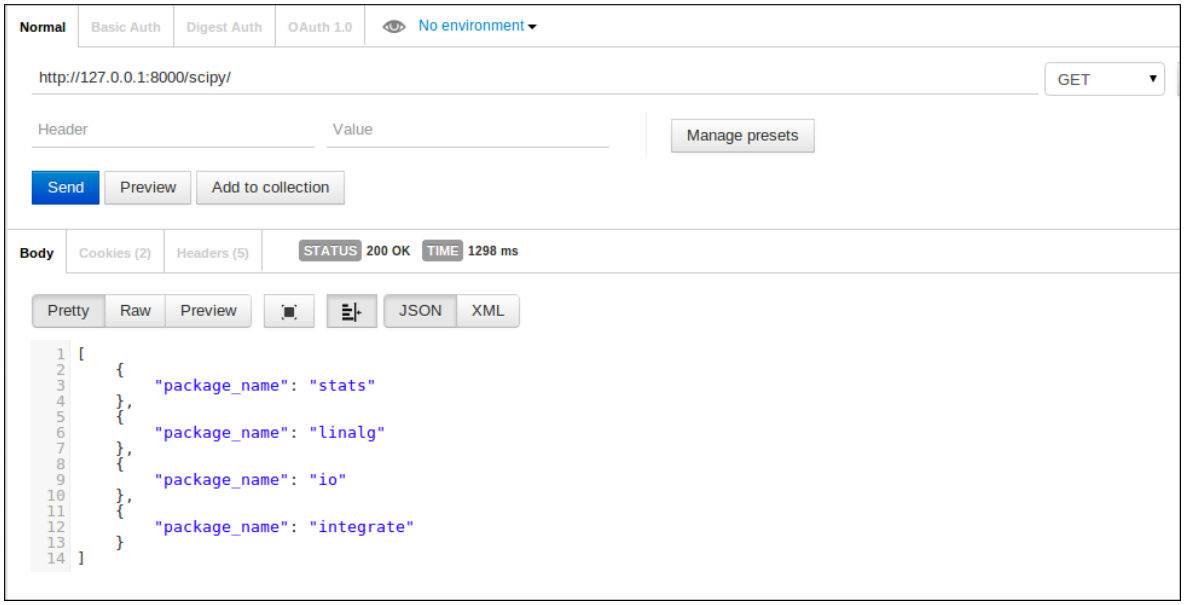

Figure 7.1: Request and response using POSTMAN API Client

[1]http://www.getpostman.com/

file storage system. Each tenant has a directory with sub-directories for the users. The results are stored in files and the location of the file is shared with the user which can be used for retrieving result. Other users if permitted can also use the location for referring to the result of another user. The files which are shareable are stored in the public folder of the tenant which can be accessed by any user across different tenants. For long running computations, the location of the file is saved but the user must poll to retrieve the result.

7.2 Validating REST APIs

Considering the scope of this thesis, the application is run on a local server and the data storage is done on the static storage on the server like in files on the system. The REST API Client used for the validation is Postman [pos], and the various operations have been performed on the resources to observe the response. The design of the REST API has considered all the resources and the interaction between them to allow the functionalities to be performed as defined in the requirements. The use cases as mentioned in the section before have led to responses like:

- Add and Remove Tenants - For the functionality of Add tenant, a POST method is requested providing the tenant details and a Status code is generated to show the success or the failure of the request. Similarly for removing a tenant, DELETE method is requested and the Status Code displays the success or failure of the request as a response.
  - URL:`/tenants/`
  - Method: POST
  - POST parameters with tenantName and databucket size
  - Status Code: 201(CREATED).
  - URL: `/tenants/{tenantID}/`
  - Method :DELETE
  - URL parameter as tenantID
  - Status Code: 200(OK).
- View Tenant Registry - For enabling to view the available tenants within the system at any point in time, this functionality provides the list of tenants registered for using the service.
  - URL: `/tenants/`
  - Method: GET
  - Status Code: 200(OK). The response contains the list of members for example in JSON format as:

```
[
    {
        "tenantID": 1,
        "tenantName": "ABC",
        "bucketSize": "5 MB"
    },
    {
        "tenantID": 2,
        "tenantName": "XYZ",
        "bucketSize": "5 MB"
    },
    {
        "tenantID": 3,
        "tenantName": "abc",
        "bucketSize": "5 MB"
    }
]
```

