Cloud Credential Vault

Huan Liu
Accenture Technology Labs
50 W. San Fernando St., Suite 1200
San Jose, California, USA
huan.liu@accenture.com

***Abstract*—While cloud computing presents strong value propositions, it also presents significant headaches to enterprise IT departments, including incompatible billing and purchasing process, no policy enforcement and control, and difficult data sharing across users. We describe Cloud Credential Vault – a central repository of cloud access credentials, which is designed to solve these problems facing enterprise IT departments. We describe the Cloud Credential Vault's architecture, design, and how it solves each of the described problems. We also describe its current implementation, where we have already integrated with Accenture's billing system. Our early experience with the Cloud Credential Vault indicats that it can meet the challenges facing the enterprise IT department when managing access to cloud resources.**

***Keywords*-Cloud management, Credential Vault**

I. INTRODUCTION

Cloud computing is already widely used at small and medium businesses. Even large enterprise customers are increasingly evaluating and piloting cloud usage. There are several features of cloud that make it attractive to IT consumers. First, it is on-demand. A user requests a virtual server and the server would be available in a few short minutes. Second, it is pay-per-use. A user no longer needs to buy capital equipment upfront. Third, it is programmable. When an application needs additional capacity, it is a simple API call away. There is no longer the need to over-provision just in case it is needed.

Cloud computing may include many different types of cloud services. One sample service is Infrastructure as a Service (IaaS), such as Amazon EC2, where a user can request Virtual Machines (VM). Other services may include key-value storage services, such as Amazon S3, semi-structured storage services, such as Amazon SimpleDB, or messaging services, such as Amazon SQS.

Although the value propositions of cloud computing is strong, it brings significant disruptions to the current enterprise IT landscape for several reasons. First, its purchasing model does not conform to the standard enterprise purchasing order process. A user can simply pull out a credit card and sign up for cloud services without any IT approval. The charges do not appear in the IT budgeting process until at the end of the month during reimbursement when it is too late. An IT manager has no visibility into the current charges and the spend trend.

Second, an IT department has no control over cloud resources usage and cannot enforce corporate policy. Since a cloud account is under a user's total control, the user could easily abuse the system. For example, a policy may mandate that all data stored in a cloud should be encrypted, but a user can easily ignore the policy, knowing that the IT department has no ability to audit.

Third, a cloud makes it difficult to manage credentials securely. Many cloud services are invoked through a web services API. A user must present valid credentials in order to successfully invoke these APIs. Although this is no different than web services in Service Oriented Architecture (SOA), a cloud makes it more difficult. In a cloud environment, a cloud VM image could be easier shared between users. If the VM needs to access other cloud services (e.g., SQS, SimpleDB, S3), the VM may have to embed the cloud credential. Unfortunately, when the VM image is shared with other users, the credential is inadvertently shared as well. Even if the VM image is never shared, since the image is stored in the cloud, there is the danger that a hacker may hack the image file to obtain the credential. In addition, when the credential is changed (rotating credential regularly is one of the best cloud practices), the VM image must be changed, which is a significant hassle.

Fourth, data sharing in a cloud environment is difficult. When a user needs to share data (either cloud data or VM image) with other users, she must obtain the other users' cloud account ID and then share the data with the ID. Since the mapping from a user to her cloud account ID is maintained manually, it is cumbersome and time consuming to manage data sharing.

In this paper, we describe Cloud Credential Vault (CCV or Vault), which hosts all cloud credentials centrally. By centrally hosting credentials, we enable an IT department to automatically monitor cloud usages and enforce corporate policy. Although CCV is designed to support multiple cloud vendors, our first release currently only supports Amazon services. To be concrete, in the following, we use Amazon services to describe the architecture, design and implementation as needed.

