National College of Ireland
Project Submission Sheet – 2014/2015
School of Computing

Student Name: Jitendra Kumar Sharma
Student ID: 01315057
Programme: MSc Cloud Computing Year: 2015
Module: Dissertation
Lecturer: Dr. Horacio González-Velez
Submission Due Date: 17/12/2015
Project Title: OpenStack cloud federation with Single Sign-On via an Identity Management System
Word Count: 15896

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project. ALL internet material must be referenced in the bibliography section. Students are encouraged to use the Harvard Referencing Standard supplied by the Library. To use other author’s written or electronic work is illegal (plagiarism) and may result in disciplinary action. Students may be required to undergo a viva (oral examination) if there is suspicion about the validity of their submitted work.

Signature: Jitendra Kumar Sharma
Date: 16/12/2015

PLEASE READ THE FOLLOWING INSTRUCTIONS:

1. Please attach a completed copy of this sheet to each project (including multiple copies).
OpenStack cloud federation with single sign-on via an Identity Management System

Jitendra Kumar Sharma

Submitted as part of the requirements for the degree of MSc in Cloud Computing at the School of Computing, National College of Ireland, Dublin, Ireland.

December 2015

Supervisor Dr. Horacio Gonzalez-velez
Abstract

The increasing popularity of cloud computing has led to the increased use of virtualization technologies and underlying features. Many virtualization and cloud platforms are being used independently or in conjunction with another environment to offer cloud services. Most of these cloud services are being offered and shared across multiple customers. Multiple customers on the same cloud enable features like multi-tenancy and resource sharing. However, this also raises a few concerns on the identity of the cloud users and confidentiality of the data and its privacy. Many researchers are still working on multi-tenancy, data protection and identity management in the field of cloud computing. Hence, this paper presents a Single Sign-On (SSO) solution for a data center that addresses the issues of multi-tenancy and cloud security for IaaS and SaaS applications. The research is carried out with the goal to develop a pluggable middleware for cloud authentication. The middleware should be able to provide SSO using a custom built identity management system called ‘Membership’. It is possible to integrate cloud services on the service level or the database level. Hence, this paper further discusses the research and testing executed to ensure optimal approach on how Single Sign-On, IDS system ‘Membership’ and cloud services can be integrated to mitigate security risks with the multi-tenant cloud. As a proof of concept, this paper develops a fully functional cloud, offering IaaS and SaaS solutions served over a Single Sign-On.

Keywords: Single Sign-On, SSO, Identity Federation, OpenStack, Cloud, Keystone, IDM, OAuth.
Declaration

I hereby declare that the dissertation entitled ‘OpenStack cloud federation with Single Sign-On via an Identity Management System’, is a bonafide record composed by myself, in partial fulfilment of the requirements for the MSc in Cloud Computing (2014/2015) programme. This has not been used and accepted in any previous application for a degree. All sources of information have been properly acknowledged.

Signed: Jitendra Kumar Sharma

Date: 16/12/2015
Prima facie, I am grateful to National College of Ireland, Dublin and Cork Internet eXchange (CIX), Cork for giving me the opportunity to carry out this research.

This thesis ‘OpenStack cloud federation with Single Sign-On via an Identity Management System’ for Masters in Science degree was carried out in the Cloud Competency Center, National College of Ireland, Dublin and Cork Internet eXchange, Cork.

I would like to express my sincere gratitude to my academic supervisor, Dr. Horacio Gonzalez-velez for the continuous support in completing this research, for his patience, feedbacks and suggestions and sharing immense knowledge. I am grateful to him for all his time and efforts helping me complete the thesis. It would have never been possible for me to carry out the research up to this level without his support and encouragement.

With an immense pleasure, I would also like to thank my industry supervisor Mr. Jerry Sweeney (Managing Director, CIX), Karen O’Connell (Director & Financial Controller, CIX) and Bob Grantham (Business Development Manger, CIX) who provided me an opportunity to join their team for this research and allowed access to the research facilities. Without their precious support, it would not have been possible to conduct this research.

Besides, I would also like to thank Vikas Sahni, Keith Brittle, Dr. Adriana Chis, NCI staff and IT support for helping me in the successfull completion of this course.

Last but not the least, I would like to thank my friends and family for continuous support and motivation throughout writing this thesis.
Disclaimer

All the information and content in this thesis is the property of Cork Internet eXchange (CIX), Cork, Ireland and has been developed under the guidance of National College of Ireland, Dublin. The content is protected by copyright laws, trademark and design rights.

Any unauthorised use of the content will be considered a violation of Cork Internet eXchange intellectual property rights. Unless otherwise stated in this document, CIX reserves all tacit and direct rights to patents, trademarks, copyrights or confidential information relating to the content of this thesis.

Unless otherwise stated in this document, no content may be copied, distributed, published or used in any way, in whole or in part, without prior written agreement from Cork Internet eXchange.
Contents

Abstract ii

Declaration iii

Acknowledgement iv

Disclaimer v

1 Introduction 1
   1.1 Background and Motivation 1
   1.2 Problem Statement 2
   1.3 Hypothesis 2
   1.4 Contribution 3
   1.5 Outline of the Thesis 3

2 Literature Review 5
   2.1 Single Sign-On (SSO)- Application and Advantages 7
   2.2 Single Sign-On Frameworks 8
   2.3 Insight on OAuth 9
   2.4 Identity Management 11
   2.5 Membership 12
   2.6 Identity Federation 14
   2.7 OpenStack Identity Federation 16

3 Design 20
   3.1 Specification 20
   3.2 Design Overview 21
   3.3 Abstract Protocol Flow 22
   3.4 Application Registration 23
   3.5 Workflow 24
   3.6 Openstack Federation Design 25
3.7 Additional Functionalities ................................. 28
  3.7.1 Problem Description ................................. 28
  3.7.2 Required Functionalities ............................. 28
  3.7.3 Impact on Clients ................................. 28
  3.7.4 Client Implementation ............................. 28
  3.7.5 Client Workflow ................................. 29

4 Implementation ........................................ 30
  4.1 SSO Installation and Configuration ....................... 31
    4.1.1 Refreshing a Token ............................. 32
    4.1.2 SSO Login and Token Grant ....................... 33
    4.1.3 Auth Server ................................. 36
  4.2 Keystone Installation & integration with Membership ........ 38
    4.2.1 Keystone Installation ........................... 38
    4.2.2 keystone.conf configuration to use CloudCIX-Keystone ....... 40
    4.2.3 Test Run ..................................... 41

5 Evaluation ........................................ 42
  5.1 Requirements and Setup ................................ 42
  5.2 Use Cases ......................................... 43
    5.2.1 Use–Case Basic Model ............................ 43
    5.2.2 List of Use Cases ............................... 44
    5.2.3 Use Case Diagrams .............................. 44
    5.2.4 Use Cases Details .............................. 47
  5.3 CloudCIX Performance testing with Rally .................. 50

6 Conclusions ...................................... 56
List of Figures

3.1 Helicopter Overview of CloudCIX .......................... 21
3.2 CloudCIX Architecture ........................................ 22
3.3 Abstract Code Flow ............................................ 23
3.4 Authorization Code Flow ....................................... 24
3.5 Implicit Code Flow .............................................. 25
3.6 Keystone Identity Manager ................................. 26
3.7 Keystone Components ........................................ 27
3.8 OpenStack Ecosystem .......................................... 27

4.1 Network Design ................................................ 31
4.2 CloudCIX Login ................................................... 34
4.3 CloudCIX Dashboard .............................................. 37
4.4 CloudCIX Compute ............................................... 38
4.5 CloudCIX Instances .............................................. 41

5.1 User authenticated for one of the frontend .................. 44
5.2 Browser session expired-Token Valid ......................... 45
5.3 Browser Session valid-Token Invalid ......................... 46
5.4 Rally Architecture ................................................ 51
5.5 Total durations .................................................... 54
5.6 Parallel Iteration .................................................. 54
5.7 Distribution ........................................................ 55
List of Tables

2.1 OpenStack Federation Comparison . . . . . . . . . . . . . . . . 19
5.1 Installation Requirements . . . . . . . . . . . . . . . . . . . . . 42
5.2 Use Case:1 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 47
5.3 Use Case:2 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 48
5.4 Use Case:3 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 49
5.5 User Creation Response . . . . . . . . . . . . . . . . . . . . . . . 53
Listings

4.1 Access Token Request ........................................... 32
4.2 Access Token Response ........................................... 32
4.3 Refresh Token Request ........................................... 32
4.4 Refresh Token Response ........................................... 33
4.5 CloudCIX Login Form ............................................. 35
4.6 Backend Authentication ........................................... 37
4.7 Keystone Installation ............................................. 39
4.8 keystone.conf ..................................................... 40
4.9 Test Run .......................................................... 41
5.1 Keystone authentication Testing ................................. 50
5.2 Rally Installation .................................................. 52
5.3 Environment Variable ............................................. 52
5.4 Benchmark Scenario .............................................. 53


Chapter 1

Introduction

1.1 Background and Motivation

This thesis aims at developing a commercial cloud while considering the risks associated with multi-tenancy and identity implementations. The purpose is to eliminate security risks associated with the cloud. The research focuses on the implementation of a Single Sign-On solution using a proprietary identity management solution. This identity management solution is proprietary to the data center and is called ‘Membership’ and is tightly coupled with the cloud services and offerings.

