Adding Federated Identity Management to OpenStack's Keystone Client.

Ben Lawrence Miller
blm4@kent.ac.uk

Supervisor:
Professor D. Chadwick

Word Count: 12,776 (approx.)

Abstract

The security of cloud services is being jeopardised by a proliferation of weak username and password combinations. As organisations continue to satisfy their computational requirements with multiple cloud services, users are failing to devise numerous unique username/password combinations. Federated identity management offers an enticing evolution in cloud service authentication; to offer greater interconnectivity and enhanced security. As an established Cloud operating system, OpenStack is developing federated identity capability in Keystone - its core Identity module. Until now, no functioning implementation of federated authentication has been reported for the keystone client. The work discussed in this project presents two implementations of federated authentication using the SAML 2.0 protocol. A standalone Keystone client has been extended with simultaneous compatibility for version 2 and 3 of the Keystone Identity API and a universal client has been adapted for the latest Icehouse release. An overview of the authentication procedures for the Grizzly, Havana and Icehouse releases are presented along with a discussion of current limitations. In doing so, we present a significant step towards functional federated authentication in OpenStack.

Table of Contents

Abstract **2**
1.0 Glossary Of Terms **5**
2.0 Introduction **7**
2.1 Objectives 7
3.0 Background **8**
3.1 What is cloud computing? 8
3.2 A brief history of OpenStack 9
3.3 Authorisation Techniques 10
3.4 Federated Authentication 10
3.5 Single Sign On (SSO) 11
3.6 Kerberos 12
3.7 SAML 13
3.8 Shibboleth 15
3.9 Keystone - OpenStack's Identity module 16
3.10 Fine-grained access control 17
3.11 The Keystone clients 18
3.12 Development Tools 18
4.0 Analysis and Design **20**
4.1 The server-side implementation of Federation 20
4.2 The Identity API v2, and OpenStack Grizzly 20
4.3 The Identity API v3, OpenStack Havana and Icehouse 21
4.3.1 OpenStack Havana 22
4.3.2 OpenStack Icehouse 22
4.4 Analysis of existing implementation - The Swift Client 23
4.5 Analysis of existing implementation - The Keystone Client 23
4.6 Analysis of existing implementation - The OpenStack client 24
4.7 Design of proposed changes 24
5.0 Implementation **25**
5.1 Project Management and development methodology 25
5.2 Changes to the Software Requirements 25
5.3 Changes to the Swift & Keystone clients 25
5.4 Dual v2 & v3 compatibility 27
5.5 Federation and the Icehouse release 29
6.0 Testing and Validation **32**
6.1 Testing with Tox 32
6.2 Validation of the Keystone Client 32
6.3 Validation of the OpenStack Client 32
6.4 Benchmarking the Universal client 36
6.5 Known Bugs 37
6.6 Acceptance criteria. 37

7.0 Conclusion **38**
8.0 Limitations and Future work **39**
9.0 References **40**
Appendix 1a - Software Requirements **43**
Appendix 1b - Software Acceptance Criteria **44**
Appendix 2 - Analysis of the Swift Client and Federated API **45**
Appendix 3 - UML Module diagram of Swift Client **47**
Appendix 4 - Analysis of the Keystone Client **48**
Appendix 5 - A Gantt planning chart for the project **50**
Appendix 6 - Small changes to the Swift client **51**
Appendix 7 - Documentation of the Federated API **53**
Appendix 8 - A Scoped Token **55**
Appendix 9 - Installation guide **56**

1.0 Glossary Of Terms

ABAC	Attribute Based Access Control
AC	Access Control
API	Application Programming Interface
AWS	Amazon Web Services
CI	Configuration Item
DDoS	Distributed Denial Of Service
DNS	Domain Name Server
E2C	Elastic Cloud Computation
ECP	Enhanced Client Proxy
FIM	Federated Identity Management
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure as a Service
IdP	Identity Provider
ISSRG	Information Systems Security Research Group (University of Kent)
JSON	JavaScript Object Notation
KDC	Key Distribution Centre
KVM	Kernel-based Virtual Machine
MDSSO	Multi-Domain Single Sign On
PaaS	Platform as a Service
RBAC	Role Based Access Control
REST	Representational State Transfer
SaaS	Software as a Service
SAML	Security Assertion Markup Language
SETI	Search for Extra-Terrestial Intelligence.
SP	Service Provider
SSO	Single Sign On
TGS	Ticket Granting Service
TGT	Ticket Granting Ticket (See Kerberos)

UML	Universal Modelling Language
URL	Uniform resource locator
VCS	Version Control Software
VM	Virtual Machine
WAYF	Where Are You From
XML	Extensible Markup Language

2.0 Introduction

In recent years, the Cloud has become a significant technology, representing a paradigm-shift from the traditional models of computing. Pallis states that the Cloud provides convenient, on-demand access to a shared pool of resources that may be dynamically redistributed with minimal human interaction (2010). As a leading "cloud computing platform for public and private clouds" (OpenStack. 2014), OpenStack is shaping the way that businesses and users are managing their resources.

OpenStack was created in July 2010 via a collaborative project between RackSpace and NASA. It was developed to provide Infrastructure as a Service (IaaS), a cloud platform to facilitate shared computation and storage (Baset, 2012). The broad capability of an OpenStack installation is defined by its 5 core services. These may be used in isolation or combined to provide a rich feature-set. Most notably, the Swift service offers large scale file storage, whilst Nova manages images of virtual machines(VMs) stored within the Glance module. In order to be granted access to any one of these services, a user must authenticate via Keystone, a central authentication service.

The wealth of infrastructure afforded by the Cloud has led to large-scale collaborative projects that necessitate the management of large groups of users. Until now, OpenStack has relied upon a centralised database to manage permissions. Each user must be individually registered in Keystone, by a systems administrator or bulk-loaded from a database (Chadwick et al. 2013). Federated Identity Management seeks to simplify the management of large volumes of users by transferring the authentication process to an external source, trusted by Keystone. This new approach brings benefits, such as the ability to support a wide range of authentication protocols and is an important step in the development of federated clouds.

2.1 Objectives

The primary objective of this project is to implement a system of federated authentication to the command-line client of OpenStack's Keystone module. In July 2013, the Information Systems Security Research Group (ISSRG) at Kent University developed an Application Programming Interface (API) to facilitate federated authentication in OpenStack (Chadwick et al. 2013). The initial implementation focused on the distributed object storage system, Swift and provided a working method of federated authentication as a proof of concept. This work is a continuation of these achievements and aims to integrate the federated API into a command line client for Keystone, the core security component of the OpenStack platform.

From detailed discussion with the project supervisor, a formal acceptance criteria has been documented (see Appendix 1.0). As such, it serves to clearly illustrate the explicit objectives of the project and the criteria for success. These may be broadly divided into three complimentary objectives. The first, to observe the existing implementation of the federated API within the Swift client; developing an understanding of its behaviour and requirements. Secondly, to analyse the Keystone client and plan the necessary enhancements. Finally, the new authentication method will be implemented. Any changes must be then be validated via testing procedures that include a suite of standardised OpenStack regression tests.

3.0 Background

3.1 What is cloud computing?

Cloud computing is a collective term given to applications delivered as a service across the internet and the server hardware tasked with their provision. (Armbrust, 2010). It represents a realisation of a long-held desire to provide a distributed and highly scalable computing environment. As a recent technology, the Cloud's short history can be traced back to 1999, when the SETI project implemented an early example of distributed computing. SETI@home took a divide and conquer approach to complex mathematical calculations. Individuals would connect to a server that would distribute small components of a larger task and each client would crunch numbers toward a single goal (Weiss, 2007).

In 2006 the first commercial web-based computation and storage facilities were offered by Amazon (Amazon, 2014). Amazon Web Services (AWS) is renowned for its Elastic Cloud Computation (E2C) which allows scalable computation capacity on demand. (Cloud, 2011). Furthermore, services such as the Simple Storage Service (S3) gave access to a scalable, reliable and inexpensive data storage infrastructure (Cloud, 2011). From a business perspective, the cloud abrogates the need for large capital investment in physical hardware. Instead, adopting a variable model that allows resources to be scaled back if over-provisioned or extended in situations of unexpected demand (Armbrust et al. 2010). The act of delocalisation "opens doors to multiple, unlimited venues from elastic computing to on demand provisioning to dynamic storage and computing requirement fulfillment[sic]." (Behl & Behl, 2012). As a consequence, sharing data across multiple enterprises has been simplified and the low cost of deployment is sympathetic to the risk averse nature of modern business.

Today, "cloud computing has emerged as a viable and readily available platform" (Kaufman, 2009) and a large number of competing projects provide similar functionality to AWS. These "open source Clouds" have the capability to provide a wide range of services - broadly divided into three general categories:

Infrastructure as a Service (IaaS) concerns itself with the provision of all necessary resources required to install and run arbitrary software. Whilst the consumer has no control over the infrastructure, they may choose the operating system, storage behaviour and applications deployed.

Software as a Service (SaaS) offers less control. A provider's applications may run on a cloud infrastructure but the user has no control over the physical elements of server hardware or its operating system. Instead, a user accesses the applications with a browser or thin client.

Platform as a Service (PaaS) facilitates the deployment of arbitrary software by an independent developer. They are granted high-level control of an abstracted layer, deploying software that runs on the physical layer beneath. (Mell & Grance, 2011).

When these resources are made available to the public as a paid service, it is referred to as a *public cloud*. Contrastingly, a *private cloud* can be thought of as a cloud infrastructure solely controlled by a large organisation. There are numerous open-source cloud platforms currently available, including (but not limited to) Eucalyptus, OpenNebula, Nimbus and OpenStack.

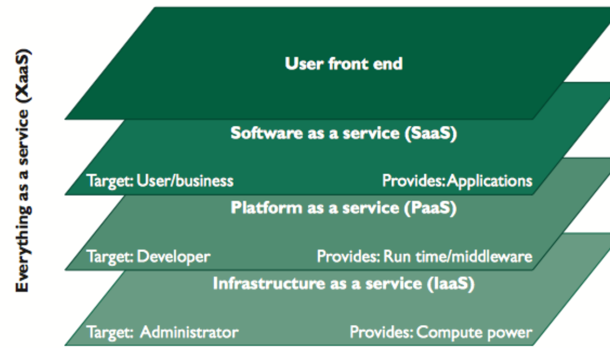


Figure 1. A schematic showing the conceptual layered-hierarchical structure of the service model types. Source: Pallis, 2010.

3.2 A brief history of OpenStack

Since its appearance in 2010, OpenStack has quickly become a popular platform for the provision of IaaS for both public and private Clouds. The project emerged from a unification of the computational capability developed by NASA and the object storage system developed by RackSpace. Now combined, an extended OpenStack platform, comprising five core services is available for Linux under the Apache 2 open-source software licence (Xiaolong, 2012). Figure 2. shows the current core architecture for OpenStack IceHouse, the ninth and most recent release.

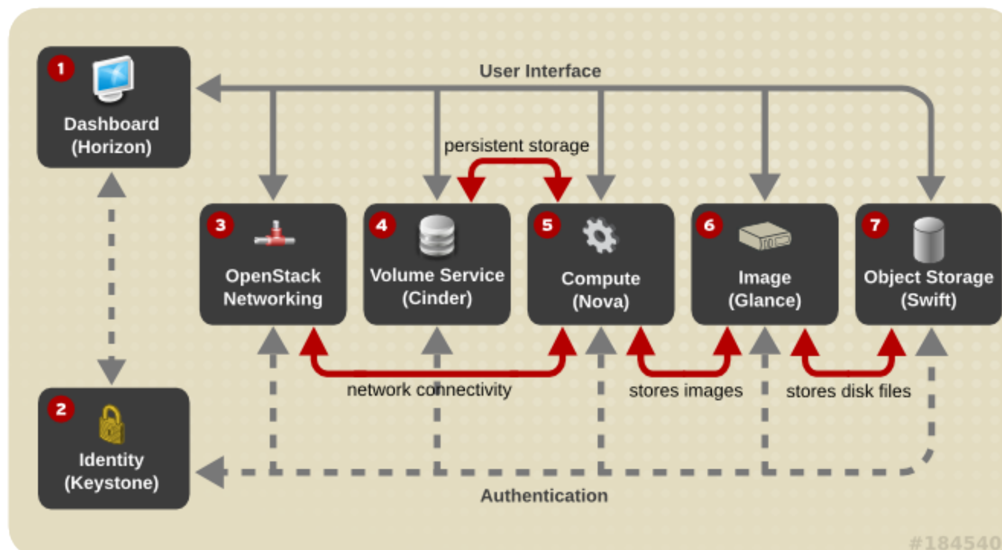


Figure 2. The core architecture of OpenStack. Source: Redhat, 2014.

From the figure, Swift facilitates the storage and retrieval of files. The machine state of virtualised servers is handled within Cinder and in turn stored as an image file within Glance. Nova is a computational service that facilitates the concurrent execution of numerous virtual devices. It provides support for a wide range of hypervisors, such as KVM or IBM SoftLayer. In a recent collaboration, IBM and Mirantis demonstrated the stability of OpenStack by scaling a single cloud at a sustained rate of 9000 new virtual servers an hour. A total of 75,000 virtual machines were created from a mere 350 physical devices. (Mirantis, 2014).

Of particular interest to this study is the behaviour and the internal workings of the Keystone module. Keystone is a core identity service that provides token, policy and catalog functions via an API (Rhoton, 2013). It's purpose is simple; to establish the identity of a user based on their credentials and manage the actions that each person can perform. It thus becomes clear that the authentication credentials used to access any service must be unique and proprietary to each individual. (Rhoton, 2013). A discussion of the authentication procedure within Keystone is presented in section 3.9 - Keystone in depth.

3.3 Authorisation Techniques

The security issues surrounding Cloud computing are compounded by an additional risk originating from within the cloud itself. Indeed, "security is one of the most-often cited objections to cloud computing" (Armbrust et al. 2010) with companies expressing concern over the safety of remote storage.

Over the years, a multitude of authentication techniques have been tested, developed and abandoned. A simple username and password combination provides the cheapest and easiest method of authentication. Consequently, it has seen near-unanimous incorporation into every new Cloud service. (Zhu et al, 2014). Whilst the weakness of password-based login has been well-documented (Ziqing et al. 2011)(Zhu et al, 2014)(Morris & Thompson, 1979) a viable universal replacement is yet to be adopted. To enhance security, *multi-factor* authentication requires a user to provide additional information such as a key from a physical key-generator, shown in Figure 3. Whilst these systems have shown improved security they have been deemed ineffective, "due to their security design or additional overheads". (Xuguang & Xin-Wen, 2012).

Figure 3. A RSA SecurID 700 Authenticator KeyGen. (TokenGuard, 2014)

3.4 Federated Authentication

Today, an average user is likely to have in excess of ten accounts for their various cloud services. (Zhu et al, 2014) Furthermore, the near-exponential growth of cloud applications is presenting a significant challenge in the management of the passwords required for their access. Users are reluctant or unable to memorise many, complex passwords hence repeatedly use one simple password for multiple accounts (Zhu et al, 2014). Mass centralised storage of user credentials in conjunction with key-derivation functions has been criticised as a risky, high-value target for hackers. (Zhu et al, 2014). Federated management techniques have been developed to share identity attributes between services without centrally storing this information (Shim et al. 2005). Using a federated identity the security of Cloud services increases, since a user is able to choose a single, strong password that may be verified by a specialised Identity Provider (IdP)(Jensen, 2102).

Federated Identity Management (FIM) relies on a mutual agreement between service providers on a standardised list of attributes that refer to a user. (Oasis, 2014). Furthermore, services must adopt a common protocol in order to facilitate this exchange of information. Once in place, organisations can use a shared name identifier to aggregate user data across their multiple domains. The proliferation of FIM methods can be seen with the common use of Facebook, Google and MySpaceID as Identity Providers for third-party services (Ko et al. 2010).

The level of access control afforded by FIM is of great significance to large scale organisations and their collaboration. If we consider the example of a university library (Smith, 2008), students may use a single identity in order to gain access to online journals and resources. These service providers are not concerned with the individual identities of each reader. Rather, they seek to ensure that each user accessing their material is from a valid, subscribing institution. The university manages the identity of each of its students. This becomes a key factor in the understanding of the conceptual model of FIM; service providers do not need to collect and maintain identity-related data (Oasis, 2014). Moreover, users "should experience increased privacy protection by having more control over [sic] own identity attributes." (Jensen 2012). In fact, the user may assume a position of relative anonymity since they are treated as a member of an institution rather than an individual.

Federated technologies were initially developed to facilitate web-based Single Sign On (SSO)(Smith, 2008). Whilst there appears to be a conflation in terms (Toronto, 2014), SSO can be thought of as a subset of Federated Identity Management; a functionality that results from its successful implementation. Section 3.5 discusses SSO in greater detail.