In Section II, we first describe CCV's architecture. Then, we get into more details of CCV's capabilities in Section III. We cover related work in Section IV, and conclude in

 ISBN: 978-1-61208-106-9

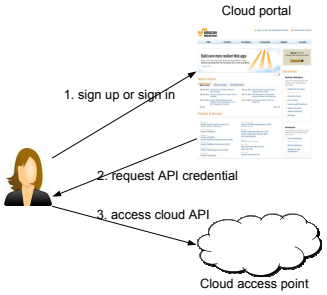

Figure 1. Cloud access model.

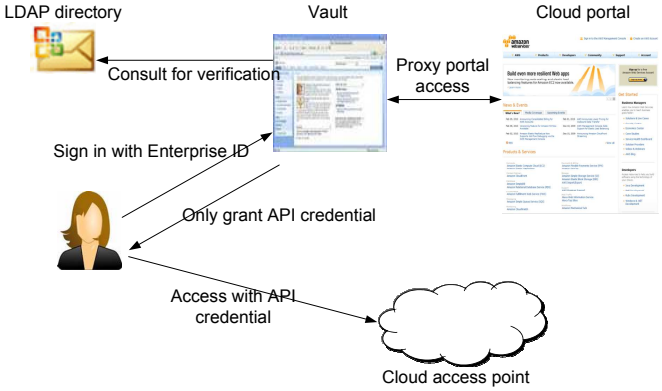

Figure 2. CCV architecture.

Section V.

II. Vault Architecture

In this section, we describe CCV's architecture.

A. Cloud access model

CCV exploits a unique feature of the cloud access model. Since it underpins CCV's design, we first describe the cloud access model, which is shown in Figure 1.

To access cloud resources, a user must first sign up for an account at the cloud vendor's portal page [1]. As part of signing up, the user establishes a username and password pair, which is used to login to the cloud portal. We refer to the username/password pair as the *master credential*, since knowing this pair would allow a person a complete control over the cloud account.

Using the master credential, a user can login to the cloud portal to obtain an *API Credential* – a credential needed to make programmatic API calls to access cloud resources. In Amazon, the API credential consists of an access key and a secret key. In GoGrid, it consists of an API key and a shared secret. In Rackspace, the API credential is an authorization token. Although the authorization token is not obtainable directly from the cloud portal, a user must first login to obtain a username and an API access key, then use them to further obtain the authorization token through an API.

Knowing the API credential is not enough to completely control the cloud account. For example, it is not possible to edit profiles and view/change the billing credit card. However, knowing the API credential is enough to access cloud resources. Although the cloud portal (only accessible through the master credential) typically consists of a GUI dashboard for accessing cloud resources, this functionality can be easily replicated by using the cloud APIs, which only requires the API credential. There are already a large number of third-party GUI console tools available which makes accessing cloud resources easier. For example, ElasticFox [2] is a popular dashboard for Amazon EC2, and S3Fox [3] is a popular dashboard for Amazon S3, both only need the API credential (access and secret key) to function properly.

Many cloud vendors, especially those with programmable APIs, follow this access model: a master credential is used to control the overall account and an API credential is used to access cloud resources. We exploit this separation of credentials in our CCV design.

B. CCV architecture

The basic idea behind CCV is that we split the master credential and the API credential. CCV, which is under the direct control of IT, holds the master credential so that IT can maintain a complete control of the cloud account. When a user is approved to have access to cloud resources, she is handed a unique cloud account which is not shared with others. A user is only given the API credential so that she can access cloud resources, but she is never given the master credential which would have given her total control over the account.

Figure 2 shows CCV's architecture. CCV is a web application, for which users can access directly from their web browser. It integrates with an enterprise's LDAP directory, so that it allows single sign on. Although it currently only works with one administrative domain, we plan to support some form of federated identity, such as that described in [4]. With federation, a CCV can support multiple organizations, making it possible to offer credential management as a service.

CCV interacts with the cloud portal directly. Since a user no longer has access to the master credential, she no longer can perform some functions that she is able to do through the cloud portal. Besides not being able to download the API credential, a user may not be able to perform other functions, e.g., downloading billing statements in the case of Amazon. CCV replicates those functions so that the user still have access to the same information. For Amazon, we screen-scrap the portal page in order to download all billing statements. From the cloud portal, a user could also perform functions that change account settings, such as changing the billing credit card number. Since we do not want the users to view or make those changes, we do not replicate those functions in CCV.

