Cloud Environment Manager

Design Document

Adis Osmankic - Team leader

Zane Seuser - Cloud Architect

Jet Jacobs - System Architect

Rishabh Bansal - Report Manager

Gavin Monroe - Meeting Scribe

Team Number: sdmay21-39

Client: PwC

Adviser: Lotfi Ben Othmane

Team Email: sdmay21-39@iastate.edu

Website: https://sdmay21-39.sd.ece.iastate.edu

Last Revised: Nov 15, 2020

Executive Summary

Development Standards & Practices Used

- Agile Development
- Cloud IAC (Infrastructure As Code)
- Modular Code
- Well-Documented Code

Summary of Requirements

- Simple User Interface
- Easily Create & Manage Any Amount of Lab Environments
- Able To Provision Labs On Multiple Cloud Providers

Applicable Courses from Iowa State University Curriculum

- COM S 309 (Software Development Practices)
- COM S 319 (Construction of User Interfaces)
- COM S 362 (Object-Oriented Analysis & Design)
- COM S 363 (Database Systems)
- SE 329 (Software Project Management)
- SE 339 (Software Architecture & Design)

New Skills/Knowledge Acquired That Was Not Taught In Courses

- Knowledge Of Different Cloud Providers
- Knowledge Of Infrastructure As Code
- Python Development
- React Development

Table of Contents

List of Figures/Tables/Symbols/Definitions/Diagrams

Definitions:

AWS	Amazon Web Services
ECS	Elastic Container Service
ECR	Elastic Container Repository
GCP	Google Cloud Platform
IAC	Infrastructure As Code
UI	User Interface

List of Tables:

1 Introduction

1.1 Acknowledgement

Thank you to our client PwC, and our contact, Matt Weidman, for the technical support and architectural guidance. We would also like to thank Lotfi Ben Othmane, our advisor for this project, for helping us through the planning and development process.

1.2 Problem and Project Statement

Currently, our sponsor, PwC, provisions lab environments for capture the flag events within a variety of cloud providers and on-premise resources by hand. This does not scale well because there is no way to quickly create and manage a large amount of lab environments. There is also no way to easily make sure that all these environments are the same.

By developing the Cloud Environment Manager application, we hope to provide PwC with a clean and simple user interface that allows them to quickly and easily deploy lab environments to their desired cloud providers. The application will also allow them to manage and destroy their existing lab environments once they are no longer needed.

1.3 Operational Environment

This main application will be hosted within Amazon Web Services. The front-end application will be accessible from a web browser and the back-end API it interfaces with will be able to deploy lab environments to different cloud providers such as AWS, Azure, GCP, etc.

1.4 Requirements

- **Functional Requirements**
  - A user shall not be able to have access without signing in via Google OAuth2
  - A user should be able to sign in to the application
  - A user should be able to view a list of existing lab environments
  - A user should be able to view the status of existing lab environments
  - A user should be able to view attributes of existing lab environments
  - A user should be able to create lab environments within AWS, GCP, and Azure
  - A user should be able to destroy existing lab environments
- **Non-Functional Requirements**
  - The application should be available at all times
  - The application should properly handle errors behind-the-scenes
  - The user interface should be visually appealing
  - The user interface should be easy to use

1.5 Intended Users and Uses

This project is designed for use by an organization that needs to deploy lab environments to different cloud platforms for a variety of different users. This project is focused on education and deploying capture the flag events to large groups of people for training. By using this product, organization admins will be able to deliver training resources to members of the organization quickly and easily.

