Oregon State University
College of Engineering

CS Capstone Project Archive

May 30, 2020

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 provides a complete archive of the billing project, including details about the requirements, architecture, current status, and documentation, and peer reviews. 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. 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.

CONTENTS

1 Introduction

1.1 Problem

At Oregon State University, the IT department has the ability to provision virtual machines through vSphere. They can fully customize the resource allocation of these virtual machines (e.g. cores, RAM, storage) meaning the cost of one machine is variable. On top of this, certain machines get charged at different rates (some do not get charged at all).

While the entire IT department can create virtual machines, the responsibility of billing comes down entirely on the infrastructures department. Currently, the billing process is a mix of automatic scripts and manual manipulation. The statistics of the machines are automatically dumped weekly into a spreadsheet and sent out to infrastructures staff to manually tabulate costs.

The variability in the cloud computing costs means any manual labor in the billing process is extremely time consuming. The main purpose of this project is to automate away the annoyance of the billing project. That is, automatically scrape the vSphere API for data and automatically generate the needed financial spreadsheets as necessary. As a byproduct, this project will also serve to refactor the code base to Python 3, and improve the fidelity of the billing calculations by increasing the scraping granularity.

1.2 Development Team and Clients

This project was requested by the Oregon State University Infrastructures department. Our clients were specifically Stacy Brock, Marjorie McLagan, and Gaylon DeGeer. The clients supervised the development of the project, clarifying requests and approving architectural ideas.

The development team consisted of Faisal Khan and Andrew Quach. Faisal Khan oversaw the development of the virtual machine tagging process. Andrew Quach oversaw the development of scraping data from vSphere and the generation of the spreadsheet.

1.3 Impact of COVID-19

COVID-19 has unfortunately and unexpectedly halted the progress of the development of this project. While this project is nearing the realm of completion, there is unfortunately still a lot of testing to do before pushing everything to production. The main issue to overcome was the integration of the two halves of the project. Due to the structure of the development environment being different from that of production, the two parts of the project operate in different environments. The tagging portion only works on the development vSphere environment as it mutates data. The scraping portion, relying on the specific production directory structure and naming conventions, only works on the production environment. Due to COVID-19 and personal circumstances, the development team was unable to consistently find time to meet together to integrate the sections.

1.4 Suggestion to future teams

The codebase is still rather small and it is feasible to comb through the entirety of it. The main barrier to understanding is figuring out how to properly interact with the API using pyVmomi. Concerning the setup of the project, the setup section should be sufficient for building the project. As for future development, the first steps should be setting up a better test suite. Due to time constraints, the product was mainly tested by hand. However, by mimicking the production environment on the development environment, integration testing should not take too much time. The other point of

consideration is deployment. Although deployment became out of scope for this project, we would highly to future teams to consider using Jenkins.

2 Requirements Document

2.1 Billing Calculation Requirements

The primary aim of the first application is to automate the billing calculation process. The end goal is automatically generating a spreadsheet in a propriety format–the "FUPLOAD" or "FERPA Upload"–monthly for accounting.

There are two sources of data that the application will ingest: virtual machine data that we will scrape from the vSphere API and storage (currently the NetApp network attached storage) data. The storage may be swapped in the future, so the support for storage data needs to be modular in nature. As for the vSphere data, there are two types of billable virtual machines: chargeable VMs and managed VMs. The billing for chargeable VMs and managed VMs (along with storage) will be processed and sent to accounting, grouped based on the configured billing period.

This program will need access to the central configuration file to determine what indexes to bill against.

1. FUPLOAD spreadsheet generation sent to accounting.
    a. Scrape data from chargeable VMs.
    b. Ingest data from storage.
    c. Calculate bill based on (a) and (b). Send report based on configured indexes billing periods.

Since VMs can be turned on and off many times during the month, the application has to operate with relatively fine granularity. As a start, the application scrape data hourly, tracking which VMs are turned on and off (and what resources the VMs are using). This will allow for more accurate billing. For example, if the application were to only run weekly, a user could allocate a huge VM at the beginning of the week then de-allocate the VM at the end of the week to not get charged. The application would store this scraped data into a database, then query the database at the end of a month/quarter to generate the billing spreadsheet.

At the basic level, here are the tasks for the billing application.

1. Scrape data from vSphere for VMs.
2. Scrape data from storage.
3. Design/create a database to store data.
4. Implement billing calculations, querying the database for information.

As for the technology stack, pyvmomi with Python 3 must be used for the main scripts.

2.2 Tag Pushing and Configuration Requirements

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 is likely to be YAML syntax. 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'.

Example YAML Configuration:

```
vars:
  vm_name: "Test"
  tag_name: "Managed-VM"
  tag_op: "tag"
- name:
  vmware_tag_facts:
    hostname: "{{ vcenter_hostname }}"
    username: "{{ vcenter_user }}"
    password: "{{ vcenter_password }}"
    validate_certs: no
  register: tag
```

With this example Yaml configuration file, we will be able to tag the Test VM in Vcenter through our python tagging script. The tag name will be Managed-VM in this case which can be specified as needed. 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.

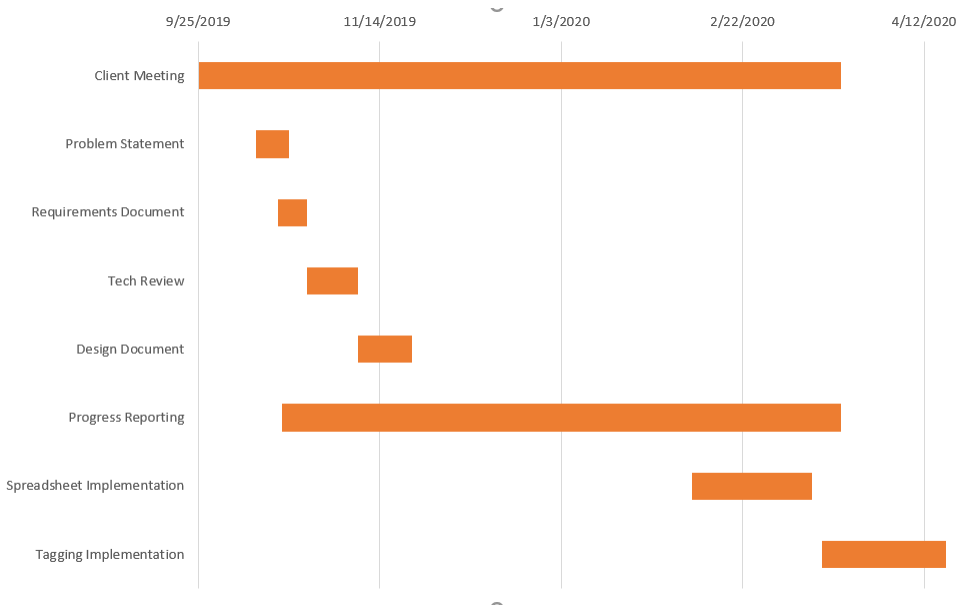


Fig. 1: Finalized Gantt chart for the billing application

Requirement	Change
Deployment	Due to a group member leaving, deployment is no longer in scope
Tagging	The required way to tag the VMs was to loop through the folders in vCenter and look for untagged VMs to tag them. However, our script will tag VMs one by one manually

TABLE 1: Change table for requirements document

3 Design Document

3.1 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.

3.2 Automatic Billing

The Virtualization Management Object Management Infrastructure (VMOMI) backs the network services used to communicate with vSphere products. 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

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.

