Oregon State University
College of Engineering

CS Capstone Design Document

December 6, 2019

Cloud Computing Billing Infrastructure

Prepared for

OSU IT Infrastructures

Stacy Brock _____ _____
Signature *Date*

Prepared by

Group 75

Beavs on Cloud

Andrew Quach _____ _____
Signature *Date*

Faisal Khan _____ _____
Signature *Date*

Abstract

This document describes the overall architecture of the billing project. The automatic billing will query the vSphere API using the pyVmomi module, then store the information into a SQLite3 database. Each month, a report will be generated and sent to accounting based on the information stored in the SQLite DB and the information about external storage utilization. Another report will be generated quarterly for CoSINe. In order to distinguish the types of VMs, a tagging application will automatically parse a YAML configuration file and tag VMs based on directory structure. The project will be deployed using Jenkins.

CONTENTS

1 Introduction

This document covers the overall architecture of the two main components of the Cloud Computing Billing project: the automatic generation of the billing spreadsheet and the automatic tagging of the virtual machines in vSphere.

The automatic billing portion of the project aims to generate billing spreadsheets for accounting monthly, and pass along managed virtual machine data to CoSINe quarterly, by querying the vSphere API. The application will be constructed in such a manner that quarterly reports can be easily extendable to other departments who request them. This portion will be handled by Andrew Quach.

The tagging portion of the project aims to provide a simple way to label the virtual machines based on a configuration file (managed/unmanaged, chargeable/unchargeable). This effectively feeds needed information into the automatic billing application by tagging based on directory structure. This portion will be done by Faisal Khan.

2 Automatic Billing

2.1 Pulling Information from vSphere

The Virtualization Management Object Management Infrastructure (VMOMI) backs the network services used to communicate with vSphere products [1]. pyVmomi is the Python module that binds to the exposed VMOMI REST API. By using pyVmomi to send a "list" call to the API, we are able to collect the identifiers of all virtual machines in vCenter. This "list" call allows for filtering, meaning we can separate the identifiers based on their tags. In particular, we are separating based on two tags: chargeable VMs, and managed VMs.

Once we have identifiers, we can use the "get" call on each individual identifier to collect crucial hardware statistics and relevant tags: CPU core count, memory usage, disk space allocated, billing rate, ownership information. This information is sufficient to calculate the price point for the virtual machines.

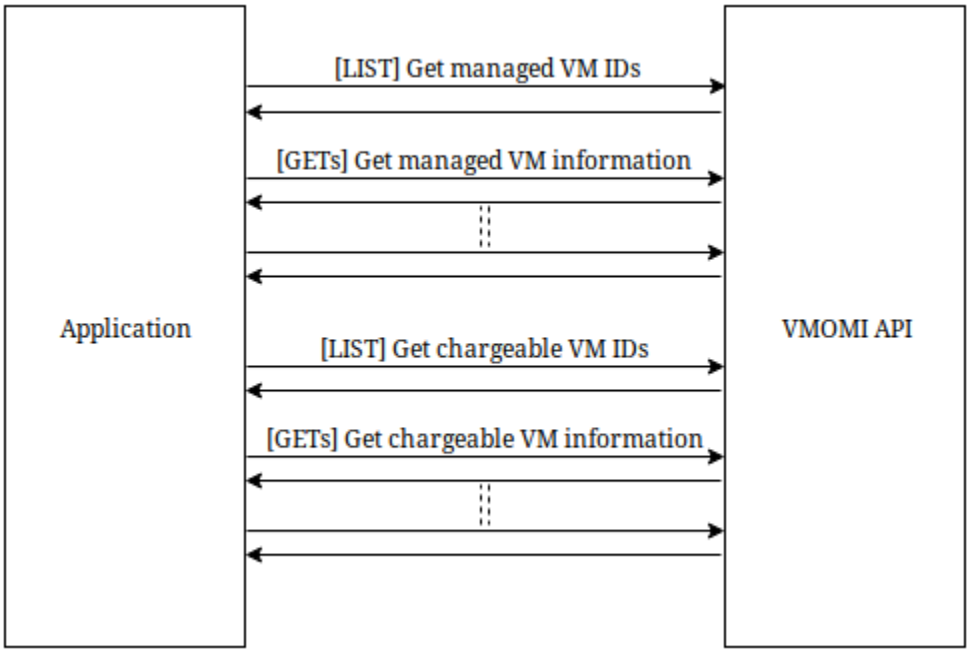

Fig. 1. API call flow for pulling managed and chargeable VM hardware data.

2.2 Storing Information from vSphere

We will run the information pulling script once per hour. This gives us fine enough granularity without querying the API to an excessive degree. Note that this figure can be reconfigured and optimized with further testing in the future. The data gathered by the pulling script will be stored in a SQLite3 database (a lightweight and server-less database). Since a SQLite3 database is just a file, this allows us to easily archive information from month to month–simply compress and store the old database file and create a new database file monthly.

The initial application will have two tables in our database: one for chargeable VMs and one for managed VMs. Naturally, the chargeable VM data will be placed in the chargeable VM table (likewise with managed VMs). The tables will store timestamp of access, CPU core count, memory usage, and disk space allocated, billing rate, and ownership information.

Since virtual machine hardware typically does not change from hour to hour, naively storing all the data would result in a lot of duplicate information and wasted storage. Instead, we can check whether the new data for a VM is different from the previous entry. If it is, we insert it into the database, otherwise we disregard it. Even though we are querying the API hourly, this optimization means that the majority of that information will not be stored (unnecessarily) into the database.