The importance of identity management is vital in managing the identity of an entity in cloud. It relates to multi-tenancy and the extent to which it can help mitigate the security risks. ‘Membership’ is also used as a basis for Software as a Service (SaaS) applications developed for customers. It is capable of implementing true multi-tenanted SaaS applications, as distinct from multi-instance SaaS applications. With the proposed cloud, internal systems, and the SaaS product offerings operating from a single identity framework is the optimal solution for the data center.

The importance of Single Sign-On explained below, describes the concepts and advantages of choosing the right framework for implementing Single Sign-On. This also suggests the reasons why Single Sign-On is important for multi-tenancy. There are many popular open standard framework for authentication and authorization. Some of these are SAML, OpenID, and OAuth which are considered and researched for this SSO solution. Single Sign-On to access cloud IaaS resources and SaaS applications reduces the complexity of managing multiple user accounts for different services.
The proposed SSO solution provides the platform using which cloud members registered in ‘Membership’ handshake with the cloud services to authenticate the user. In general, most of the cloud platforms handle authentication by using native username and password. These credentials are stored in its local database. However, with the help of proposed SSO solution and integration with ‘Membership’, a specific URL is created that acts as a provider of authentication for all frontends. This will also be able to create, refresh and scope access tokens on behalf of other frontends. After receiving the access token, the client shall validate it against cloud services. Based on the validation, it either redirect to SSO login (if token becomes invalid during the process) or create a local browser session using data received from the SSO response. The Single Sign-On along with ‘Membership’ allow users to use same credentials to authenticate for all the resources. It does not need to register individually for each cloud resource offerings. This helps eliminate the security risks and complexity of managing users and resource access in a multi-tenant environment.

1.2 Problem Statement

The data center already has few SaaS applications in place which are used by different customers with varying needs. The problem was to develop a cloud to offer IaaS services as well and link all the SaaS and IaaS offering together. Both the offerings should be accessible via a Single Sign-On. The challenge was to write a new Identity Management System keystone which should be pluggable to any cloud environment to authenticate users. The new Identity Management System should allow access rights based on privileges defined in the data center’s proprietary IDM system ‘Membership’. This custom built keystone should not only manage users for IaaS offerings but also control access to SaaS offerings through a Single Sign-On.

1.3 Hypothesis

The hypothesis for the research problem ‘Is it possible to federate cloud with Single Sign-On’ in this thesis can be claimed as ‘This is possible to federate cloud with Single Sign-On’. The hypothesis will be tested against the null hypothesis. The null hypothesis can be listed as ‘This is not possible to federate cloud with Single Sign-On’. The null hypothesis will be tested and evaluated against few test cases which are explained in detail in ‘Evaluation’ section.
1.4 Contribution

The contribution of this dissertation is the research carried out to find the feasibility of integrating OAuth based Single Sign-On with custom built Identity Management System. During the evaluation, the requirements considered are to provide SSO to the tenants. They should be able to access SaaS applications and OpenStack-based IaaS offerings using the same login. In order to provide this solution, multiple experiments have been carried out. As a result, it offered us a robust private cloud named CloudCIX. This CloudCIX can be commercially used eliminating the risk of multi-tenancy. Few of the requirements to carry out the experiment are as listed below

1. Implementation of a private cloud offering Iaas services.
2. Creation of multiple python based SaaS applications using python and .Net frameworks offering SaaS services.
3. Creation of OAuth based configuration for SSO to access resources.

The research component to make the above experiments feasible is as listed below.

1. To build a trust with cloud internal Identity Management System and ‘Membership’.
2. Implementation of required configuration in the cloud configuration files, ‘Membership’ and SaaS applications. This includes importing public certificates, importing key pairs for the service provider, defining custom scopes in ‘Membership’, disabling inbuilt user management of cloud, configuring SaaS applications to be controlled via this trust and ‘Membership’.
3. Testing if the users are validated based on the scope defined in ‘Membership’ and if the user belonging to a tenant is granted access to the resources defined for the tenant.
4. Testing if an independent user is granted access native to particular resources and no other offerings for a tenant.
5. Authentication is based on the configuration defined in ‘Membership’.

1.5 Outline of the Thesis

The research problem in this industrial dissertation: ‘Cloud federation with Single Sign-On and an Identity Management System’ studies different available Single
Sign-On frameworks. The purpose is to provide a cloud with safe and secure multi-tenant cloud environment. More significantly, this research claims that OAuth based solution is the best suited SSO framework for proposed cloud federation. It supports cloud customization for better identity management in a multi-tenant cloud.

To research the feasibility of Single Sign-On mitigating multi-tenancy risks in the cloud, this research has been divided into four sections. The first section focuses on literature review on the related work. This section analyzes similar works done by different authors. It discusses the importance of Single Sign-On, Identity Management and details on proprietary Identity Management System ‘Membership’. This section also compares and contrasts previous cloud federations and their challenges. The second section ‘Design’ includes the strategies on the integration of the proposed framework with CloudCIX and ‘Membership’. This section discusses in detail the architecture of cloud’s inbuilt Identity Management System. It also discusses ‘Membership’ and its working schema with the underlying database. Further, this section tries to include the algorithm for integrating CloudCIX with ‘Membership’ and support Single Sign-On with the help of OAuth.

The third section of this thesis includes all the work involved in implementing the private cloud, integration of cloud with ‘Membership’, design, implementation and configuration of SSO to support multi-tenancy. The last section ‘Evaluation’ records the performance of cloud Single Sign-On with the help of test user accounts and presents the findings while accessing multi-tenant cloud resources and their access management from ‘Membership’.
Chapter 2

Literature Review

The rapid growth of technology and increased user dependency on it has led to the development of new computing paradigms. The usage of computing resources can vary at times with changing user needs. Mishra et al. (2012) suggest that typical data centers are provisioned to cater for the peak hour demands. It results in wastage of resources during non-peak hours. These challenges of varied resource demand, usage, and efficient resource utilization are sorted by the use of virtualization.

However, resource utilization on different virtualized servers can also vary based on the services and applications it is hosting. Mishra et al. (2012) mention that changing workload on VMs can create ‘hot spots’ i.e. resource crunch or ‘cold spots’ means underutilization of resources. This condition defies the whole purpose of virtualization as both conditions result in no uniformity in resource utilization and performance. This further paved path for new paradigm called cloud computing. Cloud computing offers compute, storage and network services on demand and users pay for only what they use.

Cloud computing today has become a crucial component for IT industry. It is extensively helping companies manage resources and meet resource requirements within the budget. National Institute of Standards and Technology (NIST) in furtherance of its statutory responsibilities under the Federal Information Security Management Act, outlined the formal definition of cloud computing and its architectural reference.

Mell and Grance (2010) of NIST define cloud computing as ubiquitous, on-demand access to shared computing resources. They say these resources are connected via a network and provisioned over the internet. They further add that these shared computing resources should be easily configurable and should be provisioned with minimum
management efforts. Mell and Grance (2010) compare cloud computing close to distributed computing where efficient use of computing resources can help to optimize capital expenditure and operational expenditure.

Similarly, Armbrust et al. (2010) in their definition of cloud computing, include both the applications which are provided over the internet and the hardware and the system software in the datacenters that provide those services. Armbrust et al. (2010) refer cloud services as Software as a Service defying other vendors who also refer it as IaaS (Infrastructure as a Service) and Paas (Platform as a Service). They consider both the services to be together and more alike than different.

Mell and Grance (2010) from NIST add to what Armbrust et al. (2010) suggest. They define cloud computing further with minute details and explain cloud with five essential characteristics, three service models and four deployment models for cloud computing. The five characteristics of the cloud they list are On-demand self-service, Broad network access, Resource pooling, Rapid Elasticity and Measured services. They categorize the cloud service offerings by defining three service models as Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS). Mell and Grance (2010) also suggest four deployment models which can be listed as Private Cloud, Community Cloud, Public Cloud and Hybrid Cloud.

As all the above definitions suggest cloud resources to be shared and delivered through the internet, it is hence also prone to various limitations and internet related vulnerabilities. Few of the authors have tried to list few challenges common in cloud computing. For example Dillon et al. (2010) consider availability, performance, security, hard to integrate with in-house IT, not enough ability to customize, bringing resources back to in-house IT etc as some of the major challenges. Dillon et al. (2010) also point out various security issues with multi-tenancy in the cloud. First being, shared resources like hard disks, data and virtual machines etc on the same physical server. This makes it prone to malicious side channels between a malicious resource and a genuine resource. They point out the second security issue with multi-tenancy as ‘reputation fate-sharing’ where multiple cloud users share same computing resource and network addresses. As a result, any notorious activity by any such users may be attributed to all the users without differentiating any real subverts from genuine users.

As they outline generic challenges related to cloud, Grobauer et al. (2011) from Siemens identify more detailed security issues and vulnerabilities specific to the IaaS and SaaS service models. They distinguish cloud vulnerabilities based on difficulties in implementing the controls on them. Grobauer et al. (2011) refer those vulnerabilities as control challenges. Other than control challenges, Cloud Security Allowance (2015)
points one of the main issues of cloud called multi-tenancy and related risks.

Cloud Security Allowance (2015) also explain that multi-tenancy issues can be of different types and related to different resources based on service models. They identify multi-tenancy issues in IaaS as same hardware shared by different tenants and in SaaS as the same application being accessed by multiple cloud users. This is in line to the work of other authors mentioned above who have identified issues with the cloud. Also, we can see above that multi-tenancy is a major concern for cloud consumers considering the security of identity and access management.