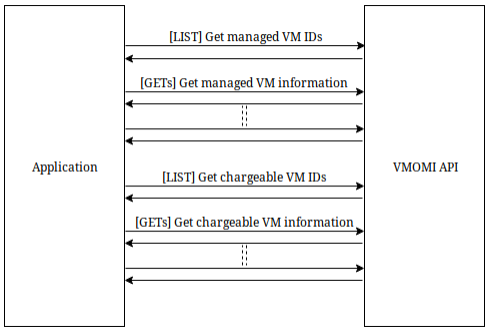


Fig. 2: API call flow for pulling managed and chargeable VM hardware data.

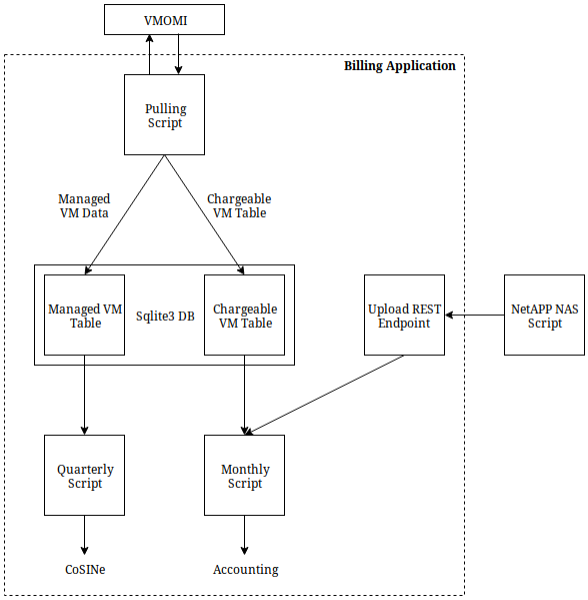


Fig. 3: Overall architecture of billing application.

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.

