Self-Service Financial Control and Organizational Governance in Cloud

Chunqiang Tang, Chang-shing Perng, and Salman A. Baset

IBM T.J. Watson Research Center
{ctang, perng, sabaset}@us.ibm.com

***Abstract*—Cloud computing touts the benefits of self service and auto scaling, but its negative impact on an enterprise's business process and culture change is often neglected. This paper addresses two risks introduced by Cloud, i.e., losing financial control in IT spending and losing organizational governance. The root cause of these risks is due to the fact that the account structure in today's Cloud does not reflect the typical hierarchical organization structure in an enterprise. Therefore, it is hard to implement in Cloud the traditional business processes that enforce IT financial control and organizational governance. We propose a solution where the Cloud provider offers *mechanisms* to enforce financial control and organizational governance, while the *policies* are provided by customers through self service in the Cloud portal. Intuitively, we extend the concept of self service from resource provisioning into the area of financial control and organizational governance. Specifically, we introduce the concept of *credit token* in Cloud to resemble how budget allocation trickles down in a traditional organization, and introduce a hierarchical structure among Cloud accounts so that an ancestor account has privileges over a descendant account to enforce governance.**

I. Introduction

Cloud Computing is widely considered as the next big thing in IT evolution, and is getting rapid adoption in the industry. Cloud promises customers with the benefits of a more convenient way of provisioning IT resources at a faster speed and with a lower cost. According to NIST [6], two key features of Cloud are:

- "On-demand self-service, where a consumer can provision compute and storage capabilities without requiring human intervention from provider."
- "Rapid elasticity, where resources used can be rapidly and in some cases automatically increased or decreased to handle demand."

For enterprises, Cloud means not only a technology shift but also changes in IT organization, business process, and culture. On the one hand, self-service and elasticity improve productivity by eliminating human intervention and manual approval. On the other hand, they bring the risk of circumventing the traditional business processes that enforce IT financial control and organizational governance. We illustrate the challenges using several examples.

A. Challenges

Example 1: A student over-spends his professor's credit card on Cloud resources.

In a USENIX ATC'09 invited talk [5], Prof. Malan presented the experience of using Amazon EC2 [1] for computer class education, eliminating the need for maintaining a physical lab. The per-student cost of using the Cloud was only about $15. However, one concern reported was the network bandwidth cost and lack of control over student spending.

Example 2: A large enterprise continuously adjusts its IT budget allocation and organization structure, making it hard for frontline engineers to balance spending.

Example 1 is relatively simple. The situation is much more challenging for a large enterprise. It is common that an enterprise's budget allocation across the business units is continuously adjusted throughout the year. For example, the IT budget for an engineering team may be cut by 10% due to stagnant sales. With the self-service model, it is the frontline engineers who provision resources in Cloud. However, it is hard for them to have a real-time view of the team's latest budget situation and the expense already accrued by other team members. As a result, they can easily make mistakes in one way or another, e.g., overspending and missing the financial target, or underspending the budget and missing the accelerated product launch cycle.

Example 3: Due to a bug in a Cloud application's auto-scaling controller, it mistakenly creates 1,000 virtual machines (VMs) instantaneously.

Suppose an unexpected workload triggers the auto-scaling controller to react, but that particular code path in the controller was rarely exercised before. Due to a bug, it mistakenly treats the elapse time one millisecond as one second, i.e., a 1,000x mistake. Instead of correctly adding 1 VM, the controller mistakenly adds 1,000 VMs. As a result, the application uses up its whole-year IT budget in a single hour.

Example 4: An employee provisions in the Cloud a public facing VM using the company's domain name, but it exposes inappropriate Web contents, due to either mistake or abuse.

Suppose this problem is reported by an outsider to the company's top management team, and the decision is to shut down the VM immediately. However, the action may be delayed either because the order cannot quickly reach the employee who owns the VM (e.g., on vacation), or because the employ does not comply. This kind of governance problem is not unique to Cloud, but it is aggravated by Cloud's self-service model and the convenience of setting up a VM.

One naive solution of the problems above is for an enterprise to blindly embrace self-service and auto-scaling, and bear the potential damage of losing financial control and organizational governance, which is not acceptable for many serious enterprise users of Cloud. Another equally naive solution is to give up the self-service and auto-scaling features of Cloud, and simply use the new Cloud technology in a restricted, old-fashioned way. For example, individual developers are not allowed to directly provision VMs. Instead, all VM provisioning requests go through a manual pre-approval process and then the requests are manually submitted into the Cloud by a dedicated IT support team. This solution, however, does not fully benefit from Cloud's capability to improve productivity and make it impractical to implement auto scaling.