The next chapter of the thesis covers related work in the field of identity management and cloud security. It also discusses how Single Sign-On can help reduce the security risks of cloud offerings. Further, this paper critically analyzes various single Sign-On frameworks. It also discusses in detail the existing identity management application ‘Membership’ that can be integrated with these frameworks. The last segment of this section explains recent work on OpenStack cloud federation and our contribution to it. All the above-discussed sections are as explained below.

2.1 Single Sign-On (SSO)- Application and Advantages

SSO has been defined by Cloud Security Allowance (2015) as a methodology used to communicate the identity of the user to the service provider. Ghazizadeh et al. (2012) point the challenge of maintaining multiple username and password with growing number of applications. They advocate the need for a single login to address this challenge. Supporting this Tsai and Sun (2013) point that any web user would have multiple web accounts which are protected by passwords. They add that any user must be using around eight password protected applications per day. Referring this challenge Tsai and Sun (2013) suggest using Single Sign-On as a rescue. They mention that it helps in using one account that is linked with multiple applications and can provide access to them from the same login.

Revar and Bhavsar (2011) explain SSO as a process of gaining access to multiple resources by authenticating once. They explain the objective of SSO being reducing the number of user credentials in a heterogeneous environment and create balance for security, efficiency, and usability. Karunanithi and Kiruthika (2011) consider SSO as emerging technology employed in the process of identity management. It manages information about the identity of the users and control access to the company resources. Similar to Revar and Bhavsar (2011), they also enlist the objective of SSO to reduce
the number of credentials a user needs to remember. They suggest using SSO to create a more efficient and streamlined work environment for the user. Karunanithi and Kiruthika (2011) also suggest that SSO was developed to reduce the number of logins between various systems. They mention that as the name suggest, SSO is designed to take a various number of login credential user needs to use and reduce them to a single operation. Karunanithi and Kiruthika (2011) explain that the data that is required during the login are presented to an application without the user having to remember these values themselves.

Almulla and Yeun (2010) suggest same by highlighting that a user can be part of multiple sub-domains in an organization. They do not need to remember credentials for different organization within the domain but can use the same trusted credentials to log in. There can be multiple solutions to implement this. One such solution suggested by Celesti et al. (2010) is where trusted credential of one cloud can be used to log into the other. They call it SSO where authentication is done once to access various cloud resources. Albino Pereira et al. (2014) support the same suggesting that cloud users will not need to authenticate each and every time they need to access an application. Albino Pereira et al. (2014) support the concept of IAM that provides federation between organizations and provides SSO. These all point towards reducing the number of registration on the web. This will help reduce the trouble of remembering multiple user credentials. These users can be of different types as suggested by Ghazizadeh et al. (2012) who classify it as Identity Provider (IDP), Service Providers (SP) and web users. Ghazizadeh et al. (2012) describe the role of IDP is to authenticate users based on the credentials they supply. They further explain Service Provider that should trust and agree on the authenticated credentials to leverage access to resources.

Though, Tsai and Sun (2013) compare Single Sign-On with Relying Party (RP) model. They explain it as a model where IDP is also considered an application which is responsible for maintaining user identity and authenticating users.

The literature review above discussed the views of various authors on Single Sign-On and different elements of it. It is notable that almost all of them support SSO for efficient resource management and ease of operation. In continuation, the next section discusses different frameworks that support SSO implementation.

### 2.2 Single Sign-On Frameworks

There are many proprietary as well as open-source SSO frameworks. Few of them can be listed as Active Directory Federation services from Microsoft, CA single Sign-On
from CA Technologies, IBM Tivoli Access Manager and few open source like OAuth, SAML, OpenID etc. The SSO solution for CloudCIX is based on one of the above open source solutions. Hence, this section discusses few of these open standard SSO solutions in detail.

These open standard SSO solutions are widely used and supported in the industry. Cloud Security Alliance advocates this suggesting that all the cloud-based applications should be compatible with open federation standards. They suggest using OAuth, OpenID, and SAML. Similarly, Tsai and Sun (2013) suggest using SAML, OAuth and OpenID to design SSO for applications.

In support to these authors, Murukutla and Shet (2012) also proposed using the open source standard. They mention that there are three de-facto standards for providing Single Sign-On. They name them as SAML, OpenID, and open authorization mechanism. They suggest that where SAML and OpenID are standard for authentication, OAuth is more inclined towards authorization.

Different to these de-facto standards, there are other solutions as well like information card. Information cards are the personal digital identities which people use online and it is the key component of identity metasystems. Identity metasystems provides the architecture for digital identity which employs multiple digital identities based on multiple underlying technologies. Tsai and Sun (2013) support same stating that information cards are one of the frameworks that can be used for SSO solutions. Though many authors support using information card but it in actual, it is not that successful as compared to other solutions. Ferdous and Poet (2012) discard information cards citing the example of Microsoft. Microsoft discontinued their project of CardSpace due to increasing standard and wide acceptance of other open standard SSO solution.

### 2.3 Insight on OAuth

The official website for OAuth Community (2015) defines it as “an open protocol which allows secure authorization in a simple and standard method from the web, mobile, and desktop applications”. E. Hammer-Lahav (2010) from the Internet Engineering Task Force (IETF) states that OAuth helps in providing a method to clients for accessing resources on a server. This is done on behalf of a resource owner which can be different users. He further adds that using OAuth, an end user can authorize third party access to their resources without being user credential shared across different parties.

Sun and Beznosov (2012) define it as an open and standardized web resource authorization protocol. They suggest that it allows users to grant access to third party
application to their web resources. This is further done without the need of sharing their login details or all the data. They focused on the architecture of the OAuth and related security issues.

Mitchell (2015) explains that OAuth works by delegating user authentication to the service that hosts the user account. It authorizing third-party applications to access the user account. He explains in detail the OAuth roles and authorization grant types. According to Mitchell (2015), OAuth consists of four roles which can be listed as Resource Owner, Client, Resource Server and Authorization Server. He described each role as mentioned below:

**Resource owners:** They are the users who own a resource and authorize applications to access their accounts. This access to the account is controlled by the ‘Scope’ of the authorization granted. The scope can be extended to read or write access.

**Resource Server/Authorization Server:** The resource server and the authorization server work hand in hand. Where resource server hosts the protected user accounts, the authorization server verifies users and then issues an access token with defined scope to the application.

**Client:** A client is the applications which need access to the user accounts. The clients need to be authorized by the user and should be verified by the resource and authorization server.

He further explains that in OAuth Abstract Protocol Flow, there are four steps in obtaining an authorization grant and access token. This grant type obtained depends on the method used by the client to request authorization. This also depends on the grant types supported by the API. Mitchell further explains the four types of OAuth grants as:

**Authorization Grant:** This grant type is mostly used with server-side applications.

**Implicit Grant:** This grant type is used with web applications or mobile apps or, other words, applications that run on the user’s device.

**Resource owner password credential:** These grant types are used with trusted applications such as the ones owned by service itself.

**Client Credentials:** These are used with applications API access.

The design section discusses in detail how these grant types have been used in our implementation.

Sun and Beznosov (2012) also discuss OAuth and its security risks. They suggest that
Single Sign-On via OAuth is mostly implemented on existing web infrastructure and hence vulnerable to Cross-Site Scripting (XSS), Cross-Site Request Forgery etc. They also suggest that any vulnerability found in the browser can also lead to serious security breaches when messages pass between the RP and IDP. They focused their research on enhancing the security of OAuth by researching the extent of web vulnerabilities that can be leveraged to compromise OAuth SSO system. They also studied the causes and consequences, the frequency of compromise and how they can be prevented at the practical level.

In our implementation, we used SSO in combination with cloud federation for better identity management. Cloud federation and related work and contribution are discussed in next chapter.

2.4 Identity Management

According to Hansen et al. (2008) creating and managing individual identity is a central challenge of the digital age. Hansen et al. (2008) define identity management as programs or frameworks to validate the collection, authentication or use of identity and information linked to an individual. They suggest that traditional identity management systems are run by organizations who administer all mechanism for authentication. This is establishing confidence in an identity claim about ‘who’ he is and authorization i.e. what he should be allowed to do as well as any additional profiling or scoring of individuals.

Similarly, Spencer (2012) explains identity management by categorizing it further in five subgroups. These subgroups can be listed as authorization, authentication, federation, user account management and auditing & logging of users. Same as the above two authors, Faraji et al. (2014) also define identity management by categorizing it in three essentials. These three essentials are authentication, management of user access and audit and reports. These all authors suggest the purpose of identity management is to validate who is who and ensure the one claiming to be a person is really he is.

Hansen et al. (2008) differentiate traditional identity management system from recent work being more user-centric. They suggest that recent works attempt to put users in charge of when, where, how and to whom they disclose their personal information. They suggest that Identity Management System can help realize the potential of the IT age. It is making e-commerce transactions more secure and seamless, using multiple devices together, combating fraud or enabling yet unimagined services. Hansen et al. (2008) point out that digitization of information by collecting, storing and sharing of a
large amount of data can make privacy risks inherent in identity management systems. It is hence important for system designers not to apply blanket privacy rules to it to address the privacy risks. Instead, they must test and evaluate how good an IDM system protects privacy in the context that is accounting for the system’s purposes, participants and possible abuses. They suggest the need of a more user-centric Identity Management System which is complemented by Eap et al. (2007).