Whilst the discussion of Federated technology may have demonstrated great benefits, the adoption rate within the industrial domain has fallen short of expectation (Jensen & Jaatun, 2013). Stolen credentials or the theft of a security token have the capacity to affect "all federation partners" (Jensen, 2012). Furthermore, the possibility of token theft undermines security within systems with more secure processes, such as two-factor authentication. Once an identity has been stolen, access to all federated service providers is possible, greatly simplifying the job of the attackers (Jensen, 2102). In a critique of FIM methods, Landau & Moore state that "federated identity management blurs security boundaries and thus creates liability and privacy risks." (2011). Whilst this criticism is valid, it derives from the inherent weakness incurred by the continued use of simple username and password combinations and provides compelling motivation to implement a viable alternative.

3.5 Single Sign On (SSO)

Single Sign-on (SSO) technologies have been developed as a form of access control that links a user to multiple independent Cloud services (Fengming et al. 2012). Once a user has provided their login credentials, they are permitted to access multiple resources without the need to re-authenticate. SSO removes the need for a user to remember multiple passwords and simplifies the administration of a system. In order to make this possible, a trusted relationship must exist between the authenticating system (or Identity Provider IdP) and the Service Provider (SP), often referred to as a Federation. With this link in place, the SP may assume that the user is both, valid and properly authenticated and create a local session in response (Oasis, 2014). Figure 4. shows a high-level conceptual model of a SSO scenario.

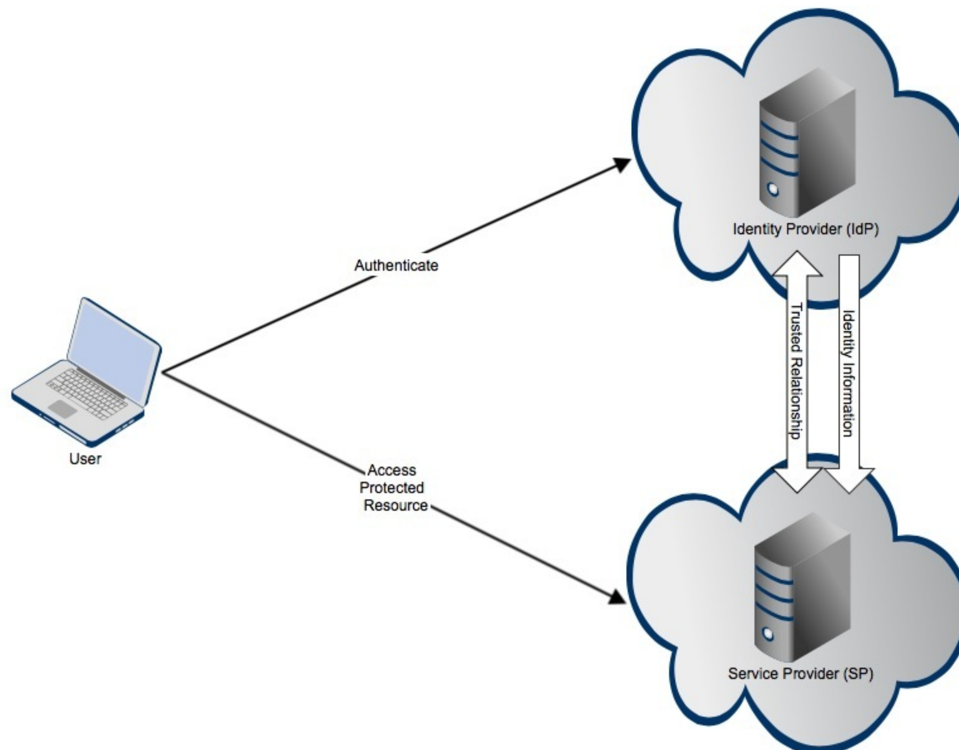


Figure 4. A Simple SSO conceptual model. Based on the model presented by Oasis(2014).

SSO deployment profiles permit the IdP to itself be a service provider. Hence the Authentication step shown in Figure 4 may involve the use of a service provided by the IdP. Historically, browser cookies have been used to store authentication status, but this has been problematic with cross-domain access. Since "browser cookies are never transmitted between DNS domains" (Oasis, 2014) authentication information is lost in translation. The multi-domain SSO (MDSSO) problem is addressed by a number of open source protocols that seek to standardise the transfer of authentication information from one server to another. These include: OpenID, OAuth, SAML, Kerberos.

3.6 Kerberos

Kerberos (Neuman & Ts'o, 1994) was one of the earliest implementations of a security protocol that required a third-party Authentication Server (Chadwick et al. 2013). An encrypted symmetric-key is used to obtain a ticket for a specific service from a central Kerberos server. However, the symmetry of the algorithm used to encrypt each ticket has been identified as a single point of weakness (Zhu et al, 2014). Kerberos is of particular interest as the identity service in OpenStack Keystone has been built on the Kerberos model. However, Kerberos has been further criticised by Chadwick et al for failing to address issues of discovery or "interoperability in a heterogeneous environment." (2013). Figure 5. (overleaf) shows a simplified Kerberos authentication protocol.

Authentication in Kerberos follows the protocol structure described below (Matthews, 2010):

1	The user connects to a Key Distribution Centre (KDC) and provides their authentication credentials. These are checked by the Authentication Server component of the KDC.

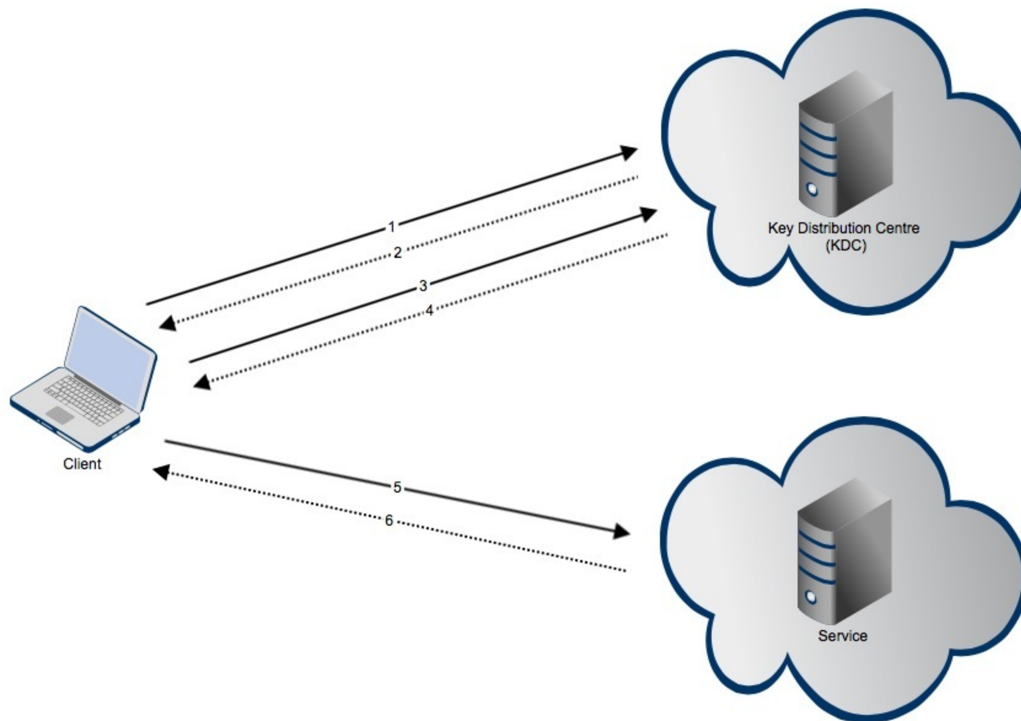


Figure 5. The basic Kerberos authentication protocol (simplified).

2	If correct credentials are provided, a Ticket Granting Service (TGS) returns a Ticket Granting Ticket (TGT) to allow the user to apply for a session ticket to permit access to (potentially) multiple servers on the network.
3	To access a service the client presents the TGT to the KDC to obtain a service ticket.
4	The TGS authenticates the TGT to issue a service ticket, comprising a ticket and session key, matched to the server the client wishes to access.
5	The client creates a session with the server by presenting the service ticket. The information from the TGS is decrypted using the server's key and the user is authenticated.
6	The service is returned to the client.

An important observation in this model is the necessity for a trusted relationship between the service/server and Key Distribution Centre. In Kerberos, this exists via a indirect connection between the two entities, using the client as a proxy to forward the key to the service. However, in some of the protocols, we shall see that this can be a direct connection between the authenticating party and the service.

3.7 SAML

Security Assertion Markup Language (SAML) is an example of a widely used authentication standard that is used extensively in SSO Federated environments (Oasis, 2014). SAML makes use of an assertion - an encrypted XML file that summarises identity and attribute information (Juniper, 2013). Owing to a multitude of potential deployment profiles, SAML may choose to omit any direct link between the Service Provider (SP) and the Identity Provider (IdP). Instead, SAML makes use of a series of HTTP redirect instructions, maintaining its trusted relationship via the

client seeking authentication. Figure 6. shows a SAML configuration using Service-Provider-Initiated SSO.

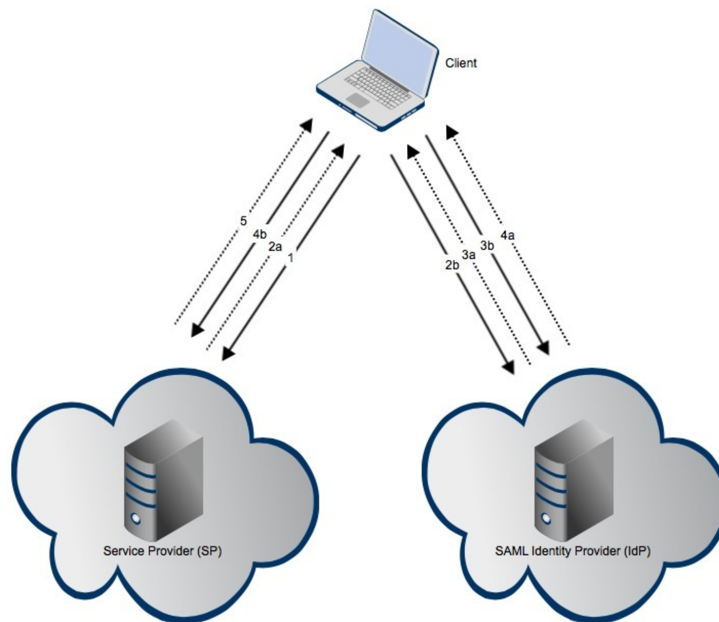


Figure 6. A SAML protocol in a Service-Provider-Initiated SSO scenario.

In the deployment configuration shown in Figure 6, the SAML protocol adheres to the following authentication sequence (Juniper, 2013):

1	The user attempts to access a service on at the Service Provider (SP).
2a	The service provider redirects the user with a HTTP 302 status code. A SAML request is returned to the client.
2b	The client forwards the SAML request to the IdP in an HTTP GET request.
3a	Stage 3a & 3b are not necessary if the user has a session with the IdP. However, if there is no current session, the user is prompted for their login credentials.
3b	The user returns the necessary credentials.
4a	The IdP returns a SAML assertion along with an HTTP 200 success code.
4b	This assertion is redirected to the SP, which is verified.
5	The user is now authenticated and may access the protected resource on the Service Provider.

A SAML exchange occurs between an asserting party - a system entity that makes SAML assertions and a relying party - a system entity that uses the assertions it has received. In situations of SSO, an assertion contains an *audience restriction*, to limit its use to a specified Service Provider (SP). The assertion not only conveys the identity of the user, such as their name and email address but important ancillary information such as the name of the SP and the access policies used to grant access to protected resources. (Oasis, 2014).

3.8 Shibboleth

Shibboleth is an interesting example of an implementation of the SAML standard (Chadwick, 2013), using a SAML assertion with OASIS-defined request and response protocols. The inclusion of an optional Where Are You From (WAYF) service allows for dynamic determination of a user's IdP. (Cantor et al, 2005). This flexibility partially explains its universal success as an authentication mechanism for universities in Europe and America. (Chadwick, 2013). Like many others, Shibboleth relies on a federation of SPs and IdPs to exchange authentication information. The authentication process is outlined in Figure 7.

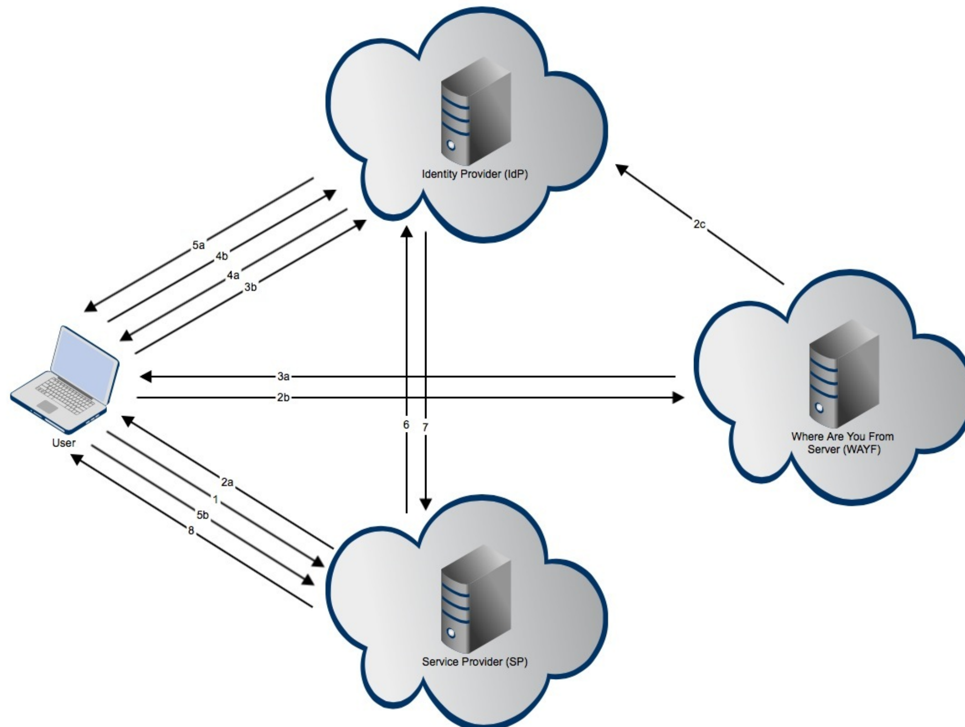


Figure 7. A high level overview of the Shibboleth authentication protocol.

The complexity of the Shibboleth authentication process is aggravated by the WAYF server and its associated traffic. However a simplified configuration without a WAYF server is also possible. Authentication via Shibboleth WAYF follows the protocol stages as discussed by Cantor (2005).

1	The user attempts to access a resource at the SP.
2	The SP issues an authentication request via the user to the WAYF server. The user chooses an IdP from a metadata file listing all the members of the federation.
3	The user chooses the IdP from the list returned by the WAYF server and the authentication request is redirected.
4	The IdP authenticates the user.
5	A <samlp:Response> 'authenticated' message is sent from the IdP to the SP, via the user.

6	On receipt of this message, the SP verifies with the IdP with a <samlp:AttributeQuery> command.
7	The IdP returns a SAML assertion to the SP.
8	Based on the user's permissions read from the SAML assertion, the SP returns the protected resource or an HTTP error.

The Shibboleth procedure may be shortened by specifying a combined authentication and attribute request at stage 2. This removes the attribute query specified at stage 6 amalgamating the authentication and authorisation operations into a single step.

3.9 Keystone - OpenStack's Identity module.

The Keystone module has two primary functions; to register users and their permissions and to provide a catalogue of available services and their associated API endpoints (Slideshare, 2012). The core services in OpenStack subscribe to a policy of token-based access, hence a user must first authenticate themselves before use. Keystone is the identity module responsible for managing all the token, catalog and policy features that govern access to any of OpenStack's services (OpenStack, 2014). Once a user is enrolled in Keystone they may be registered with a project (historically referred to as a tenant). A project is a collection of services configured to a specific application. Roles provide a mechanism to define the extent of a user's permission to each service and a user may be assigned any number of roles within a given project. This complex combination of authorisation information is managed within a single entity, a token. Once a user has been authenticated, each service will refer to that token and grant access where applicable. In order for this to occur all services in the OpenStack family must have a trusted relationship with Keystone.

To obtain a token via conventional username and password authentication a user would complete the sequence shown in Figure 8.