B. Solution Overview

We propose a different approach to address the challenges. The Cloud provider builds *mechanisms* in Cloud to enforce financial control and organizational governance. The *policies* for financial control and organizational governance are provided by each customer through self-service in the Cloud portal. Intuitively, we extend the concept of self-service from resource provisioning into the area of financial control and organizational governance.

Our solution consists of two parts, one for financial control and another for organizational governance. First, we allow a customer to use the Cloud portal to conceptually construct a directed acyclic graph (so-called *budget-flow-DAG*) that resembles how budget allocation trickles down in an enterprise. A vertex in the DAG represents an account and an edge $P \rightarrow C$ from account P to account C represents budget allocation from P to C. The edge $P \rightarrow C$ is annotated with a set of (potentially sophisticated) financial control policies called *credit token*.

Below, we discuss how the budget-flow-DAG helps address the challenges in financial control.

- For *Example 1* above, the professor account (as a parent node in the DAG) can impose a policy on a student account (as a child node in the DAG) that its total spending cannot exceed $30.
- For *Example 2* above, if a senior manager wants to transfer some budget from one department X he manages to another department Y he manages, he can do so easily through self-service in the Cloud portal, with neither higher-level management nor lower-level management involved. Then the Cloud ensures that department X cannot spend beyond its new, reduced budget.
- For *Example 3* above, the account owner of the auto-scaling application can impose a policy with three conjunctive conditions: 1) the hourly spending of the account cannot exceed $50, 2) the daily spending of the account cannot exceed $300, and 3) the monthly spending of the account cannot exceed $3,000. The buggy program's attempt to create 1,000 VMs would be rejected, because the hourly spending of 1,000 VMs exceeds the hourly limit of $50.

The second part of our solution enables organizational governance. The budget-flow-DAG described above is related but not identical to an enterprise's organization structure. For example, a development team may get funding not from its parent organization but instead from a parallel sales organization. Therefore, the budget-flow-DAG does not necessarily reflect the organization structure.

We propose a separate concept called *organization-tree* to help enforce organizational governance. The administrators of an enterprise can use the self-service Cloud portal to construct an organization-tree that resembles the enterprise's structure. Alternatively, the organization-tree may simply be pulled from the enterprise's LDAP server. An edge $P \rightarrow C$ in the organization-tree means that account P is the upper-line management of account C. An ancestor account in the tree has authority over the resources provisioned by its descendant accounts.

For *Example 4* above, suppose the problematic VM is created by the owner of account C_1 and its upper-line management account is P_1. Then the owner of account P_1 can use the Cloud portal to directly shut down the VM and disable account C_1 without the owner of account C_1 involved. In this case, the Cloud governance mechanism supports a scenario that resembles what may happen in an enterprise outside Cloud.

II. Organizational Governance

We first define some terminology about *account*, *user*, and *resources*. An *account* in Cloud represents an organization entity, large or small. Just like organization entities in the real world forming a hierarchy, accounts in Cloud form a tree structure, where a parent node in the tree is the upper-line management organization of a child node in the tree.

Multiple *users* can be associated with a single account, performing operations on behalf of the account. The users can have different roles. For example, an *administrator* can provision VMs, whereas a *restricted user* can only use the account's existing VMs but cannot provision new VMs.

Resources such as VMs are owned by *accounts* rather than *users*. Suppose a user u provisions a VM v on behalf of an account P. Even if the user u leaves the company, the account P owns the VM v, and other users of the account P can still de-provision the VM.

Below, we present some major use cases of self-service organizational governance in Cloud.

a) User creation: An employee can create a user ID in Cloud on his own, but the ID is not linked with any account initially and cannot provision VMs on behalf of any account.

b) Standalone account creation: A user u can create an account P in Cloud and becomes the administrator of the account P. The user u can associate other users with account P so that they can also perform operations on behalf of the account P. In order to provision resources, the account P must be backed by either an actual line of credit (e.g., credit card or contract) or a virtual *credit token* (see Section III).

c) Child account creation: The administrator u of an account P can create a child account C, and link P to C, i.e., creating an edge $P \rightarrow C$ in the organization-tree. The administrator u can associate more users of different roles with the new child account C so that they can perform operations on behalf of C. The child account creation process can be performed recursively to construct a complex organization-tree. Using the self-service Cloud portal, the administrator of an account in the organization-tree can independently create its own child accounts, without other accounts involved. There is no need for a central authority in the enterprise to manage every change to the organization-tree. Otherwise, the solution would not be scalable.