Eap et al. (2007) suggest that being proactive and vigilant is the best defense against identity theft and invasion of privacy. They suggest that no identity management system can provide full-proof security. According to them, the challenge of identity protection is, even more, complex in Service-Oriented Architecture. It is because users have their identity scattered across many services and they have no control over the management of these identities. They advocate making user control and consent as the key concept for identity management.

Celesti et al. (2010) also indicate the issues encountered by cloud services in authentication. They explain a scenario where cloud services manage their identity and security management using their own internal mechanism to authenticate users.

Eap et al. (2007) propose a ‘Personal’ Identity Management (PIM) framework which is a Service-Oriented Architecture framework. This proposed PIM gives users full control over the management of their identity data and reduces the complex management of trust and privacy.

When implemented in the cloud environment, Faraji et al. (2014) suggest authentication as the privilege to a user to the cloud resources and Spencer (2012) defines authorization as the amount of access to cloud resources given to a user. Faraji et al. (2014) also suggest the need for separating the authentication logic from user credential so as to support any authentication mechanism. The different definitions and views suggested by above authors clearly indicate that single Sign-On and federated identities can help manage users more efficiently in the cloud. This in turn effectively enhance cloud interoperability.

2.5 Membership

‘Membership’ is the proprietary identity management system of the data center which uses PostgreSQL as the backend to store user details. It uses Single Sign-On to authenticate users for different services within the data center’s proposed cloud. ‘Membership’ is also used as a basis for Software as a Service (SaaS) applications developed
for customers within the cloud. It is capable of implementing true multi-tenanted SaaS applications, as distinct from multi-instance SaaS applications. The data center intends to develop a cloud to offer IaaS services from within the same cloud so that customer can leverage the benefits of IaaS services along with all the SaaS capabilities. Though the challenge faced was that most of the cloud platforms like OpenStack have their own identity management service. These cloud Identity Management System manages their own database for users, authentication, and authorization details.

OpenStack also has its own management portal to manage different services provided by it. All the database members need to be recreated within OpenStack and need to use a different interface to manage IaaS services. In this research at the data center, we tried to integrate OpenStack cloud internal identity management with ‘Membership’. To authorize users registered with them without being recreated by in OpenStack. The purpose was to maintain a single database that will be maintained by ‘Membership’. All the SaaS and IaaS services will be authenticated based on the user privileges defined in ‘Membership’. Though this integration was possible at service level as well at the database level, it was achieved considering the benefits at the database level.

The integration at service level means that system would remain separate, some data duplication occurs and system can be slower (in theory). Integration on the database level would mean that ‘Membership’ and OpenStack Keystone databases are merged. This could benefit in the speed and data integrity. However, any upgrades will need to be executed with extreme caution to not cause any problems in the ‘Membership’ database. This paper further presents the different approaches used to modify ‘Membership’ and OpenStack keystone to achieve this. Also, the OpenStack management portal called dashboard is merged with Data center’s login page to manage all the SaaS and IaaS services from one place.

In version 1 of ‘Membership’, a user is referred as a ‘Person’ which can belong to a single address at a time. Although working, this approach creates problems when the user switches the address, it affects not only what that particular user sees but how system sees him as well.

With V2 iteration, ‘Membership’ will extend Keystone and will merge the ‘Member’ with the ‘Domain’ and ‘Person’ with the ‘User’.

The new important change will be the approach to how ‘Person’ belongs to an address. The current address should only affect what the user sees, not how the system interacts towards this user. With suggested structure change, we should be able to separate concerns of system and user view.
This solution is based on how users in keystone are assigned to the projects. A person is assigned a role in an address and he is able to interact with that address for as long as the assignment is active. Assignments are never deleted physically, once created they stay in the system forever so that old data structure can remain intact.

The person will select his/her default address and will work within that address but will be able to switch it at any time. This said the switch will affect only the person changing the address (it will change the context of what person sees), it will not affect the system.

2.6 Identity Federation

Identity federation is an arrangement where user’s electronic identity and attributes stored across multiple distinct systems are linked together. It allows a user to use same identification data across multiple applications. Our research is primarily focused on building such a cloud which allows us to federate user identity based on our IDM system ‘Membership’. This should also allow all the applications to be accessible over a Single Sign-On using OAuth.

Single Sign-On can be considered as a subset of identity federation in which user is granted an authentication token that is trusted across multiple systems. In the case of this cloud, it should be trusted across all the SaaS and IaaS offerings. The identity federation for us is a two-step process where authentication is done using OAuth and authorization is via ‘Membership’.

Identity federation has been explained in detail by Maler and Reed (2008). They define federated identity management as a set of technology and process that allows computer systems to dynamically distribute identity information and delegate the identity task across multiple domains. Maler and Reed (2008) further add that identity federation is the mean by which web applications can allow users to Single Sign-On across multiple domains. They suggest that it allows users to authenticate once and thereafter gain access to multiple resources and web applications elsewhere. Where this system provides ease of user management, this also has its own risks. Maler and Reed (2008) correctly point out that an identity federation not only increases the cost but also entails new increased privacy and security risks. They highlight that valuable information is shared across domains using lightly coupled network protocols and hence put them at risk.

Similarly, Dreo et al. (2013) have also raised same concerns but considering domains as different clouds and identity federation between intercloud. They raise concerns over the amount of user information that should be exchanged across domains and using
which protocol and in which format. Dreo et al. (2013) point that a user may have different access rights across different domains. Hence, there may be need of extending traditional and conventional authorization models, such as role-based access control or mapping of permission between domains.

A similar concept has been used for CloudCIX Single Sign-On where authentication is done using OAuth and authorization is done based on permission and access mapped in ‘Membership’. A good comparison between ‘Membership’ and Identity Management System developed by others may give an idea of how authorization in our work is different and adding value to the existing concept of IDM.

For example Claußand Köhntopp (2001) developed an Identity Management System that provides a tool for multilateral security and identity management. They focused on controlling the amount of user data transmitted across the domains. More focus was given to allow the user to control its interaction across domains and amount of data it wants to share. Claußand Köhntopp (2001) considered pseudonymity as the core mechanism of their identity manager and the identifier for an object or user. These pseudonyms may have few characteristics linked by the means of digital signature to disclose partial information about the user. The security management interface for their Identity Management System provides users the possibility to configure their security goals and preferred security mechanism. Their IDM system is more focused towards allowing users to manage their identity and attributes while communicating with the different application. Though this type of Identity Management System would not be successful in the case of multi-tenant cloud environment like CloudCIX. The reason being access and roles are defined based on tenants and services they are availing. The access and roles are inherited based on the attributes of tenant and users being a member of it. The attributes that will be associated with the user will be controlled by the administrator of the tenant and not individual users. Once declared by administrator, user will be granted an attribute and on authentication, he will be issued a token that will allow him access to resources without sharing much information of the user as the application will agree to the token. This added an extra level of security and identity management where designated members of the tenant can act as an admin and control user access to the resources.

The next section discusses in detail OpenStack, identity service keystone, keystone federation and related work.
2.7 OpenStack Identity Federation

OpenStack is an open source software that provides IaaS layer for building private as well as public clouds. The clouds build on OpenStack offer IaaS platform. It consists of different modules offering individual services. Keystone is an OpenStack module that provides identity management services, including authorization and authentication. OpenStack currently is seeing a huge growth in popularity and have widely been adopted by corporations, small and medium businesses, service providers and value-added resellers. As OpenStack is getting tremendously popular and more enterprises begin to deploy it and is becoming very important for these enterprises to integrate OpenStack Keystone with their existing federated Identity Management System. Identity federation was not supported in OpenStack until 9th release, Icehouse. OpenStack Icehouse introduced new federation capabilities to keystone which enabled keystone to integrate with popular identity management products like IBM Tivoli identity manager and RSA access manager.

In an OpenStack deployment which works with non-federated authorization model there are users, groups, and projects. A user once authenticated against a project is granted an access token. This access token contains the attributes that specify his role on that project. Any action performed by a user in OpenStack is verified against his token. The role associated with the token helps determine if user is allowed for that action or not. As opposed to it in a federated environment, the user information is no longer stored in keystone but is managed by the identity provider. This creates a challenge for associating OpenStack roles to the users as roles are defined by keystone, but the users are not managed by keystone. As a solution to this keystone provides group support that allow leveraging the attributes that are returned from the identity provider for a user. It allows leveraging through a rule mapping that maps a user to a keystone group based on the associated attributes.

This was an area of research for this data center as the identity provider ‘Membership’ needs to be built with roles and those roles needs to be replicated in keystone. Also, the access policies need to be defined for these custom roles. Similar work with OpenStack federation has been tried by few other enterprises and independent researcher though it still remains an area of research. As there is not much research done on it, there is no best tried and tested formula for OpenStack federation and it is still being researched with individual experiments.

David Chadwick from the university of Kent worked extensively on OpenStack federation. He points out that though OpenStack federation makes the user management easier, it has its own challenges as well. Chadwick et al. (2014) points IDP discovery
problem as one of the most challenging factors. It makes it difficult for OpenStack to decide which IDP user wants to use. There are different IDP providers offering a different set of option making it confusing for OpenStack. For example, Nascar solution displays a set of IDP icons to the user. Similarly, shibboleth lets user pick/search for his IDP. He also points out that it is difficult to decide which protocol (OpenID, SAML, OAuth) will work best. Chadwick et al. (2014) worked extensively with OpenStack keystone federation.