- Add and Remove Users from the tenant - For the functionality of Add user, a POST method is requested providing the user details and a Status code is generated to show the success or the failure of the request. Similarly for removing a user, DELETE method is requested and the Status Code displays the success or failure of the request as a response.
  - URL:`/tenants/{tenantID}/users/`
  - Method: POST
  - POST parameters with tenantID, userName, bucketsize, userRole, email, shareBucket and shareData
  - Status Code: 201(CREATED).
  - URL: /tenants/tenantID/users/userID/
  - Method :DELETE
  - URL parameter as tenantID and userID
  - Status Code: 200(OK).
- Trigger Computation - To allow the users to use the computational library, the user must provide the function name and the input parameters in the URL as query string to perform the calculation.
  - URL: `/scipy/packagename/subpackagename?function_name={}&params={}`
  - Method: GET

– Query string parameters as function name for the provided Package and sub-package name and the input parameters for the same.

– Status Code: 200(OK) and the computed result is shown in the desired format. e.g., for *scipy.stats.alpha.rvs()* function with input parameters as array [10,122,30,3], the result is shown in the figure 7.2 using POSTMAN API Client[2]:

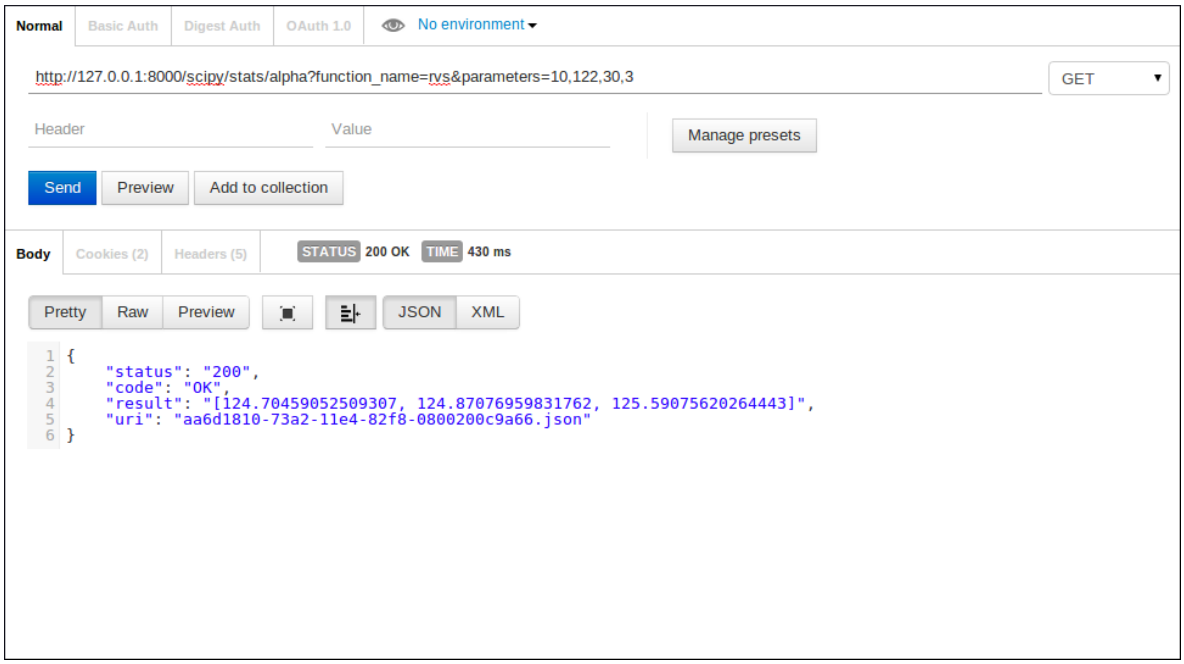

Figure 7.2: Request and response using POSTMAN API Client

[2]http://www.getpostman.com/

8 Outcome and Future Work

Cloud computing has opened gates to newer means to achieve computational results in the field of e-Science which made the recent development and researches to focus on cloud and move away from grid computing. From the previous times, grid computing due to its distributed nature was an obvious choice for the frameworks and toolkits to be developed for supporting e-Science works. But with the economical support the cloud offers, it has become interesting to dig deeper into providing better services. Using computational libraries and building applications around it will gain more popularity soon. The dynamic scaling of resources on the cloud helps the scientists and researchers from exact approximation of the need of resources or the computing power ahead in advance unlike for Grid Computing. Exposing functionalities as web services enables easy usability of such scientific libraries and provides a way to handle large results and share data among the group for the analysis and re-usability features. To built such web service applications, SaaS as a model proves to fit well providing all the necessary properties. PaaS and IaaS can further be dealt as and when the need grows for the application to be developed.