 ISBN: 978-1-61208-106-9

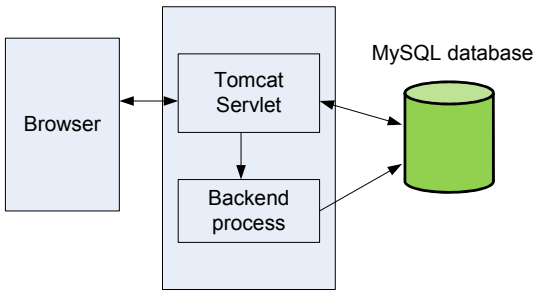

Figure 3. CCV implementation architecture.

Even though not shown, CCV is designed to support multiple cloud providers. In the back end, CCV not only holds the credentials for multiple cloud providers, but it also interacts with each of the cloud portals. For users, CCV can supply cloud accounts from multiple cloud vendors, depending on what a user requests.

A user of CCV can login to CCV to perform functions that she normally would use the cloud portal to perform, such as downloading the API credential, or viewing her cloud usage. Once the user has the API credential, she can access cloud resources directly through the cloud access point using third-party tools. Since only the infrequent action (e.g., viewing statement once a month), but not the more frequent action of accessing cloud resources, goes through CCV, CCV is unlikely to be a performance bottleneck.

Figure 3 shows the current implementation architecture. We use Google Web Toolkit (GWT) to develop the front end browser UI. The backend is implemented as Java Servlet running in a Tomcat container. The backend servlet is responsible for communicating with the frontend through a RPC mechanism. When the frontend invokes RPC calls to retrieve information, the servlet checks (and authenticate if necessary) the user's identity, and then returns the appropriate information authorized for the user.

There is a separate backend process running on the same server. It performs bulk actions that are not part of the web UI's request/response exchange. For example, once a month, the backend process logs into the cloud portal, screen-scraps and downloads the billing statement for each account. In our current implementation, downloading the complete billing statement at Amazon for one account takes roughly 30 seconds, thus it is not suitable to download on-demand when a user requests it. When a user requests for a billing statement that has not been downloaded yet, the servlet passes a request to the backend process, and it then returns immediately. The user is notified that the statement is being updated and that she should wait for a short while before viewing it again.

Screen-scraping simulates a user accessing for information, and a program automatically parses the output for needed information [5]. There are various ways to screen-scrap. For example, we could use the accessibility layer of an operating system to access UI components[6]. However, since all cloud vendors provide a web portal, we choose to use HtmlUnit[7] to parse the returned web page for relevant information.

Besides downloading billing statements, the backend process also performs auditing and policy enforcement. For example, if a policy states that no cloud data should be unencrypted, the backend process periodically takes a sample, and checks if any file conforms to the corporate policy. Since CCV not only has the master credential, but also the API credential, it can easily invoke cloud API to perform the auditing. Section III-B describes the kind of policies that we currently support.

Information downloaded from the cloud portal, such as the billing statement and the credentials, are stored in a MySQL database. For security purpose, all credentials are encrypted before stored in the database. The servlet does not keep any billing or credential data in memory. It always queries the database for the latest information. Thus, the database is our central state storage, which simplifies the synchronization between the servlet and the backend process.

III. Solution Details

This section describes the CCV solution in details. In particular, we describe how we address each of the problems described in Section I.

A. Billing integration

One of the goals of CCV is to integrate with an enterprise's internal billing process. CCV accomplishes this goal by acting as a broker between the internal billing system and the cloud.

In the cloud portal, CCV configures each cloud account to use a corporate credit card, which is charged each month by the cloud vendor and then paid directly by the IT department. In Amazon, we enable *consolidated billing* for all cloud accounts under CCV's management. This allows us to benefit from volume discount.