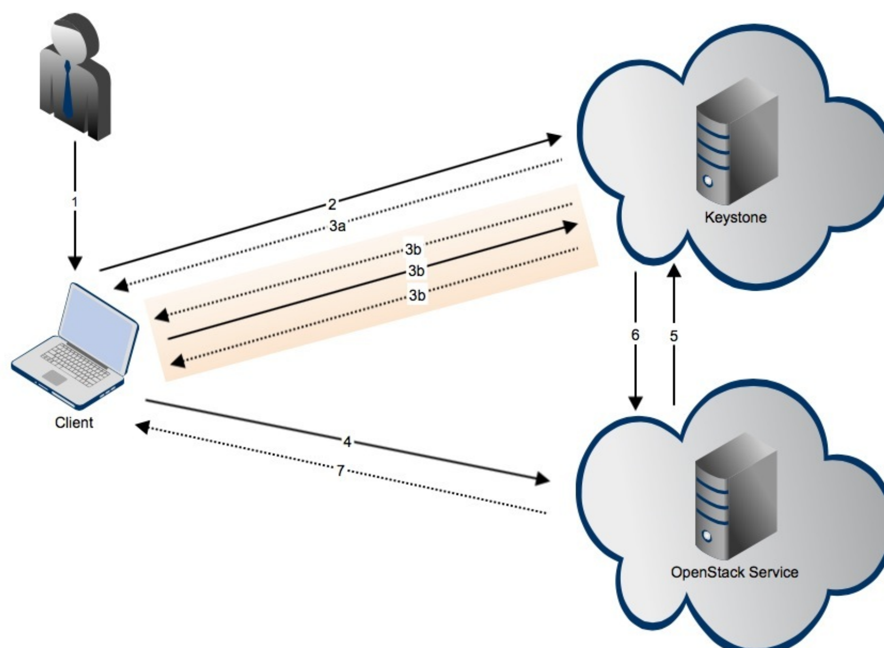


Figure 8. The authentication process for 'regular' username/password login.

1	The user launches the client for the intended service. In doing so they issue a command, providing their user credentials and an endpoint for the service location.	
2	If the user has no token, the client makes a request to Keystone, forwarding the necessary credentials for validation.	
3	Assuming the credentials are valid, Keystone may choose one of the following actions:	
	3a	If the user specified a project then a token tailored to that project is returned to the user, along with a list of endpoints for the services the project makes use of. This is commonly referred to as a scoped token.
	3b	If no project is specified, Keystone can only provide a token that validates the user. This unscoped-token may be used to list available projects. Once the user specifies a project ID, the unscoped-token is returned to Keystone and exchanged for a scoped token. In OpenStack, tokens are scoped to projects. With no scoped token, no access can be granted to any of the services.
4	From the list returned by Keystone, the user chooses the endpoint of the service and makes a request, this time including the scoped token.	
5	Here the trusted relationship is utilised: The token sent to the service is returned to Keystone for validation.	
6	Assuming all is correct, the user is authenticated with the service. Keystone returns the project and the associated roles of the user.	
7	If the request made by the user is within the scope of their role, it is authorised and the response is returned to the client.	

In order to prevent repeating this process every time a request is issued, a token is assigned a timeout and can be stored locally within the client on a 'keyring'. Once the token expires, the user must repeat the authentication process and acquire a fresh scoped-token. In special cases an admin token may be used to perform administration tasks but this carries "no explicit authorization" (OpenStack, 2014) and should be disabled once the Keystone server has been configured.

3.10 Fine-grained access control

Historically, models of access control have been coarse-grained and static (Yuan & Tong, 2005), relying on a policy list mapped to each user. As cloud infrastructures have driven information sharing, legacy mechanisms have become incapable of managing security across multiple domains at an enterprise level. Organisations need to be able to share data yet retain careful control over the access privileges surrounding proprietary material. The mechanisms of Attribute Based Access Control (ABAC) and Role Based Access Control (RBAC) present a solution to the problem by decoupling "authentication from authorisation, separating 'who you are: from what you can do'" (Jericho Systems, 2014).

The RBAC model facilitates cross-domain authorisation by mapping the role of a user in one domain to an equivalent level of permission in another. As a consequence, administration overheads are reduced as "permissions no longer need to be repeatedly assigned to individual users." (Yuan & Tong, 2005). Whilst this may initially appear a viable solution, a number of competing Access

Control (AC) models coexist. A clear limitation of RBAC is the reliance on other domains to adopt the same RBAC model (Long et al. 2010). Furthermore, RBAC has been criticised for failing to scale well, requiring the "definition and management of static roles." (Yuan & Tong, 2005).

In contrast ABAC offers a flexible model that defines permissions based upon any security-relevant attribute. This allows ABAC to offer the functionality of RBAC or extend the AC to encompass a multitude of attributes. For the first time, ABAC allows AC based upon the situation, date, time or context of the access request. These environment attributes and the fine-grain control they afford is largely overlooked in most access control policies (Yuan & Tong, 2005).

The proposal to implement ABAC in OpenStack was submitted as a blueprint in 2013 (Jin). Whilst initial work has been completed to customise user and object attributes, the enhancement is yet to receive formal acceptance by the foundation (Jin, 2013). For the time being, Keystone operates a RBAC model. Whist a scoped token from Keystone permits a user to access a project, we must examine the token to determine the *role the user has in the project.* Only if a user fulfils the role requirements imposed by the owner may they perform the desired operations.

3.11 The Keystone clients.

In order to issue commands to an OpenStack installation, a user may choose to use a command-line client or a web-based graphical user interface (Horizon). Whilst Horizon simplifies the user experience its current feature-set lacks the capability to perform batch processes. In situations where an administrator may wish to automate a large number of tasks, the command-line affords an efficient form of interaction, with greater capability than its graphical counterpart.

OpenStack's command line utilities make use of standardised Representational State Transfer (REST) with Application Programming Interfaces (APIs) tailored to each service. REST is an architectural principle that facilitates communication between a client and server, without either entity requiring knowledge of each other. REST is said to be stateless as all the necessary data required for a successful request is sent with each communication. In the case of OpenStack, RESTful requests are made to server endpoints with the standard HTTP protocol.

Historically, each of the OpenStack services has relied upon a dedicated client to manage user interaction. With the release of OpenStack Havana these standalone clients have been deprecated in favour of a universal client. This new OpenStack client unifies and extends functionality, providing compatibility with Havana's v3.0 Identity API. In order to preserve back-compatibility, standalone clients have been retained within the source code, but lack the necessary functionality to be able to interact with the newest versions of the API. In time they will be phased out, in favour of the universal client. The differences between the APIs and their clients is discussed in section 5.0.

3.12 Development Tools

OpenStack is written predominantly in Python, 2.7 widely accepted as the most stable version of the language (Kerner, 2013). Whilst new functionality is being implemented in Python 3, migrating the 1.25 million line code-base to the new standard is a huge task (Kerner, 2013). Consequently, the code-base can be considered a hybrid of Python 2.x and 3.x. Python is an interpreted language that compiles its source to an abstract byte-code intermediate run on a Python virtual machine

(Python, 2014). The development overheads of Python are small, requiring a base installation for execution and a text-editor, such as Vim, for development. Any dependencies required by Python are handled by Pip (Pip, 2014), a Python-specific package manager analogous to the standard package manger that accompanies linux installations. Once a package has been installed it may be integrated into a Python script with a simple import statement.

Openstack makes extensive use of GIT, a popular form of Version Control Software (VCS) that tracks the changes made to designated Configuration Items (CIs). VCS was developed to address the overlapping update scenario, where two developers would simultaneously work on the same file. On a small development project with a single developer this scenario is impossible to encounter. However, GIT offers many additional features, such as allowing a user to observe the chronology of changes made to a code base. Once the code has been finalised it may be 'pushed' to an online repository for review.

Any new code implemented in OpenStack is subjected to a suite of regression tests using the Tox framework (Tox, 2013). Tox is a generic, virtual environment (virtualenv) based tool that automates testing over a number of different deployment profiles. Since a number of versions of Python exist, Tox allows code to be tested against environments that may be markedly different to that of its development. In each case, an isolated virtualenv is configured to contain the libraries and dependencies required by the application (Fruiapps, 2012). Tox is primarily used to perform regression testing and ensure compliance with pep8 syntax standards. Tox is lightweight, easy to configure and highly compatible with agile methodology.

4.0 Analysis and Design

4.1 The server-side implementation of Federation.

Currently, a keystone server has the capability to support two variants of its Identity API - version 2 or 3. For the purpose of this discussion we may safely assume that version 1 of the identity API has been deprecated and the installed user-base have made the necessary upgrade. What is not immediately clear, are the large differences in the way a request is made to each API, the server response and the broad incompatibility between the two.

4.2 The Identity API v2, and OpenStack Grizzly.

The Grizzly release of OpenStack and its v2 identity API had no native provision for federated authentication, nor support for proxy to an external service. This presented a major obstacle for authentication processes that required redirection to an external IdP. An early implementation of federated identity management in OpenStack was discussed by Chadwick et al. (2102). Here, they discuss the mechanism of detection and dispatch that all Keystone traffic is subjected to; a request is passed though a two-stage processing pipeline. First, the nature of the request is detected, then redirected to a relevant service module via a *router*. These software components, termed middleware have been shown in Figure 9.

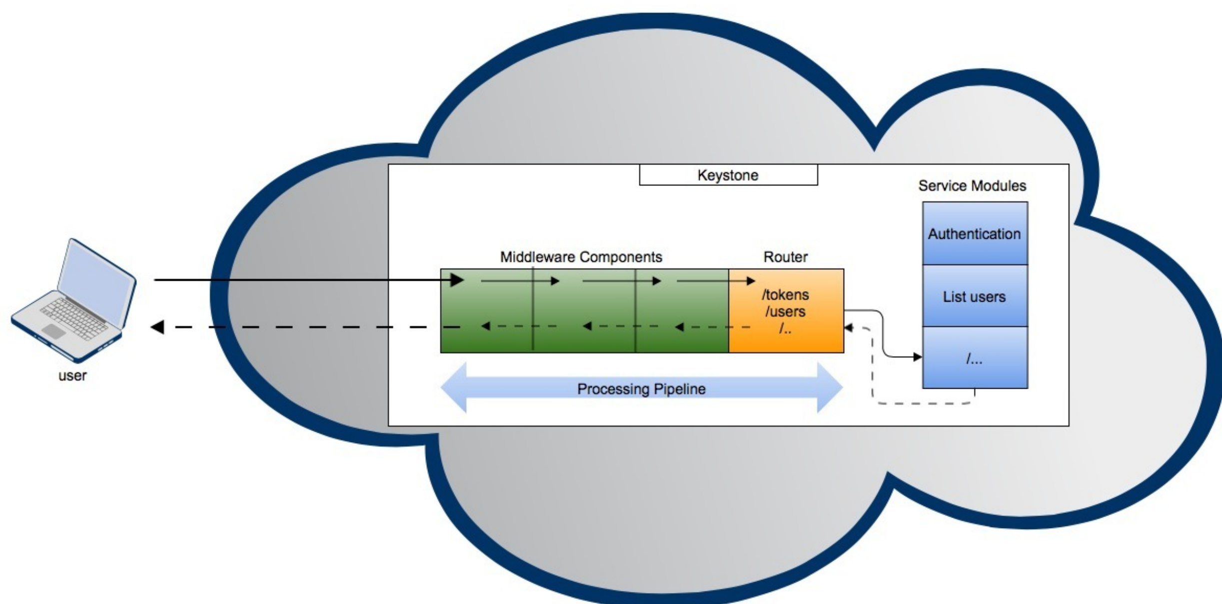


Figure 9. Keystone's Middleware Components. Adapted from Chadwick et al. (2012).

Realising that "new software components are easy to create and easy to add into the pipeline" (Chadwick, 2012) the group added a new middleware component to interrogate HTTP messages for a specific *X-Authentication-Type:federated* header. Once detected, the message would be intercepted and processed by a new federated module. In a manner analogous to a switch, when the federated module detected a federated message with a blank body it would return a list of IdPs from Keystone's directory service. Different commands could be sent to the federated module by appending a new element onto the HTTP body in the JSON data format.

Once the Keystone server had been modified the service clients needed extension to support federation and its multiple protocols. As a proof of concept, a new command was added to the

Swift client to indicate a user wished to authenticate via federated methods. In response, the client would load a new federated library and build a HTTP request with the necessary *X-Authentication-Type:federated* header. Once a user had chosen from a list of trusted IdPs the client would load a corresponding protocol module and contact the IdP in an appropriate manner. When the authentication process was complete Keystone could issue either a scoped or un-scoped token and return to normal flow.

The amended authentication process can be described as follows:

1	The user launches the Swift client with a command, a Keystone endpoint and an instruction to use federated authentication.
2	Recognising the user is not authenticated, the client contacts keystone with a request for federated authentication. The client sends an HTTP GET request with a blank body to Keystone.
3	The federated component of the keystone middleware intercepts the request and returns a list of supported IdPs to the client.
4	The user is presented with a list of IdPs and asked to make a selection. In the proof of concept implemented, three protocols are supported: Keystone Proprietary, ABFAB and SAML.
5	The user chooses an IdP. The client sends this choice to Keystone and receives and authentication request.
6	The client interrogates the server response to determine the necessary protocol and loads the corresponding protocol module.
7	The client redirects the user to the IdP along with the authentication request. The user authenticates with the IdP and an assertion is returned to Keystone via the client.
8	Keystone generates an un-scoped token which is returned to the client and stored in its 'keyring'.

Whilst the project can be considered a successful proof of concept it was not without its problems. A major drawback of the chosen design is the necessity for the middleware to interrogate the header of every HTTP message reaching the server. This additional overhead introduces greater latency and could be problematic in situations of high network volume. Furthermore, the use of SAML relies on SSO persistence by storing a cookie in a web browser. Clearly, the command-line environment lacks this functionality. Consequently, the user environment has to change focus and a browser automatically opens for the user to complete authentication in the GUI. This hybrid approach is somewhat inelegant and a integrated command-line implementation warrants further investigation.

4.3 The Identity API v3, OpenStack Havana and Icehouse.

The release of OpenStack Havana in December 2013 was the first to make use of version 3.0 of the Keystone Identity API. In April 2014 its successor, Icehouse, began to use the unified client, providing full v3 compatibility at the command line. Whilst Havana can be considered closer in behaviour to Grizzly, Icehouse is a marked departure, with native provision for federated authentication within the API.

4.3.1 OpenStack Havana

The inherent shortcomings of the federated design in Grizzly led the Kent security group to a more refined approach to its successor. The Havana architecture abandons the middleware pipeline for a dynamic dispatch centre to issue commands to the relevant authentication service. By adding a new plugin component, the server could interpret and issue commands to a new federated module.

In the Havana server, federated requests target a single endpoint: */auth/tokens.* The subsequent authentication procedure has been divided into four distinct stages:

Discovery

The /auth/tokens endpoint is targeted with a POST message containing 'phase': 'discovery' in the body of the request. The server returns an error; that additional authentication is necessary, accompanied by a list of supported Identity Providers.

Request Issuing

The request phase is declared within the body of a POST message along with an ID of the chosen IdP (read from the discovery response). The server responds with a SAML authentication request (other protocols are supported) and an endpoint for the IdP.

Negotiation

This optional stage is used in instances where the chosen protocol necessitates multiple client-server interactions. A message is POSTed, containing a declaration of the negotiation phase, the name of the IdP and any additional data required for authentication. The resultant response contains all the protocol-specific data needed to authenticate in Keystone. In the case of SAML, this will be an assertion.

Validation

The SAML response message is forwarded to the Keystone server along with the ID of the IdP and the 'validate' keyword. Keystone responds with a scoped or un-scoped token accordingly.

The dispatch module interrogates the body of every HTTP request in order to determine its destination. It thus follows that every message must declare its phase in order to ensure the efficacy of the process. A detailed discussion of this procedure can be found in the design document "A more robust design for a federated authentication plugin" (Siu n.d.).

4.3.2 OpenStack Icehouse.

The Icehouse Keystone server makes use of an external Apache front end, loading a modified Shibboleth module (mod_shib), responsible for directing users to the IdP and communicating the result using a SAML assertion. Although SAML is the only protocol currently supported, mechanisms such as OAuth and OpenID are planned for future versions (OpenStack, 2014). The mod_shib installation is backed by an Apache HTTP server to manage the transfer of authentication requests. These two technologies are used in conjunction to provide an additional layer of abstraction; treating Keystone as if it was a service provider.

The V3 API has been designed such that federated authentication can be achieved via a sequence of requests to specific endpoints. The conceptual model of federation remains remarkably similar to the process outlined for the Grizzly server, albeit without the inefficiencies discussed.
First the user must target the IdP endpoint:

auth-url/OS-FEDERATION/identity_providers

where auth-url is the authorisation URL of the Keystone server, provided by the user. In turn, Keystone will return a list of trusted IdPs and their supported protocols. (Note the capability for a single IdP to support multiple protocols). The user may choose an option and make an authentication request to the following endpoint:

auth-url/OS-FEDERATION/identity_providers/*{realm}*/protocols/*{protocol}*/auth

Where {realm} is the name of the IdP and {protocol} one of its supported protocols.