3.3 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:

```
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 percent and 75 percent 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.

Requirement	Change
Deployment	Deployment is no longer in scope and has been removed from the document.
Tagging	The required way to tag the VMs was to loop through the folders in vCenter and look for untagged VMs to tag them. However, our script will tag VMs one by one manually.

TABLE 2: Change table for design document

4 TECH REVIEW - ANDREW QUACH

4.1 Overview

This document specifies the technology and design decisions made for the billing application portion of the project. That is, pulling metadata about virtual machines that have been deployed by the OSU infrastructure team, and storing

that metadata inside a database to be queried quarterly/monthly.

4.2 Technology Stack: Language and API SDK

Per the client's specifications, we will use Python3 as the main language for this application. The majority of the scripts written by/for the OSU infrastructure team is in Python3 (or in the process of being converted into Python3). Keeping everything consistent in a long-standing, yet still relevant, language will aid maintainability in the long run. Python3 has the stability that comes with a being a well-established scripting language, and its popularity means it will continue being supported and maintained in years to come.

In addition, the fact that Python3 is a highly readable language, and the fact that most of infrastructure team already knows the language, means that it will be easier to transfer ownership of the code-base when the project is complete. This makes Python3 a clear winner among the countless scripting languages (e.g. Ruby, the language used for the current billing script).

An advantage of using a popular language is that there are numerous libraries and frameworks to accomplish common tasks. For this project, in order to collect virtual machine metadata, we will use pyVmomi, the official open-source Python software development kit (SDK) for the vSphere API. Since it is an official SDK, it will garner a lot of open-source support from the Python community and will likely have a long lifespan. In fact, vmWare directly supports the API. This fact ensures that the core technology stack of this application will not become outdated anytime in the near future. As long as Python remains a relevant language, pyVmomi will be supported.

Note that the alternative Python SDK for the vSphere API is pySphere. It was last released Aug 23, 2013 and is only available for the depreciated Python 2. This makes pyVmomi the clear choice to use.

4.3 Parallel vs Serial Architecture

In essence, we will use pyvmomi to pull crucial data about the virtual machines (VM) that are deployed. There are a few specific criteria that determine a how much a VM "costs". These criteria include the amount of storage allocated, the amount of CPU cores allocated, the amount of RAM allocated, the type of VM it is, and the uptime.

The information about the raw specifications of the VM can easily be pulled using the vSphere API (CPU, RAM, Storage, category). Note that the categorization of VM will be set by the tagging application. The main challenge is keeping track of uptime.

Suppose that client wishes to spin up a virtual machine and pay as little for the VM as possible. Since billing happens monthly, a naive implementation would include pulling the VM information every month. However, using this billing system, the client could destroy the VM right before our application runs. Our application would then not detect this VM and the client would be charged nothing.

Alternatively, suppose a client wishes to conserve as much resources as possible. During peak-hours, the client would allocate more resources (vice versa during off-hours). If we up our application to running daily, the hour at which we ran the application would now matter.

The core problem the application faces is the granularity at which we query the API. If we query the API too little, it results in practical exploits/unfairness concerning billing. If we query the API too much, we might hit the API rate limit/struggle to process the data. The application, then, must be built to handle as much incoming data as needed.

Effectively, the architectural decision to make is whether or not to parallelize querying for finer granularity by increasing throughput.

Parallel computing allows us to divide a larger task into concurrent smaller tasks. For this project, the larger task at hand is repeatedly querying the vSphere API. It is possible to break up this task by setting up multiple instances of the application to query the API at different timing offsets. For example, we can set up five instances of our applications running every fifteen minutes, offset three minutes from each other. In this scenario, we would have a fine granularity of three minutes. This is an example of perfect parallelism, where each job is completely independent of every other job.

The downsides of parallel computing include decreased maintainability and predictability due to an increase in complexity. Instead of only maintaining one instance of the application, we would have to maintain and deploy multiple. And even though the problem at hand is embarrassingly parallel, there are a few inter-process issues that could potentially arise (e.g. concurrent database access, concurrent logging).

Parallelization is only worth it to implement if and only if it is necessary. And the necessity of parallelization depends on two factors: what level of granularity is sufficient to prevent exploitation and whether a single process can handle that level of granularity.

After careful discussion with the client, it was agreed upon that a granularity of roughly an hour is sufficient. The error incurred (in comparison to real-time querying) is minimal to a degree that is considered inconsequential in the grand scheme of the entire bill (that is cost-err $<<$ total-bill). Testing showed that a naive implementation of the API call, querying everything, was in the order of minutes (roughly a thousand VMs queried). This means that the throughput of a serial implementation is more than sufficient for the task at hand, even if the number of VMs grows in the future. Optimizing the API call will further ensure that no blocking will occur (blocking will happen if time-to-query $>$ granularity).

The application will use a serial architectural design, favoring the advantages of simplicity and maintainability.

4.4 Technology Stack: Database and Framework

To store the information pulled from the API, we need a suitable database. There were three databases considered for the project: SQLite3, PostgreSQL, and MongoDB. The three databases that were considered span a large variety of database design choices available on the market–from server-oriented to server-less, from relational to document stores.

Living up to its name, SQLite3 is the lightest of the three databases. Its motto: ”Small. Fast. Reliable. Choose any three”. The main architectural difference between SQLite3 and other relational databases is that it is server-less. SQLite3 is an embedded database that can run within the application. There is no need to communicate with another server process. Instead, SQLite3 directly performs reads and writes to disk. The database itself is a simply file on disk. Because of this architecture, SQLite3 is very fast compared to other databases, however it does not handle concurrency very well. SQLite3 uses a lock to handle multiple accesses. In order to write to the database, the mutex must be acquired. This means that if one process is writing to the database, no other process can read/write to the same database. It is for mainly this reason that SQLite3 is not used for larger applications despite its speed, size, and reliability.

PostgreSQL, on the other hand, is a very typical relational database. It uses a client-server model, meaning a database server must be set up and ran over the network. PostgreSQL, being a more heavy duty solution, provides more features compared to SQLite3. These features include XML support, concurrency, indexing optimizations, write-ahead logging, authentication, stored procedures, procedural languages, so on and so forth. Because of the extensive features, PostgreSQL is a very common database choice for large applications in production.

MongoDB is a step away from the traditional relational database design. It instead uses a document model, not a table model. In comparison to the table model, documents are flexible. Each document can store data with different

attributes. There is no central database schema that restricts what attributes can and cannot be present. This results in a more natural representation of data. Data for a specific entity is stored in a single document, instead of spread across many tables..

Based on previous design decisions, there is one database that is the clear winner: SQLite3. The pros of SQLite3 are very advantageous, and the cons are irrelevant considering the proposed the application design. SQLite3 is easy to deploy, small, and fast thanks to its server-less architecture. And since we are using a serial architecture, the poor handling of concurrent accesses is not an issue. On the other hand, the advantages of PostgreSQL and MongoDB are not very useful with regards to this application. The heavy-duty features implemented in PostgreSQL are unnecessary for a simple application. The polymorphic behaviors that MongoDB gives us are also unnecessary, as the data we are pulling from the API has a static structure by nature.

However, to maintain modularity, in the case we wish to swap out databases, we will use SQLAlchemy as the framework for object-relational mapping. This abstracts away the actual database underneath, and allows us to work in our language of choice (Python 3).

4.5 Conclusion

The design decisions made for the billing application focus on being light and simplistic. The design of the application should promote maintainability and avoid unnecessary features and over-engineering.

5 Tech Review - Faisal Khan

5.1 Overview

The primary aim of the first application is to automate the billing calculation process. The end goal is automatically generating a spreadsheet in a propriety format–the “FUPLOAD” or “FERPA Upload”–monthly for accounting.

There are two sources of data that the application will ingest: virtual machine data that we will scrape from the vSphere API and storage (currently the NetApp network attached storage) data. The storage may be swapped in the future, so the support for storage data needs to be modular in nature. As for the vSphere data, there are two types of billable virtual machines: chargeable VMs and managed VMs. The billing for chargeable VMs and managed VMs (along with storage) will be processed and sent to accounting, grouped based on the configured billing period.

This program will need access to the central configuration file to determine what indexes to bill against.

1. FUPLOAD spreadsheet generation sent to accounting. a. Scrape data from chargeable VMs. b. Ingest data from storage. c. Calculate bill based on (a) and (b). Send report based on configured indexes billing periods.

Since VMs can be turned on and off many times during the month, the application has to operate with relatively fine granularity. As a start, the application scrape data hourly, tracking which VMs are turned on and off (and what resources the VMs are using). This will allow for more accurate billing. For example, if the application were to only run weekly, a user could allocate a huge VM at the beginning of the week then de-allocate the VM at the end of the week to not get charged. The application would store this scraped data into a database, then query the database at the end of a month/quarter to generate the billing spreadsheet.

At the basic level, here are the tasks for the billing application.

1. Scrape data from vSphere for VMs.

2. Scrape data from storage.

3. Design/create a database to store data.

4. Implement billing calculations, querying the database for information.

5. Create a deployment script that runs the scraping hourly.

6. Create another deployment script that runs based on configured billing periods, and queries only chargeable VMs and storage, generating the FUPLOAD spreadsheet.

7. Automatically send the spreadsheet to accounting.

As for the technology stack, pyvmomi with Python 3 must be used for the main scripts. My role in this project is to provide with a front end web page that will be used by the client. I will mostly be working with UI toolkits, UI organization and Embedded data visualization on the web page which are described in detail as follows.