1.6 Assumptions and Limitations

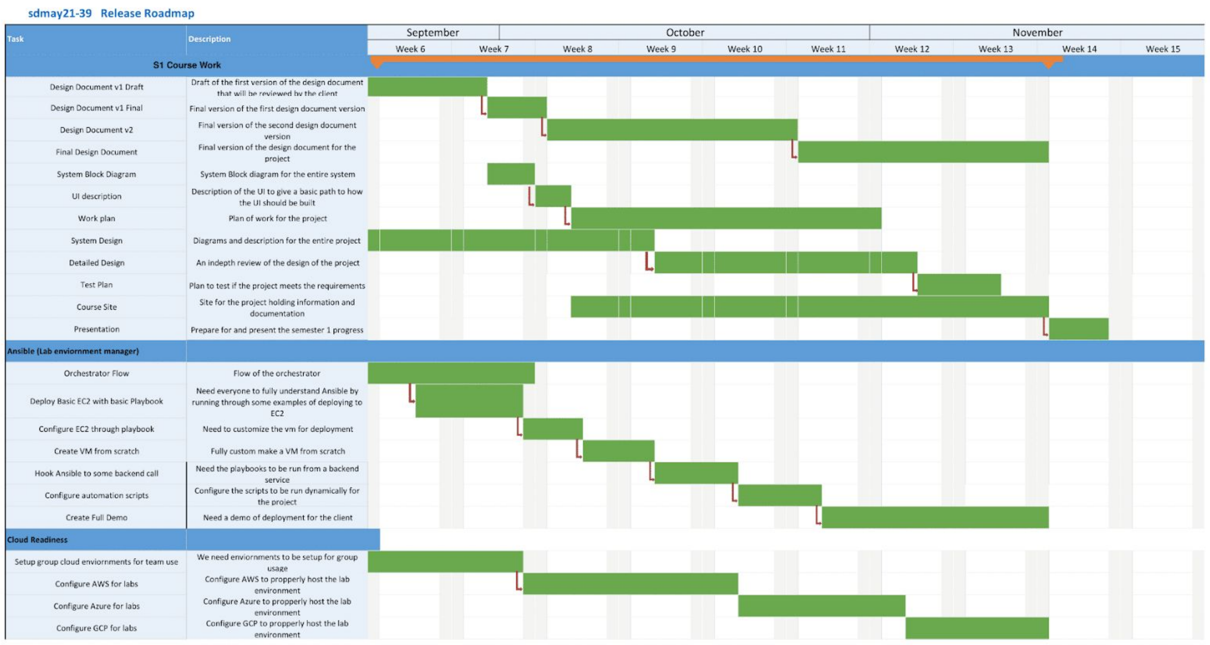
- **Assumptions**
  - The organization has a cloud platform
  - The organization has the resources to host the user interface and backend
  - The organization can configure the lab environments to accept programmatic access to the virtual machine resources in the cloud
- **Limitations**
  - There are some configurations that need to be manually changed by an organization
  - Only administrators can set up the application

1.7 Expected End Product and Deliverables

- Cloud-Hosted Cloud Environment Manager And Source Code
- Project Documentation
- May 2021 Hand-Off

2 Project Plan

2.1 Task Decomposition



Task	Description
S1 Course Work	
Design Document v1 Draft	Draft of the first version of the design document that will be reviewed by the client
Design Document v1 Final	Final version of the first design document version
Design Document v2	Final version of the second design document version
Final Design Document	Final version of the design document for the project
System Block Diagram	System Block diagram for the entire system
UI description	Description of the UI to give a basic path to how the UI should be built
Work plan	Plan of work for the project
System Design	Diagrams and description for the entire project
Detailed Design	An indepth review of the design of the project
Test Plan	Plan to test if the project meets the requirements
Course Site	Site for the project holding information and documentation
Presentation	Prepare for and present the semester 1 progress
Ansible (Lab enviornment manager)	
Orchestrator Flow	Flow of the orchestrator
Deploy Basic EC2 with basic Playbook	Need everyone to fully understand Ansible by running through some examples of deploying to EC2
Configure EC2 through playbook	Need to customize the vm for deployment
Create VM from scratch	Fully custom make a VM from scratch
Hook Ansible to some backend call	Need the playbooks to be run from a backend service
Configure automation scripts	Configure the scripts to be run dynamically for the project
Create Full Demo	Need a demo of deployment for the client
Cloud Readiness	
Setup group cloud enviornments for team use	We need enviornments to be setup for group usage
Configure AWS for labs	Configure AWS to propperly host the lab environment
Configure Azure for labs	Configure Azure to propperly host the lab environment
Configure GCP for labs	Configure GCP to propperly host the lab environment

Task	Description	Deadline
Orchestrator Flow	Flow of the orchestrator	10/3
Deploy basic EC2	Need everyone to fully understand Ansible by running through some examples of deploying to EC2	10/2
Configure EC2	Need to customize the vm for deployment	10/7
Create VM From Scratch	Fully custom make a VM from scratch	10/13
Hook Ansible to Backend Call	Need the playbooks to be run from a backend service	10/20
Configure Automation Scripts	Configure the scripts to be run dynamically for the project	10/27
Create Full Demo	Need a demo of deployment for the client	11/15