Whilst the API has support for federation, changes are yet to be implemented in the universal client. This forms the justification behind the work conducted in this project. By examining the previous implementation of federation in the Swift client, a greater understanding of the federated functionality can be gained and integrated into the various stages of the project.

4.4 Analysis of existing implementation - The Swift Client

The Swift client is the command line interface for issuing commands to OpenStack's object storage module. It was targeted by the Information Systems and Security group at Kent University as an easily accessible candidate for modification and formed the basis for the work discussed in 2013 (Chadwick).

The swift client is launched from the command line using the following command:

swift -F -A http://fedkeystone.sec.cs.kent.ac.uk:5000/v2.0 list

Where Swift is the name of the OpenStack service (the binary executable), -F the Federated option and -A indicates we wish to specify a specific endpoint. The endpoint URL is appended with ': 5000', the default port address for an OpenStack installation. v2.0 states our intention to use v2 of Keystone's Identity API. Finally, we issue the command we wish the swift module to complete. In this case we list all the projects for the authenticated user.

From analysis of the source code a number of key functions have been identified. A full discussion of the code is presented in Appendix 2 which has been used to construct a UML module diagram of the core functionality of the client (Appendix 3). Here we can observe low coupling and high cohesion of the Federated API and it's two utility modules; federated_exceptions and federated_utils.

4.5 Analysis of existing implementation - The Keystone Client.

The Keystone Client will form the starting point of this project, hence a detailed understanding of its functionality is required in order to correctly integrate the planned functionality. A detailed description of the key functions is provided in Appendix 4. Analysis of he Keystone Client. From this analysis, a call to the Federated API needs to be made before the pathway to regular authentication is invoked (via the addition of another conditional statement). Conditional checks for authentication within the current implementation, rely heavily on the attributes of an *args* variable created by the sub-parser. Here, we aim to make the necessary checks and bypass regular authentication altogether. Successful authentication by an IdP should return a token from the federated API before the program returns to normal flow.

4.6 Analysis of existing implementation - The OpenStack client.

The universal client is the most recent implementation of the command line interface for the OpenStack services. It is the first client to provide compatibility with Keystone's v3 Identity API. In comparison to the functional complexity of the Keystone client, the OpenStack client is more simple, unifying the existing functionality of the standalone programs into a single executable.

The OpenStack client defines its own command parsing functionality within the shell.py module; taking attributes specified by the user and building a client via the clientmanager.py module. To use the v3 API, an OS_IDENTITY_API_VERSION environment variable must be set to 3. In all other instances a version 2 client will be created by default. The OpenStack client imports the functionality of the keystone client and creates an object of type Client(v3) directly from the library. The HTTPclient.py superclass initialiser is invoked creating the necessary server request, managing authentication and invoking the relevant call-back functions to perform the required action. It thus follows that once the keystone client has been extended, it should be a simple step to integrate it into the OpenStack client.

4.7 Design of proposed changes

The large, established codebase of the keystone client already exhibits a highly modular structure. In continuation of this pattern, the Federated API will reside in its own module; as such displaying high cohesion with minimal coupling with existing classes. This simple choice of design is compliant with Bertrand Meyer's open/closed software principle (Meyer, 1988). Meyer states that effective code should be open to extension but closed for modification. By taking this approach, one can minimise the chances of introducing errors and emergent behaviour.

It is important to recognise the existing conventions of the current design and incorporate these into the new functionality. From this analysis the following have been observed;

- Program errors caused by the user failing to supply sufficient parameters for authentication are handled as an Exception of type CommandError. The error message is incorporated into the exception and the result printed to the screen.
- The exceptions.py module defines specific exception types for the program. New functionality should make use of these existing definitions, or define a new type and import the library accordingly.
- The definition of command-line options follows a convention of a single hyphen and a capital letter (e.g. -X) or two hyphens and a descriptive name, separated by hyphens (e.g. --os-auth-url).
- New code should be Python 3 compatible.

5.0 Implementation

5.1 Project Management and development methodology

As part of the planning process, a detailed Gantt chart was produced, showing the chronology and duration allocated to each task (See Appendix 5). Making an accurate time prediction was troublesome, thus a simple time-boxing strategy was employed. "Time-boxing allocates a fixed time period, called a time box, to each planned activity" (Wikipedia, 2014). Whilst simplistic, time-boxing is ranked 23rd in the top 200 list of best practices (Jones, 2010) and is considered one of the "six common features to the various Agile methods" (Coram & Bohner, 2005). With a low planning overhead and compatibility with a broad range of methodologies, time-boxing is highly suited to this project.

Once a detailed schedule had been created, tasks were transferred to TRAC for dynamic management. TRAC is an "enhanced wiki and issue tracking system for software development projects" (TRAC, 2013). It's features allow for easy subdivision of tasks into job tickets, whilst additional tasks or outstanding issues may be added or reassigned where necessary. Importantly, TRAC acts as the hub of the project, presenting the resources generated at each stage in an accessible manner.

Undertaking an individual research project requires a modified approach in comparison to many aspects of conventional software methodology. A significant proportion of time was spent understanding OpenStack, Keystone and its supporting technologies. As a consequence, there were few iterations of development and greater time spent comprehending the existing code base. The explorative approach required for this work led to an an Agile-like methodology, with each small change being integrated and tested. For example, when the new federated flag was added to the client, it was validated with functional and regression tests. Whilst this can be considered a stage of development, it doesn't offer sufficient new functionality to be considered an iteration in its own right. The day-to-day management of these iterations can be seen in the accompanying log-book.

5.2 Changes to the Software Requirements.

Within the early stages of development the software requirements were changed by the supervisor. Consequently the scope of the project has been extended, to include additional objectives. Namely:

1. Extend the federated library to facilitate v3 API compatibility in the Havana release.
2. Refactor the federated library and integrate into the OpenStack client, compatible with the Icehouse release.
3. Refactor the federated library to support SAML ECP in Icehouse.

In response to this, any contingency built into the schedule in Appendix 5 has been replaced by additional development and validation activities.

5.3 Changes to the Swift & Keystone clients

To understand the planned implementation within the Keystone client a number of small changes have been made to the source code of the Swift client. A full description of this work is provided in Appendix 6. Having gained an understanding of the command parsing infrastructure, attention was turned to the keystone client.

The convention of command formatting in the OpenStack clients permits both, short and long strings. To continue this practice, it was decided that -F and --federated flags would be adequately

descriptive of the functionality offered. The command parser was amended to handle these commands and the help function updated to reflect these changes.

To authenticate via federated methods, the user must use either flag, in conjunction with an authentication url. This requirement is enforced by an auth_check() function that acts as a filter to ensure the user always provides the correct parameters. If the values specified are of the correct combination the conditional statements succeed and the function completes. auth_check() has been extended to test for a *'federated'* boolean variable - assigned a value of True if the user specifies either of the -F or --federated flags. Figure 10 shows a decision tree for the auth_check() function; the new logic has been highlighted in green. If the user chooses federation, but fails to provide an authentication url, the program follows convention and exits with a CommandError exception and a message describing the error encountered. Note the core behaviour of the client has been preserved: If the user has a token (such as an admin token), this is favoured over all other forms of

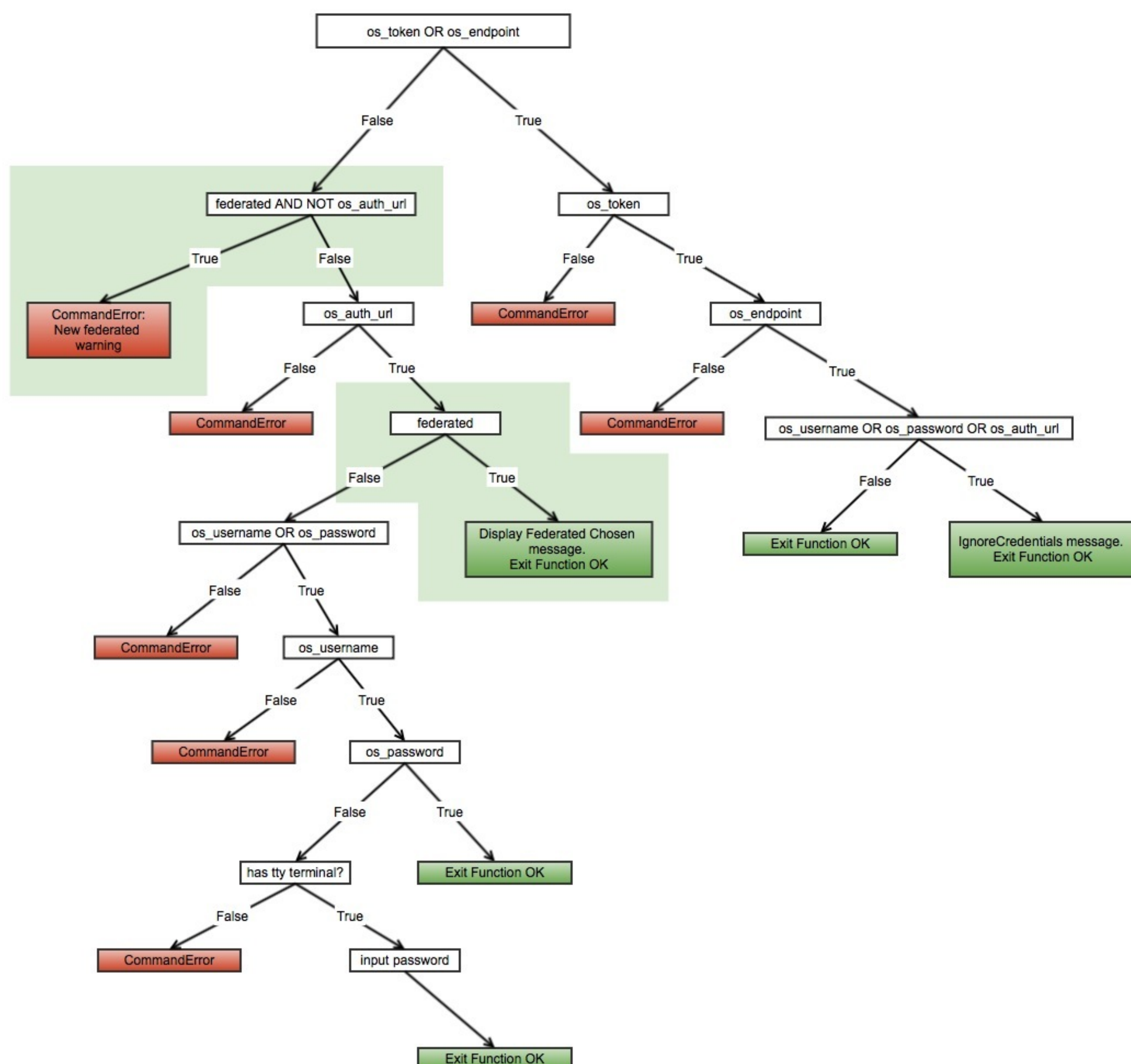


Figure 10. A logic-tree of the auth_check() function and the location of code extension.

authentication. It thus follows that in order to use federation any admin tokens stored in environment variables must be unset to permit selection of an alternative branch.

5.4 Dual v2 & v3 compatibility

The existing federated API was integrated into the client and its v2 compatibility tested. The pathway to creating a connection for authentication is complex, centring on the creation of a client object. In turn, functions from the HTTPClient superclass are invoked, to obtain a token and store it in a keyring. Finally, the keystone command is matched to a corresponding callback function performing the desired action. Once this work had been completed, the new code was tested and validated, before moving onto the v3 compatibility.

To offer the capability of dual API support, the client needs to be aware of the Keystone version specified within the authentication url. The user is required to provide the API version by appending the endpoint. In the example below, we target the v3 API;

http://fedkeystonev3.sec.cs.kent.ac.uk:5000/v3

The client detects the explicit use of a '/v3' substring within the --os-auth-url and sets a corresponding v3 flag to True. Once determined, the federated API can generate a HTTP request with a body compatible with the server at the specified endpoint.


```
{u'error':{
        u'message': u'Additional authentications steps required.',
        u'code': 401,
        u'identity': {
            u'methods': [u'federated'],
            u'federated': {
                u'providers': [
                    {
                        u'description': u'Abfab Service',
                        u'type': u'idp.abfab',
                        u'id': u'123456',
                        u'links': {
                            u'self': u'http://129.12.3.223:5000/v3/services/123456'
                        },
                        u'name': u'ABFAB'
                    },
                    {
                        u'description': u'Kent Proxy Identity Service',
                        u'type': u'idp.saml',
                        u'id': u'da0f5973f42e45b29ecbff6df17174d8',
                        u'links': {
                            u'self': u'http://129.12.3.223:5000/v3/services/
                        da0f5973f42e45b29ecbff6df17174d8'
                        },
                        u'name': u'Kent Proxy Identity Service'
                    }
                ]
            }
```


Figure 11. A Havana server response, displaying the '401 error' and a list of IdPs.

Choosing v3 authentication appends the keystone endpoint with a */auth/tokens* substring and an HTTP POST message is made with a body declaring the 'discovery' phase. As the user has no

token, the server responds with an HTTP 401 error in combination with a list of available IdPs, shown in Figure 11. Using this list the user may select an IdP and make a second request to Keystone for an authentication request. The same HTTP 401 error is encountered and the IdP protocol contained within the response is used to load the relevant protocol module. Figure 12 shows a SAML authentication request returned from the Keystone server. In the case of SAML, a


```
'info: ', {
        u'error': {
          u'message': u'Additional authentications steps required.',
          u'code': 401,
          u'identity': {
               u'methods': [u'federated'],
               u'federated': {
                    u'data':
u'SAMLRequest=nVXbkppAEH33KyzyaLFcRAVKrUJZjYriet31JTXACKMwA8wQZL8%2BuNYmxl1TSXjs
Pn1O92lo2hREYawbGQvwAiYZpKx6ikJM9bdEh8tSrBNAEdUxiCDVmasvjamlyw
%2BiHqeEEZeE3FXJnysApTBliGCuOjIv0CxDnq5CoGieJvGiI7m8oioir6qewquy6ABNaUravs5VNzClZW2H
K6lKAkozOMKUAczKkCgpvNjiZXUlNfS6psvSjqsa73p9gmkWwXQJ0%2B
%2FILcs8eOpwJYvBWIqcjMELAmH%2FFtKtVK%2Be9nlI
%2FU077U5gsWQEw36IIGZt4Tr3sSrWZ6UhI3NOQuQWVSMMSd5PIWCww7E0g1x1QNIIsD9beI4gj9%2B
%2FQXWWAkzP4pxwo7hEPgYsS%2BFlOR0uYCzWBSHP84e8%2FkBSX5BFURRETSgBHkX%2BF
%2B4TCuiN8J7cJPoAE4xcEKJXcHZ3CllAvHIin6SIBdEdMUmQxLMYD08u70oK%2FnK%2F67%2Bk
%2FK3%2FlAKeBkD6wLqAe5hC7MKb%2BOrs3tlJei%2Fxbw1A%2FB2GJIYeT9%2Fn
%2BNCLcFfURH75%2Bf3P4J8OfaHbgDCD3QRMWi%2F95mpJjhT6WpCY
%2BX6I1j2lRTtt4Rp50%2Bw954S778bPDV74bNRS1NXOzjMhFqX8NA41J88LOX7smeYU5wcnP2x7cygk
%2FZmQUDDamHKmKbuJXcyeR1pUCZPnY9%2B10omjamQy2FpJMRB2x9PMc5o9D6Dda20lNVVRFori
MJyNW464WRu%2BJzmqF%2FiPUSVx10P5kFrhXlgspgNHni4Oe3v5AutP88BC
%2FnFlOVZjtCCnWUacx6Rnur2jxOTyuJ2sRXNegc0iXIhxkO1wuMxrGDTCLXjtjcRj82tCg
%2BLrdCQNchebm%2BmToyVG0CKtoWIXSPGlxilqVVa%2BuTHGMhrUn1QRr%2B3Y9saB8ZhYx
%2BfiabYjgVWTG3t7DHsrYo%2BzGdvW1lvJwyvjqGlN0644J4RfZqhmb7X5sOfmhVBWGJ3OZQ1Xbv%2B
%2BivI8fbKgMnq%2BQ93zK1qeu29HWDzEMGoL74mbTX%2FO8kv5NvHxj9L9AQ%3D%3D',
                    u'provider_id': u'da0f5973f42e45b29ecbff6df17174d8',
                    u'endpoint': u'https://persistence.kent.ac.uk/simplesaml/saml2/idp/
SSOService.php',
                    u'protocol': u'saml'
               }
```


Figure 12. A SAML authentication request, returned from the Havana server.

browser is opened, targeting the IdP endpoint specified in the authentication request. The user may now authenticate with the IdP within the browser and a SAML assertion is returned to the client to be sent to Keystone. The resultant un-scoped token is stored in the keyring of the Keystone client or exchanged for a scoped token using the functionality in the federated API.