When a user signs up for a cloud account in CCV, she must configures a charge code associated with the cloud account. The charge code is charged each month for the actual cloud usage. In Accenture, the charge code is referred to as the WBS element. Although each company has a different internal billing mechanism, we have designed CCV to be flexible enough that it can easily integrate with a different mechanism.

When a charge code owner logs into CCV, she can see all cloud accounts that are charging to her charge code. She can view billing statements for all those cloud accounts. Since CCV can download partial billing statement when a user or the charge code owner requests, even if it is not the end of the month yet, the charge code owner can view up-to-date spends.

 ISBN: 978-1-61208-106-9

B. Control and policy enforcement

The charge code owner has a full control over the cloud account. She can optionally disable an account (e.g., if the user abuses the account or if the user has left the company), in which case, the API credential is changed so that the user can no longer use it. In addition, the charge code owner can optionally stop all cloud resources usage (stop all servers and/or remove all storage) to stop incurring further charges.

We also plan to support a flexible set of policies that a charge code owner can specify and enforce. However, initially we only support two sample policies.

The first policy states that "no more than x servers are running each day at time y". Both x (a number) and y (a time) are specified by the charge code owner. This policy is designed to catch run-away instances – instances that the user forgot to turn off, which happens frequently in the cloud environment. When enforcing this policy, we start a cron job at the specified time y and use the API credential to query how many EC2 servers are running at the time. If it is more than x, we send an email alert to the charge code owner.

The second policy states that "all cloud data should be encrypted". When this policy is enabled, CCV periodically samples a few files stored in the cloud, and checks whether they are encrypted. For demonstration purpose, we currently only check whether the file is a plain text file. However, in the future, we intend to employ more sophisticated algorithms to detect whether a file is encrypted.

While these two policies are designed to demonstrate the capabilities of CCV, we intend to support a wider range of flexible policies. We are in the process of gathering requirements to understand which set of policies are the most useful.

C. Credential on demand

When a VM needs to access other cloud resources, such as S3, SQS and SimpleDB, a common practice today is to embed the needed API credential inside the VM image. The reason is because it is cumbersome to copy over the API credential every time the VM starts. However, this practice is not secure, for two reasons.

First, the server image could be shared with other cloud users. When other cloud users launch the same image, they would have access to the image owner's API credential.

Second, changing API credential frequently is a cloud best practice, since it minimizes the damage of losing an API credential. Unfortunately, when the API credential is changed, the credential embedded in the VM images remains the same, requiring the image to be recreated again.

Instead of embedding the *image creator*'s API credential, we believe a VM should be entitled to the *image user*'s API credential. CCV provides such a facility to VMs to query the API credential used to launch the VMs in the first place. The VM can request this by querying a URL at the IP address of CCV, but with URI of "/apicredential", e.g., a VM can query https://ccv.com/apicredential.

When a VM queries this API, CCV first has to find the API credential used to launch the VM. We currently iterate over each stored API credential and query Amazon API in sequence to see which API credential the VM was launched under. Although this is inefficient, we have not run into performance problems yet during our pilot trial. We are actively looking into alternative approach to determine the launching credential,

When the launching API credential is found, CCV passes back the API credential to the VM, so that is can start accessing other cloud services such as S3, SimpleDB, and SQS. By providing this mechanism of API credential on demand, we prevent any need to hard-code credential information inside a VM image, thus greatly enhance cloud security.

D. Data sharing

Sharing data across cloud account is cumbersome. A user has to find out about other users' cloud account ID (a 12 digits number in Amazon) and enables sharing with the ID instead of a user name.

In CCV, a user can share cloud data and server images with a user name or a group alias. The group aliases are from the LDAP directory. When a user chooses to share with a group (e.g., HR.global group), CCV looks up in the LDAP directory to expand the group into a list of user names. From the list of user names, CCV then expands it into a list of cloud account IDs by checking the cloud account IDs that belong to each user. It then shares the data or image with the list of account IDs. Even though CCV still uses the cloud's native capability to share data with an account ID, the user operates at a much higher abstraction layer, sharing with a user or a group of users.