Chadwick et al. (2014) present a protocol independent federated identity management to the OpenStack services. They point out that though most of the cloud deployments are stand alone, there still is a need for community clouds and inter-clouds. Further, they suggest that a protocol independent federation will enable authentication and authorization to be flexibly enforced across federated environments. Chadwick et al. (2014) in support of their statement presents a detailed federated identity protocol sequence and implements protocol independent system components. Finally, they also incorporate two other identity federation protocol i.e., SAML and OpenStack default keystone. They further compare the performance against protocol independent federation. In support of the need for protocol independent identity federation, they quote to the Mell and Grance (2010) US Government Cloud Computing Technology Roadmap guidance. It suggests using Frameworks to support federated community cloud. This guidance is to enable all manner of international, government-to-government, agency-to-agency, or business-to-business collaborations.

Similar to these authors, Martinelli and Topol (2014) from IBM also worked on OpenStack keystone federation. However, Martinelli and Topol (2014) restricted their work mainly to SAML-based federation using IBM proprietary Identity Management System called ‘Tivoli Federated Identity Manager’. As Keystone federation support is based on few libraries provided by Apache HTTPD, they implemented keystone within apache and enabled keystone extensions to support federation and SAML-based authentication. Further using the Tivoli Federated Identity Manager admin console they created a SAML 2.0 Identity Provider federation. They used all the default options, including the default ip_saml20.xsl mapping rule. Also, they disabled all the artifact profiles and used only the browser-post profile for this integration. Further, they exported the SAML 2.0 metadata for the Tivoli Federated Identity Manager and used it with apache module ‘Shibboleth’ to communicate with Tivoli. To complete the setup, they added Keystone as a SAML 2.0 Partner to the Tivoli Federated Identity.

Similarly, Chadwick et al. (2014) also configured the federated environment for OpenStack, however, they used the concept of Virtual Organization (VO) to design their
Identity Management System. Chadwick et al. (2014) explain that Virtual Organization is a security context in which each member of VO is linked with a set of authorization attributes (roles). These attributes can generally be managed from multiple administrative domains. The administrative domains use an external VO Membership Service (VOMS) which maintains all the information for a set of VOs. Each VO maintains its own administrator who can define all the groups, roles in a VO and can grant or revoke any membership in that VO. Chadwick et al. (2014) further add that when a user authenticates to VOMS for any VO, they receive SAML assertion defining the user’s authorizations. The user’s client uses this to build an X.509 proxy certificate. This proxy certificate is based on the user’s primary certificate, and this is used for authorization for the protected service.

Chadwick et al. (2014) used this concept implicitly in their design though they used an attribute mapping server to eliminate the need for a separate VOMS server. They used this attribute mapping server to solve the problem of semantic interoperability of mapping user identity into keystone’s model of tenant, project and roles. This mapping allows in making proper authorization decision which was missing from Kerberos and shibboleth.

They developed a protocol independent module responsible for both authentication and attribute request messages and also getting them to the Identity Provider. These messages are protocol dependent. However, converts it to standard format while returning the user identity information to the keystone. Keystone does not, therefore, need to process it and act as a relay to pass them across protocol specific modules. They focused on making the module replaceable and making multiple modules supported simultaneously. This is to allow keystone be able to support different identity providers. This helped them keep the keystone intact without any change in other OpenStack services and how they interact with keystone.

However Chadwick et al. (2014) point to the challenge that vast majority of the IDPs provide users browser-based logins for authentication, whereas the OpenStack clients for different services are the command line. They investigated the feasibility of command line client interacting with identity providers via HTTP. However, as they found it to be very complex, they resorted to the web browser for the authentication phase of the federated identity management implementation. They used SAML v2.0 for this implementation as their request issuing function supports the SAML Web Browser SSO profile.

Different to Martinelli and Topol (2014) and Chadwick et al. (2014) where both worked with SAML and using existing approaches, we tried Single Sign-On using OAuth for
authentication and an IDM system built in-house for authorization. The Table 2.1 below compares CloudCIX OpenStack federation with other similar works.

<table>
<thead>
<tr>
<th>Researcher</th>
<th>Framework</th>
<th>Identity Management</th>
<th>Integration Level</th>
<th>Limitations</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM</td>
<td>SAML</td>
<td>Tivoli</td>
<td>Service Level</td>
<td>Browser Based</td>
</tr>
<tr>
<td>Chadwick et al.</td>
<td>SAML</td>
<td>VOMS</td>
<td>Service Level</td>
<td>Browser Based</td>
</tr>
<tr>
<td>CloudCIX</td>
<td>OAuth</td>
<td>Membership</td>
<td>Database Level</td>
<td>CLI and Browser</td>
</tr>
</tbody>
</table>

Table 2.1: OpenStack Federation Comparison
Chapter 3

Design

3.1 Specification

This section explains all the strategies, planning and design concepts adopted to implement Single Sign-On using OAuth and Keystone federation. The OpenStack keystone federation was carried out in phases. The first phase is the development of OAuth middleware for SSO, integration of keystone with ‘Membership’, and customization of dashboard for the user login, account, and Resource management. The major challenge was to take the existing ‘Membership’ application and make it capable of supporting the Keystone API. The figure 3.1 below shows how it is proposed that the components in this stack will be integrated together.
The stack shown above is to include Infrastructure as a Service (IaaS). The IaaS Services are based on OpenStack plus extensions such as billing, DNS etc and a SaaS platform that exposes RESTful Web Services that can be combined to build business applications. Such Web Services would support CRM, SCM, Financial and data/IOT processes. With the introduction of Keystone API version 3, it is now possible to create a database schema that is a superset of both the Keystone and ‘Membership’ API. This suggests that with the recent release of the Kilo version of OpenStack this project became more feasible.

### 3.2 Design Overview

As the research presents using Single Sign-On for CloudCIX, given below is the process and data flow for the design. The figure 3.2 shows the overall architecture of the federated OpenStack. This also depicts how ‘Membership’ has been positioned in the stack.
The dashboard shown above is developed using Django framework with python and OAuth. The credentials supplied through the dashboard is validated by ‘Membership’. Post validation, based on authorization defined, they are assigned a token which is passed to Keystone. Keystone analyzes the token for associated attributes and allows access to other OpenStack resources accordingly.

3.3 Abstract Protocol Flow

Explained below is the process flow for authentication and obtaining access token using OAuth. Given below is the abstract protocol flow for the one of the four OAuth roles used in CloudCIX and explained in above sections. The figure 3.3 below depicts this flow and is explained in detail below.
The abstract protocol flow for OAuth shown in the figure 3.3 above is as explained below:

1. The application requests the authorization to access service resources from the user.

2. On user authorizing the request, an authorization grant is provided to the application.

3. An access token is requested by an application from the authorization server by presenting authentication of its own identity and the authorization grant.

4. On application identity being authenticated and authorization grant being valid, an access token is granted to the application by the authorization server (API). This completes the authorization process.

5. The application presents the access token for authentication and requests the resources from the resource server.

6. Resource server serves the resources, on access token being valid.

3.4 Application Registration

Before using OAuth, The CloudCIX must register the applications with the service. This is done through the API part of the service website where following information
is provided.

- Application name
- Application Website
- Redirect URL

Once the user is authorized or denied the application, they will be redirected to Redirect URL that will manage the authorization code or access token.

### 3.5 Workflow

The authentication process is based on OAuth 2.0 standard and implements section 4.1 and 6 of the RFC6749. After receiving the access token, the client shall validate it against Keystone. It will either redirect to SSO login (if token becomes invalid during the process) or create a local browser session using data received from Keystone response. The above explained Authorization workflow is shown below in the figure 3.4.

![Figure 3.4: Authorization Code Flow](image)

In addition to the Authorization Grant type, as shown above in the figure 3.4, there is also an Implicit Grant type used. The implicit grant type is used for those mobile apps and web applications where the client secret confidentiality is not guaranteed. This grant type also has a redirection-based flow. Though, in this case, the token is granted
to the user-agent to forward to the application. As a result, the token may be exposed to the user and other applications which are running on the device. This authorization flow will not validate the identity of the application but rely on redirect URI to serve the purpose. Also, the implicit grant type does not support refresh token. A detailed flow for the implicit grant type is as shown below in the figure 3.5.

![Figure 3.5: Implicit Code Flow](image)

**3.6 Openstack Federation Design**

As the above section discussed in detail the design for SSO, given below is the design for CloudCIX keystone and different components interacting with each other. A general overview of keystone architecture is as shown below in figure 3.6.
The OpenStack Identity service along with ‘Membership’ performs the following task.

1. Track users and their permissions.

2. Provide a catalogue of available services with their access API endpoints.

While installing each service of OpenStack, they are registered with identity service. This is for keystone to keep a track of available services on the network. OpenStack keystone confirms an incoming request by validating the credentials supplied by the user. Once credentials are validated, keystone issues an authentication token which allows users to access different services. As in the case of CloudCIX OpenStack, the user is validated from the identity service ‘Membership’. On successful validation, they get the access token which allows unrestricted access to all the resources on CloudCIX including OpenStack. A small overview of how different components are distributed in the OpenStack ecosystem can be shown as given in the figure 3.7 below:
Figure 3.7: Keystone Components

A flow chart to depict Customized OpenStack keystone interaction with other services can is shown below in the figure 3.8. It shows how every user and service is authenticated against Membership and keystone before getting access to different OpenStack resources.