Full integration of v3 compatibility in the Keystone client was fraught with difficulty. Whilst requests and responses to the server were handled effectively, a fundamental difference in operation between the two Identity APIs prevents full functionality. The scoped-token shown in Appendix 8 is a response from a Havana server running the v3 Identity API. We can clearly see the 'projects' field used to indicate the project to which the token is scoped. The Keystone client only has the capability to process v2 server responses, which make no use of the *projects* keyword. In the v2

API, *projects* were termed *tenants*. In the token shown, the client cannot identify the necessary fields, resulting in the early exit of the program with a 'key error'. Regardless of the client's ability to recognise the API version specified, these errors occur when the v3 scoped token is received by the client and the program returns to normal flow. In order for authentication to complete, a token is converted to an AccessInfo object within the client.py module. It is highly likely that the *key error* occurs at this point. However, the complexity of determining the root cause is considered beyond the scope of the project. Since Havana has already been superseded, it was decided to cease further development and treat the code as a proof of concept.

A criticism of the previous federated functionality was the inability for a user to cancel the process of authentication. Once federated login had been chosen it had to be followed through to completion, regardless of the outcome. As part of this project, a new quit function has been added to the client, allowing the user to abandon the authentication process when prompted to choose an IdP or project.

The enhancements discussed in this section may be viewed on the *v3_ compatible* branch of the GIT repository at *https://github.com/kingBenny/python-keystoneclient.*

5.5 Federation and the Icehouse release

The large differences in the implementation of the Icehouse server necessitate a very different approach to providing federated functionality. A significant difference in the proposed design is the decision to remove support for the v2 Identity API. This was due to a number of reasons. Firstly the deployment of the v3 API was accompanied by an associated change in vernacular; tenants were re-named to projects. As we learnt with the implementation for Grizzly and Havana servers, this change has been reflected deep within the code-base. The use of the *projects* keyword in the server response will cause a miss-match with the callback functions that depend on the *tenants* keyword. Since the migration to *projects* is considered a permanency, back-compatibility may be left to the jurisdiction of the standalone clients.

In addition, the formal adoption of federation as a valid method of authentication negates the need for the middleware components added to previous Keystone servers. In the Icehouse server, there is no processing pipeline to handle HTTP requests nor a modular dispatch system. Hence the 'old' federated library must be refactored to become Icehouse compatible.

The OpenStack client requires the user to set a UNIX environment variable to state the OS_IDENTITY_API_VERSION. Having decided to support v3 servers exclusively, federation requires a specific combination of three parameters; a federated flag, an authentication url and an OS_IDENTITY_API_VERSION set to a value of 3. The presence of these credentials is verified by the authenticate_user() function. This was amended to report errors accordingly. Once present a v3 client could be built, in turn invoking the authenticate functionality in its HTTPClient superclass, calling the federated library.

Figure 13 displays a high-level overview of the new pathway to federated authentication. We may now consider the full pathway to federated authentication as follows;

1	The user issues a command using the OpenStack client. They specify a federated flag, an authentication URL and a command

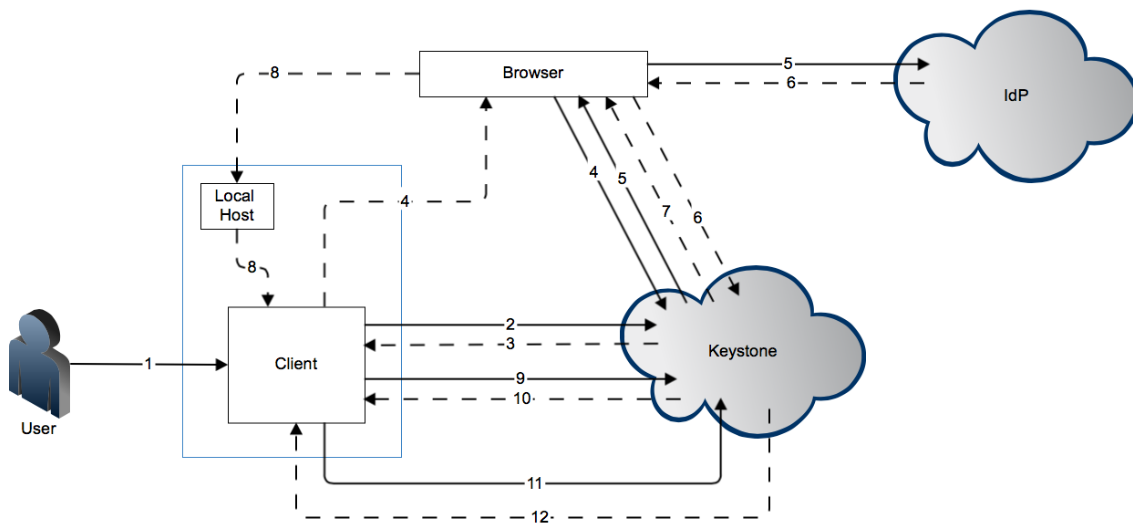


Figure 13. An overview of client-server requests in Icehouse federated authentication.

2	The client checks for the correct combination of parameters and sends an HTTP GET request to the Keystone endpoint.
3	Keystone returns a list of IdPs to the client. The client interrogates each IdP listed and makes a further HTTP GET request to the server to discover the protocols supported. (omitted from the figure)
4	The authentication endpoint is built from the user's selection and the system's default web-browser is used to target the Keystone server
5	Keystone's Apache2 webserver detects that the endpoint specified is protected and initiates an authentication process with the IdP via mod_shib.
6	On successful authentication, a SAML assertion is returned to the Keystone endpoint via the browser. Now the user is authenticated, mod_shib populates the environment with the user's identity attributes and forwards the request to Keystone.
7	Keystone processes the attributes in the environment using a mapping engine. The attributes in the assertion are converted to local attributes such as username or group. An un-scoped token is created from the local user identity details. This is returned to the client via the browser.
8	The browser redirects the token to it's local-host. An HTTP server listens for a response and handles the redirection forwarding the token to the client.
9	In order to be able to select from a list of projects the client makes a HTTP GET request to the Keystone server, this time including the un-scoped token in the request header.
10	Keystone returns a list of projects for the specified user to choose from.
11	The user makes a selection and the project ID is extracted from the response. An HTTP POST request is made, including the un-scoped token and project data.
12	Keystone returns a scoped token to the client.

Since the universal client acts as a wrapper for the standalone clients, adding federated flags to this component ensures they can be used with all of OpenStack's services. As a result, two repositories

have been worked upon. The openstack client for the command-line flags and the keystone client to integrate the federated API. The Federated API has been completely refactored and uploaded to the Icehouse_compatible branch of the GIT repository; *https://github.com/kingBenny/python-keystoneclient/tree/icehouse_ compatible* and the command-line flags have been added to the *master* branch of *https://github.com/kingBenny/python-openstackclient.* Furthermore, installation instructions for the OpenStack Client have been provided in Appendix 9 - Installation Guide.

A full description of the functionality of the federated library has been documented in Appendix 7. One aspect of the software acceptance criteria requires meaningful error messages to be relayed to the user. To this end, a new Federated_Exception class has been defined to handle all errors relating to federated functionality. This has been defined within the exceptions.py module and imported accordingly.

A surprising consequence of the new implementation is the need to return a scoped-token at all times. Since the Keystone service permits limited functionality with un-scoped tokens, previous versions of the client have no problem authenticating to this extent. The unification brought by the OpenStack client has forced keystone's, apparently unique, behaviour to be ignored. Instead, all authentication functions must return a scoped token in order to operate correctly. This means the user must choose a project as part of the authentication procedure and the federated API has been extended to accommodate this requirement.

The greatest challenge encountered in this phase of development was a persistent HTTP 401 error, returned to the client every time a request was made to any of the federated endpoints. Under scrutiny it was discovered that this relates to a design choice made by the OpenStack foundation. "To communicate with the API, you will need to be authenticated - and the keystone provides multiple options for this." (OpenStack, 2014) Unfortunately this level of protection extends to making requests for a list of IdPs or protocols. In the current configuration, a user must specifically know their IdP endpoint and protocol in order to authenticate. For the sake of this project, the Identity endpoints were amended (by removing a Python decorator) to remove protection, allowing an unauthenticated user to gather a list and make a relevant selection.

6.0 Testing and Validation

6.1 Testing with Tox

Testing was performed using the Tox harness on an installation of Ubuntu 13.04 LTS running as a virtual machine in VirtualBox OSX. In use, Tox is a far from intuitive. The current release (1.7) generates a persistent error message of the form:

tox.ConfigError: ConfigError: substitution key "posargs" not found

This error is documented as a core-infrastructure bug (number #1274135) and may be temporarily resolved by reverting to a previous version (1.6.1) with full compatibility. Future development projects should note that an official resolution is not planned until Tox 1.8 is released. (OpenStack, 2014).

By default, Tox suppresses error messages and must be used in conjunction with the --debug flag in order to display a verbose stack-trace of the errors encountered. Any attempt to include a print statement within the testing script has the undesirable result of halting the current batch of tests and returning a success message without further warning. As a laborious workaround, print statements were routinely added to the source code at the point of error reported by the stack-trace. The client could then be executed in an attempt to ascertain the relevant information needed to fix the bug. The lack of any meaningful interaction with the tests being written made for lengthy and frustrating debugging sessions.

6.2 Validation of the Keystone Client

To gain confidence in the federated API a series of unit tests were written to simulate improper user input and erroneous function parameters. In all cases the new code passed, including tests written for pre-existing functionality. However, whilst the regression tests show the code is robust, the failure of the v2 client to recognise a token in the v3 format is a significant stumbling block. Recognising that the server is returning a valid token can be considered a successful proof of concept. But further testing is redundant, given the incompatibilities discussed and the significant changes in architecture experienced in later versions of OpenStack.

6.3 Validation of the OpenStack Client

The implementation of new functionality of the OpenStack client has been subjected to a series of functional tests. Figure 14 shows the results of a series tests to cover the new functionality and its integration into the existing code base. From the chart, three tests have been flagged with failure:

fail to use v3 in the --os-auth-url - The client fails to fulfil an element of the acceptance criteria and report an error message when the user fails to append 'v3' to the authentication url. A better design would ensure automatic checking; automatically appending the substring or exiting with an error message. This is a simple fix but the error was discovered late in the final phase of development, with insufficient time to complete.

invalid values of environment variables - To facilitate easy extension, the OpenStack client has no upper limit to API version specified in the environment variables. However, this has an unexpected result. Should the user set their OS_IDENTITY_API_VERSION to a non-existent version or a non-numeric value (Not a Number - NaN) the program will exit with a command error. Clearly, this is unrelated to the command issued and parameter checks should be extended to validate the environment variables of the system.

Functional Testing of OpenStack Universal client.

Test Case	Expected Outcome	Actual Outcome	Result	Notes
use ‘v2’ in the —os-auth-url (i.e. contact an API v2 server with the openstack client)	Error reported: “must target a v3 endpoint and set OS_IDENTITY_API_VERSION=3”	Error reported: ERROR: cliff.app Federated authentication has only been, configured to work with the v3 API you must set env[OS_IDENTITY_API_VERSION]=3 and target a v3 Keystone endpoint.	PASS	All user input errors raise a exception of type CommandError. This causes each error to have the prefix “ERROR: cliff.app”
use 'v2' and ‘v3’ in the --os-auth-url	The client will prevent potential API incompatibility failure by rejecting the request to any auth_url with a ‘v2’ substring.	Error reported: ERROR: cliff.app Federated authentication has only been, configured to work with the v3 API you must set env[OS_IDENTITY_API_VERSION]=3 and target a v3 Keystone endpoint.	PASS	The existence of ‘v2’ as a substring in the auth_url could be considered a bug. Consider the scenario of a v2 server that implements the v3 API.
fail to use ‘v3’ in the --os-auth-url	The client reports an error that the incorrect auth_url format has been used	Client returns a blank list of IdPs and protocols followed by a python key error: ERROR: cliff.app 'identity_providers'	FAIL	Do we correct endpoint by adding ‘v3’ to the auth_url or send a message, or both?
fail to set OS_IDENTITY_API_VERSION=3	Error reported: “must set OS_IDENTITY_API_VERSION=3 if you want to use federated authentication”	ERROR: cliff.app If using Federated authentication, you must set env[OS_IDENTITY_API_VERSION]=3	PASS	
set OS_IDENTITY_API_VERSION=5	Error reported: “must set OS_IDENTITY_API_VERSION=3 if you want to use federated authentication”	ERROR: cliff.app Unknown command ['project', 'list']	FAIL	It should support an error message. this mssge is as expected for the designated API.
set OS_IDENTITY_API_VERSION=NaN	Error reported: “must set OS_IDENTITY_API_VERSION=3 if you want to use federated authentication”	ERROR: cliff.app Unknown command ['project', 'list']	FAIL	correct error message for the specified API - the command isn't implemented so it’s not recognised
set SERVICE_TOKEN=password	No change	Normal flow, no change in behaviour	PASS	The SERVICE_TOKEN environment variable is used for the Keystone client. As v3 functionality is invoked by the OpenStack client it is never read.
set OS_TOKEN=password	Error reported: That an endpoint for the token must be supplied	ERROR: cliff.app You must provide a service URL via either --os-url or env[OS_URL]	PASS	Priority will always be given to tokens. This behaviour is preserved across all the openstack services.
use --os-token in command	Error reported: That an endpoint for the token must be supplied	ERROR: cliff.app You must provide a service URL via either --os-url or env[OS_URL]	PASS	see above.

Figure 14. A functional testing chart of the OpenStack client.

Test Case	Expected Outcome	Actual Outcome	Result	Notes
Try to run OpenStack with malformed —os-auth-url openstack -F --os-auth-url http://icehose.sec.cs.kent.ac.uk:5000/v3 project list	Error reported: “Reached the maximum number of retries.”	ERROR: cliff.app HTTPConnectionPool(host='icehose.sec.cs.kent.ac.uk', port=5000): Max retries exceeded with url: /v3/OS-FEDERATION/identity_providers (Caused by <class 'socket.gaierror'>: [Errno -2] Name or service not known)	PASS	Provides a name or service not know error.
Try to run OpenStack without any --os-auth-url	Error reported: "must provide an --os-auth-url if wanting to use federated authentication.”	ERROR: cliff.app If using Federated authentication, you must specify an endpoint with --os-auth-url	PASS	
Use both -F and --federated flags at once	Normal flow	Normal flow, no change in behaviour	PASS	use of multiple federated flags should have no detrimental affect on the parsed args.
Type openstack --help	help will display the amended list of commands to include the federated option.	--federated, -F Login via Federated Authentication		
Use --FEDERATED	Unrecognised command	openstack: error: argument --federated/-F: ignored explicit argument ‘EDERATED'		The first char of the flag is recognised
The Keystone server is offline	Error reported that the server has timed out and unreachable	ERROR: cliff.app HTTPConnectionPool(host='icehouse.sec.cs.kent.ac.uk', port=5000): Max retries exceeded with url: /v3/OS-FEDERATION/identity_providers (Caused by <class 'socket.gaierror'>: [Errno -2] Name or service not known)	PASS	This error was encountered within a different eduroam subnet.
The client is disconnected at IdP request phase	Error reported: timeout	There was no response from the Identity Provideror the request timed out. An error occurred, please try again	PASS	Syntax error in error message. Needs fix.
The client is disconnected at project selection phase	Error reported: maximum retries timeout	ERROR: cliff.app HTTPConnectionPool(host='icehouse.sec.cs.kent.ac.uk', port=5000): Max retries exceeded with url: /v3/auth/tokens (Caused by <class 'socket.gaierror'>: [Errno -2] Name or service not known)	PASS	Could be more descriptive, but there is no way to discern between malformed url and failed connection.
Choose Service from list = -1	Display error message	Display error message: “An error occurred with your selection.”	PASS	
Choose Service from list = “£$%^&*	Display error message	Display error message: “An error occurred with your selection.”	PASS	
Choose Service from list = ben	Display error message	Display error message: “An error occurred with your selection.”	PASS	
Choose Service from list = 1000	Display error message	Display error message: “An error occurred with your selection.”	PASS	
Choose Service from list = 10000000000000000000000000000000000000 00000000000000000000000000000000000000 0000000000000000000000000000	Display error message	Display error message: “An error occurred with your selection.”	PASS	

Figure 14 continued. A functional testing chart of the OpenStack client.

Test Case	Expected Outcome	Actual Outcome	Result	Notes
Quit from Service list and cancel login	Display message	Display user message: “Quitting”	PASS	
Client subjected to 100 run cycle	No errors	Full 100-cycle run with no errors or crashes.	PASS	
Client subjected to 1000 run-cycle	No errors	Full 1000-cycle run with no errors or crashes.	PASS	

Figure 14 continued. A functional testing chart of the OpenStack client.