5.2 UI Toolkit

Since the project that my team is working on is an enterprise project and will be used by Oregon State Universities Infrastructure department in the future, we need to work in an agile development environment and need to have shared understanding across the board before we start working on the UI for this project. The intended audience will be the whole team which will include: designers, developers, Professors and the client.

There is a number of common UI toolkits available online which can be used such as Twitter Bootstrap or ZURB Foundation which focus on framework of code to pattern libraries. There is a lot of benefits for the enterprise UI toolkit which includes the growth of product to re-use of assets in the application. We need to be really careful with naming of the menus on our front end UI as it is a matter of billing, it can cause big issues for Infrastructure accounts team.

A technology that I am planning to use for this project is Twitter Bootstrap which is really easy to use. It makes it easier to create roles for administration of the site for viewing and editing which will be needed in the future if administration decides to make any changes to User Interface to make it more user friendly. The required abilities that the front end User Interface will need to have will include tags and attributes which will have specific departments under them, ability to show the date and time of recent updates as our project is time sensitive in relation to billing, recent changes made by infrastructure such as adding or pulling managed devices and the historical events and billing information for specific departments.

Other than Twitter Bootstrap, there is enormous amount of other toolkits that can also be used to develop a front end User Interface for this project such as Adobe Comp, Axure and MockFlow but Bootstrap is the easiest and most user friendly toolkit that I have used so far. It is free and contains an open source CSS framework for front end web development. It also contains templates which might be helpful for us for this project.

We will need to focus on the layout of the web app such that the client is satisfied with it. The focus will primary be on the color choice, size and font which catches the attention of human eye for specific option available on the web app for example the font size for managed machines menu need to be more visible as it will be the major and most commonly used part of the web page. Auto complete is also one of the functionality that can be used to auto fill the name of the department or of specific machine in search bar or input fields. Bootstrap provides easy access to all these functionalities to be added on the web page by providing HTML structure, CSS declarations and in some cases JavaScript code. This will be a helpful tool for us that will save some time and make things easier for us.

5.3 UI Organization

Displaying of data on the webpage is also one of the challenges that needs to be met for this project as Infrastructure will be handling the data which will consists of the managed and used resources provided by infrastructure to the departments across Oregon State University. The displaying of the data on the web page will be confidential and needs to be handled with care.

There is different technologies available for the display of data on web page such MySQL or Mariadb database management systems. However, I think that Amazon DynamoDB will be a reliable source to be used for this project as database management system. It provides a number of services that make it easier and renowned around the world. With DynamoDB, we do not have to worry about hardware provisioning, setup and configuration, replication, software patching or cluster scaling. It also offers encryption at rest which excludes the operational burden and complexity involved in protecting sensitive data. We will be able to store and retrieve any amount of data and serve any level of 4 traffic requests. It lets us create an on demand backups and enable point in time recovery for our Amazon DynamoDB tables. Point in time recovery will help us protect our tables from accidental write or delete operations, we will also be able to restore data within the last 35 days with this feature. DynamoDB allows user to delete expired items from tables automatically to help reduce storage usage and cost of storing data which is no longer needed. We will use JavaScript code to display the data from DynamoDB to our webpage. Our main script which will pull the data of usage and managed devices from Vsphere will be written in Python, that data will be added to DynamoDB and with the help of JavaScript code, it will be displayed to the web page. DynamoDB is known for maintaining consistency and fast performance and it also spreads the data and traffic for tables over a sufficient number of servers to handle throughput and storage requirement. All the data is stored in solid state disks and is automatically replicated across multiple availability zones in Amazon web services region providing builtin high availability and data durability which is required by our project.

This is a proposed tool for the project that we might be using but there will be other possibilities too. We will provide this solution to the client to see what they think of it but it can always be changed to something else depending on clients needs and expectations. I have personally never used DynamoDB but after doing research on how to securely save and display the data on to the web page, I came up with DynamoDB after the amazing reviews and its usage by big companies such as Netflix, Lyft and New Relic who trust the technology and store their data on it. Although DynamoDB has big competitors in the market such as Google Cloud Datastore and MongoDB, it is still preferred by big companies.

5.4 UI INTERACTION MODEL FOR EMBEDDED DATA VISUALIZATION

We will need a data visualization tool to display data or create data visualization on our webpage. There are multiple resources available for data visualization but I did some research to find the easier tool that can represent large data sets. The data visualization can be used for a variety of purposes for this project such as annual reports, sales and displaying information virtually anywhere elsewhere it needs to be interpreted.All the data visualization tools available on market have some pros and cons. Some have excellent documentation and tutorials while others exceed in ease of use. The tool that I was looking for in an app is the capability of handling large sets of data or multiple sets of data in a single visualization. It will make our webpage for this project look more decent if it has graphs and maps of the data that is being pulled from the usage of the customers. Different kind of software available online includes Tableau, Infogram,

Chartblock etc. However, I find Tableau most interesting and easy to use for our project. It provides us with hundreds of data import options, Mapping capability and lots of online tutorials available online to walk us through on how to use it. It also lets us to keep data analyses private. Tableau supports complex computations and data blending and transform them in decent visualizations which is hard to derive from spreadsheets.

Tableau is also much more easier to learn as compare to Python or any other data visualization tool. We will be able to provide live data visualization using Tableau which will make it easier for Infrastructure to analyze the usage of virtual machines. This is not the final decision of using Tableau but I did some research and came up with this software to be used for our project. We will finalize the data visualization tool after meeting with the client to make sure that it will completely work with our project. We also have to talk to the client if they even need a visualization tool added to the web page or not. This was discussed as a group but not with the client so I already did my research on it in case if the client wants it included in the project.

6 Weekly Blog Posts - Andrew Quach

Fall Week 4: We have finalized all the requirements with the client (for both services we plan to create: billing calculation and tag pushing). We still need to set up our accounts and get access to the APIs, however that will be a discussion for next week. We work with the client to start drafting architecture plans in the near future.

Fall Week 5: We have successfully figured out the technology stack that we plan to use. We foresee no issue with the stack at this time. We plan to get everything reviewed by the client next week.

Fall Week 6: We met with the client and finalized our technology stack. We have the majority of the architecture mapped out as well now. There are no problems at this moment. We plan to finish the design document and have the client review it by next week.

Fall Week 7: We've discussed how to deal with our team member leaving. We talked with the client on how to deploy our application (the part the team member was supposed to take over). Hopefully we can get another person Winter term to join us in development.

Fall Week 8: We finished up our design document and we are planning to give them to the client to look over. We have a meetig with the client December 2nd for a final review.

Fall Week 9: Gave the design document to client for review. Planning to discuss it next week and get it signed off.

Winter Week 1: We reunited with our client and completed the team evaluation. We plan to schedule a meeting with our client next week. No problems this week.

Winter Week 2: Began the implementation of the project. Having minor issues with API but it should be resolved eventually. Planning to meet with client next week to discuss progress.

Winter Week 3: Continued development on Capstone project. Planning to this weekend for a sprint. Planning to meet with client to discuss specifications next week. No problems as of now.

Winter Week 5: Continued development. Reversed the API. Plan to finish up project for alpha.

Winter Week 6: I began to polish up the spreadsheet output itself. I plan to have everything polished up on my end by the design review.

Winter Week 7: Made further progress on my section by polishing up the spreadsheet output.

Winter Week 8: Continued development on my spreadsheet section. I'm planning to help with the tagging section this weekend.

Winter Week 9: Started focusing more on the demo for the design review. Polished up some potential loopholes with the scraping script. Talked with partner how his section. Project seems a little more on track now.

Winter Week 10: Was out most of this week for an interview. I'm currently working on creating the video + final report.

7 Weekly Blog Posts - Faisal Khan

Progress Report 1

We met with the client last week and they walked us through the 10,000 foot view of the project giving us the basic details and told us the major requirements. Progress: After meeting again this week, things got a little more clearer. We will be able to write a draft in a more clear manner as compare to the problem statement that we turned in last week. We will show it to the client and get it approved and signed off by them before we turn it in. All of the group members got their jobs assigned accordingly for writing a draft. Problems: We are still facing a little problem regrading the meeting time with client since two people in group are pretty much booked every week day. However, we are planning to shift meetings with client to teleconference unless it is necessary to meet in person Plans: A lot of things got clearer after the second meeting but we plan to meet them again next week to ask further questions that we are sure are going to come up once we start writing a draft so the plan for next week is to meet with the client and ask further questions that comes up.

Progress Report 2

Progress: After writing the requirements document, the project got much more understandable. We met with the TA and setup a time to meet every week. We also planned to meet at teleconference just in case everyone is busy and can not meet in person. We split the task for the requirements document and it was easier to get it done that way. Problems: We worked on the requirements document this week and it took a while because no one in the group was familiar with the code in latex. However, we ended up completing the document and turned it in. Plans: We will meet with the client earlier next week and show them the requirements document draft 1 and double check that we got everything correctly. We will get clients feedback and make changes for Draft 2 accordingly.

Progress Report 3