Figure 3.8: OpenStack Ecosystem
3.7 Additional Functionalities

3.7.1 Problem Description

With the introduction of Keystone tokens, it is no longer possible to extend token validity. Instead, multiple tokens should be used to maintain the user session. By default, the tokens are issued as a unscoped token. For some functionality (vault/swift), we require project scoped tokens. In this case, SSO should act as the center point of communication with keystone (except for token validation). If a frontend requires a new token (scoped or just refreshed) it should fire a request to SSO. It will revoke the previous token and force other frontends to refresh their tokens with the new instance.

This functionality will be implemented as per the specifications in section 6 of the RFC6749.

3.7.2 Required Functionalities

1. Ability to refresh an expiring token or existing token.

2. Ability to request a scoped token.

3.7.3 Impact on Clients

Other frontends should be able to refresh tokens in the background. Furthermore, if the request is a POST, frontends should be able to refresh token without affecting the action. If a user POSTed some data and is required to log in, we should have some ability to ‘record’ his action and ‘reply’ it after he logged in.

3.7.4 Client Implementation

If a client validates the token with every request and if keystone returns code 401 upon token validation, the client should redirect to SSO login page. Secondly, If a client caches the token, it should wrap around the API call and expect a code 401 response from the service. If the service returns code 401, the client should check with SSO for a new token and reply the API request if SSO returns a new valid token.
3.7.5  Client Workflow

The above explained workflow can be represented diagrammatically as shown below:

Reacting to Token Refresh

1. apps.cloudcix ---- change user ----> services
2. apps.cloudcix <-------- OK --------- services
3. apps.cloudcix --- refresh token ---> SSO
4. apps.cloudcix <----- new token ----- SSO
5. apps.cloudcix ----- more work ------> services
6. iaas.cloudcix ----- more work ------> services
7. iaas.cloudcix <------ NOES! 401 ----- services
8. iaas.cloudcix --- redirect to login ---> SSO
9. iaas.cloudcix <----- user already signed in, code ---- SSO
10. iaas.cloudcix ------ retrieve token ----> SSO
11. iaas.cloudcix <----- token ---- SSO
12. iaas.cloudcix ----- more work -----> services

Reacting to a no longer valid token

1. iaas.cloudcix ----- more work -----> services
2. iaas.cloudcix <------ NOES! 401 ----- services
3. iaas.cloudcix --- redirect to login ---> SSO

As this section described in detail the architecture, process flow, and design for authentication and authorization within CloudCIX, Next section discusses in details the implementation of OAuth and OpenStack installation and keystone integration with Membership.
Chapter 4

Implementation

This part of the thesis discusses in detail, the steps carried out to install, configure and test different component of CloudCIX. This includes installation and configuration of OpenStack, OAuth and keystone integration with ‘Membership’. This section explains in detail the customization of Keystone and different changes made to its configuration to interact with ‘Membership’.

The CloudCIX consists of different components serving different purpose in the network. A brief network design for these components along with some other hardware supporting CloudCIX are configured on the network as shown in the figure 4.1 below:
Discussed below is the setup and configuration of different components for CloudCIX to offer SSO and other cloud services.

### 4.1 SSO Installation and Configuration

A single page is implemented at a specific URL `https://auth.cloudcix.com`. All frontends created by CIX should use that URL as an authority provider.

1. `https://auth.cloudcix.com` will be able to derive from which URL the authentication request came from. ([https://apps.cloudcix.com](https://apps.cloudcix.com), [https://apps2.cloudcix.com](https://apps2.cloudcix.com)) will be able to redirect back to that site.

2. `https://auth.cloudcix.com` will only allow specified URLs to contact with it and will only redirect to URLs on that list, any unsupported clients will receive an error response as per OAuth 2.0 specifications.

3. `https://auth.cloudcix.com` will act as a provider of authentication for all frontends and will be able to create, refresh and scope keystone tokens on behalf of other frontends.
4. Other frontends will only connect to keystone to validate tokens. Other frontends will direct all CUD token operations to SSO.

NOTE: State returned in the response from SSO is the same as state specified by the client in the request. Both should be compared by the client upon retrieval to ensure that the response is an actual response to send the request.

NOTE: The code will be a one-time use valid for 10 minutes after the redirect URL has been created. After receiving the code, Client \texttt{https://apps.cloudcix.com} shall use the code to request the Access Token and the Refresh Token from the SSO Provider using the request shown in listing 4.1 over TLS:

```
POST /token HTTP/1.1 Host: apps.cloudcix.com Authorization: Basic3465y3y3tg3346hrtgbwvf45gursfv Content-Type: application/x-www-form-urlencoded
grant_type=rant_type=authorization_code&code=3425g4weg54getg3 &redirect_uri=https%3A%2F%2Fapps%2Ecloudcix%2Ecom%2Fcb
```

Listing 4.1: Access Token Request

This request will result in a response containing the Access Token (used for requests to services) and the Refresh Token (used to request a new Access Token before its expiry or to request a Scoped Token). The code for same is shown in the listing 4.2.

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache
{
  "access_token":"2YotnFZFEjr1zCsicMWpAA4tg35trbws",
  "token_type":"Bearer",
  "refresh_token":"tGzv3JOkF0XG5QxZT1KWIA"
}
```

Listing 4.2: Access Token Response

### 4.1.1 Refreshing a Token

The token is about to expire, is expired or a scoped token is required, client shall send the following request mentioned in listing 4.3 to the SSO Provider over TLS:

```
POST /token HTTP/1.1 Host: apps.cloudcix.com Authorization:
Basic 3465y3y3tg3346hrtgbwvf45gursfv Content-Type: application/x-www-form-urlencoded
```
Listing 4.3: Refresh Token Request

This request will result in a response containing the access token (used for requests to services), refresh Token will be omitted, example shown in listing 4.4.

```
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache
{
  "access_token":"2YotnFZFEjr1zCsicMWpAA4tg35trbws",
  "token_type":"Bearer",
}
```

Listing 4.4: Refresh Token Response

After receiving the access_token, client shall validate it against Keystone and either redirect to SSO login (if token became invalid during the process) or update the local browser session using data received from Keystone response.

### 4.1.2 SSO Login and Token Grant

The OpenStack login page called Dashboard has been integrated with data center login portal for SSO and this single page will be used for all logins. SSO login page is as shown below in the figure 4.2.
Figure 4.2: CloudCIX Login

This has been deployed on Apache2. Apache uses mod_proxy to create a pass from one server to the second server for the sake of downloading files (Background jobs are executed on the second server and that’s where the resulting files are located).
The Application framework is connected to messaging queue rabbitmq on the worker server. This means that every celery task started by the services is executed by the worker server. Apache config was adapted to run with 8 processes and 1 thread. Application framework accesses keystone on the admin port (35357) to distribute resource usage a bit easier on the keystone server. Once there are enough resources it can be switched to pub (5000) port. Application framework settings.local.py allow for 50 requests per second which should be tuned down. Clients should be informed that they should implement a back-off procedure in case of HTTP 429 - TOO MANY REQUESTS error codes.

A simple implementation of login using the form above is as depicted below in the codes 4.5

```python
class AuthenticationForm(forms.Form):
    
    Basic authentication form for users, it checks credentials and if correct returns instance of CIX User that is stored in local Session.
    
    username = forms.CharField(label=_("Email address"), max_length=50,
        widget=forms.TextInput(attrs={
            'class': 'form-control',
            'required': 'required',
            'placeholder': "Email address",
            'autofocus': 'autofocus'})
    
    password = forms.CharField(label=_("Password"),
        widget=forms.PasswordInput(attrs={
            'class': 'form-control',
            'required': 'required',
            'placeholder': "Password")
    
    error_messages = {
        'invalid_login': _("Please enter a correct username and password. "
            "Note that both fields are case-sensitive.")},
        'no_cookies': _("Your Web browser doesn’t appear to have cookies "
            "enabled. Cookies are required for logging in.")
    }

    def __init__(self, request=None, *args, **kwargs):
        
Listing 4.5: CloudCIX Login Form
4.1.3 Auth Server

SSO Auth

https://auth.cloudcix.com auth server is the single point of authority for all CloudCIX frontends. It is responsible for providing Single Sing-On functionality and issuing any (scoped and unscoped tokens). Scoping is also done via interaction with the Single Sign-On server.

IMPORTANT NOTES

1. ALLOWED_LOGIN_REDIRECT_HOSTS should be a list of domains that SSO should be able to redirect to. After successful login, the user will be redirected to those domains so that a call to retrieve a token can be made.

2. CLOUDCIX_SSO_CLIENTS should contain pairs of client_id: client_secret. Only frontends that use a pair listed in this dictionary will be allowed to use SSO functionality. Currently, only 2 pairs are used. This should be extended to allow better security auditing: - Each frontend should have its own client_id client_secret pair - Each SSO usage and token retrieval should be logged along with client_id and IP address

3. In the case when the SSO is distributed on many machines, each deployment should use the same SECRET_KEY, DATABASE and CACHE instances should be shared.

4. Frontends should use PickelSerializer as SESSION_SERIALIZER until the OpenStack auth is able to serialize the Token instance.