Regardless of the number of possible functionalities operable from a web service, the ease of using the application can be achieved by using an architectural style which is simple, modifiable and portable. REST being such an architectural style allows the user to identify the resources by using HTTP methods with appropriate URLs. These resource locators are designed to be comprehensible and understandable to the users. The standard HTTP methods available help the user to realise the operations that can be performed on the resources. The feasibility of providing the user with the choice of expecting response in the desirable format, makes REST an appropriate choice. Using SciPy libraries for such an application is beneficial for the e-Science teams as it is one of the most accepted and recognized open source libraries used due to its robustness, varied features and can easily be used with simple, expressive, and accessible syntax [VM08]. Using multi-tenant architecture makes it realistic to realize the use of such applications with the researchers and scientists at different geographical locations, in different organisational structures and data isolation needed at various level within the structure.

In chapter 2, the fundamentals relating to the technologies used in this thesis have been explained in detail and a foundation for the work is built. To describe what features of cloud computing and REST-based architecture for web service development are important for the implementation and how they supersede the other styles has been explained. Whereas in chapter 3, the ongoing researches in the development of similar kind of applications for e-Science have been considered to understand their design patterns, used technologies, reusable concepts and work if any close to our thesis and challenges faced to built or own application.

The functional and the non-functional requirements have been specified in chapter 4 which must be provided by the system. The focus is on expressing the requirements for a multi tenant system with key features of data isolation and data sharing at a restricted level. Defining the necessary use cases and consolidating all the requirements for the application, chapter 5 defines the design for the prototype to be built. While dealing with resources, the interactions and possible operations that can be performed on them as per the user roles have been defined in detail so as to built the prototype. The database schemas as Tenant and Configuration registries have been defined to express the kind of information stored. The design of the data storage for an individual user in form of data buckets has been defined and the storage capabilities have been explained. Using the concepts developed in chapter 4 and the design explained in chapter 5 leads to chapter 6 for the implementation details.

Using Django REST framework for designing the API has been explained in chapter 6 and the data models for the resources and possible operations to provide the SciPy function to the users has been explained. Chapter 7 discusses the validation for the system by describing the HTTP responses for the operations on the resources. Also, as the system allows varied format of responses, hence the JSON and XML responses are shown as well. The application is validated by running it on the local server and extending the data stored in static storage and performing operation on it.

Future work related to the system can be expressed in terms of allowing more functions to be exposed as the service for the users. Considering more libraries offered by other languages can be integrated together to enable freedom of usability under a single web service. Since, the actual prototypical implementation does not deal with hosting the application on the cloud, the dynamic scaling of the resources can be expressed with extensible design and implementation. Similarly, implementing different algorithms to enable the cloudburst scheduler, monitor and manager is considered a prospective future work. Designing the databuckets in order to share it partially or completely with the tenant users can be dealt in with different approach and can be handled with other implementation techniques. Customizing the properties of data bucket for maintaining data versions in order to restore results can be considered as future work. Furthermore, more secured authentication mechanisms can be dealt with and refinement of the Role based access and defining permissions at an individual level can prove to be an extended work of interest as well.

Bibliography

[ABK+05] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, and I. Foster. The Globus Striped GridFTP Framework and Server. In *Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*, SC '05, pages 54–, Washington, DC, USA, 2005. IEEE Computer Society.

[ABLS13] V. Andrikopoulos, T. Binz, F. Leymann, and S. Strauch. How to Adapt Applications for the Cloud Environment. *Computing*, 95:493–535, 2013.

[ACK12] P. Aghera, S. Chaudhary, and V. Kumar. An Approach to Build Multi-tenant SaaS Application with Monitoring and SLA. In *Proceedings of the 2012 International Conference on Communication Systems and Network Technologies*, CSNT '12, pages 658–661, Washington, DC, USA, 2012. IEEE Computer Society.