d) Ancestor account privilege: Just like the hierarchical line management structure in the real world, an ancestor account in the Cloud has authority over descendant accounts. For example, the administrator of an ancestor account can suspend a descendant account, or de-provision or power off a VM owned by a descendant account.

e) Cloud consumption reporting: Using the Cloud portal, the administrator of an account can easily produce reports at all levels, e.g., resources consumed by the account itself excluding resources consumed by its descendant accounts, or resource consumed by itself plus all its descendant accounts.

f) Account migration: As an enterprise's organization structure evolves over time, the changes need be reflected in the organization-tree maintained in Cloud. Suppose an account C's parent account need be changed from P_{old} to P_{new}, because C's upper-line management changes. The administrator of account P_{old} initiates the change, by using the Cloud portal to send to account P_{new} a migration request for account C. If the administrator of account P_{new} approves the request, account C becomes a child of account P_{new} in the new organization-tree.

g) VM ownership transfer: Suppose a VM is provisioned for account C_{old}, but it need be transferred to another account C_{new}. The administrator of account C_{old} initiates this change, by using the Cloud portal to send to account C_{new} a transfer request for the VM. If the administrator of account C_{new} approves the request, the transfer completes and the VM shows up in the Cloud portal under account C_{new}. Accordingly, the cost of the VM will be charged to account C_{new}.

III. Financial Control

Our financial control solution allows a customer to use the Cloud portal to conceptually construct a *budget-flow-DAG* that resembles how budget allocation trickles down in an enterprise. A vertex in the DAG represents an account and an edge $P \rightarrow C$ from account P to account C represents budget allocation from P to C. The edge $P \rightarrow C$ is annotated with a set of financial control policies called *credit tokens*.

A. Credit Token

Definition 1: A credit token is defined as a tuple $C = (R, P, F, T)$. Here R is a list of resources the credit can be used on. $P = \langle S, E \rangle$ is the lifetime of the token, where S is the starting date and E is the ending date. F is the amount of the fund. $0 \leq T \leq 1$ is the ratio of surplus that can be retained by the credit recipient.

A "root" credit token is obtained from the Cloud provider and backed by an actual line of credit, e.g., credit card or contract. The owner account of a credit token can either spend the fund by itself, or divide the token and issue "child" credit tokens to other accounts, meaning passing on part of the fund.

There are some constraints for dividing a credit token:

Rule 1: A credit token $C = (R, P, F, T)$ can be divided into a set of child credit tokens $\{C_i | i \in I\}$ where I is an index set and $C_i = (R_i, P, F_i, T_i)$. The division has to satisfy the following conditions:

1) The credit can only be used on the same or the subset of the originally specified resources, i.e., for $i \in I$, $R_i \subset R$.
2) The credit can only be used in the originally specified spending period, i.e., for $i \in I$, $P_i \subset P$.
3) The total fund of the divided tokens cannot exceed the original fund, i.e., $\sum_{i \in I} F_i \leq F$.
4) The amount of the fund that can be retained does not increase, i.e., $\sum_{i \in I} F_i \times T_i \leq F \times T$, because the owner of the original credit token still needs to return the fraction $(1 - T)$ back to the source account.

A credit token can be issued manually or automatically. By defining and registering a *credit-issuing function* (cif) with the Cloud, an account can automate the task of periodically re-creating credit tokens based on changing conditions.

Definition 2: A credit-issuing function is of the form $cif(B, \langle L_1, \cdots, L_n \rangle) = \langle C1, \cdots, C_n \rangle$ where B is the total amount of allocated base credit or expected incoming credit for the period, L_i is the surplus returned by the receiving account i, C_i is the new credit token to be distributed to account i.

Credit-issuing functions can be written in a safe script language. Below is one example that can be supported by a credit-issuing function. Suppose an account has decided to allocate \$4,000 each week for a year for the two child accounts A and B, such that each month at least \$2,000 goes to A and at least \$1,000 goes to B. Each child account can retain 50% for future use and return 50% to this account. The surplus is distributed to the accounts by the amount inversely proportional to the surplus they had for the previous period.

B. Spending Monitoring

Proactive spending monitor helps an organization manages its financial status. Following the ECA paradigm (event, condition, and action), our solution allows accounts to define spending monitoring rules. When an event specified by a rule is detected, the system checks the corresponding condition. If the condition is satisfied, the action is fired.

a) Events: There are three types of events.