Unfortunately, the universal client could not be tested in Tox, due to a cryptic failure relating to a *MissingAuthPlugin: Token Required* error. With insufficient time to fully track its source, no further testing was conducted and the code was refactored to fully comply with pep8 syntax standards.

6.4 Benchmarking the Universal client

In order to fulfil the software requirements of the project and demonstrate the stability of the federated library, the universal client was subjected to a series of benchmarks. To achieve this, elements of the program that rely on human interaction had to be removed. Without this fix, the response-delay of the user becomes the rate-determining step and the resultant data adversely affected.

Monkey Patching is a technique of dynamically patching a function at runtime to alter its behaviour. Whilst many communities frown upon the practice (Grimm, 2008) it's flexibility is well-suited to rigorous testing. By redefining functions in the federated API, the user input can be substituted for dummy or default data to allow for fully automated testing. Figure 15 shows the results of a 100-cycle test run on two different wireless-networks: the University of Kent Eduroam network and a home broadband connection. The test system was a 2.66 Core i7 MacBook Pro with 8Gb DDR3 ram running Ubuntu 14.04 LTS as a virtual machine in OSX 10.9.3 (Mavericks).

The data displayed in Figure 15 is intended to consider the likely working environments encountered by a typical user. It is unsurprising that the Home broadband network exhibits inferior performance to an optimised high-volume network. In both environments, the data shows a small upward trend within the standard deviation of 68.749ms and 56.851ms for the Eduroam and home broadband connections respectively.

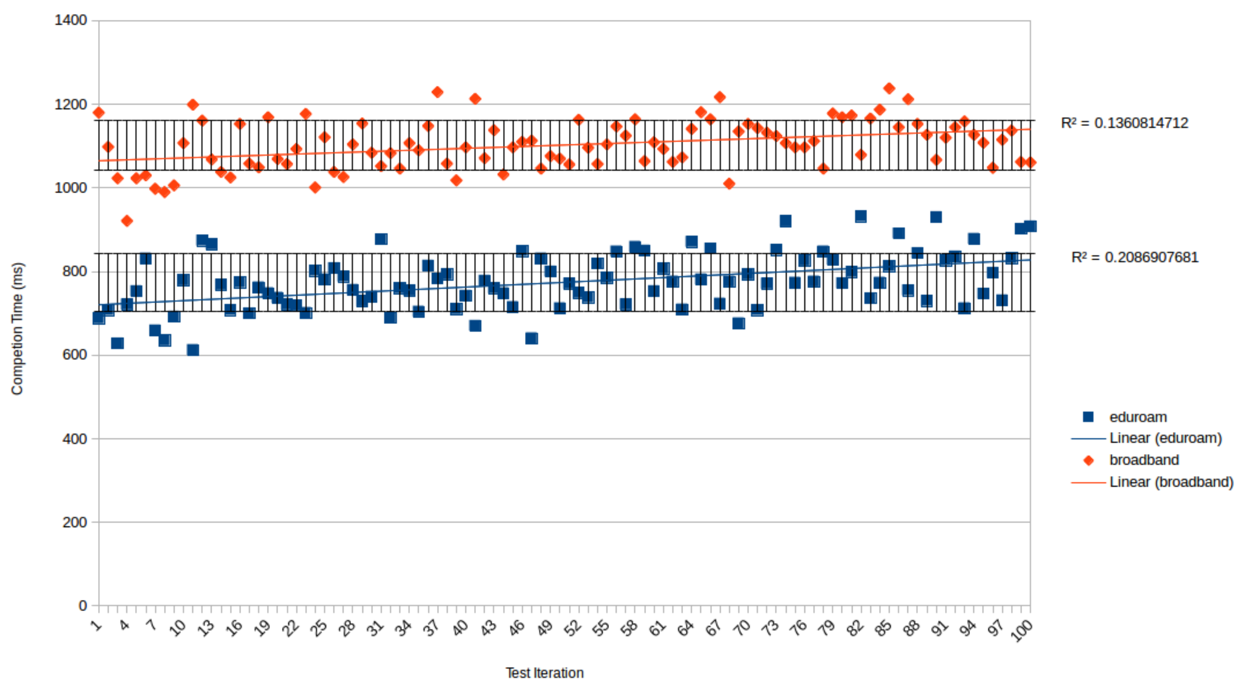


Figure 15. Results of a 100-run benchmark using the OpenStack client (v3 API).

The upward trend shown in the figure is surprising, although it could be attributed to the rapid sequential invocation of the client under testing conditions. With the reliance on the web-browser

in the authentication process, multiple sources could be responsible for this behaviour. Unfortunately, there was insufficient time to determine the cause of this trend and whether this is a consequence of memory leaks.

6.5 Known Bugs

A number of known bugs may be experienced by the user, dependent upon the web browser used. In both implementations of the client, users of Firefox will experience the following error message in the terminal window:

*(process:2862): GLib-CRITICAL **: g_slice_set_config: assertion `sys_page_size == 0' failed*

Various web-searches conclude that this is a much discussed issue, relating to accessibility API features (Bugzilla, 2011). Whilst no fix for the error is presented, there does not appear to be any detrimental effect to the functionality of either of the clients.

Users of Google Chrome will experience multiple errors until the security certificate is trusted and cached. The user must explicitly trust the certificate and restart the client in order to authenticate successfully.

6.6 Acceptance criteria.

With the changes in project requirements, the acceptance criteria created in the early stages of the project have become partially irrelevant (see Appendix 1b). Owing to the problems encountered in the development of the Keystone client, simultaneous v2/v3 compatibility was deprecated to a proof of concept. Unfortunately, this meant that no efforts were made to address the protocol independence of the Keystone client. Casual testing was conducted to ensure correct v2 behaviour with SAML, but there has been no verification using the ABFAB protocol. We thus address all relevant aspects of the acceptance criteria from the perspective of the Icehouse development.

The behaviour of the extended OpenStack client fully complies with sections 1,2,5 and 6, offering the necessary functionality, code quality and performance. Sections 3 e/f/g/i relate to the client's knowledge of the server's state during authentication. The client is unable to discern the difference between an overloaded server, a slow server and an IdP that crashes mid-request. As a result, the client will report a generic timeout message when applicable. Section 4 is largely fulfilled, with the exception of 4c; support for three authentication protocols. The mod_shib/Apache technology used to route authentication requests the Icehouse server limits authentication to SAML. This will be extended in future OpenStack releases.

7.0 Conclusion

The work completed in this project has presented two different approaches to federated authentication in the command-line clients for OpenStack Keystone. Attempts to provide simultaneous v2/v3 compatibility within the Keystone client have faltered, due to the fundamental incompatibility between the token format of the two servers. However, the code demonstrates the functionality of the federated modules in Havana and Grizzly servers, documenting the authentication steps necessary to obtain a scoped token.

The rapid pace of development within the OpenStack community has resulted in a new middleware-free approach to federation in the Icehouse release. The OpenStack client unifies the command-line clients and has been extended to support federation with the SAML SSO protocol. This provides an important step in delivering this functionality to all OpenStack services. As federated identity management becomes increasingly popular, a robust system in OpenStack cements the longevity of the platform and facilitates the benefits of greater interconnectivity.

8.0 Limitations and Future work

A clear limitation of the Keystone client is its lack of compatibility with version 3 of the Identity API. The change from tenants to projects has had far-reaching repercussions regarding the ability of the client to interrogate the token returned by the server. Despite its limited functionality, additional testing of the v2/v3 Keystone client with the ABFAB protocol would be a useful step in the verification of its protocol independence.

The Openstack client has been extended to offer federated authentication with Icehouse servers, running the v3 identity API. With further time for development, it would have been possible to locate the cause of the failed regression tests. Having successfully implemented the functionality in Keystone, it seems logical to extend this project to include libraries for all the OpenStack clients. The federated API is highly modular and facilitates simple integration into the other services with minimal additional coupling.

The new OpenStack client has only been tested with a single Icehouse server. Whilst other Keystone servers and IdPs could be configured, this is a lengthy and complex process considered beyond the scope of the project. Furthermore, the only protocol tested with the new code was SAML, as this was the only standard configured on the Icehouse server. Whilst this may appear an oversight, the mod-shib/Apache technology used for Icehouse federation will only support SAML at this time (OpenStack, 2014).

A clear shortcoming of this work is the persistent need for a hybrid authentication environment. Having to change focus from the command-line to a browser window is an inelegant solution that limits users to those who possess a GUI. It is an implicit requirement of a SAML-web implementation to manage redirection requests and the transfer of assertions with a browser and cookie. In recognition of this, work is planned to incorporate a variant of SAML SSO, called the Enhanced Client Proxy (ECP) (Denis, 2014). ECP is a SAML profile that negates the need for redirections or a cookie for persistence. As such, browser authentication stages can be omitted and full authentication may be carried out on the command line.

When choosing a v3 server with the OpenStack client, a user must append the authentication url with a '/v3' substring. This additional information seems erroneous, since it mirrors the value held in the OS_AUTHENTICATION_API_VERSION environment variable. The process could be simplified to automatically concatenate the '/v3' string to the end of the --os-auth-url only when required.

Choosing to protect the federated endpoints is of questionable worth for a user-friendly system. Having to enter the specific URL for an authentication request within an *openstack* command would be a laborious and error-prone activity. Instead a user should be able to pick from a list, as shown in this work. A possible reason for this design may be a desire for maximal security. Protecting the API makes Keystone less susceptible to Distributed Denial Of Service (DDoS) attacks that overload the server with requests attempting to render it useless. However, the protection of specific endpoints should be a decision for a system administrator and not require changes to be 'forced' at a code level. Whether this behaviour is by design or oversight is debatable and should be reported to the Keystone project for further discussion amongst its development community.

9.0 References

Ahn, G.J. & Ko, M. (2007). User-centric privacy management for federated identity management. Collaborative Computing: Networking, Applications and Work-sharing, CollaborateCom 2007. International Conference on. pp.187 - 195.

Amazon (2014) [Online]. About AMS. http://aws.amazon.com/about-aws/ [Accessed on 8/07/2014].

Armbrust, M. et al. (2010). Comm's of the ACM. A View of Cloud Computing. 53(4). pp. 50-58.

Baset, S.A. (2012). Open source cloud technologies. In Proceedings of the Third ACM Symposium on Cloud Computing (SoCC '12).

Behl, A. & Behl, K. (2012). An analysis of cloud computing security issues. Information and Communication Technologies (WICT), 2012 World Congress on. pp.109-114.

Boronine, A. (2014). [Online]. Tox-Driven Python Development. http://www.boronine.com/2012/11/15/Tox-Driven-Python-Development/ [Accessed 20/07/2014].

Bugzilla. (2011). [Online]. Bug 672671 - "GLib-CRITICAL **: g_slice_set_config: assertion `sys_page_size == 0' failed" on startup, for version 7 and earlier. https://bugzilla.mozilla.org/show_bug.cgi?id=672671. [Accessed 4/09/2014].

Cantor et al. (2005). [Online]. Shibboleth Architecture: Protocols and Profiles. http://shibboleth.internet2.edu/shibboleth-documents.html. [Accessed, 13/07/2014].

Chadwick, D. W., et al. (2013). Adding Federated Identity Management to OpenStack. Journal of Grid Computing. [Online] accessed http://dx.doi.org/10.1007/s10723-013-9283-2

Cloud, Amazon Elastic Compute (2011). Amazon web services. [Online]. http://dclug.tux.org/200611/AmazonEC2.pdf [Accessed 08/07/2014].

Coram, M. & Bohner, S. (2005). "The impact of agile methods on software project management," Engineering of Computer-Based Systems. ECBS '05. 12th IEEE International Conference and Workshops on the. pp.363-370.

Denis, M. (2014). [Online]. Python client library for Keystone: Add authentication plugins for keystoneclient authentication. https://blueprints.launchpad.net/python-keystoneclient/+spec/add-saml2-cli-authentication. [Accessed 01/09/2014].

Fengming, N., Feng, X. & Rongzhi, Q. (2012). "SAML-based single sign-on for legacy system," Automation and Logistics (ICAL), IEEE International Conference on , pp. 470-473, 15-17 Aug.

Fruiapps. (2012). [Online]. An Introductory tutorial to python virtualenv and virtualenvwrapper. http://blog.fruiapps.com/2012/06/An-introductory-tutorial-to-python-virtualenv-and-virtualenvwrapper. [Accessed 16/07/2014].

Grimm, A. (2013). [Online]. Monkey Patching is destroying Ruby. http://devblog.avdi.org/2008/02/23/why-monkeypatching-is-destroying-ruby/. [Accessed 20/08/2014].

Jensen, J. (2012). "Federated Identity Management Challenges," Availability, Reliability and Security (ARES), 2012 Seventh International Conference on, pp.230-235.

Jensen, J.& Jaatun, M.G. (2013). "Federated Identity Management - We Built It; Why Won't They Come?," Security & Privacy, IEEE. 11(2). pp.34-41.

Jericho Systems. (2014). [Online]. ABAC (Attribute-Based Access Control). http://www.jerichosystems.com/technology/glossaryterms/attribute_based_access_control.html. [Accessed 15/08/14]

Jin. X. (2013). [Online]. OpenStack Identity (keystone): Attribute Based Access Control. https://blueprints.launchpad.net/keystone/+spec/attribute-based-access-control.

Jones, C. (2010). Software Engineering Best Practices: Lessons from Successful Projects in the Top Companies. Mc Graw Hill publishing. pp. 25.

Juniper. (2013). [Online]. Secure Access Service SAML 2.0 Supported Features Reference. http://www.juniper.net/techpubs/en_US/sa8.0/topics/reference/general/secure-access-saml-sso-support-reference.html. [Accessed 10/07/2014]

Kaufman, L.M. (2009). "Data Security in the World of Cloud Computing," Security & Privacy, IEEE , 7(4), pp.61-64.

Kerner, S.M. (2013). [Online]. How Open Source Python Drives the OpenStack Cloud. http://www.datamation.com/cloud-computing/how-open-source-python-drives-the-openstack-cloud-video.html. [Accessed 16/08/14]

Ko, M.N., et al. (2010). "Social-networks connect services." *Computer* 43(8). pp. 37-43.

Landau, S. & Moore, T. (2011). "Economic Tussles in Federated Identity Management," Proc. 10th Workshop Economics of Information Security (WEIS 11). [Online]. http://weis2011.econinfosec.org/papers/Economic%20Tussles%20in%20Federated%20Identity%20Management.pdf [Accessed 13/07/2014].

Leavitt, N. (2009). "Is Cloud Computing Really Ready for Prime Time?," *Computer* , 42(1), pp. 15-20.

Long. Y. et al. (2010). "Attribute mapping for cross-domain access control," Computer and Information Application (ICCIA), 2010 International Conference on. pp. 343-347.

Lori, M. (2009). Data security in the world of cloud computing. Co-published by the IEEE Computer And reliability Societies, pp. 61-64.

Matthews, D. (2010). [Online]. How Kerberos Works. http://mccltd.net/blog/?p=1053 [Accessed 09/07/2014].

Mell, P. & Grance, T. (2011). [Online]. The NIST Definition of Cloud Computing: Recommendations of the National Institute of Standards and Technology. Special Publication 800-145. [Acessed on 07/07/2014]. http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

Meyer, Bertrand (1988). Object-Oriented Software Construction. Prentice Hall. ISBN 0-13-629049-3.

Mirantis. (2014). [Online]. Mirantis and IBM Set New OpenStack Benchmark, Standing Up 75,000 Live VMs in Multi-Datacenter Cloud. http://www.mirantis.com/company/press-center/in-the-media/mirantis-ibm-set-new-openstack-benchmark-standing-75000-live-vms-multi-datacenter-cloud/ [Accessed 08/07/2014].

Morris, R. & Thompson, K. (1979). "Password security: a case history," Commun. ACM, 22(11). pp. 594–597.

Neuman, B.C. & Ts'o, T. (1994). "Kerberos: an authentication service for computer networks," *Communications Magazine, IEEE*, 32(9). pp. 33-38.

Oasis. (2014). [Online]. Security Assertion Markup Language (SAML) V2.0 Technical Overview. https://www.oasis-open.org/committees/download.php/27819/sstc-saml-tech-overview-2.0-cd-02.pdf. [Accessed 13/03/2014].

Open source software for building private and public clouds. (2014). [Online]. https://www.openstack.org [Accessed on 2/07/2014]

OpensStack. (2014). [Online]. Configuring Keystone for federation. http://docs.openstack.org/developer/keystone/configure_federation.html. [Accessed 01/09/2014].

OpenStack. (2014). [Online]. Configuring Services to work with Keystone. http://docs.openstack.org/developer/keystone/configuringservices.html. [Accessed 14/08/2014]

OpenStack. (2014). [Online]. Keystone. https://wiki.openstack.org/wiki/Keystone[Accessed 14/08/2014].