[ACL+05] S. Arana, F. Curbera, F. Leymann, T. Storey, and D. F. Ferguson. *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More*. Prentice Hall PTR, 2005.

[ASV13] A. Afanasiev, O. Sukhoroslov, and V. Voloshinov. MathCloud: Publication and Reuse of Scientific Applications as RESTful Web Services. In V. Malyshkin, editor, *PaCT*, volume 7979 of *Lecture Notes in Computer Science*, pages 394–408. Springer, September,2013 2013.

[BYV+09] R. Buyya, C. Yeo, S. Venugopal, J. Broberg, and I. Brandic. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, Volume 25 Issue 6:599–616, June 2009.

[BZ10] C. P. Bezemer and A. Zaidman. Multi-tenant SaaS Applications: Maintenance Dream or Nightmare? In *Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE)*, IWPSE-EVOL '10, pages 88–92, New York, NY, USA, 2010. ACM.

[CALB10] D. Candeia, R. Araujo, R. Lopes, and F. Brasileiro. Investigating Business-Driven Cloudburst Schedulers for E-Science Bag-of-Tasks Applications. In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pages 343–350, November 2010.

[CCMW11] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1, March 2011.

[CDK+02] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana. Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI. *IEEE Internet Computing*, 6(2):86–93, 2002.

[Che11] R. Chen. *Intelligent computing and information science*, volume 135 of *Communications in Computer and Information Science*. Communications in Computer and Information Science 135. Berlin:Springer., January 2011. Page Number 115.

[com14] O. community. GNU Octave 3.8.1, 2014.

[CSB10] S. Cholia, D. Skinner, and J. Boverhof. NEWT: A RESTful service for building High Performance Computing web applications. In *Gateway Computing Environments Workshop (GCE), 2010*, pages 1–11, Nov 2010.

[dMRF+10] J. V. der Merwe, K. Ramakrishnan, M. Fairchild, A. Flavel, J. Houle, and H. L.-C. J. Mulligan. Towards a ubiquitous cloud computing infrastructure. In *Local and Metropolitan Area Networks (LANMAN), 2010 17th IEEE Workshop*, pages 1–6, May 2010.

[DWC10] T. Dillon, C. Wu, and E. Chang. Cloud Computing: Issues and Challenges. In *Proceedings of the 2010 24th IEEE International Conference on Advanced Information Networking and Applications*, AINA '10, pages 27–33, Washington, DC, USA, 2010. IEEE Computer Society.

[Far08] D. Farber. Oracle's Ellison nails cloud computing. http://www.cnet.com/news/oracles-ellison-nails-cloud-computing/, 2008.

[FEL+12] C. Fehling, T. Ewald, F. Leymann, M. Pauly, J. Rütschlin, and D. Schumm. Capturing Cloud Computing Knowledge and Experience in Patterns. In *Proceedings of the 5th IEEE International Conference on Cloud Computing, CLOUD 2012*, pages 726–733. IEEE Computer Society, 2012.

[Fie00] R. T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000. AAI9980887.

[Fos02] I. Foster. What is the Grid? - a three point checklist. *GRIDtoday*, 1(6), July 2002.

[FZRL09] I. T. Foster, Y. Zhao, I. Raicu, and S. Lu. Cloud Computing and Grid Computing 360-Degree Compared. *CoRR*, abs/0901.0131:1–10, 2009.

[GHJV95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[GLT09] J. C. Glover, V. Lazzarini, and J. Timoney. SIMPL: A Python Library for Sinusoidal Modelling. *DAFx 09 proceedings of the 12th International Conference on Digital Audio Effects, Politecnico di Milano, Como Campus, Sept. 1 - 4, Como,*

Italy, pages 1–4, September 2009. The authors would like to acknowledge the generous support of An Foras Feasa, who funded this research.

[GLZ+10] C. Gong, J. Liu, Q. Zhang, H. Chen, and Z. Gong. The Characteristics of Cloud Computing. In W.-C. Lee and X. Yuan, editors, *ICPP Workshops*, pages 275–279. IEEE Computer Society, 2010.