The group membership in LDAP directory may change over time, e.g., new members joining HR.global or existing members leaving HR.global. CCV periodically checks the group membership and adjusts the permission as needed.

IV. Related work

RightScale[8] and EnStratus[8] all address the same management challenge facing enterprises trying to adopt cloud computing. However, both of them take a different approach than ours. They use one cloud account to support a whole enterprise, and build an interface on top to multiplex multiple cloud users through the same cloud account. Although it has the ability to limit a user to a subset of cloud functionalities, it has two disadvantages. First, their interfaces have to be very scalable since all cloud access go through those interfaces. Second, they cannot perform accurate accounting between different projects. For example, a VM could consume significant bandwidth charges, but since its bandwidth usage does not go through the management interfaces, it is

 ISBN: 978-1-61208-106-9

not possible for those interfaces to meter and bill projects for those bandwidth charges.

MyProxy[9][10] is a repository of X.509 proxy certificates[11]. CredEx [12] extends MyProxy to further support heterogeneous authentication methods. These repositories all treat the credentials as opaque, whereas we take advantages of the API credentials to enable control and auditing.

Amazon cloud manages SSH public key in a similar mechanism as we used for passing the API credential to an image. The SSH public key is available at a fixed IP address (169.254.169.254) and a fixed URI (/latest/meta-data/public-keys). To pass the API credential, we also use a fixed IP address (CCV's IP address) and a fixed URI (/apicredential).

V. Conclusion and future work

We have presented the architecture, design and implementation of the Cloud Credential Vault – a central repository of cloud credentials. By centralizing the credentials and by separating the master credential from the API credential, CCV solves many management problems facing enterprises that are adopting cloud computing.

We have only completed a prototype implementation supporting only one cloud vendor (Amazon). Beyond supporting more cloud vendors, there are also several open questions that we need to address to make CCV more efficient. First, we need a more efficient mechanism to look up the API credential used to launch a particular VM. Second, we need to define a more comprehensive set of policies to support. Third, we are also looking at more efficient ways to monitor group membership changes so that we can adjust data sharing permission as needed.

References

[1] Amazon Web Services, "Amazon Web Services Portal," http://aws.amazon.com, 08.23.2010.

[2] Elasticfox, http://sourceforge.net/projects/elasticfox, 08.23.2010.

[3] Suchi Software, "S3fox," http://www.s3fox.net/, 08.23.2010.

[4] E. R. Mello, M. S. Wangham, J. Silva Fraga, E. T. Camargo, and D. Silva Böger, "A model for authentication credentials translation in service oriented architecture," pp. 68–86, 2009.

[5] B. Myers, "User interface software technology," *ACM Comput. Surv.*, vol. 28, no. 1, pp. 189–191, 1996.

[6] M. Grechanik, K. Conroy, and K. S. Swaminathan, "Creating web services from gui-based applications," in *Proc. SOCA*, 2007.

[7] Htmlunit, http://htmlunit.sourceforge.net, 08.23.2010.

[8] Rightscale, http://www.rightscale.com, 08.23.2010.

[9] J. Novotny, "An online credential repository for the grid: Myproxy," in *Proceedings of the Tenth International Symposium on High Performance Distributed Computing (HPDC-10), IEEE*. Press, 2001, pp. 104–111.

[10] J. Basney, M. Humphrey, and V. Welch, "The myproxy online credential repository," *Software: Practice and Experience*, vol. 35, pp. 801–816, 2005.

[11] S. Tuecke, V. Welch, D. Engert, L. Pearlman, and M. Thompson, "Internet X.509 public key infrastructure (PKI) proxy certificate profile." RFC 3820 (Informational), 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3820.txt

[12] D. D. Vecchio, M. Humphrey, J. Basney, and N. Nagaratnam, "Credex: User-centric credential management for grid and web services," *Web Services, IEEE International Conference on*, vol. 0, pp. 149–156, 2005.

 ISBN: 978-1-61208-106-9