Setup group cloud environments for team use	We need environments to be setup for group usage	10/2
Configure AWS for labs	Configure AWS to properly host the lab environment	10/20
Configure Azure for labs	Configure Azure to properly host the lab environment	11/3
Configure GCP for labs	Configure GCP to properly host the lab environment	11/15
Design Document v1 Final	Final version of the first design document version	10/4
Design Document v2	Final version of the second design document version	10/25
Final Design Document	Final version of the design document for the project	11/15
System Block Diagram	System Block diagram for the entire system	10/3
UI description	Description of the UI to give a basic path to how the UI should be built	10/6
Work plan	Plan of work for the project	11/1
System Design	Diagrams and description for the entire project	10/13
Detailed Design	An in depth review of the design of the project	11/4
Test Plan	Plan to test if the project meets the requirements	11/11
Course Site	Site for the project holding information and documentation	11/15

Table 2: Task Summary

2.2 Risks And Risk Management/Mitigation

All risks are evaluated based on the perceived impact that it might have on the completion of the project.

Risk	Evaluation	Mitigation
A cloud environment may	Low	This can be mitigated by using school resources to do the hosting of the application.This in

not be provided		combination with the free tier AWS should be enough for the initial development.
Cloud platforms may not have much overlap in functionality	Medium	This does pose a fairly considerable barrier to implementation, but we can mitigate using a few different techniques. Firstly, by prioritizing the most important cloud platforms. Second, by using Ansible we hope to minimize the feature set required by cloud platforms. **Note:** GCP and AWS both support Ansible
With all projects there is a possibility that members may leave	Low	While this is low likelihood, the best mitigation strategy is to not divide into clear roles, in which only one person knows how to fill any given role. We should all have a firm grasp of each area, so that we may spot fill in the event of a member leaving.

Table 3: Risk Table

2.3 Project Proposed Milestones, Metrics, and Evaluation Criteria

Orchestrator Flow	Software flow, how will everything flow together
Deploy Basic EC2 with basic Playbook	Playbook on to deploy a EC2 on AWS cloud platform.
Configure EC2 through playbook	Playbook on to configure a EC2 on AWS cloud platform.
Create VM from scratch	Create a VM image from scratch that we can use on the EC2
Hook Ansible to some backend call	Hook up Ansible for backend software calling and usage.
Configure automation scripts	Configure scripts that will perform automatic commands, and processes.
Create Full Demo	Create a demo for the client showing

	everything working from backend to frontend.

Table 4: Milestones Table

2.4 Project Timeline/Schedule

Orchestrator Flow	October 3, 2020
Deploy Basic EC2 with basic Playbook	October 2, 2020
Configure EC2 through playbook	October 7, 2020
Create VM from scratch	October 13, 2020
Hook Ansible to some backend call	October 20, 2020
Configure automation scripts	October 27, 2020
Create Full Demo	November 15, 2020

Table 5: Project Schedule Table

2.5 Project Tracking Tools & Procedures

- **Team Communication**
  - Discord
- **Source Code Repository**
  - GitLab
- **Task Tracking**
  - Jira
- **Procedure (Agile)**
  - Determine most pressing tasks from project assignments and sponsor
    - Place into our Agile Board TODO
  - Volunteer for, assign, and commit to project tasks in weekly meetings
    - Move to Agile Board In-Progress
  - Give status update on completed tasks in weekly meetings
    - Move to Agile Board Done
  - Hold each other accountable for completing assigned tasks by deadlines

2.6 Personnel Effort Requirements

Task	Person hours	Explanation
Orchestrator Flow	**10**	To establish a cursory flow for the orchestrator is fine.

Deploy basic EC2	**40**	For each member to complete a deploy and be knowledgeable on the topic 8 or so hours each seems fair.
Configure EC2	**40**	Seems fitting as, it will likely take some research, but ultimately shouldn't be too time exhaustive.
Create VM From Scratch	**70**	This will likely be one of the most difficult steps moving from using premade VMs. Has multiple parts, but this may also end up being completed much faster that expected.
Hook Ansible to Backend Call	**20**	With a backend established, having Ansible send a message should be fine.
Configure Automation Scripts	**10**	This may break this prediction, but it appears to be rather straightforward so long as the other steps are complete.
Create Full Demo	**L**	
Setup group cloud environments for team use	**10**	Should be rather quick environments that we can use are pretty easy to set up.
Configure AWS for labs	**25**	Fairly major step, but should be fairly simple.
Configure Azure for labs	**30**	The cloud platform that we as a group know the least about.
Configure GCP for labs	**30**	The second least known cloud platform.
Design Document v1 Final	**15**	Not too bad about 3hrs per person seems reasonable
Design Document v2	**15**	Not too bad about 3hrs per person seems reasonable
Final Design Document	**20**	Not too bad about 3hrs per person seems reasonable
System Block Diagram	**10**	Diagram shouldn't be too time extensive
UI description	**15**	Fairly easy, for the most part would be working with our client to ensure that it is acceptable
Work plan	**10**	Pretty straight forward, and shouldn't require