[GSH+07] C. J. Guo, W. Sun, Y. Huang, Z. H. Wang, and B. Gao. A Framework for Native Multi-Tenancy Application Development and Management. In *E-Commerce Technology and the 4th IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services, 2007. CEC/EEE 2007. The 9th IEEE International Conference on*, pages 551–558, July 2007.

[GVB11] S. Garg, S. Versteeg, and R. Buyya. SMICloud: A Framework for Comparing and Ranking Cloud Services. In *Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on*, pages 210–218, Dec 2011.

[HKM09] A. Holovaty and J. Kaplan-Moss. *The Definitive Guide to Django: Web Development Done Right, Second Edition*. Apress, Berkely, CA, USA, 2nd edition, 2009.

[HT03a] T. Hey and A. Trefethen. *The Data Deluge: An e-Science Perspective*, chapter 36, pages 809–824. John Wiley & Sons, Ltd, 2003.

[HT03b] T. Hey and A. Trefethen. e-Science and its implications. *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 361(1809):1809–1825, 2003.

[HWWL10] H. Hugo, P. Watson, S. Woodman, and D. Leahy. *E-Science Central: Cloud-based E-science and Its Application to Chemical Property Modelling*. Technical report series. Newcastle University, Computing Science, 2010.

[IG96] R. Ihaka and R. Gentleman. R: A Language for Data Analysis and Graphics. *Journal of Computational and Graphical Statistics*, 5(3):299–314, 1996.

[JBBvN05] S. Jansen, S. Brinkkemper, G. Ballintijn, and A. van Nieuwland. Integrated Development and Maintenance of Software Products to Support Efficient Updating of Customer Configurations: A Case Study in Mass Market ERP Software. *2013 IEEE International Conference on Software Maintenance*, 0:253–262, 2005.

[JOP+01] E. Jones, T. Oliphant, P. Peterson, et al. SciPy: Open source scientific tools for Python, 2001. [Online; accessed 2014-10-13].

[JOP11] V. Josyula, M. Orr, and G. Page. *Cloud Computing: Automating the Virtualized Data Center*. Cisco Press, 2011.

[KNL08] T. Kwok, T. Nguyen, and L. Lam. A Software as a Service with Multi-tenancy Support for an Electronic Contract Management Application. *2013 IEEE International Conference on Services Computing*, 2:179–186, 2008.

[Koz11] H. Koziolek. The SPOSAD Architectural Style for Multi-tenant Software Applications. In *WICSA*, pages 320–327. IEEE Computer Society, 2011.

[KP88] G. Krasner and S. Pope. A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80. *Journal of Object-Oriented Programming – JOOP*, 1(3):26–49, August/September 1988.

[KSAK14] R. Krebs, S. Spinner, N. Ahmed, and S. Kounev. Resource Usage Control in Multi-tenant Applications. In *2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, Chicago, IL, USA*, pages 122–131. IEEE, May 26-29, 2014 2014.

[LLLC08] X. H. Li, T. Liu, Y. Li, and Y. Chen. SPIN : Service Performance Isolation Infrastructure in Multi-tenancy Environment. In A. Bouguettaya, I. Krüger, and T. Margaria, editors, *Service-Oriented Computing - ICSOC 2008, 6th International Conference, Sydney, Australia, December 1-5, 2008. Proceedings*, volume 5364 of *Lecture Notes in Computer Science*, pages 649–663, 2008.

[Mas11] M. Masse. *REST API Design Rulebook - Designing Consistent RESTful Web Service Interfaces*. O'Reilly Media, 2011.

[Mat09] Matlab. MATLAB - The Language Of Technical Computing. 2009. The MathWorks.

[MG11] P. Mell and T. Grance. The NIST Definition of Cloud Computing. 2011.

[Muh12] D. Muhler. Extending an open source enterprise service bus for multi-tenancy support focusing on administration and management. Master's thesis, Universität Stuttgart, Holzgartenstr. 16, 70174 Stuttgart, 2012.

[Nat12] Y. V. Natis. Gartner reference model for elasticity and multitenancy. Technical report, Gartner, Inc, 2012.

[OI06] H. Ohsaki and M. Imase. Performance Evaluation of Data Transfer Protocol GridFTP for Grid Computing. November 2006.