OpenStack. (2014). [Online]. Keystone command line utility. http://docs.openstack.org/developer/python-keystoneclient/man/keystone.html [Accessed 13/08/2014].

OpenStack. (2014). [Online]. Welcome to Keystone, the OpenStack Identity Service! http://docs.openstack.org/developer/keystone/. [Accessed 13/08/2014].

OpenStack. (2014). [Online]. Infrastructure-wide issues with tox 1.7.0. Bug report #1274135. https://bugs.launchpad.net/openstack-ci/+bug/1274135. [Accessed 30/07/2014].

Pip. (2014). [Online]. Pip. http://pip.readthedocs.org/en/latest/installing.html [Accessed 14/08/2014].

Python. (2014). [Online]. General Python FAQ. https://docs.python.org/2.7/faq/general.html#what-is-python [Accessed 16/08/14].

Pallis, G. (2010). Cloud Computing: The New Frontier of Internet Computing, *Internet Computing, IEEE*, 14(5), pp.70-73.

RedHat. (2014). [Online]. Architecture. https://access.redhat.com/documentation/en-US/Red_Hat_ Enterprise_Linux_OpenStack_Platform/2/html/Getting_Started_Guide/ch01.html. [Accessed on 07/07/2014].

Rhoton, J. (2013). [Online]. Discover OpenStack: The Identity component Keystone Available at: http://www.ibm.com/developerworks/cloud/library/cl-openstack-keystone/index.html?ca=dat- [Accessed 08/07/2014].

Shim, S.S.Y., Bhalla, G. & Pendyala, V. (2005). "Federated identity management," *Computer.* 38(12), pp.120-122.

Siu, K. (n.d.). A more robust design for a federated authentication plugin. School of Computing, University of Kent. Design document.

Slideshare. (2012). [Online]. OpenStack keystone identity service. http://www.slideshare.net/openstackindia/openstack-keystone-identity-service. [Accessed 13/08/2014].

Smith, D. (2008). The challenge of federated identity management [Online], Network Security, 2008(4). pp. 7-9. http://dx.doi.org/10.1016/S1353-4858(08)70051-5.

TokenGuard. (2014). [Online]. http://www.tokenguard.com/RSA-SecurID-SID700.asp. [Accessed 08/07/2014].

Toronto University. (2014). Single Sign-On and Shibboleth. [Online]. http://www.utoronto.ca/security/projects/shibboleth.htm. [Accessed on 13/07/2014].

Tox. (2013). [Online]. Welcome to the Tox automation project. http://tox.testrun.org. [Accessed 17/08/2014].

TRAC. (2013). [Online]. Welcome to the Trac Open Source Project. http://trac.edgewall.org [Accessed on 7/07/2014].

Wikipedia (2014) TimeBoxing. [Online]. http://en.wikipedia.org/wiki/Timeboxing [Accessed on 7/07/2014].

Weiss, A. (2007). Computing In The Clouds. Networker, 11(4).

Xiaolong W. et al. (2012). "Comparison of open-source cloud management platforms: OpenStack and OpenNebula," Fuzzy Systems and Knowledge Discovery (FSKD), 2012 9th International Conference on. pp.2457-2461.

Xuguang R. & Xin-Wen W. (2012). "A novel dynamic user authentication scheme," Communications and Information Technologies (ISCIT), 2012 International Symposium on. pp. 713-717.

Yuan, E. & Tong, J. (2005). "Attributed based access control (ABAC) for Web services," Web Services, 2005. ICWS 2005. Proceedings. 2005 IEEE International Conference on. pp.569.

Zhu, B., Fan, X. & Gong, G. (2014). "Loxin — A solution to password-less universal login," Computer Comm's Workshops (INFOCOM WKSHPS), 2014 IEEE Conference pp.488-493.

Ziqing M., Florencio, D., Herley, C. (2011)."Painless migration from passwords to two factor authentication," Information Forensics and Security (WIFS), 2011 IEEE International Workshop on. pp.1-6.

Appendix 1a - Software Requirements.

1	Add a new authentication option to the OpenStack Keystone module facilitating Federated login via an external identity provider.
2	Maintain the convention of the Keystone client, by offering two flag options; a single char (e.g. -F) and a descriptive word (e.g. - -federated), to maintain consistency of the interface.
3	The new functionality should be robust, to handle user errors such as: • incomplete commands, • malformed URLs, • server errors and • the inability to connect to a server. In these instances, the system will return an intelligible error message to the user.
4	The change must be sufficiently robust to manage its own exceptions and take the necessary corrective behaviour (recovery).
5	The code should be easily extensible to account for new federation authentication protocols.
6	The code should be well-documented and easy to maintain
7	The performance of the federated Keystone login should be comparable to that of other federated login systems
8	The system should function correctly under the most popular operating system for OpenStack installations (to be determined).
9	OPTIONAL REQUIREMENT: The system should work effectively on the second most popular operating system for OpenStack

Appendix 1b - Software Acceptance Criteria

1.0	a	The user can choose a federated option to log in via an external identity provider (IdP)
	b	The user may choose to log in as usual (non-federated) without any change to functionality
	c	The user can initiate help and see the federated login as an option with a short help message.
2.0		The user can choose any one of the federated option flags (short or long) with identical behaviour.
3.0		The system will be able to handle the following errors, reporting an intelligible error message (where appropriate) when...
	a	The user doesn’t enter the correct federated option (malformed option flag)
	b	The user doesn’t specify the correct end-point
	c	The user doesn't specify the correct port number for the OpenStack installation (not 5000)
	d	The user uses the wrong API version with the wrong flag, i.e. a long option with a V1.0 API and a short option with a V2.0 API
	e	The IdP server is down/unreachable
	f	The IdP crashes whilst processing a request
	g	The IdP is overloaded and/or slow to respond
	h	User provides incorrect federated credentials
	i	Keystone crashes during the federated log-in procedure
	j	The Keystone client must be able to handle a non-protocol message
4.0	a	The user is given a list of Identity Providers (IdP’s) to choose from.
	b	Protocols are loaded via a modular structure, facilitating easy addition or removal.
	c	The user experience should be the same for the three currently implemented federation protocols.
5.0		The code has been inspected by a systems engineer and deemed well-documented.
6.0		The execution time has been measured over a 100-run cycle to determine the standard deviation and average time of federated login and deemed acceptable by the project supervisor.

Appendix 2 - Analysis of the Swift Client and Federated API.

The 'swift' command launches the client via the formal entry point, swift.bin.

swift.bin

The client makes use of the OptionParser library to parse the arguments provided by the user. Each option is registered with the parser at initialisation via a series of add_option() statements. Once the parser has parsed the arguments an *options* object is returned, containing the values of the user input and any further values that derive from UNIX environment variables.

parse_args()

parse_args() is a helper function that ensures the correct credentials have been supplied. If a user chooses federated authentication, this function tests for the correct combination of arguments required to authenticate. Specifically, this must be an OS_AUTH_URL Linux environment variable set or overridden with --os_auth_url.

To login via regular authentication we must provide an OS_AUTH_URL, OS_USERNAME, OS_PASSWORD and OS_TENANT_NAME via Linux environment variables. These may be overridden (or set) with --os_auth_url, --os_username, --os_password, and/or --os_tenant_name flags at the command line. Additionally, a SERVICE_TOKEN may hold a special token, used for administration duties. By providing an administration token we may by-pass other forms of username/password authentication and specify the --os_auth_url alone.

get_conn()

Returns a new connection object based on the command line options provided by the user. Of particular interest is the auth_version variable that will have been set to 'F' by the previous call to parse_args().

swiftclient.client.py

The client.py module contains the class declaration for the Connection class. A connection object establishes connections and defines their retry behaviour. Each Swift command entered on the command line invokes _retry() which creates an HTTPConnection or an HTTPSConnection from a call to the HTTP_Connection() function. Importantly, _retry() invokes get_auth() which is responsible for choosing the correct API version and authentication method based on the auth_version passed to the Connection object when created. At this point that the code can branch to facilitate federated authentication via _get_auth_federated()

_get_auth_federated()

This function acts as the gateway to the Federated API developed by Kent. If the user already possesses a federated_endpoint and federated_token_id they are already authenticated and these details are returned. However, if the user has no federated_token_id the url, realm and tenant_name are passed to federatedAuthentication() within the federated API module.

contrib.federated.federated.py

The federated module comprises methods to handle security tokens, authentication protocols and requests to identity providers (IdPs). It relies on a number of helper functions from the federated_utils.py library

federatedAuthentication()

This function is the the super-function that calls various API methods in order to return a scoped token. It requires a keystone URL, IdP realm and a tenant name. Failure to supply a recognised realm results in a call to the helper function selectRealm(). This function prompts the user to choose an Identity Provider (IdP) from a list provided. The choices are populated

from a query to the Keystone Endpoint via a middleware request. The user may now make a choice from the options provided (shown below).

```
➜ bin  ./swift -F -A http://fedkeystone.sec.cs.kent.ac.uk:5000/v2.0 list
("'ben' flag has been chosen with the value ", 'http://fedkeystonev3.sec.ca.kent.ac.uk:35357/v2.0')
("'ben' flag has been chosen with the value ", 'http://fedkeystonev3.sec.ca.kent.ac.uk:35357/v2.0')
Please use one of the following services to authenticate you:
        { 0 }  Moonshot
        { 1 }  Kent Keystone Identity Server
        { 2 }  Kent Proxy Identity Service
Enter the number corresponding to the service you want to use: 1

Initiating Authentication against: Kent Keystone Identity Server
```

The relevant protocol is loaded dynamically by the load_protocol_module() function, based on the choice of the user. Here, a protocol is represented in the source code by a file of the same name. It is suggested that this could be improved to utilise an associative mapping between keyword and protocol module using the Python dictionary structure. This would allow for a more complex protocol directory structure in the future. Once chosen, the user is prompted for their login credentials...

```
Initiating Authentication against: Kent Keystone Identity Server

Please enter your username: Guest
Password:
Successfully Logged In
```

getTenantOrDomain() then returns a list of tenants and domains the user has permission to access...

```
You have access to the following tenant(s) and domain(s)on Kent Keystone Identity Server:
        { 0 }  A project in the V2 Keystone
Enter the number corresponding to the tenant you want to use: 0

Successfully Authorised to access: A project in the V2 Keystone

You have access to the following tenant(s)and domain(s):
Enter the number corresponding to the tenant you want to use:
```

Appendix 3 - UML Module diagram of Swift Client.

For a legible version of this diagram, please view online, at
https://projects.cs.kent.ac.uk/projects/co880_os_keystone/trac/attachment/wiki/WikiStart/Swift%20Client%20-10.svg

Appendix 4 - Analysis of the Keystone Client

The shell.py module represents the formal entry point to the Keystone client and creates a new OpenStackIdentityShell object on initialisation. There are five key functions responsible for processing commands and handling the subsequent authentication.

main()
Main is the first call after initialisation, it creates a command parser to handle the command-line input, determines the API version and builds permissible sub-commands, based upon API version. If the user fails to input any arguments (or chooses help) the help function is invoked accordingly. The callback function that performs the required Keystone action is mapped to the arguments, this is called later.

The user's os_token and os_endpoint are determined from the UNIX environment variables, although these can be passed at the command line via the --os-token and the --os-endpoint options. If the user has not authenticated, a generic client may be created to perform rudimentary tasks, such as listing the server version. Otherwise a v2 client is instantiated to communicate with the server. Note that a v3 client is never created in the existing version, despite capability within the source code. This will be handled by future implementations of the OpenStack universal client, which will act as a wrapper, using the functionality of this client as a library.

get_base_parser(self)
This method is responsible for registering the command-line options with the parser. To add a -F or --federated option will require the addition of another add_argument() call with a corresponding action, default value and help string. When the parser is instructed to parse its arguments, the results are stored in an *options* variable. The federated choice will be stored as a boolean variable deriving from the parser's action=store_true attribute.

get_subcommand_parser(self, version)
The subcommand parser handles the Keystone command words based on the API version previously determined. Currently, however, it is only the v2.0 module that is consulted for its Keystone commands. Five modules are consulted and their commands are added to the sub-parser via the _find_actions() method call. These commands become the Keystone commands that the user can invoke.

_find_actions(self, subparsers, actions_module)
Each actions_module is probed for its member-methods that begin with a 'do_' prefix. This is subsequently removed and the remaining method name becomes a Keystone command with standardised-hyphen-form to separate multiple word commands (e.g. do_user_update() becomes user-update). The method is extracted and added to the parser in the form of the command, its help description and arguments. Finally the command is mapped to its function as a call-back which will be invoked by default, every time the parser recognises the command. Changes are not necessary for this method, as federated authentication should not make any changes to the functional behaviour of Keystone.

auth_check(args)

auth_check() is a helper method that checks to see that the user has provided the necessary combination of arguments required for authentication. The user is required to specify a token and an endpoint, the omission of either will raise a CommandError(). Should the user successfully provide both details, the inclusion of username, password and/or authentication URL will be ignored and the user notified. If no token or endpoint are provided, the requirements for authentication credentials change. We therefore need an authentication URL, username and password. Should the user fail to provide all three, a CommandError() is raised and an error message printed. If the user is on a UNIX OS (with a tty terminal), they may be prompted to input their password at this stage. auth_check() will need to be amended in order to accommodate the necessary parameter checks for federated authentication.

v2_0.client.py

If the user attempts to send a command to the Keystone server, already possessing a token, authentication is by-passed and a v2_0.client is created to manage the request. The creation of this Client object invokes the initialiser in the HTTPClient super-class. In turn this creates a Session object that handles building the requests to the server.

get_raw_token_from_identity_service()

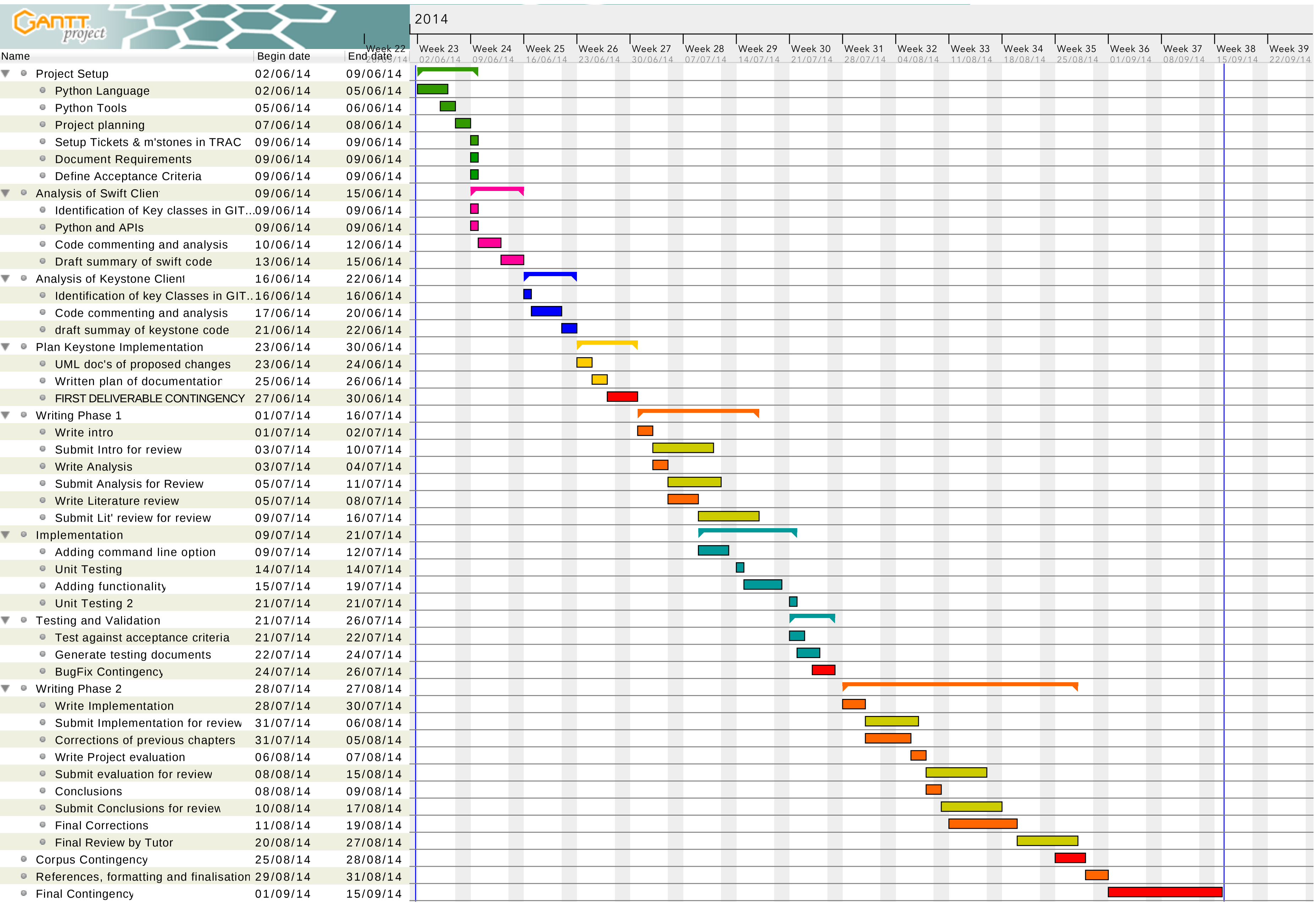
This method authenticates the users credentials using v2 of the identity API. If a token is present it is chosen over username and password credentials. However, the client is blind to the validity of the token and will accept any token as a valid parameter. If Keystone does not validate the existing token or the credentials are incorrect an AuthorizationFailure is raised.