A simple backend authentication is done by calling different modules and can is shown below in the listing 4.6
Listing 4.6: Backend Authentication

A successful sign-on gives access to all the CloudCIX applications from where they can be installed to appear on the dashboard. An installed app will appear on the top of the screen as shown below in the figure 4.3.

Figure 4.3: CloudCIX Dashboard
Further clicking on ‘Compute’ app redirects to the customized dashboard for OpenStack. It allows the authenticated user to access the resources based on the attributes in the token. The customized dashboard for CloudCIX is as shown below in the figure 4.4

![CloudCIX Compute Dashboard](image)

**Figure 4.4: CloudCIX Compute**

## 4.2 Keystone Installation & integration with Membership

OpenStack keystone has been integrated with ‘Membership’ with the help of python libraries which has been developed as a plugin for keystone. While this document describes the installation process step by step, it is advised to upload cloudcix_keystone to a /opt/ folder on a destination server and execute install script for keystone.

### 4.2.1 Keystone Installation

1. Install Keystone on the machine. The preferred way is to download the correct release and install via python setup.py. A detailed installation instruction is shown in listing 4.7
Listing 4.7: Keystone Installation
2. Remove any apache sites enabled by default and enable the newly copied key-
stone.conf NOTE: keystone.conf is set to use mod-SSL (cloudCIX certs are re-
quired) and run as administrator user

NOTE: If the root user can’t clone the repo from the https://git.cix.ie it is
better to create a symbolic link in from /opt/cloudcix_keystone to just-packages
as it makes updates easier.

4.2.2 keystone.conf configuration to use CloudCIX-Keystone

If the configs were copied from CloudCIX-keystone, the keystone is already pre-
configured to use CloudCIX Stage environment. Notable differences between original
keystone.conf and CloudCix-keystone keystone.conf are denoted below in listing 4.8.
NOTE: If keystone is supposed to run as a production system, the only option that
needs to be changed is [database]/connection to point to the live database.

```
[database]
connection = postgresql+psycopg2://postgres:xxxxxxxx@x.x.x.x/Membership

[identity]
driver = cloudcix_keystone.backends.identity.Identity

[token]
provider = cloudcix_keystone.backends.token.ExtendedUUIDProvider
driver = cloudcix_keystone.backends.token.Token

[resource]
driver = cloudcix_keystone.backends.resource.Resource

[auth]
methods = external,password,token,cloudcix_auth
token = cloudcix_keystone.backends.auth.token.Token
external = cloudcix_keystone.backends.auth.domain.Domain

#The cloudcix_auth plugin module#
cloudcix_auth=cloudcix_keystone.backends.auth.cloudcix.CloudCIXAuth

[trust]
driver = cloudcix_keystone.backends.trust.Trust
```

Listing 4.8: keystone.conf
4.2.3 Test Run

Keystone can be started without apache and ensured it is in working state by starting it from command line and requesting a token as shown in code 4.8.

```
%item From same Console%
#keystone-all -d

%From other console window %
curl -i -H "Content-Type: application/json" -d '{"auth":{"identity":{"methods": ["cloudcix_auth"], "cloudcix_auth": {"username": "you@cloudcix.com", "password": "your_password"}}}" http://keystone:5000/v3/auth/tokens
```

Listing 4.9: Test Run

On successful completion of the test, the user can log in to CloudCIX and can access Compute App to launch an instance. A successful authentication and validation will allow him to access his project and instances as shown below.

![Figure 4.5: CloudCIX Instances](image)

Next section explains the evaluation for CloudCIX to assess the performance. A detailed load test is run to test the load on different nodes including keystone to validate the number of requests it can process.
Chapter 5

Evaluation

This section of the thesis discusses in detail the environment used for implementing and testing the CloudCIX for the proposed SSO solution. Different use cases have been defined and CloudCIX is evaluated against them. Also, few testing tools are used to test the number of requests processed by the customized keystone implemented in this solution. Given below is the detail for test and evaluation of CloudCIX.

5.1 Requirements and Setup

This section discusses in detail setting up the pre-requisites for the CloudCIX Compute offerings. Table 5.1 shows the hardware resources required for installation and configuration of CloudCIX.

<table>
<thead>
<tr>
<th>Service</th>
<th>Number of Nodes</th>
<th>Hardware</th>
<th>vCPU</th>
<th>RAM(GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Controller</td>
<td>1</td>
<td>Dell Poweredge R710</td>
<td>24</td>
<td>24</td>
</tr>
<tr>
<td>Compute</td>
<td>4</td>
<td>Dell Poweredge R710</td>
<td>96</td>
<td>200</td>
</tr>
<tr>
<td>Network</td>
<td>1</td>
<td>Dell Poweredge R710</td>
<td>24</td>
<td>24</td>
</tr>
<tr>
<td>Keystone</td>
<td>2</td>
<td>Dell Poweredge R710</td>
<td>24</td>
<td>24</td>
</tr>
</tbody>
</table>

Table 5.1: Installation Requirements
5.2 Use Cases

5.2.1 Use–Case Basic Model

Actors

Tenant: Tenants are the companies registered in ‘Membership’. Tenant is a collection of persons who are linked to an address for that tenant. The cloud access rights are granted on tenants and are inherited by the users with additional attributes.

Users: Users are the persons listed under any particular tenant. They are the actual cloud users. An attribute associated with the tenant decides if a person can use the cloud resources or not. Further, the additional attributes for the user decide its role.

Admin Users: The Tenant admin manages the users within a tenant and defines their access level.

Tokens: Tokens are the attribute bearers which define the role and access rights for the users who have been granted the token.

Additional Information

The users and the tokens only, seen in the use case are considered essential to the system developed. Of the three essential use cases, User authenticated for one of the Frontends, Browser session expired-Token Valid, Browser session valid-Token Invalid, the use case considered highest priority in the system is User authenticated for one of the Frontends and has been focused on. The figures shown in section 5.2.3 shows the currently implemented use case for illustrative purpose.

System Under Design

The system under design is a multi-tenant cloud, offering Single Sign-On. The actors described above represent the system and actions it takes.

Cloud Administrator

A cloud administrator is a user who administers the whole system by controlling the overall aspects of the cloud. Few of the tasks include registering new tenants, granting them access rights, allocation of resources etc.
5.2.2 List of Use Cases

CloudCIX User Use Cases

1. User authenticated for one of the Frontends
2. Browser session expired-Token Valid
3. Browser session Valid-Token Invalid

5.2.3 Use Case Diagrams

Use Case: “User authenticated for one of the Frontends”

Figure 5.1: User authenticated for one of the frontend
Use Cases: “Browser session expired-Token Valid”

Figure 5.2: Browser session expired-Token Valid
Use Case: “Browser Session valid-Token Invalid”

Figure 5.3: Browser Session valid-Token Invalid
### 5.2.4 Use Cases Details

**Use Case:** User authenticated for one of the Frontends

<table>
<thead>
<tr>
<th>Use Case Name: User authenticated for one of the Frontends</th>
<th>Use Case ID: UAF</th>
<th>Priority: High</th>
</tr>
</thead>
</table>

**Primary Actor:** CloudCIX User  
**Source:** auth.cloudcix.com  
**Use Case Type:** Evaluation  
**Level:** Overview

**Interested Stake Holders:**  
CloudCIX Users, Tenant admin, Cloud Administrator

**Brief Description:**  
This use case describes the authentication of a cloud user which is the key function of the system. In this use case, the actor's goal is to obtain an authentication token.

**Goal:**  
The successful authentication of a cloud user.

**Success Measurement:**  
The user is authenticated and granted access to cloud resources.

**Precondition:**  
i. User is registered under a tenant in Membership.  
ii. Cloud access is enabled for the tenant.  
iii. User has a role defined for that tenant.

**Trigger:**  
CloudCIX user has reached a point in their workflow where he has passed his credentials to be validated against membership.

**Typical Flow of Events:**
1. User goes to the login page @ https://auth.cloudcix.com.  
2. He logs in and is redirected to https://apps.cloudcix.com  
3. He tries to access the dashboard called Horizon which is hosted on a different server @ https://apps2.cloudcix.com  
4. Browser session for apps2.cloudcix.com does not exist.  
5. User is redirected to the auth.cloudcix.com.  
6. Since user already logged in @ https://auth.cloudcix.com, code auth automatically.  
7. https://apps.cloudcix.com receives the code, uses it, to retrieve the token and create a browser session for user behind the token.

**Assumption:**

1. It is assumed that user is already registered in Membership and providing correct credentials.

---

**Table 5.2: Use Case:1**

---

47
**Use Case: Browser session expired - Token Valid**

<table>
<thead>
<tr>
<th>Use Case Name:</th>
<th>ID:</th>
<th>Priority:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Browser session expired - Token Valid</td>
<td>SE-VT</td>
<td>High</td>
</tr>
</tbody>
</table>

**Primary Actor:**
CloudCIX User

**Source:**
auth.cloudcix.com

**Use Case type:**
Evaluation

**Level:**
Overview

**Interested Stake Holders:**
CloudCIX Users, Tenant admin, Cloud Administrator

**Brief Description:**
This use case describes the redirection of user to login page in case browser session expires.

**Goal:**
To allow users acquire same token which is still valid from last browsing session.

**Success Measurement:**
User is redirected to login page when Browser Session expires.

**Precondition:**

i. User has already authenticated.

ii. User already has an authentication token and a browsing session.

**Trigger:**
CloudCIX user has reached a point in their workflow where browser session expire passes his credentials to be validated against membership.