2.3 Generating Reports

Each month, a billing spreadsheet will be generated in the Finance UPLOAD (FUPLOAD) format. This spreadsheet will be calculated from information contained in the chargeable table in the database and (currently) storage metrics from a NetApp Network Attached Storage (NAS). This storage system may change in the near future, so the design needs to be accommodating in that regard. A script running on the NAS should transfer the data to the machine our billing application is on. One potential way to handle this data transfer would be setting up a REST API endpoint for the storage script to POST a file to. The quarterly report is similarly handled (albeit without the dependency on the external NAS script).

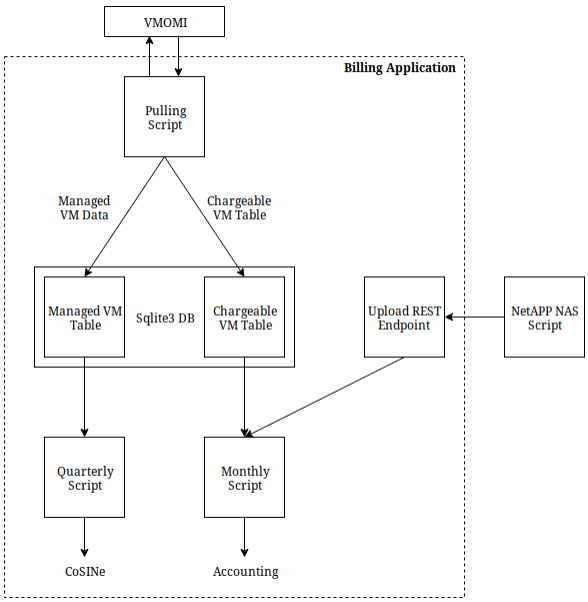

Fig. 2. Overall architecture of billing application.

3 AUTOMATIC TAGGING

To specify a specific attribute for a virtual machine or vSphere object, VMware's tagging feature built into vSphere is an amazing asset that can be used. For this project, tagging plays a critical part to figure out if either the device is managed or chargeable within the virtualization platform. Two most important things to remember when figuring out the best way to implement VMware tags are the tags that are going to be used along with the categories that are going to be put into them. Following is the way we are going to be using tags in our project to figure out the chargeable and managed devices through indexes.

For the tagging rules engine, all the departments will fall under a location directory (such as Corvallis). Each department will then be tagged as managed or chargeable. Managed departments will be sent all the VM information quarterly and will be billed quarterly while chargeable departments will be billed each month. The ”indexes” field will specify the division of the bill. Tagging rules engine will look like this:

```
Corvallis:
  Library: Everything charged
  NWS:
    Folder A: [Chargeable]
        item 1: Managed
        item 2: Chargeable
        item 3:
    Folder B: [Chargeable]
        item 1: Chargeable
        item 2: Managed
        item 3:
```

```
UHDS: Everything Charged
```

Managed VMs: COSINE – Do billing quarterly

Chargeable VMs: Infrastructure - Do billing monthly

In order to describe which VMs are billed for which rates, there needs to be a configuration file that describes the vSphere directory structure for each client, along with what tags to apply to which directories and what index(es) to charge to. A client may want to split billing across multiple indexes with different percentages of the total applying to each index. This program should either run continuously, or before the billing calculation program is ran. This program will read the configuration and push tags to all VMs that fall under the directories configured, using the vSphere API. The configuration will be in YAML. There would be a top-level hash with the key 'clients', with each client's user or other identifying information as a sub-hash with the keys 'root', 'indexes', and optionally, 'billing'. The 'root' hash would describe the directory structure along with tags that apply to certain directories. The 'indexes' hash would contain a list of indexes and optional custom billing percentages. The billing calculation program would have to verify that the percentages add up to 100%. The 'billing' key-pair describes the billing period in plain English. It can accept 'monthly' or 'quarterly'.

```
Clients:
  Cosine:
    Root:
      Managed:
        Tags:
          - Manage
          - Chargeable
      Unmanaged:
        Tags: [Chargeable]
    Billing: quarterly
    Indexes:
Name: IDGF9001
Percent: 25
Name: MCSCKS3000
Percent: 75
```

With this example configuration, CoSINe would be billed each quarter, split across indexes IDGF9001 and MCSCKS3000 at 25% and 75% of the total bill, respectively. The tags 'managed' and 'chargeable' would be applied to VMs under their/managed directory, and the 'chargeable' tag would be applied to their /unmanaged directory. Items are charged within the folders as well so if an item within the folder is not specified as managed or chargeable, it will be charged according to if it is in a chargeable folder or not. This script will be written is Python 3.

4 Deployment

Deployment for this project will be done using Jenkins deployment tool. Jenkins provides a machine consumable REST style API for programmatically interacting with Jenkins server. This comes in three flavors currently which are XML, JSON, and Python. Remote access API is offered in a REST-like style.

For this project, the client take over the deployment of the scripts. There is already has a well-defined pipeline for deploying applications using Jenkins. The scripts will be ran in Python 3.6 with all the dependencies in a requirements.txt file. The known dependencies are pyVmomi, SQLite3, and SQLAlchemy. More dependencies may be added as needed. The scripts will configured to run hourly/quarterly/monthly (as previously specified) using the job scheduling feature of Jenkins.

5 Conclusion

The entirety of this project can be thought of as loosely coupled scripts that aims to automate away the rest of the billing process. All together, there are three main components of the project: pulling information from the API and storing it, synthesizing the data pulled, and pushing information to vSphere. Since these parts are relatively independent of one another, the complexities associated with the scripts are encapsulated from each other. This design allows for ease of testing (using Python Unit test) thanks to its modularity.

REFERENCES

[1] vmWare. (2014). Vmomi, [Online]. Available: https://github.com/vmware/pyvmomi/wiki/VMOMI.