Code Errors

The following code errors have been logged as trivial bug-fixes in TRAC:

shell.py	Line 369 declares a class-variable parser that is initialised but never referenced.
	Line 367 is duplicated at line 415.

Appendix 5 - A Gantt planning chart for the project.



Appendix 6 - Small changes to the Swift client.

To demonstrate understanding of the way in which the client parses arguments, a test option has been added to the command line options (Figure 1).

```
1080        parser.add_option('--os_auth_url', dest='os_auth_url',
1081                          default=environ.get('OS_AUTH_URL'),
1082                          help='Openstack auth URL. Defaults to env[OS_AUTH_URL].')
1083        parser.add_option('--os_username', dest='os_username',
1084                          default=environ.get('OS_USERNAME'),
1085                          help='Openstack username. Defaults to env[OS_USERNAME].')
1086        parser.add_option('-T', '--os_tenant_name', dest='os_tenant_name',
1087                          default=environ.get('OS_TENANT_NAME'),
1088                          help='Openstack tenant name.' \
1089                               'Defaults to env[OS_TENANT_NAME].')
1090        parser.add_option('--os_password', dest='os_password',
1091                          default=environ.get('OS_PASSWORD'),
1092                          help='Openstack password. Defaults to env[OS_PASSWORD].')
1093        parser.add_option('-B', '--b-test', dest='ben_test',
1094                          default=environ.get('BEN_TEST'),
1095                          help='A Test option for new command. Default taken from env[BEN_TEST]')
1096        parser.disable_interspersed_args()
```

Figure 1. A new '-B' option.

When the user fails to provide arguments to the Swift client, or makes an input error, a help message is displayed to the user, including a description of the new -B command (Figure 2.).

```
Example:
  swift -A https://auth.api.rackspacecloud.com/v1.0 -U user -K key stat

Options:
  --version             show program's version number and exit
  -h, --help            show this help message and exit
  -s, --snet            Use SERVICENET internal network
  -v, --verbose         Print more info
  -q, --quiet           Suppress status output
  -A AUTH, --auth=AUTH  URL for obtaining an auth token
  -V AUTH_VERSION, --auth-version=AUTH_VERSION
                        Specify a version for authentication(default: 1.0)
  -U USER, --user=USER  User name for obtaining an auth token
  -K KEY, --key=KEY     Key for obtaining an auth token
  -F, --federated       Enable the federated authentication.
  -R REALM, --realm=REALM
                        Realm to use for the federated authentication
  --os_auth_url=OS_AUTH_URL
                        Openstack auth URL. Defaults to env[OS_AUTH_URL].
  --os_username=OS_USERNAME
                        Openstack username. Defaults to env[OS_USERNAME].
  -T OS_TENANT_NAME, --os_tenant_name=OS_TENANT_NAME
                        Openstack tenant name.Defaults to env[OS_TENANT_NAME].
  --os_password=OS_PASSWORD
                        Openstack password. Defaults to env[OS_PASSWORD].
  -B BEN_TEST, --b-test=BEN_TEST
                        A Test option for new command. Default taken from
                        env[BEN_TEST]
➜ bin
```

Figure 2. The amended Swift client help screen.

The user may now use the command with a parameter which is stored within the client.
In keeping with the existing commands, three options are possible;

1. The user chooses to use the '-B' flag

```
➜ bin  ./swift -B this_is_a_test
("'ben' flag has been chosen with the value ", 'this_is_a_test')
Usage: swift <command> [options] [args]
```

2. The user chooses the verbose version '--ben-test'

```
➜ bin  ./swift --ben_test hello_world
("'ben' flag has been chosen with the value ", 'hello_world')
Usage: swift <command> [options] [args]
```

3. The user does not make an explicit use of the flag. In this case the value is derived from the environment variables of the system. Note, the *-F* flag has been used to escape from the default help screen that's displayed when a user fails to input any arguments.

```
➜ bin  ./swift -F
("'ben' flag has been chosen with the value ", 'http://fedkeystonev3.sec.ca.kent.ac.uk:35357/v2.0')
Usage: swift <command> [options] [args]
```

If no environment variable has been set, no value is derived and an error should be raised if required for successful operation.

Appendix 7 - Documentation of the Federated API.

The federated API consists of 5 key functions. The entry point of the library is via the federatedAuthentication() super-function.

federatedAuthentication(keystone_endpoint, realm=None, scoped=True)

This is the main function responsible for invoking the necessary functions in the library. The function first calls get_IdP_List() to populate a list of IdPs for the user to choose from. They are prompted to choose from the list displayed, via the select_IdP_and_protocol() function. The authentication endpoint is appended with the corresponding values and get_unscoped_token() is invoked. The token is exchanged for a scoped token with a second request to Keystone with the unscoped token in the header of the HTTP GET request.

Parameters:
- keystone_endpoint - The url of the keystone server
- realm=None - The name of the IdP (default None)
- scoped=True - Indicates the need for a scoped/un-scoped token. Note, in the case of the OpenStack client we must always return a scoped-token. (default True)

Returns:
- *A* scoped token
- The body of the server response, without the header

get_IdP_List(keystone_endpoint)

Makes a HTTP GET request to the specified endpoint using the Python *requests* library. Appends */OS-FEDERATION/identity_providers* to the *keystone_endpoint* provided.

Parameters:
- keystone_endpoint - The url of the keystone server

Returns:
- A list of list of IdPs in JSON format.

get_protocol_List(keystone_endpoint, realm)

Gets a list of protocols for a specified IdP endpoint using the Python *requests* library. Extracts the IdP endpoint from the realm[‘links’][‘protocols’] field of the Python. dictionary.

Parameters:
- keystone_endpoint - The url of the keystone server
- realm - The the name of the IdP.

select_IdP_and_protocol(keystone_endpoint, identity_providers)

Allows the user to choose an IdP and protocol from the list displayed onscreen. The function maps the protocols to each IdP and stores each combination in a list of key-value pairs. The user may quit the client at this stage, with the options provided onscreen.

Parameters:
- keystone_endpoint - The url of the keystone server

Returns:

- a selected IdP and Protocol ID.

Raises:

- FederatedException - if there are no available IdPs at the authentication URL specified.
- FederatedException - if there are no available protocols for the IdP selected.

get_unscoped_token(authentication_endpoint)

Loads a security certificate and creates a localhost HTTP server. Opens a browser with the authentication endpoint and a query string appendage - redirecting to the local host (port 8080). A RequestHandler class is defined locally, to override the do_GET and do_POST messages in its BaseHTTPRequestHandler superclass. The IdP response is captured and returned.

Parameters:

- authentication_endpoint - the newly constructed endpoint based on the authentication URL, user's choice of IdP and protocol.

Returns:

- an un-scoped token

get_scoped_token(keystone_endpoint, unscoped_token, selected_protocol)

Makes an HTTP GET request to the *keystone_endpoint* with the *unscoped_token* in the *X-Auth_Token* header. A list of projects is returned by calling the select_project() function in the futils library. The user chooses a project and the selection with the unscoped token is sent with a POST message to keystone.

Parameters:

- keystone_endpoint - The url of the keystone server
- unscoped_token - the token returned from previous authentication steps
- selected_protocol - the name of the selected protocol, chosen by the user.

Returns:

- a scoped token.

Appendix 8 - A Scoped Token

{"token": {"methods": ["saml2"], "roles": [{"id": "975228c312fa4e2d8adf6475f5dd5cb7", "name": "member"}], "expires_at": "2014-08-08T14:53:55.372224Z", "project": {"domain": {"id": "default", "name": "Default"}, "id": "065e5a706ee24a9dbb15d2d0b869aa4a", "name": "myProject"}, "catalog": [{"endpoints": [{"url": "http://192.168.1.46:8000/v1", "region": "RegionOne", "interface": "admin", "id": "5b5e7cfb29bc418d8a3559b1906b509e"}, {"url": "http://192.168.1.46:8000/v1", "region": "RegionOne", "interface": "internal", "id": "851b0f8e654f4984bda9c52a3fc10a82"}, {"url": "http://192.168.1.46:8000/v1", "region": "RegionOne", "interface": "public", "id": "97182bbb317a4bf1b43993227cca75d4"}], "type": "cloudformation", "id": "0d2d2dbd9a074cce9f5bc3e979cfd738", "name": "heat-cfn"}, {"endpoints": [{"url": "http://192.168.1.46:8774/v3", "region": "RegionOne", "interface": "public", "id": "09fd435287134479ab02972eb28a713c"}, {"url": "http://192.168.1.46:8774/v3", "region": "RegionOne", "interface": "internal", "id": "4106cbd7820045719dd61938738b54bf"}, {"url": "http://192.168.1.46:8774/v3", "region": "RegionOne", "interface": "admin", "id": "7bcee4d802224a56a6c79fe617923b07"}], "type": "computev3", "id": "1120591e1d494f0087c627cbacdf8a72", "name": "novav3"}, {"endpoints": [{"url": "http://192.168.1.46:8004/v1/065e5a706ee24a9dbb15d2d0b869aa4a", "region": "RegionOne", "interface": "internal", "id": "26ea104dcb39420185f6bc906e7d4c5c"}, {"url": "http://192.168.1.46:8004/v1/065e5a706ee24a9dbb15d2d0b869aa4a", "region": "RegionOne", "interface": "admin", "id": "84c3d33218e64325bee92363ca190801"}, {"url": "http://192.168.1.46:8004/v1/065e5a706ee24a9dbb15d2d0b869aa4a", "region": "RegionOne", "interface": "public", "id": "a2ac3f92880747a993c3d8db9e4847a9"}], "type": "orchestration", "id": "1549412ceb3d49f989efd150ca4b9b7e", "name": "heat"}, {"endpoints": [{"url": "http://192.168.1.46:8773/services/Admin", "region": "RegionOne", "interface": "admin", "id": "993a249cba3b4043a78dccdb67c4f107"}, {"url": "http://192.168.1.46:8773/services/Cloud", "region": "RegionOne", "interface": "public", "id": "a826b2dd9b6040178e435503a7516384"}, {"url": "http://192.168.1.46:8773/services/Cloud", "region": "RegionOne", "interface": "internal", "id": "fa96e3771ab84e4d9ccace13819b7b66"}], "type": "ec2", "id": "29f5b6af354c49f3a15a6be6fccf1bbe", "name": "ec2"}, {"endpoints": [{"url": "http://192.168.1.46:8776/v1/065e5a706ee24a9dbb15d2d0b869aa4a", "region": "RegionOne", "interface": "admin", "id": "0c205dfa8b0c4d7e86e2cfa60ea787bd"}, {"url": "http://192.168.1.46:8776/v1/065e5a706ee24a9dbb15d2d0b869aa4a", "region": "RegionOne", "interface": "public", "id": "5bebd1d1740543ccb0480c4fb8233d58"}, {"url": "http://192.168.1.46:8776/v1/065e5a706ee24a9dbb15d2d0b869aa4a", "region": "RegionOne", "interface": "internal", "id": "7f6bf298588343649e0a940d46edcec6"}], "type": "volume", "id": "8b1abb03520c4da4a9862b5f5b759bbb", "name": "cinder"}, {"endpoints": [{"url": "http://192.168.1.46:9292", "region": "RegionOne", "interface": "admin", "id": "591b51f4250f47b4819c6763156cb953"}, {"url": "http://192.168.1.46:9292", "region": "RegionOne", "interface": "internal", "id": "9404fc709d3846fb8848093b05bf0b90"}, {"url": "http://192.168.1.46:9292", "region": "RegionOne", "interface": "public", "id": "f4680edcf0b24ae087cd1c9f0aad3ec6"}], "type": "image", "id": "9ae9104d3e6c4132ad60d40a75d2d03e", "name": "glance"}, {"endpoints": [{"url": "http://192.168.1.46:8774/v2/065e5a706ee24a9dbb15d2d0b869aa4a", "region": "RegionOne", "interface": "admin", "id": "1d26ea8e63764f30a96ad5eab00385e7"}, {"url": "http://192.168.1.46:8774/v2/065e5a706ee24a9dbb15d2d0b869aa4a", "region": "RegionOne", "interface": "internal", "id": "53a42c17de024e40bf05a86ec2a59d35"}, {"url": "http://192.168.1.46:8774/v2/065e5a706ee24a9dbb15d2d0b869aa4a", "region": "RegionOne", "interface": "public", "id": "88d26e5dc41f4ff5a6c19a884686dc57"}], "type": "compute", "id": "ae4bbeccc1514230a913d5f1ffff6b90", "name": "nova"}, {"endpoints": [{"url": "http://192.168.1.46:8776/v2/065e5a706ee24a9dbb15d2d0b869aa4a", "region": "RegionOne", "interface": "public", "id": "71f9c2eb8c9d4f9091c67762329453ef"}, {"url": "http://192.168.1.46:8776/v2/065e5a706ee24a9dbb15d2d0b869aa4a", "region": "RegionOne", "interface": "internal", "id": "87563f8cc4074086b96b04c59bba308e"}, {"url": "http://192.168.1.46:8776/v2/065e5a706ee24a9dbb15d2d0b869aa4a", "region": "RegionOne", "interface": "admin", "id": "9c7c50b1aaba4a03a28ed0b5c0135943"}], "type": "volumev2", "id": "c53a5aac29494108a483b46b43cb7347", "name": "cinderv2"}, {"endpoints": [{"url": "http://192.168.1.46:3333", "region": "RegionOne", "interface": "admin", "id": "0c83aaa89af6436e96fec2a3885a4059"}, {"url": "http://192.168.1.46:3333", "region": "RegionOne", "interface": "public", "id": "6f573b0c3b9243a695745c4ca4223030"}, {"url": "http://192.168.1.46:3333", "region": "RegionOne", "interface": "internal", "id": "d3e56f2022a2477d83f4a6d8a7a6269d"}], "type": "s3", "id": "ced8e46a45794933a59c8018119c0549", "name": "s3"}, {"endpoints": [{"url": "http://icehouse.sec.cs.kent.ac.uk:5000/v2.0", "region": "RegionOne", "interface": "public", "id": "7a7f118aba764b628b916c9d8557e3ed"}, {"url": "http://129.12.3.224:5000/v2.0", "region": "RegionOne", "interface": "internal", "id": "8ac9edfad28f4b4581c89138ef0254a3"}, {"url": "http://icehouse.sec.cs.kent.ac.uk:35357/v2.0", "region": "RegionOne", "interface": "admin", "id": "c91ac515e60d46a399284de27c42468c"}], "type": "identity", "id": "f10041f0063f4e3890f80e6af61a0ec2", "name": "keystone"}], "extras": {}, "user": {"id": "blm4", "name": "blm4"}, "issued_at": "2014-08-08T13:53:55.372285Z"}}

Appendix 9 - Installation guide

In order to use the modified OpenStack client, perform the following steps in order;

1. Install Git at the command line with *sudo apt-get install git*
2. Install setuptools with *sudo apt-get install python-setuptools*
3. Install pip *with sudo easy_ install pip*
4. Clone a local copy from the python-openstackclient repository, from branch: master
5. Install the openstack universal client with *sudo python setup.py install*
6. In most cases an error message will be reported *"No module named pbr.version"* is reported. To resolve, install pbr with *sudo pip install pbr.*
7. Remove the reference to *python-keystoneclient>=0.10.0* on line 5 of *requirements.txt.* It can be commented out with a # sign at the start of the line.
8. Clone a local copy from the python-keystoneclient repository, from branch: icehouse_compatible
9. Install the openstack keystone client with *python setup.py install*
10. export an environment variable OS_IDENTITY_API_VERSION=3
11. run the client using *openstack -F --os-auth-url http://icehouse.sec.cs.kent.ac.uk:5000/v3 user list*

You may encounter error messages depending on the default browser for your system. Testing with Firefox has shown full functionality with a bug ((process:2862): GLib-CRITICAL **: g_slice_set_config: assertion `sys_page_size == 0' failed) that exists between Firefox and Ubuntu. See the evaluation section for more information.

Google chrome would appear to have a higher level of security with self-signed certificates. As a result the connection with the server is killed and the program exits, until it has added the security certificate to its cache.