		much effort beyond the few man hours to implement.
System Design	**25**	While typically simple, it may require some research, as well as maintenance if it changes.
Detailed Design	**40**	This will be quick for the initial draft, but effort to keep it current seems reasonable to push this upward in people hours.
Test Plan	**30**	This could get complex, since there doesn't seem to be a very clean way to test the configuration via some framework. It may take research, etc.
Course Site	**25**	Once all steps are done, it should be pretty easy to set up, and fill out, but may take some time.

Table 6: Effort Table

2.7 Other Resource Requirements

There will be no other resource requirements.

2.8 Financial Requirements

There will be no financial requirements. This application will be developed with free tools, and we are using the free tier of the various cloud providers.

3 Design

3.1 Previous Work

- **Infrastructure**

  We will be using Iowa State virtual machines for the infrastructure that will be hosting both our UI, Backend API, and database.

- **User Interface**

  We will be utilizing React, a Javascript library for building user interfaces, to build our project's user interface. We chose to go this route because React will allow us to build a highly functional and dynamic UI extremely quickly. We will be able to do this because of all the external libraries that we can easily import into our React application. A couple examples of libraries we will be using to build our UI are: Bootstrap and Material-UI. Both of these libraries offer dynamic components that we can pull in and start utilizing immediately to create a clean and functional user experience.

- **Backend API**

  We will be utilizing Flask, a Python API framework, to build our Backend API. We chose to use Flask, because it allows us to get an API up and running extremely quickly, and allows us to use any package that is available to Python. Having access to Python libraries is extremely important, because libraries exist that will allow us to interact with all the major cloud providers. Additionally, there is a Python library for Ansible. This is ideal because we will be using Ansible's Infrastructure As Code functionality for provisioning and deploying our cloud environments.

3.2 Design Thinking

We wanted our design to be easy to develop and efficient, so we decided upon a separately hosted frontend and backend. Another thing that we thought was very important was to use the technology that made implementing our functionality easy and fast. Because of this, we chose React for our frontend and Python Flask for our backend.

Another design that we thought about implementing was a React frontend with API Gateway fronting Lambda functions for the backend. There were 2 main reasons we didn't go this route. The first reason is that we didn't want to provision a new AWS Lambda for each different service that the frontend needed. This wouldn't have been that difficult at all, but it would have been a hassle. In Flask, all we need to do is add another route to the code to provide another API service, and ease of development was very important to us. The second reason is that we wanted our client to be able to host this anywhere, and not just AWS. Using API Gateway and Lambda would have made that impossible.

3.3 Proposed Design

We have successfully experimented with each individual technology that we will be using during development (i.e. React, Flask, Ansible, and various cloud providers) to verify that they will be able to serve the purpose that they need to.

All technologies that we have chosen to implement will be able implement all of our functional and non-functional requirements that we have listed above.

User Interface:

A React application will be created for our project's user interface. This application will authenticate users via Google OAuth2 and then present users with the ability to define new lab templates, provision labs into various cloud providers with said templates, and view and destroy existing lab environments that were previously created. All of this functionality will be available to our React application via the API.

API:

A Python Flask API will be created to provide all the functionality to our React frontend. Flask will be used to route the API paths that will provide the various different functionalities that our project needs. Depending on what a specific route or path is doing, it may interface with Ansible or MongoDB. Ansible Playbooks are what we are using to manage all of our cloud infrastructure, and MongoDB is what we are using to store any data that we need that isn't a lab template. All lab templates will be stored on our server's local filesystem.

Cloud Infrastructure (Labs):

We are giving users the ability to create labs within various different cloud providers such as AWS, GCP, and Azure. A lab will be created with the user-selected template, and placed into a network on the selected cloud provider that will make it accessible to those trying to utilize the lab.