We were unable to meet with the client this week since all of us got busy but we met last night and planned to meet with the client next week before we turn in the design document so our client can give us a feedback on anything that needs to be taken out or missing from the design.

Progress: We met on a teleconference for tech review so we know what each person is working on just in case we overlap on each other work. Specific functionality is decided by each other to work on tech review. Problems: One person is taking on the coding part to pull the data for the project and two people will be working on the Infrastructure part of the project. The problem is we need to figure out which part of infrastructure each person will work on for the tech review. Plans: As I already mentioned, we will be meeting next week to make a rough design and then meet with the client to get the feedback so all this is planned for next week to make sure that we get rid of mistakes in a first design document.

Progress Report 4

We met with the client to talk about the design document. We also figured out how Microsoft teams work as Teams is the direst and quickest way for the client to communicate and respond. The client created a license for all of us on

Teams so we can now access it. Progress: Met with the client and talked about the design document that we will create over the weekend and discuss with the client by Tuesday. Problems: Meeting time between all of us is still an issue but we have move from in person meeting to teleconference which works for all of us. We will meet when it is really necessary to meet in person such as creating design document. Plans: Plan is to get done with the design document early next week so we have enough time to find out the problems and issues for the final design document.

Progress Report 5

We had a meeting with a client and found out if we are missing anything for the design document. Worked on a first draft of design document for the peer review Progress: Started the rough draft for the design document for the peer review. Problems: We are short of one person in the group who had to work on the deployment part. We are talking with the client and professor to sort it out Plans: Plan is to finish up the draft for design document by the end of this week.

Progress Report 6

This week, we were able to finish the design document and met with the client to setup a meeting for a design document sign-off. They can meet with us a day before a signed off design document is due. We will go through the design document together and if they think there is still things that are missing and needs to be added, we will review that with them and add it so they can sign the document off and send it to Kirsten or Scott. Progress: Design document is completed and draft 1 is ready to be turned in. Meeting with the client following week to get the document signed off. Problems: We had some problems with the deployment part of the project, but things got a little clearer after meeting with the client and we were able to put together a design document. Plans: Plan is to keep researching about the deployment tool that the client has suggested us to use so we can build a solid base with it and do not need to worry in the end.

Progress Report 7

we sent the design to the client so they can take a look at it when they get time and proofread it before sending it to the professor. Progress: Sent the design to the client for proof reading and so they can sign it and send it to the professor for grading. Problems: We just have some problems with the deployment part of the project for which we are doing some research and looking at some tutorials. Plans: We are planning to Keep doing the research and reflect it with the design so we can change the design in case we get something wrong. Also, keep working with the deployment part and learn everything that we need to.

Progress Report 8

After completing the fall term, we were able to figure out what exactly are the requirements for the project that we will working on with the infrastructure team. I do not have much experience with Python3 Pyvmomi VMware vSphere Python API that we will be using in this course but I have started looking at the tutorials and PDF's to learn about it so we can start working on the project as soon as possible. We have done the first critique and I was amazed by the exact feedback I got from my teammate. Progress: I am working on learning Python3 Pyvmomi VMware vSphere Python API and will soon start to implement the code. Problems: I have a busy schedule this term so it will be hard to meet with the TA and be in class regularly, but I am working on it. Plan: Plan is to start working on the project as soon as possible so we can have a basic structure of the project for the client to review.

Progress Report 9

I tried to do as much research on creating a YAML configuration file according to what is required. Since I have not worked on YAML before, it is taking a little while for me to learn how to create a YAML file. Planning on meeting

with the group and the client next week to discuss all the questions that we have. Progress: Research going on about languages that is needed for the project Plans: Plan is to meet with the group and the client next week to discuss any questions that we have Problems: As of now in this term, we are not facing any problems and hope that we get to do all the required stuff on time.

Progress Report 10

Our plan was to meet with the group and the client this week but due to heavy workload, group meeting got scheduled to the weekend. We will plan on how to move forward from here in the meeting and when will be the good time to meet with the client. Progress: My personal progress is I have learned how to work on YAML and will start to create a configuration file that is required for the project. Will ask question from the client if I come up with any related to the implementation. Problems: As of now, I am a little confused on how to check if my configuration file is correct. I will ask this question from the client when we meet. I also have to ask my group mate about his progress and what's the plan of action from now on will be. Plans: Plan is to get done with the project on this term and work on the testing part in next term. Hopefully, we will be able to achieve our goal.

Progress Report 11

I spent time with developing on the project. I was able to create a Yaml script which I will parse in to the Python code. I will have to meet with the group mate to make sure that we are doing everything correctly. Progress: Able to create a Yaml script for a specific department Plans: Plan in to meet with the group and parse the script in to the code Problems: As of now, I do not have any problem with the project but it is important to meet with the group next week.

Progress Report 12

We got quite a bit done this week. I was able to write a Yaml script to tag the departments that Infrastructure manages or charge. I found out that we do not have access to create random VM's to test our code so we met with the client and let them know. They have setup the permissions now so we can set up VM's and play around with it. Progress: Wrote a Yaml script and parsed it in to Python code. It worked and now we can setup VM's to play around with it and try tagging the departments. Problems: We did not have permission to set up the VM's in vSphere so we talked to the client and they looked in to it and gave us the permissions. Plans: Plan is to work on the poster and come back to the development part of the project. Have to complete the tagging part and meet with the client next or the following week.

Progress Report 13

This week I tried to do as much research as I can to get the Yaml configuration written for the project. I have most of it figured out but still have to put in data in the Yaml. I am working on creating, editing and deleting Tags. I am learning on how to work the RESt API to create the Tags. I will try to get most of it done by next week. Progress: Wrote the Yaml script and python code to parse the Yaml to. Problems: Still have to learn how to use the RESt API to create, Edit and delete Tags which is required for the project. Plans: Plan is to get done with the Yaml and start creating the Tags.

Progress Report 14

This week I figured out how to create, edit and delete tags. Now the next step will be to figure out how to incorporate the yaml script in to the Python script to it loads all the data about the departments. The script to create tags was not giving any errors but it was not creating tags on Vmware because I was off campus and need to setup proxy. I will figure that out and make sure that it works by tomorrow. Progress: Script to create, edit and create tags works and the yaml script is working which contains all the data about the VM's Problems: The tags script was working but need to setup proxy to create test tags in Vmware Plans: Plan is to find out how to parse Yaml script in to the tags script so it loads all the data in test tags.

Progress Report 15

This week I was able to create the actual directory structure of the Vcenter. We did the design review and was able to showcase what has already been done so far. Next step is to develop code to read the directory and do the tagging. Progress: Design review is done and yaml script for actual directory structure has been formed. Problems: Need to come up with script to parse Yaml, create tags and push the tags. Plans: Plan is to create the script and get done with tagging in this week.

Progress Report 16

This week I was able to setup dev environment and do development. I am working on creating and pushing tags but coming up with some errors that needs to be fixed. I will try to fix it during the weekend and send it to the group mate so he can further work on it. Progress: Yaml database is created and we are able to scrape data from Vcenter database. Problems: Coming up with some errors in my code to create and push tags. I'll have to go through VMware documentation to figure out what I am doing wrong and fix the bugs. Plans: Plan is to fix the bugs during the weekend and send it to my group mate so he can further work on it.

COLLEGE OF ENGINEERING

FAISAL KHAN

ANDREW QUACH

CLOUD INFRASTRUCTURE BILLING

Automating virtualization resource billing

PROBLEM

Parts of campus are allowed to self-provision these virtual machines without any middle man. Consumers can reserve any amount computing resources they deem necessary.

This pushes the burden of tracking usage to IT infrastructure. Currently, the billing calculation process is a mix of automated and manual procedures.

To complicate things further, not all virtual machines are treated equal. Different parts of campus are charged different rates. Some are not even charged at all.

REQUIREMENTS

Continuously scrape data from vSphere for VMs and scrape data from storage and store in a database (hourly).

Generate a spreadsheet based on scraped data in FUPLOAD format and send to accounting.

Automatically push tags to VMs to add in spreadsheet calculations based on user-changeable configuration file.

Created with pyVmomi and python3 technology stack as base.

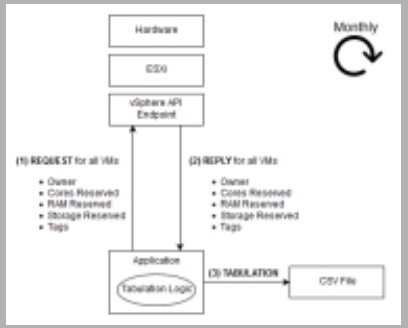
DEVELOPERS

- Andrew Quach: quachand@oregonstate.edu
- Faisal Khan: khanfa@oregonstate.edu

CLIENTS

- Majorie McLagan: Majorie.mclagan@oregonstate.edu
- Stacy Brock: brocks@oregonstate.edu

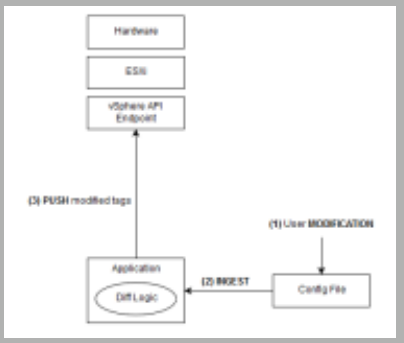
BILLING DESIGN



From the vSphere API, we will extract all necessary information to calculate billing amount. This API call mechanism (1) and (2) will happen every hour and stored in an sqlite3 database.

After pulling this information, the bill is tabulated for all VM owners and outputted to a CSV file. This (3) happens every month. The output file is automatically sent to accounting.

TAGGING DESIGN



The billing application requires some user inputted data. This input will be in a YAML configuration file. The tagging application continuously monitors for VMs without tags and updates the VMs with metadata about the index used and whether it is managed.

Oregon State University

8 Project Documentation

In the repository, there are two main folders: tagging and spreadsheet.

The tagging folder consists of the scripts necessary to tag a virtual machine. The main script is tag_op.py which performs allows for the tagging/untagging of a virtual machine in vSphere. It internally uses the Python SDK for VMware vSphere. The script has currently been tested on the development environment. It is not yet runnable in production without further testing. Tagging information is stored in the yaml file. In order to run the script, first place a configuration file called settings.py in the following format is needed.