**Typical Flow of events:**
2. https://auth.cloudcix.com finds a browser session for the user and validates the token.
3. https://auth.cloudcix.com responds with a code that should be used to retrieve the token.
4. https://apps.cloudcix.com receives the code and retrieves the token behind it.

**Assumption:**

i. It is assumed that cloud user had already authenticated itself before.

ii. It is assumed that the browser session expires before the token.

Table 5.3: Use Case:2

48
Use Case: Browser Session valid-Token Invalid

<table>
<thead>
<tr>
<th>Use Case Name:</th>
<th>ID:</th>
<th>Priority:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Browser Session valid-Token Invalid</td>
<td>BV-TI</td>
<td>High</td>
</tr>
</tbody>
</table>

Primary Actor:
CloudCIX User

Source:
auth.cloudcix.com

Use Case type: Evaluation
Level: Overview

Interested Stake Holders:
CloudCIX Users, Tenant admin, Cloud Administrator

Brief Description:
This use case describes the need for reauthentication when token expires even though browser session is valid.

Goal:
To allow users acquire a new token even when browser session is valid.

Success Measurement:
User is redirected to login page when token life time expires

Precondition:
i. User has already authenticated.
ii. User already has an authentication token and a browsing session.

Trigger:
CloudCIX user has reached a point in their workflow where browser session is still valid though lifetime of token has expired.

Typical Flow of events:
1. Calls to the service fail with the invalid/expired token.
2. https://apps.cloudcix.com recognizes the error and redirects the user to https://auth.cloudcix.com
3. https://auth.cloudcix.com nds a browser session for user viewing the page and retrieves the user token.
4. After token validation if the token is invalid, the user is displayed with the login box.
5. After user logs in he is redirected back to https://apps.cloudcix.com with a code for token retrieval.

Assumption:
i) It is assumed that cloud user already has an authentication token.
ii) It is assumed that the user already has a valid browsing session.

Table 5.4: Use Case:3
Testing Methods

The above ‘Use Cases’ have been tested and results have been verified using the browser logins as well as command line. A token authentication tests have been performed using the command shown below and the results have been analysed. The command below has been used with different user credentials with different access rights. The result has been analysed to come to the conclusion that the single Sign-On is working as intended. A sample output for this test is as shown below:

```
curl -i -H "Content-Type: application/json" -d '{"auth": {"identity": {"methods": ["cloudcix_auth"], "cloudcix_auth": {"username": "jks@cix.ie", "password": "xxxxxxxxx"}}}}', "http://keystone-server:5000/v3/auth/tokens"
```

Listing 5.1: Keystone authentication Testing

5.3 CloudCIX Performance testing with Rally

Rally

Rally is a standard tool for benchmarking OpenStack-based cloud. Rally provides different scenarios with varying workloads for stress testing. It helps improve the performance, reliability and SLA (Service Level Agreement) for the cloud. Rally benchmarks a cloud by automating the deployment and report generation. It has different test cases for benchmarking keystone and other components for its efficiency in processing requests from different services within the cloud. The figure 5.4 shows the architecture of Rally and benchmarking methodologies.
Components

The three major components of Rally are as listed below:

**Deploy Engine**: It is the deployer for Rally. It has a pluggable mechanism which helps it simplify work with different automated installers like DevStack.

**Benchmark Engine**: It contains a big repository of benchmark using which it creates parameterized load of the cloud for testing.

**Verification**: This is still in development and will use tempest to verify the functionality of an OpenStack cloud.

Rally Installation

1. Step 0: Installation

The first step includes downloading the installation packages for Rally and
running the installation from the setup files. This is shown below in the list 5.2

```bash
```

Listing 5.2: Rally Installation

2. Step 1: Setting the environment variables

This step includes registering CloudCIX with Rally. The CloudCIX login credentials are passed through an openrc file which sets the environment variable. The listing 5.3 shows how CloudCIX deployment is registered with Rally.

```bash
export OS_TENANT_NAME=admin
export OS_PROJECT_NAME=admin
export OS_USERNAME=jks@cix.ie
export OS_PASSWORD=XXXXXXXXX
export OS_REGION_NAME=RegionOne
export OS_IDENTITY_API_VERSION=3
export OS_USER_DOMAIN_ID=1
export OS_PROJECT_DOMAIN_ID=1
export OS_DEFAULT_DOMAIN=1
```

Listing 5.3: Environment Variable

With the above environmental variables the CloudCIX deployment was registered within Rally and is ready for benchmarking.

**Benchmark Scenario**

Rally offers multiple scenarios to benchmark the OpenStack Cloud. It has predefined templates and test cases defined in JSON or YAML files. These files can be modified as per the cloud environment and can be used for benchmarking. In this deployment, we are mainly focused on benchmarking the performance of keystone. A simple test is performed on keystone to verify authentication as well as performance.

Test1: Create Users
This test was performed on keystone to identify if it allows an admin to create users in the system. Also, it measures the time taken by keystone to create ‘n’ number of users. The test was performed with a different number of users and concurrency and results were compared. The test criteria are defined in the JSON file and is shown below in the listing 5.4

```json
{
    "KeystoneBasic.create_user": [
        {
            "args": {},
            "runner": {
                "type": "constant",
                "times": 100,
                "concurrency": 10
            }
        }
    ]
}
```

Listing 5.4: Benchmark Scenario

Given below are the results of the test.

**100 Users with Concurrency 10**

<table>
<thead>
<tr>
<th>Action</th>
<th>min</th>
<th>median</th>
<th>90%ile</th>
<th>95%ile</th>
<th>max</th>
<th>avg</th>
<th>success</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>keystone.create_user</td>
<td>0.737</td>
<td>1.389</td>
<td>2.357</td>
<td>2.77</td>
<td>4.07</td>
<td>1.618</td>
<td>100.0%</td>
<td>100</td>
</tr>
<tr>
<td>total</td>
<td>0.737</td>
<td>1.389</td>
<td>2.357</td>
<td>2.77</td>
<td>4.07</td>
<td>1.618</td>
<td>100.0%</td>
<td>100</td>
</tr>
</tbody>
</table>

Load duration: 17.2923059464
Full duration: 28.2163949013

Table 5.5: User Creation Response

The result above suggests that keystone is authenticating admin successfully. Also, it allows admins to create users. A detailed performance report is as shown below in the graphs.
Figure 5.5: Total durations

<table>
<thead>
<tr>
<th>Action</th>
<th>Min (sec)</th>
<th>Median (sec)</th>
<th>90%ile (sec)</th>
<th>95%ile (sec)</th>
<th>Max (sec)</th>
<th>Avg (sec)</th>
<th>Success</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>keystone.create_user</td>
<td>0.737</td>
<td>1.389</td>
<td>2.357</td>
<td>2.77</td>
<td>4.07</td>
<td>1.618</td>
<td>100.0%</td>
<td>100</td>
</tr>
<tr>
<td>total</td>
<td>0.737</td>
<td>1.389</td>
<td>2.357</td>
<td>2.77</td>
<td>4.07</td>
<td>1.618</td>
<td>100.0%</td>
<td>100</td>
</tr>
</tbody>
</table>

Figure 5.6: Parallel Iteration
The evaluation results from above test cases and benchmarks confirm that the Cloud-CIX SSO is working accurately as intended. The test results not only confirm the successful authentication but also the performance of keystone in processing a request.
Chapter 6

Conclusions

Based on the findings from the above research and experiments, it can be suggested that OAuth is the most appropriate Single Sign-On framework that could be used in a multi-tenant environment. This very well allows such an environment where the SaaS and IaaS offerings are accessed by multiple tenants from the same login. With the increase in the popularity of cloud computing, the concern related to the multi-tenancy has also increased. This thesis helped develop a robust Single Sign-On for all cloud offerings controlled by an internal identity management system. This also helped reduce risk and complexity of user and resource management in a multi-tenant environment.

This thesis analysed multiple SSO frameworks and solutions that can be used to mitigate multi-tenancy issues. Based on the findings from this analysis we can conclude that there are multiple solutions and have different use cases depending on the needs and requirements.

SSO solutions like OpenID and SAML have their own advantages and shortcomings though they do not fit in all kinds of requirements. For example as in the case of CloudCIX where SSO solution needs to be integrated at two different levels that are keystone and membership.

In the scope of this dissertation, as the requirement was to mitigate multi-tenancy risks, OAuth has been considered.

The novelty of this dissertation is to implement such a solution which uses existing solutions like OpenStack keystone and ‘Membership’ together with OAuth to provide a robust Single Sign-On Solution the for multi-tenant cloud.

We can conclude that OAuth can be integrated with keystone and ‘Membership’ to reduce the risks of multi-tenancy and manage identities for a cloud that offers IaaS and
Saas Services together.


URL: [http://dx.doi.org/10.1007/s10723-013-9283-2](http://dx.doi.org/10.1007/s10723-013-9283-2)


URL: [http://dx.doi.org/10.1016/S1389-1286(01)00217-1](http://dx.doi.org/10.1016/S1389-1286(01)00217-1)


**URL:** http://www.ibm.com/developerworks/cloud/library/cl-keystone-tfim/


Mitchell (2015), Simple cloud infrastructure for developers.

**URL:** https://www.digitalocean.com/


**URL:** http://oauth.net


**URL:** http://www.sciencedirect.com/science/article/pii/S1561372312700751