3.4 Design Analysis

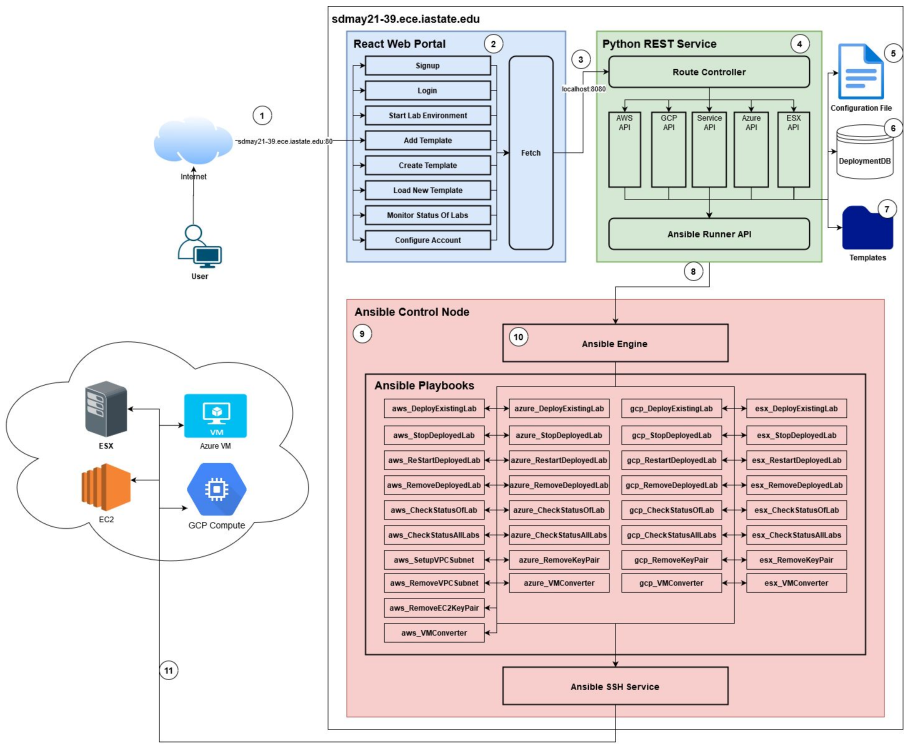
From the design standpoint, the work that was performed with 3.3 stayed the same. After multiple conversations with the client we accomplished a well fleshed out design that has stayed consistent throughout the semester. As we push forward into development our design proves to stay the same and efficient across all needs and wants when it comes to the client and work performed for the Cloud Environment platform. The design integration with cloud platforms along with third-party solutions like Ansible, overall construct layers of interfaces and services that work hand-in-hand to provide the required solution for the client.

3.5 Development Process

Using weekly meetings, and issue tracking we perform the development on the basis of weekly development. This process allows for continuous integration and progress on creating the solution. We use techniques like the roadmap approach to keep track of what needs to be done vs soft features that would be nice to have. With weekly meetings it allows us to keep track of tasks by person along with discussing what needs to be done and what has to be done next. This approach follows the SCRUM and Agile ethics while keeping the end goal in mind. Weekly meetings allow us to client-face which in returns gives us more opportunities to ask important questions that may save time in the long run. This approach allows us to keep development to high standards and quality for the client.

3.6 Design Plan

This project involves giving information to a user and giving that user control over deployed environments. Due to this, a user interface is required to give the user a convenient and easy access to the tools of the project. The user interface will then need to connect to a backend service to do the logic for every use case. In order to administer resources to the deployment platforms, there will need to be an environment orchestrator that connects to different cloud and on-premises platforms. To connect all of the individual pieces together, we abstracted everything into modules that connect to each other via interfaces.



1. User accesses the interface.
2. A React Framework interface containing pages for each use case.
3. The UI calls a backend service via HTTP REST requests.
4. A Python REST service that handles the logic for each use case.
5. A configuration file containing the configuration for several platform accounts.
6. Database containing the tracking ID and information on every deployed lab.
7. Filesystem folder containing templates to be used as predefined lab environments
8. The REST service utilizes an Ansible API that sends commands to Ansible
9. The Ansible Control node is a system that contains the Ansible
10. The Ansible Engine calls playbooks and runs them to interact with cloud
11. Ansible accesses the cloud platform via a secure SSH session

4 Testing

4.1 Unit Testing

Testing

Each class, structure, and logical module should be tested using python's “unittest” library.

Deriving Coverage

The coverage can be derived using python’s coverage.py. This will allow us to track our progress toward our 80+% coverage goal.