```
host = 'vctr-vd01.sig.oregonstate.edu'
user = '<replace with real username>'
password = '<replace with real password>'
port = 443
url = 'https://{}/api'.format(host)
```

Then, run the tagging script with the following command:

```
python3 tag_op.py <tag/untag> <tag_name> <vm_name>
```

The spreadsheet folder consists of three scripts: a scraper script (scraper.py), a spreadsheet generation script (generate.py) and a database helper script (db.py). The scraper script uses the API in order to query all the necessary information and store it into a SQLite3 database (with the help of db.py). The spreadsheet generation script, then, pulls information from the database to generate a billing spreadsheet.

The scripts currently operate on the production servers. (It does not mutate data, meaning it is safe to run without testing.) When deployed, scraper.py will be ran in 15 minute increments. generate.py will be ran monthly/quarterly depending on whether it is managed or chargeable.

In order to run the script, first place a configuration file called settings.py in the scraper directory.

```
host = 'vcenter-vp01.sig.oregonstate.edu'
user = '<replace with real username>'
password = '<replace with real password>'
port = 443
```

Then, run the scripts with the following commands:

```
python3 scraper.py
python3 generate.py
```

The scripts require a few dependencies that can be installed using pip. Use the following commands to set up the dependencies.

```
$ sudo apt-get install python3-venv
$ python3 -m venv billing-env
$ source billing-env/bin/activate
$ pip install --upgrade pip setuptools
$ pip install --upgrade git+https://github.com/vmware/vsphere-automation-sdk-python.git
```

```
$ pip install —r requirements.txt
```

One important note is that the development and production servers are only accessible when you are on the OSU network. So make sure you are on OSU VPN when running the scripts.

9 Recommended Technical Resources for Learning More

9.1 Helpful Websites

1. www.Github.com
2. docs.vmware.com
3. www.virtualizationreview.com
4. www.youtube.com
5. www.stackoverflow.com
6. www.vmware.com

9.2 Helpful Reference Books

There was not specifically a reference book need to learn about Vmware VCenter but the docs available at Vmware website were helpful. All the modules and functionalities are very clearly defined and helped us greatly to better understand Vmware VCenter modularities.

9.3 Helpful People On-campus

There was not much help needed after we went through the documentation and tutorials available online. However, the class TA and client cooperated greatly whenever we got stuck or had questions about specific functionality. Since our client was also from the technical background, they were able to provide technical guidance which was really helpful.

10 Conclusions and Reflections

10.1 Andrew Quach

Undertaking this project was definitely a difficult exercise in both hard skills and soft skills. In undertaking the project, my ability to read documentation has increased significantly. The vSphere API documentation was difficult to navigate at first. However, towards the end of the project, I felt very comfortable parsing through the specifications, quickly being able to locate what I need in moments. I've also not had the opportunity to use anything vSphere related in the past, so this project was a good first attempt at learning an in-demand technology. Figuring out how to architecture the project was also another great learning experience. In classes, the projects are pre-structured for you, so having control over the design was different.

As for the soft skills, communication was definitely an aspect that improved over the course of the project. Scheduling meetings, making sure everyone is on the same page–all these soft skills were in rough shape at the beginning of the term. However, over time, the team was able to be on the same page more consistently. Working on a year long group project definitely facilitates the building of teamwork.

In finer detail, I realized the importance of clearly dividing the work. Days of planning definitely saved us the effort of weeks of work. For larger projects, I learned that a lot of the work is done before the coding actually begins. Designing

the project, breaking down the project to small components, building a schedule–these tasks are crucial to successfully finishing a project. This is something I never appreciated until now.

This idea of project management is a skill that is difficult to learn in school. Typically, in a work environment, project management is handled by those with more experience, so planning out everything was treading new water for us. We did our best in planning how we wanted to tackle the project, however without feedback, it was hard to tell if we were doing it effectively. Nonetheless, even if the plan was not perfect, it definitely did help in the long run.

Working in a team this term definitely has made me appreciate when a team leader manages to have everything run smoothly. We ran into a lot of hiccups this year (from a member leaving, to capstone inhibiting our meetings), so there were some difficulties in effectively working as a team. However, the experience was definitely fruitful. Learning to deal with difficulties is something that is necessary in a corporate environment. I think my main takeaway is that a team should consistently communicate what they are working on, so everyone has a clear idea of how the project is progressing.

Overall, I think if I were to do this project again, I would take a more active role in the project management side of things. In our group, sometimes we both were too content of working on our own little part of the project, that it was difficult to integrate our ideas together. We often times had different ideas of how our implementation was going to look, so a lot of extra work was done that might not have been necessary if there was clearer communication. But, I attribute this to lack of experience with project management. I think if we were to do it again, we could definitely do a better and more thorough job.

10.2 Faisal Khan

Since this project was related to computer science and specifically cloud computing involving APIs, there was a lot of technical information to grasp out of this project. I got the opportunity to learn greatly about cloud computing and how the virtual machines are used to store data on the cloud. We specifically worked with billing virtual machines that are used by the departments all over OSU and managed by Infrastructure. I was able to learn how to implement a functionality which was to bill the departments according to the usage automatically on Vmware VSphere that is used by infrastructure as a cloud computing virtualization platform.

The Non-technical part that I was able to learn out of this project is how to follow through the process and fulfill the requirements one at a time to finish a project like this. We started with problem statement to design document and writing technical reviews to make sure that we meet all the requirement and leave less chance for mistakes to happen. Furthermore, I was able to learn how to meet the deadlines and how to prepare questions before meeting with the client.

After working on this project for three consecutive terms, I was able to learn how to run how to undertake the project and fulfill the requirements. The way senior capstone is designed, we were able to figure out all the essential requirement before started implementing the code. Leading the work of a functionality to achieve goals and meet success criteria at a specified time was one of the biggest challenges. However, since this was the first big project of my life, I accomplished and learned substantially about project work and project management.

Another thing that this project helped me learn is working in teams which I believe will benefit me in the future during professional life. The reason why teamwork taught me differently specifically during this project is because my team member helped me learn about cloud computing because of his prior skills working with it. This was a whole new sub-world for me in the field of computer science and it would have taken me more time to learn if I had team members

who were new to cloud computing as well. Helping each other out taught us conflict resolution skills and stay well organized during the whole course of action which led us to at least finish major tasks if not fulfill all the requirements.

We learned numerous things that can be avoided in the future or could be done differently if we could do it all over again. Great amount of time can be saved if we could be more proactive and meet with the client every other week to make sure that we are giving them updates on what has been done. This will also help us to stay on track and ask questions if stuck. Another thing will be to divide the project in sub sections and setup deadlines to finish them off. Since there were no specific deadlines to finish the implementation, it gave me a chance to procrastinate until the last moment. In conclusion, I will be more proactive and finish things off faster if I could have a chance to do it all over again.

11 APPENDIX 1

Below is an example of a test Yaml file which contains required information about a virtual machine.