[Pal07] M. Paluszek. Coordinating Distributed Loops and Fault Handling, Transactional Scopes using WS- Coordination protocols layered on WS-BPEL services. Diplomarbeit, Universität Stuttgart,Institut für Architektur von Anwendungssystemen, Universitätsstraße 38D – 70569 Stuttgart, March2007 2007. Page Number 11.

[Pap03] M. P. Papazoglou. Service -Oriented Computing: Concepts, Characteristics and Directions. In *Proceedings of the Fourth International Conference on Web Information Systems Engineering*, WISE '03, page 3, Washington, DC, USA, 2003. IEEE Computer Society.

[Pet13] D. Petcu. Multi-Cloud: Expectations and Current Approaches. In *Proceedings of the 2013 International Workshop on Multi-cloud Applications and Federated Clouds*, MultiCloud '13, pages 1–6, New York, NY, USA, 2013. ACM.

[pos] POSTMAN REST API Client.

[RJ05] G. D. Reis and J. Järvi. What is generic programming. In *In LibraryCentric Software Design, 2005. OOPSLA workshop*, 2005.

[RMCL09] L. M. V. L. Rodero-Merino, J. Caceres, and M. Lindner. A break in the clouds: towards a cloud definition. *ACM SIGCOMM Computer Communication Review*, Volume 39 Issue 1:50–55, January 2009 2009.

[RR07] L. Richardson and S. Ruby. *Restful Web Services*. O'Reilly, 2007.

[SA14] O. Sukhoroslov and A. Afanasiev. Everest: A Cloud Platform for Computational Web Services. In *Proceedings of the 4th International Conference on Cloud Computing and Services Science*, pages 411–416. SCITEPRESS - Science and and Technology Publications, 2014.

[SASL13] S. Strauch, V. Andrikopoulos, S. G. Sáez, and F. Leymann. ESB: A Multi-tenant Aware Enterprise Service Bus. *International Journal of Next-Generation Computing*, 4(3):230–249, 2013.

[SHK14] P. Schneider, N. Herbst, and R. Krebs. Optimization Method for Request Admission Control to Guarantee Performance Isolation. In *Proceedings of the 2nd International Workshop on Hot Topics in Cloud Service Scalability (HotTopiCS 2014), co-located with the 5th ACM/SPEC International Conference on Performance Engineering (ICPE 2014)*. ACM, March 2014.

[SK09] M. H. Selamat and A. A. Kharusi. Service Oriented Architecture in Education Sector. *IJCSNS International Journal of Computer Science and Network Security*, 9(5):301–308, May 2009.

[Sud03] B. Suda. SOAP Web Servcie Implementation. Master's thesis, The University of Edinburgh, November 2003. Page Number 3.

[Sá12] S. G. Sáez. Integration of different aspects of multi-tenancy in an open source enterprise service bus, 2012.

[VM08] T. Vaught and J. Millman. The State of SciPy. In G. Varoquaux, T. Vaught, and J. Millman, editors, *Proceedings of the 7th Python in Science Conference*, page 5, Pasadena, CA USA, 2008.

[W3C04] W3C Working Group. Web Services Architecture, 2004.

[W3S] W3SCHOOLS. HTTP Status Messages.

[WoNuTCS10] P. Watson and U. of Newcastle upon Tyne. Computing Science. e-Science Central, 2010.

[WWWK94] J. Waldo, G. Wyant, A. Wollrath, and S. Kendall. A Note on Distributed Computing. *Sun Microsystems Laboratories*, 1994.

[XBXW10] Z. Xuesong, S. Beijun, T. Xucheng, and C. Wei. From isolated tenancy hosted application to multi-tenancy: Toward a systematic migration method for web application. In *Software Engineering and Service Sciences (ICSESS), 2010 IEEE International Conference on*, pages 209–212, July 2010.

All links were last followed on November 25, 2014

Acknowledgement

I am sincerely thankful to my supervisors Santiago Gómez Sáez and Johannes Wettinger from the University of Stuttgart for their encouragement, guidance and support in all the phases of my master thesis. I thank them for giving me an opportunity to learn from them. Special thanks to my family and friends for their help and moral support.

Sugandha Agrawal

Declaration

I hereby declare that the work presented in this thesis is entirely my own. I did not use any other sources and references than the listed ones. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

Stuttgart, November 25, 2014 _____

(Sugandha Agrawal)