Objective

80% code coverage

4.2 Integration/Interface Testing

Interface Testing

It is important for each of our web facing APIs to be well tested and well defined. To this end we must have a good procedure for testing both our web server, as well as the engine endpoints.

To do so each interface should be appropriately tested along all routes, and operations. To do this we intend to use a REST service test utility(hoppscotch.io or postman). From here we can insure the functionality of each interface as a whole unit.

Interface List

- Front end server web interface
- Back end server REST API

Integration Testing

For this we can still utilize the tools from the interface testing plan, but now tracking effects across the environment.

Additionally we can recruit a targeted form of blitz testing to hit an integrated environment prior to moving into the well defined stage of acceptance testing.

4.3 Acceptance Testing

Acceptance Testing would be the final stage of testing that would be completed after the Unit and Integration Testing. The entire application should be fully developed and should be able to function as expected. When working on the Acceptance test, we would make sure that assumptions and the constraints are to be considered ahead of the time. For instance, the execution of a certain use case of the application can be affected by different operating systems and web-browsers; the acceptance Tests should be performed on each operating system and web browser that would be specified ahead of time. An example of the Acceptance testing could be if the user would be successfully able to select/design template by following steps:

1. User is able to login/ signup through his/her credentials.
2. User is able to select the Environment that would be needed
3. Able to add the template.
   a. By either loading a template or,
   b. By creating a new template

The expected result should be:

- The user should be able to successfully publish the template.

For Non-Functional Testing, we would take care about the performance of the Project. By Performance we would make sure that there is no wait time to load the pages. We would also be keeping security of users data as our main concern. We will try to make use of Authentication for each user. We will make sure that user credentials would be used by the user itself and no other user would have access to it.

We would try to engage our client and professor to perform acceptance testing by using the application at weekly meetings, and will demonstrate our current progress and functionality to meet the requirements.

4.4 Results

At this time we do not have any automated testing, which includes but aren't limited to: integration testing, unit testing, and cloud testing.

5 Implementation

The software implementation of the Cloud Management Platform is designed to solve real-world problems that clients are currently facing. This implementation will of course come in three different phases, the first phase being design documentation and diagrams of the software infrastructure along with understanding the core functionality of third-party software. The second phase is the prototype of the proposed solution to get a better understanding of what needs to be done along with what our client would like to see more of. This phase helps us as it provides a guide to overall help us change and mold the end product into a finished product. Finally, the third phase consists of everything that will be needed to be finalized for our client to use the said solution. This phase will have a high chance of being broken down into sub-phases as the development of the prototype phase continues.

A simple yet effective implementation of the infrastructure presented in section 3.6 is the goal of the second phase. As we reach the end of the second phase we see a constant need for more infrastructure and documentation of third-party cloud technologies. This is also true when it comes to communication with our clients. As we push forward toward our implementation using software technologies including but not limited to Python and React we find more communication is needed for the smaller details. The implementation of this software solution is split across the team evenly to create an equal work balance to help make sure the whole team understands the implementation of the design mentioned previously. We believe this implementation will give our client the best version of the end product.

6 Closing Material

6.1 Conclusion

As we look toward the future of development of the proposed solution for our client, we see constant improvement, constant communication, and change that comes with it. We have developed and implemented a prototype of the said solution to provide a concept that the client will overall enjoy and help guide us toward a finished cloud product. With this first semester wrapping up we continue to implement the designs proposed in this document along with any diagrams shown within it. Using third party cloud solutions, python, and other technologies we overall feel confident in our ability to finish the mentioned prototype and begin working toward a finished product next semester. This product will overall change throughout the process of development and communication with said client. With the coming change and development of the solution we are confident in our vision and dedication toward our client's vision of the proposed cloud product. Working hand and hand developing services, modules, and infrastructure with third party cloud technologies is just one of many ways we can accomplish this. Thank you for giving us your time and consideration when reading.

6.2 References

[1] "Chef: Enabling the Coded Enterprise through Infrastructure". [Online]. Available: https://www.chef.io/ [Accessed: 1 September 2020]

[2] "Amazon Web Services (AWS) – Cloud Computing Services", Amazon Web Services, Inc.,2020. [Online]. Available: https://aws.amazon.com/ . [Accessed: 15 September 2020]

[3] "Microsoft Azure", Azure.microsoft.com,2020. [Online]. Available: https://azure.microsoft.com/ [Accessed: 20 September 2020]

[4] "MongoDB: The Most powerful database for modern app" [Online]. Available: https://www.mongodb.com/ [Accessed: 25 November 2020]