```
vars:
    vm_name: "Test"
    vcenter_hostname: "hostname"
    vcenter_user: "user"
    vcenter_password: "<password>"
    vcenter_dc: "dc1"
    tag_name: "Managed_VM"
    tag_op: "tag"
```

These are all the required libraries and bindings to implement the tagging functionality.

```
import ssl
import time
import sys
import requests
from com.vmware.cis.tagging_client import (Tag, TagAssociation)
from com.vmware.cis_client import Session
from com.vmware.vapi.std_client import DynamicID
from pyVim.connect import Disconnect, SmartConnect
from pyVmomi import vim
from vmware.vapi.lib.connect import get_requests_connector
from vmware.vapi.security.session import create_session_security_context
from vmware.vapi.security.user_password import create_user_password_security_context
from vmware.vapi.stdlib.client.factories import StubConfigurationFactory
from settings import host, user, password, port, url
```

Pseudo-code to find the VM(virtual machine) ID and tag the VM using it's ID.

```
def get_vm_id():
```

```
    for vm in vms:
        if vm.name == name:
            return vm._GetMoId()


def tag():
    vm_moid = get_vm_id()


    if tag_op == "tag":
            if tag_id == tag_id:
                tag_attached = True
                break
        assert tag_attached
    elif tag_op == "untag":
```

This is the core tabulation logic:

```
vCPU = 38
RAM = 28
OSDisk = 1.5
CapDisk = 0.09
WinLic = 24
SvrMgmt = 91
Month = 30 * 24 * 60 * 60
def tabulate_row(row):
    total = 0
    total += row[2] * vCPU
    total += row[3] * RAM
    total += row[4] * OSDisk
    total += row[5] * CapDisk
    if 'windows' in row[7].lower():
        total += WinLic
    diff = row[11] — row[10] + (15*60)
    total *= (diff / Month)
    return total
```

This is how to determine whether we are using fast storage or normal storage.

```
fast, slow = 0, 0
for device in vm.config.hardware.device:
    if type(device).__name__ == 'vim.vm.device.VirtualDisk':
        size = hrsize2b(device.deviceInfo.summary) / 2**30
        if 'fast' in device.backing.datastore.name:
```

```
            fast += size
        else:
            slow += size
fast, slow = round(fast, 2), round(slow, 2)
```

We have to backtrack to figure out the department name.