1) Resource changes. This includes any change to the amount of resources owned by an account, e.g., provisioning or de-provisioning VMs.

2) Usage alerts. This type of event is generated when the usage of a resource exceeds a threshold. Typical resource usages include network, disk I/O, and database transaction.
3) Periodic check. The system checks a condition periodically.

b) Conditions: The condition is a boolean expression of a set of variables. The following types of variables are supported by the system and can be used in customer-defined rules.

1) Information in a credit token.
2) Amount of fund already spent.
3) Service request history, e.g., time of VM creation and persistent storage allocation.
4) Allocated and active resources, e.g., the number and type of active VMs.
5) Metered I/O, e.g., network data transfer and disk I/O.
6) Historical spending rate, e.g., hourly, daily, weekly, or monthly.
7) Sustainable spending rate, which is the maximum spending rate that can be maintained toward the end of a spending period.

The rule conditions can defect interesting situations that need attention. One situation is *usage burst*. For example, a condition may detect there are more than 50 virtual machines provisioned in the last hour. Another situation is *trend violation*, i.e., a prediction that the current spending rate would not be able to sustain till the end of the spending period covered by the credit token.

c) Actions: Typical actions fired by a rule include:

1) Notify owner account of the credit token and provide the situation summary. This is a default action for all triggered rules.
2) Disable a request that would increase spending.
3) Release resources in possession to reduce spending.
4) Disable an account until corrective actions are taken.

d) Examples: Below are some rule examples.

1) To monitor sudden burst of VM provisioning:

event:	every 60 minute
condition:	requested more than 20 VMs
action:	disable provisioning

2) To prevent unsustainable spending rate:

event:	every 4 hours
condition:	Spending rate of last 4 hours
	is not sustainable for the period
action:	de-provision non-essential servers

IV. Related Work

Large enterprises have mostly manual mechanisms for enforcing financial control and organizational governance. The self-service nature of Cloud necessitates an automated solution. Cloudability [3] provides services that monitor spending in multiple Clouds and send email alerts when a condition is met, e.g., spending beyond a certain threshold. Cloudability and our solution differ in capability and philosophy. Cloudability provides simple functions without requiring Cloud provider support, whereas we advocate building mechanisms into Cloud to provide a comprehensive solution. Cloudability does not cover organizational governance. It can monitor but cannot enforce financial control.

Credit based tokens have been described in other contexts such as spectrum sharing between different cell phone operators [4]. However, the use of credit tokens within Cloud for financial control and organizational governance is novel.

Amazon Identity and Access Management [2] manages user identities and access control for Amazon web services (AWS). These services include group management of Cloud users, federation between corporate directory and AWS services, and fine grained access to Cloud services. However, it does not provide any mechanism for financial control and organizational governance described in this paper.

V. Conclusion

In this paper, we discussed the challenges introduced by Cloud in the area of financial control and organizational governance. We proposed building *mechanisms* into Cloud so that customers can define *policies* in a self-service portal to enforce financial control and organizational governance, in a way familiar to customers outside Cloud, while not losing the benefits of self-service provisioning and auto scaling. Specifically, we presented the concept of *credit token*, which resembles how budget allocation trickles down in an enterprise. We also proposed a hierarchical structure among Cloud accounts so that an ancestor account has privileges over a descendant account to enforce governance.

To our knowledge, this is the first work that proposes self-service financial control and organizational governance in Cloud, covering an often neglected but critically important topic. Currently, we are actively implementing the proposed solution based on OpenStack [7]. We anticipate that user feedback will change and/or enhance some design decisions in this paper.

References

[1] Amazon elastic compute cloud (amazon ec2). http://aws.amazon.com/ec2/, 2008.

[2] Amazon Identity and Access Management. https://aws.amazon.com/iam/, 2012.

[3] Cloudability. https://cloudability.com/.

[4] D. Grandblaise, K. Moessner, G. Vivier, and R. Tafazolli. Credit token based rental protocol for dynamic channel allocation. In *1st International Conference on Cognitive Radio Oriented Wireless Networks and Communications*, pages 1–5, June 2006.

[5] D. J. Malan. Teaching Computer Science in the Cloud, 2009. Invited talk at USENIX ATC'09. http://static.usenix.org/events/usenix09/tech/slides/malan.pdf.

[6] P. Mell and T. Grance. The NIST Definition of Cloud Computing. NIST Special Publication 800-145, September 2011.

[7] OpenStack. https://www.openstack.org, 2012.