```
pfolder = vm.parent
while pfolder.parent.name != 'vm':
    pfolder = pfolder.parent
```

12 Appendix 2

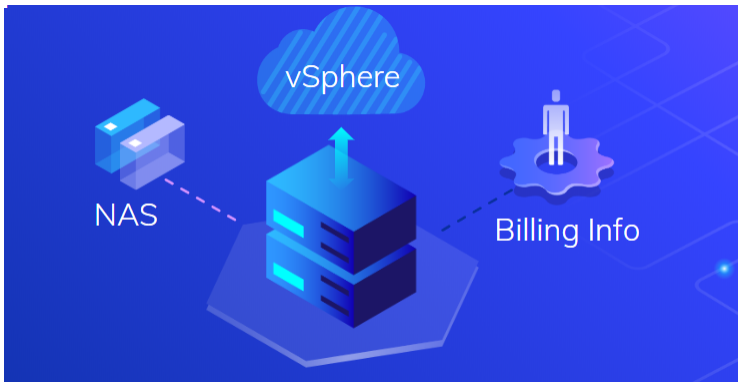


Fig. 4: Data Aggregation

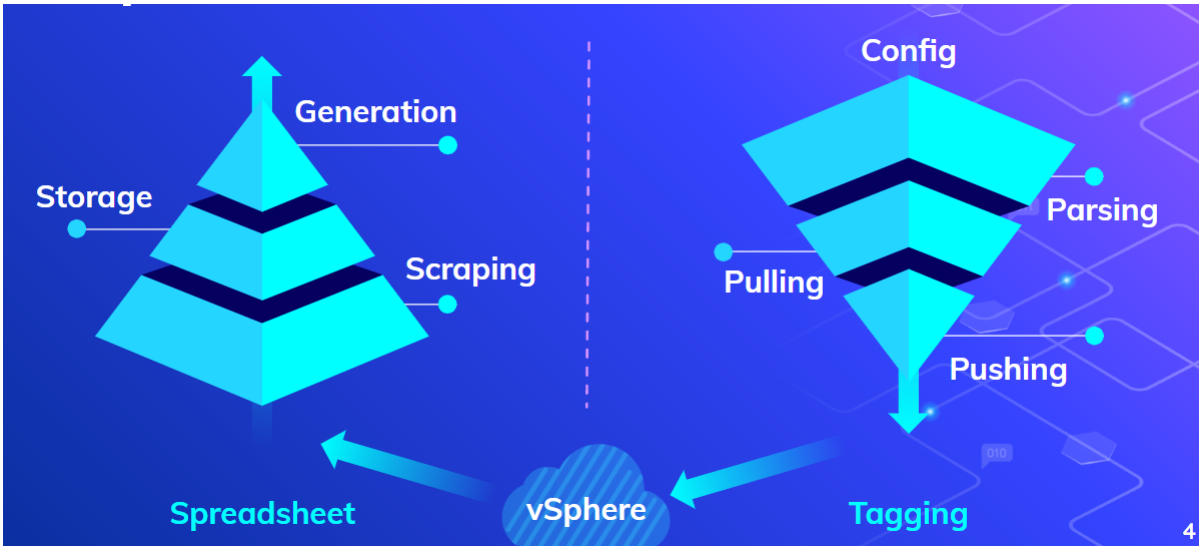


Fig. 5: Pyramid Flowchart

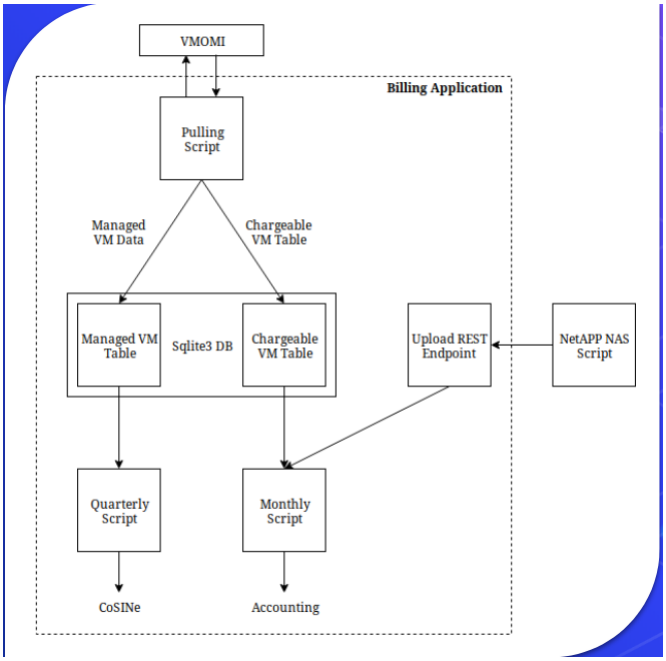


Fig. 6: Billing Application

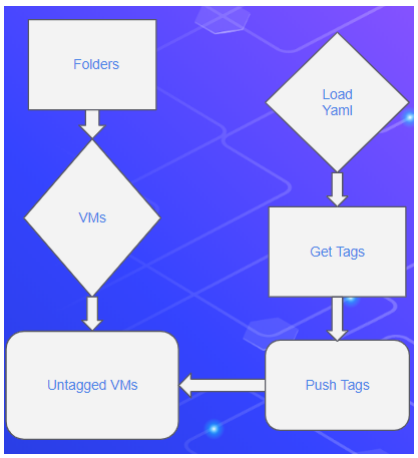


Fig. 7: Tagging Flowchart

13 Appendix 3

Review 1

Cloning and running the code took some time. I had to figure a couple of things since the environment setup didn't have any details except the commands. The READ ME file was helpful either way. It was useful since it had some instruction and information about a couple of files and what their tasks.

The code followed the general guidelines and style. The good thing about this project that the majority of the files were short so the code could be easily be represented and styled. It just makes everything make sense. There are multiple READ ME files. I found each directly has one which I believe it is helpful with a complicated project like this where you need to understand what's going on before you compile it. Otherwise, it would be hard.

The code implemented really good in the way that it was written. Looking to some files as scraper.py and generate.py,

there are very nice and organized well. I think it was helpful that there was not a lot of code involved in this project.

The unit test is built into the code which is good. It gives me a good idea as a tester or user about what the project should perform. Sometimes I feel like I wish if one of the members of the group walks me through the testing to understand some of these numbers. In general, they are there and could be understood better with some maintenance.

I wish if the requirement file had more information and more details about the project. I find it to be way short for me to grip the project idea and where their final destination ends. More information about the group teams would be good. Good job, on a cool project!

Review 2

Build: It seems like there might be some backend restrictions on running the program, but the documentation on the readme was clear and straightforward. The team provided step by step guides and specified the division on the project files to reduce confusion.

Legibility: Yes, the code is simple to understand and follows conventions developing good software. The team has also made their code variable easy to understand and straightforward. Looking through their code files as well, variables follow naming conventions and are consistent.

Report: The team also named the different project files with easy to understand conventions and provided a readme file for both parts of the project.

Implementation: The code is much more efficient than what it used to be, the team has managed to move everything to python making things much faster and much easier to read. It also, again, standardizes the projects languages.

Maintainability: The team mentioned that they are able to do visual testing checking their changes and seeing if they are posted to the database.

Requirements: The team's code satisfies all requirements.

Review 3

Build:

3 I could clone the repo, but I had some trouble getting it to run. It gave me an error on the sudo apt-get saying that I cannot run sudo. I tried on the OSU EECS server. I took one point off because the instructions were pretty vague for getting the code to run. Legibility:

4 The code looked great. Variable names were precise. The code was also incredibly clean. It looked very precise and that it had been refactored. It also adhered to Python's style. Implementation:

4 The team makes good use of functions. The functions abstract out processes. It makes the code easier to follow. The only thing I would recommend is adding some comments to the functions. That will allow people to have a better understanding of what the code does. Maintainability:

4 I did not see any unit tests, but I don't feel that the team needs any. This script is fairly small and does just a few things. It would be easy to tell if it wasn't doing the correct thing. They also stated in the code review that they are able to check other sources to make sure the code is working correctly. Requirements:

4 Yes, the code fulfills the requirements.

Review 4

OK– so I had some trouble cloning and running this project. I did connect to a VPN according to the instructions in the README file. I believe the README file would be a lot better if it had a little more information about the project itself and how to install and run it on your local machine. However, looking through the two files (scraper.py generate.py) I could easily understand the implementation of the code because of the way it was written and organized.

It seems like this project did not involve a lot of coding, up to this point and that might be the reason I was able to follow it. Building the unit tests for the project was a good idea to make sure the project performs and runs the way it is supposed to.

Review 5

Build: I was unable to fully run the application(s) as I don't think I have permissions to download the necessary software on the engineering server. Also, I had trouble telling when commands and scripts did run properly.

Legibility: This is some of the cleanest code I have seen in a long time, and while some variables are named very simply and their purpose was a bit vague, I believe that their simplicity fits in sufficiently with how succinct the scripts need to be. One suggestion would be to add more comments to make the scripts easier to understand.

Implementation: The code is already pretty concise, so the only useful abstractions may be further replacing values in the code with constants so that they can be even more easily configured in the future.

Maintainability: There are no unit tests, but I don't think they need to be added with this kind of project.

Requirements: Yes, the code fulfills the requirements.

Other: Nothing negative stands out to me in the code, but I am still thoroughly impressed by how concise the scripts are and how they still fulfill all the requirements necessary for the project. Amazing job!

Review 6

Build: I could clone the repository easily. I could not, however, complete the environment setup for the script. The first line we are told to run for the environment setup contains "sudo", which our student accounts do not have permission to use on the engineering servers. The next two lines of environment setup worked fine, but the last one failed with the following error: Could not find a version that satisfies the requirement nsx-policy-python-sdk==2.5.1.0.1.15419398 (from -r requirements.txt (line 7)) (from versions:) No matching distribution found for nsx-policy-python-sdk==2.5.1.0.1.15419398 (from -r requirements.txt (line 7)) This prevented me from preparing all the requirements necessary to run the scripts.

Legibility: Application functionality was split up into folders and files that made sense. All files contained small amounts of code and were easy to follow. Variables were given names that made sense and comments were prevalent. The two separate application functionalities (spreadsheet and tagging) were each given their own readme's that helped me understand their purposes. Overall, excellent!

Implementation: All functions are very concise and modularized. Work is separated into units that make sense. For example, the "tag" function in the file "tag_op.py" uses the functions "get_vm_id" and "get_tag_association" to tag a virtual machine with appropriate billing info. The longest function (tag) was only 23 lines long, which is pretty concise.

Maintainability: There are no unit tests. Since there are very few functions (or units), I do not believe that unit tests are necessary. It doesn't seem difficult for the units to be judged on their functionality through observation. All that said, unit tests would not hurt the quality and maintainability of the code. I just believe that they would only give very small returns compared to the amount of time it may take to create them.

Requirements: The code does fulfil the requirements. I do not believe there is much more to be said here. I did not get to run the code myself, but I was present at a demo they gave and it looked like the script was successfully gathering the billing information necessary.

Other: I think that the developers went above and beyond with refactoring their code. I can't see anything that needs improvement with the code. The environment setup, however, could be updated, as I was unable to follow the procedure that they gave.

Review 7

Build: Yes. I was able to clone it but not starting it. It was my terminal problem. Isn't clear on the pre req/requirement of doing the first command

Legibility: It was clear the functionality of each part of the code. It was organized.

Implementation: I am not really familiar with cloud development in python, but the comments help me to understand

Maintainability: I don't think they have unit test.

Requirements: Yes, it follows the guildeline and is well organized.

Other: The instruction in README could be more clear.